

```
gets(message,SIZE,fd);
rror(fd)) { perror("fgets"); }
!feof(fd) && !ferror(fd)) /*

g = strlen(message);
&& message[lg-1]=='\n') { m
("%s\n",message);
fgets(message,SIZE,fd);
rror(fd)) { perror("fgets")

close(fd);
!=0) { perror("fclose"); }

0;
```

# Problem solving through Programming In C

**Prof. Anupam Basu**  
**Computer Science and**  
**Engineering**  
**IITKGP**



# INDEX

S. No	Topic	Page No
	<b><i>Week 1</i></b>	
1	Introduction	1
2	Idea of Algorithms	10
3	Flow Chart and Pseudocode	20
4	Introduction to Programming Language Concepts	32
5	Variables and Memory	45
6	Types of Software and Compilers	59
7	Introduction to C Programming Language	73
8	Variables and Variable Types in C	89
9	Introducing Functions	102
10	Address and Content of Variables and Types	118
	<b><i>Week 2</i></b>	
11	Assignment Statement and Operators in C	137
12	Arithmetic Expressions and Relational Expressions	153
13	Logical Operators and Change in Control Flow	171
14	Use of Logical Operators in Branching	183
	<b><i>Week 3</i></b>	
15	Branching : IF - ELSE Statement	202
16	IF-ELSE Statement (Contd.)	218
17	Switch statement	234
18	Switch Statement (Contd.) and Introduction to Loops	248
19	Implementing Repetitions (Loops)	263
	<b><i>Week 4</i></b>	
20	Implementation of Loops with for Statement (Contd.)	274
21	For Statement (Contd.)	284
22	Example of If-Else	298

23	Example of Loops	310
----	------------------	-----

### ***Week 5***

24	Example of Loops (Contd.)	321
25	Example of Loops (Contd.), Use of FOR Loops	334
26	Introduction to Arrays	347
27	Arrays (Contd.)	359
28	Arrays (Contd.)	372

### ***Week 6***

29	Program using Arrays	384
30	Array Problem	396
31	Linear Search	408
32	Character Array and Strings	422

### ***Week 7***

33	String Operations	445
34	2-D Array Operation	460
35	Introducing Functions	474
36	More on Functions	486

### ***Week 8***

37	Function (Contd.)	503
38	Scanf and Printf Functions; Function Prototype	522
39	Parameter Passing in Function Revision	540
40	Parameter Passing in Function Revision (Contd.)	555

### ***Week 9***

41	Substitution of # include and Macro	573
42	"search" as a function	589
43	Binary Search	600
44	Binary Search (Contd.)	612
45	Sorting Methods	629
46	Bubble Sort (Contd.)	644

## ***Week 10***

47	Use of Pointer in Function : Context Bubble Sort	655
48	Arrays at Strings	666
49	Data Representation	682
50	Bisection Method	694
51	Interpolation	716
52	Trapezoidal Rule and Runge-Kutta Method	727

## ***Week 11***

53	Recursion	753
54	Recursion(Contd.)	768
55	Structure	782
56	Structure (Contd.)	798
57	Structure with typedef	814
58	Pointer	829

## ***Week 12***

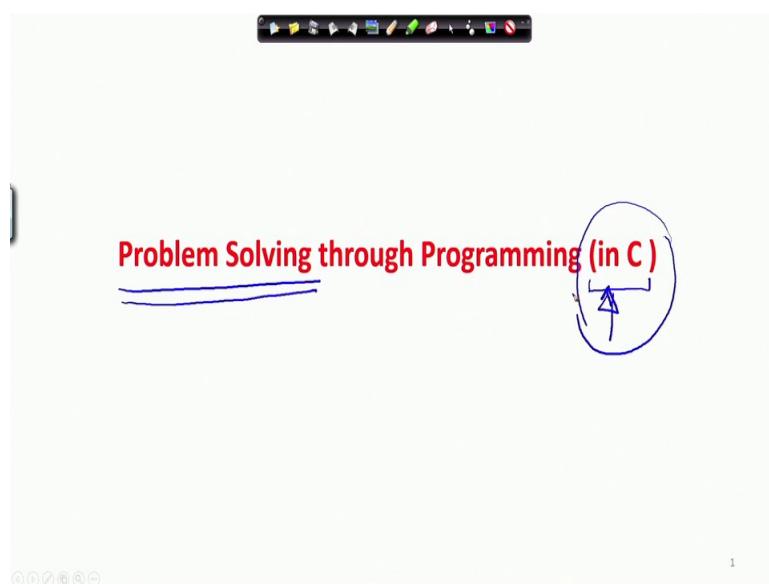
59	Pointer (Contd.)	850
60	Pointer in Structures	868
61	Dynamic Allocation and File	883

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 01**  
**Introduction**

Good morning, welcome to the course of Problem solving through programming in C.

(Refer Slide Time: 00:20)



This is a course of programming no doubt about it the first course on programming for many of you, but also the focus of the course is to show you how we can carry out problem solving; that means, solving different problems using programming. We encounter a different types of problems all the times in our life and some of them can be solved using programming, why I am saying that some of them can be solved not all can be solved there is a reason for that we will come to that later, also there is another thing in the bracket you can see that we will be discussing programming in C.

Now, C why I have put it in the bracket the reason is the C is just one of the means one of the languages using which we can do programming, there are different other languages like as many of you know Java, C plus plus and others using which we can also do programming. Now our objective is to not to just think about the c programming language, but also some programming principles, some programming logic, some programming methods that are adopted while solving problems all right.

Now, let us think of some of the problems that we often encountered in our regular life, let us start with a very simple problem of there is there is a large volume of data large volume of data maybe integer or may be numbers different types of numbers.

(Refer Slide Time: 02:38)

Mean = Average →  $\frac{x_1 + x_2 + x_3 + x_4}{4}$

n numbers       $\frac{\sum x_i}{n}$

$x_i$        $x_1 \quad x_2 \quad x_3 \dots x_n$

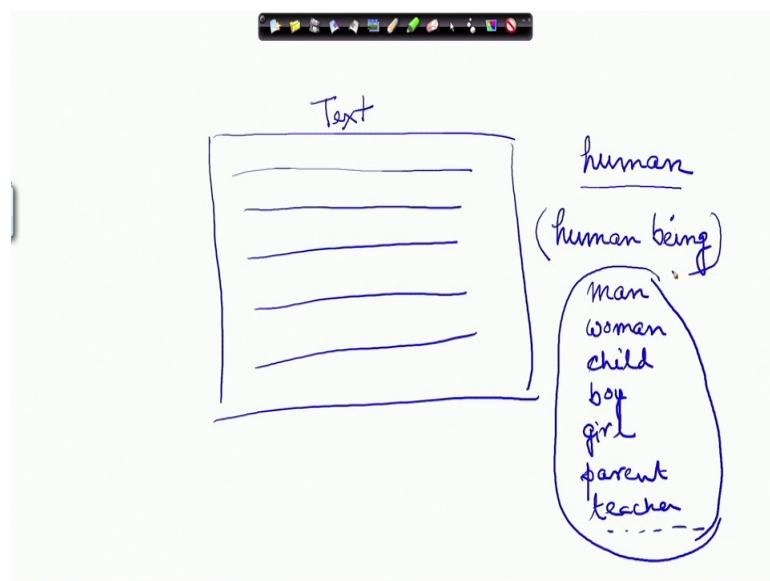
And we have to find the mean of that and mean of that many of you know mean is nothing, but the same as average. Now how do you find out the average of a large volume of numbers now all of you know how that is done, that is a relatively simpler problem. We can also have, this is this has got some standard formula if you have got 4 numbers say  $x, y, z, p$  you add them and divide them by 4 that is how we find out the average right.

Now, also for  $n$  numbers if there are  $n$  numbers then we can write it as  $\sigma x_i / n$  what does this mean, this means I am taking what is  $x_i$ ,  $x_i$  is each number,  $x_1$  is one number,  $x_2$  is another number,  $x_3$  is another number and so on up to  $x_n$ . Now each of these numbers any of these numbers I can represent with  $x_i$  and by replacing different values of  $i$ , I can get any of these numbers. So, sum of all these numbers sum of all these numbers can be designated as  $\sigma x_i$  this sigma means summation and, I do the summation of this and divide by  $n$  when it was 4 number I divide it with 4 here since there are  $n$  numbers I divide it with  $n$ .

This is a very well known mathematical formula mathematical approach that all of us the school children will, Now, therefore, we can very easily translate this problem and solve

this problem into programming, we will see how we can translate it to programming also we will also discuss why we should translate it to programming all right. Let us take a let it I mean a little different problem suppose there is a text all right there is a text of 20 sentences all right.

(Refer Slide Time: 05:29)



We will have a text of 20 sentences all sentences are here and I say that find out how many times the word human occurs in this text, this is my text and I want to see how many times the word human occurs in this text also the problem can be much more complicated.

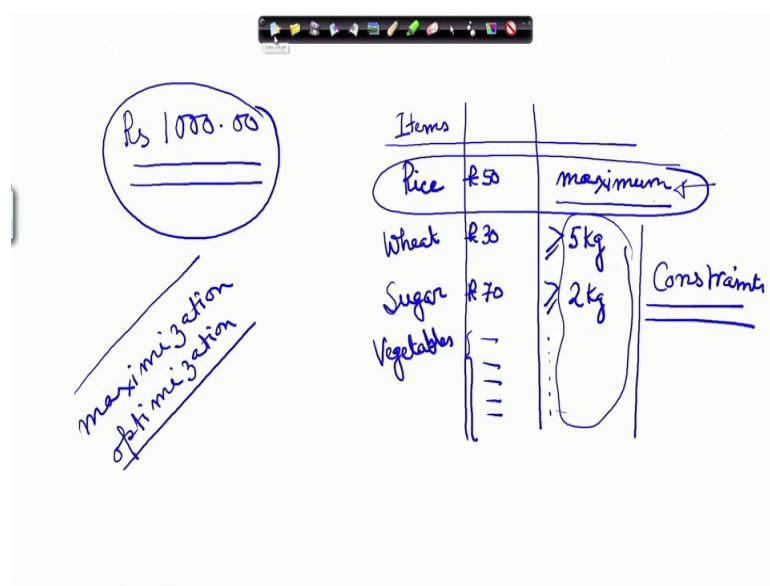
This is another type of problem where I have to look every word and see is it a human no is it a human yes. So, accordingly I will have to go on counting all right now I can also say that how many times does, the word pair human being appear in this text, so that is can also be, I get human, but I do not get being after that, I do not count it all right. If it be human followed by being then I count it once in that way I can formulate different steps by which I can solve this problem. If you just spread your imagination you can think of making the problem even complicated for example, human being and man, woman, child, boy, girl, parent, teacher etcetera all of them are human beings.

Now, if the problem is posed in a way that find out how many times earlier what the problem was earlier the problem was how many times the word pair human being appeared as it is. Now if I modify the problem statement and say that find out how many

times human being has been referred to in this text; that means, if it be a man then also it is a human being, if it be a woman then also it is a human being, if it be a child then also it is a human being, if it be a grand grandparent it is also human being, In that case the problem becomes a little more difficult little more difficult than the previous version of it where we wanted to just find out the word pair human being this is an example of a second problem and third problem.

The third problem is a little more complicated as you can see, we can go on adding examples of problems let us take another one suppose you have got a fixed amount of money.

(Refer Slide Time: 09:16)



All right whatever that amount is suppose you have got 1000 rupees and you have got you have been asked to buy some items may be rice, wheat, sugar, vegetables etcetera and each of them have got a price rice may be rupees 50, wheat may be rupees 30, sugar may be rupees 70 and vegetables depending on the variety there are different prices all right.

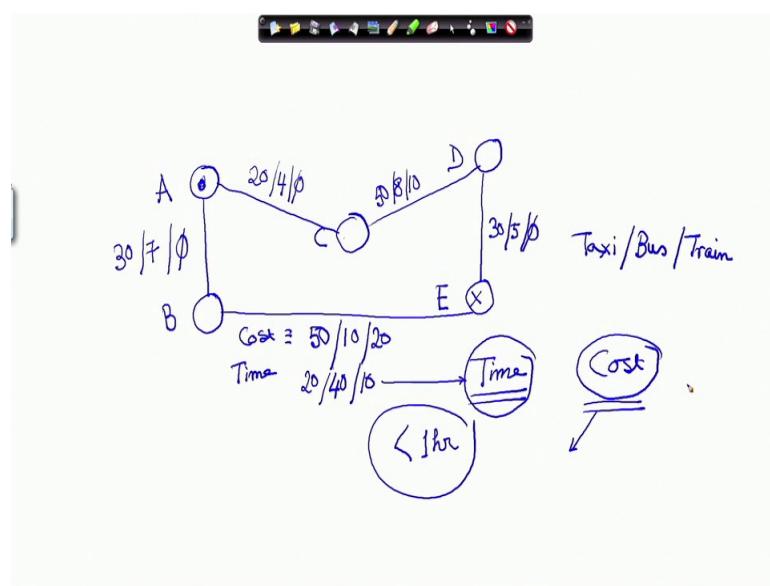
Now, suppose you have been told that you have been given this amount of money and you have to buy the maximum amount of rice possible the maximum amount of rice possible. So, with 50 rupees you can have 20 kgs of rice, but there is a constraint, there is a maximum amount of rice you have to buy, but you have to buy at least 5 kg of wheat, at least 2 kg of sugar, at least some amount of vegetables. Now given this you can buy 5

kg of wheat, you can buy 6 kg of wheat, 7 kg of wheat, you can buy 2 kg of sugar or 3 kg of sugar etcetera you can do many things, but you have to see that how you will distribute this so, that even after satisfying these requirements you can find you can buy the maximum amount of rice.

There is another problem when you go to the market I mean in such a thing is always told or we have got in mind that we have to cover these items and there is a fixed there is a cost for that and we have to satisfy the cost. This is another type of problem we can see this is an maximization problem or in problem solving terminology we also sometimes call similar problems as optimization problems; that means, I want to maximize the amount of rice that I want to buy, but these are the constraints that I have to satisfy right. So, I have to satisfy these constraints after satisfying these constraints, how can I maximize this, this is another type of problem relatively much simpler maximization or optimization problem.

Let us now move to another type of problem say I have got a number of places let us name the places A B C D E.

(Refer Slide Time: 13:02)



Now, suppose these are, suppose some cities and we have got direct paths among some cities and there is no direct path among some cities all right suppose this is the scenario. So, I can directly reach from A to C, but I cannot directly reach from A to D I can directly reach from B to E, but I cannot directly reach from B to C like that moreover

along with each of these paths we have got some cost associated the amount of money the amount of expense that we will have to bear in order to make this travel possible, suppose this traveling B to E I have got a different options say for taxi first I put the taxi thing first, then I put the cost by bus and then I put the cost by train now suppose from B to E I can go by bus taxi and train both.

So, suppose by bus it is 10 rupees and train it is 20 rupees and also of course, there can be I mean according to the mode of transport that I take the time taken will also be different, say from D to E I can go by taxi and that will cost me 30 rupees and by bus it will cost me 5 rupees and there is no train between these 2 all right, there is no train no I put it null here. So, A to B suppose I have got taxi of course, 30 rupees bus may be here in this route it is a little expensive 7 and there is no train here, A to C I have got 20 rupees by train, 4 rupees by bus and there is no train, here C to D there is 50 rupees, again 8 and by train it is say 10 rupees.

This is the picture now I am here and I have to reach this point from A to E now I can go via A to C, C to D, D to E or I can travel A to B, B to E right, if I come by from A to E by taxi then it will be 30 plus 50, 80 rupees required, if I come via this path it will be 20 plus 50, 70 plus 30, 100 rupees required, but if I come here by train there is no train. So, so I cannot come between these 2 by train if I come by bus 4 rupees and now by train 10 rupees 14 and here there is no train I cannot do that right, I cannot take train from here.

I will have to take again bus from here, in that way I have to solve the problem of how to reach from one city to another with some constraints now these constraints are coming every time and this constraints actually generate the fun in this what are the constraints that we can have in this problem the constraints it will be satisfying the time how fast I can move I have not shown the time information here that time information should also be kept here for example, from B to E by a taxi it takes 50 rupees and say takes 20 minutes and by bus 10 rupees, but it takes 40 minutes and by train is 20 rupees, but it takes 10 minutes like that.

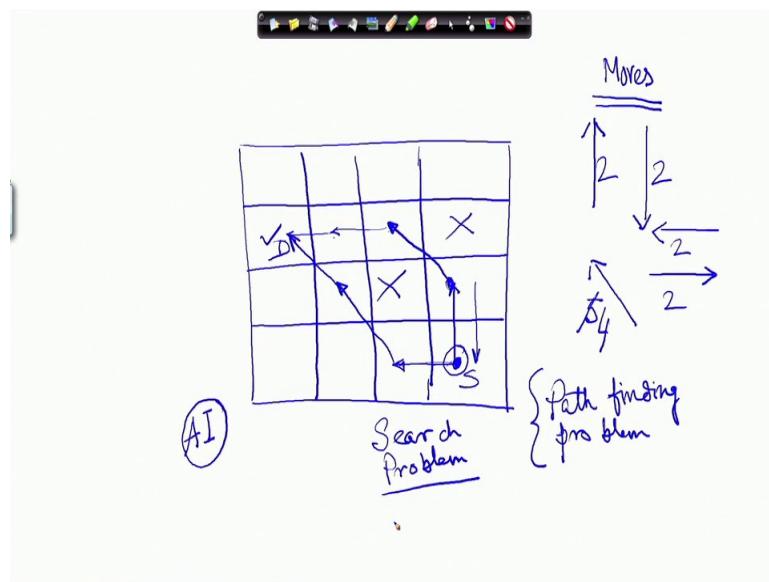
So, if the time information is also there I may be asked to minimize the time and also what is shown here this is cost, I have got the cost and the time, I may like to minimize the cost also. I may be asked that I am not much bothered about time I am relaxed it is a weekend I can devote time, let me try to minimize the cost then the problem will have

one color right one form. If I just say that well now I am little pressed with time, I will be selecting a vehicle or selecting a mode of transport I do not mind the cost, but I have to minimize the time then the problem is something different also it may be that I am with the total time spent between a less than one hour less than one hour even this constraint minimize the cost, in that way different forms of this same problem can be put forward and often we have to solve such problems in our real life.

Let me, that is the best way to reach a place right this is how we can find a best way to reach a place.

Now, I come to another interesting problem, this problem that was there it is again an optimization problem again you could see now say let us take another example.

(Refer Slide Time: 19:24)



Although I am showing it as a game it can act as a model for many real life scenarios. This is a maze and I am here and I have to reach say here, this is my source, this is my destination and from here I can either from any place each of them are places from any of these places I have got 3 moves, either I can have an up move, I can have a down move, I can have a diagonal move all right.

So, for example, from here I can with a diagonal move I can come here with an up move I can come here I cannot apply a down move now this up and down moves are allowed only if there is no bar or there are other moves also I can move left or I can move right. I

cannot move diagonally down suppose these are the moves that I have and any of these moves can be applied only if the corresponding destination place is not bared or if the destination place is free for example, if this place is bared and say this place is bared then from this point source I cannot apply the diagonal move why because this place is not free.

Suppose I apply the up move, I can I will go from here to here now I am here at this point I have what are the moves that are applicable left move cannot be applied this cannot be applied this is out of question, up move cannot be done down move I can do you see down move I can do I can come back alright and diagonal move is possible suppose I come to I find if I apply the down move I come here then; obviously, if I again apply this move then I will be just doing these 2 repeatedly and I will not progress any further may be once I come back here because of some thought I can apply this move all right.

Let us see once again let us start suppose I was here now apply a up move, then I can apply a diagonal move and then I can move a horizontal move, another horizontal move 1 2 3 4 assuming that all these moves are having the same cost, same effort all right then with 1 2 3 4 moves I can reach my destination. Let us see if I had instead of going up if I had taken this move 1 because I could not go diagonally then I move diagonally 2 then with 3 moves I can reach the destination assuming that the costs are same then this is a better move.

If the costs that is scenario becomes a little more complex when each of these moves I have got different costs it is I have to pay more or I have to put in more effort for moving from this point to this point, from this point to this point in a diagonal manner is much more easier to move in a horizontal manner all right. This is another type then the problem would have been little more complex right in that case suppose the diagonal move will costs you 5 and all other moves cost you 2, then what would be the cost in this 2 5 7 8 9 10 11.

In this path let us do it again 2 because up is 2, 5 because diagonal is 5, 7 8 9 10 11 units if I have done in this way 2 5 7 and 5 12. So, in that case this is becoming costlier, but; obviously, if the diagonal move was 4 and not 5 then what would have happen let us see

2 4 6 7 8 9 10 and if I do it in this way what it would be 2 4 and 2 6 and 4 10 in that case both of them would be equal.

This is another problem where again you can see that we are trying to look at the cost and solve a problem, this you can think of a path finding problem or a search problem what are we searching here I am calling it we are search problem. So, what are we searching here, we are searching for the path through which we can move we can reach the objective, now this is a very simple example of the type of problems that we solve in artificial intelligence, say a robot has been asked to move from this point to this point how the robot will find it is way given some obstacles.

So, this is another type of problem that we can solve using programming now having said that is it that all the problems that are there in the world can we solve them by programming when you say problem solving we are not actually meaning to address all possible problem solve the problems that are in the world we are not saying that all of them can be solved by programming for example, the problem of hunger cannot be directly solved by programming some person is feeling depressed he is feeling sad that is a problem we are not addressing that sort of problem here.

So, when we say problem solving through programming then we are talking of not all kinds of problems, but certain categories of problems, that is that will be discussing in the next module what are the problems that can be solved by programming and we will progress further.

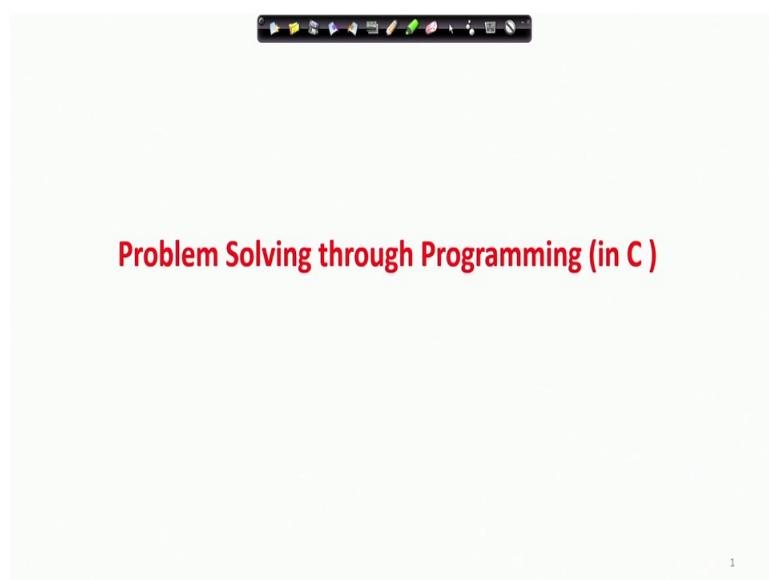
Thank you for this module.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 02**  
**Idea of Algorithms**

So we move to the next module of this course.

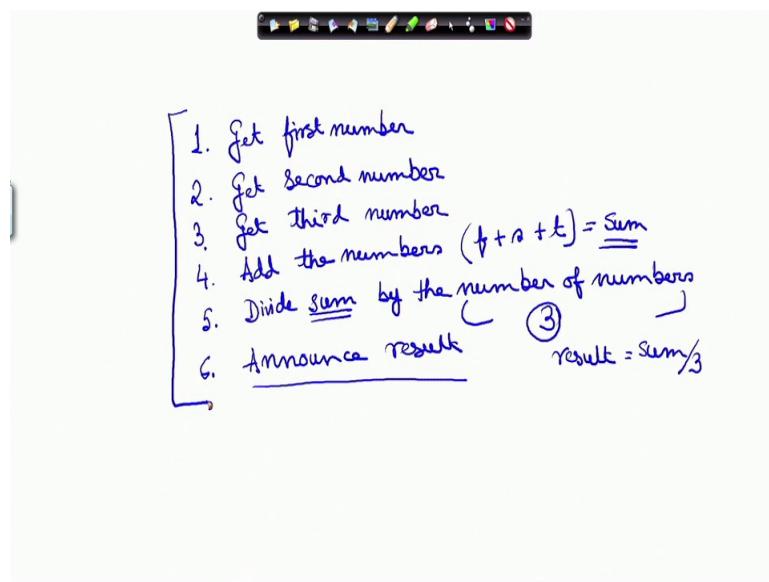
(Refer Slide Time: 00:19)



In the earlier module we have seen some example problems, and the problems which can be some example problems which can be solved through programming using computers. And also we said that all problems cannot be solved by computers. So, what are the problems that can be solved using computers? That is a natural question that can be asked. The answer to that is that, if it is known what are the steps that we must execute or perform in order to arrive at the desired result then we can solve it by programming.

Now, those steps may be directly known for example, in the case of finding the average marks in the class for n students, we know the steps right. So, let us see if we write down the steps following which we can solve this problem, how would it look like?

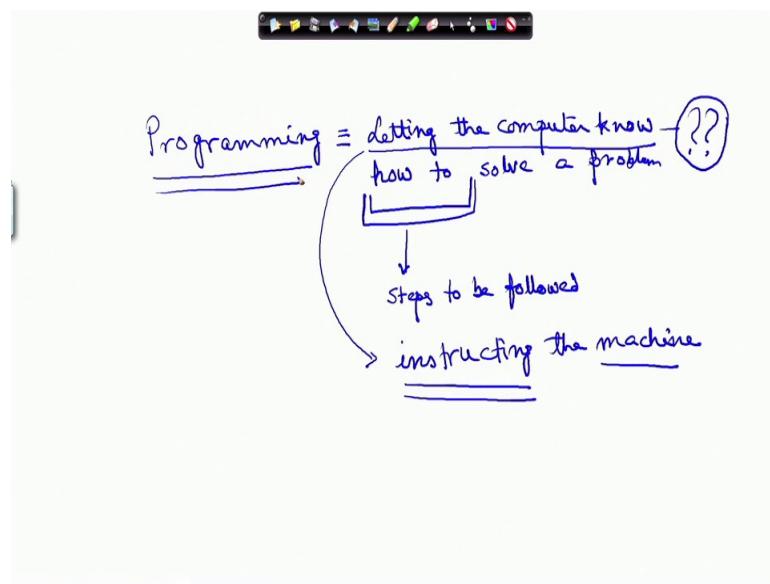
(Refer Slide Time: 01:59)



I want to read one number, suppose I am reading phone numbers, I want to find out the average of phone numbers, I want to find out the average of phone numbers read. Read means I just know get I can also say get first number, 2 get second number suppose you want to find for 3 numbers; get third number add the numbers; that means, first number plus second number plus third number and let that be the sum and then I say divide sum is this sum by the number of numbers and what is the number of numbers? Number of numbers here is 3, because there are 3 numbers and then say announce result.

So, this one divide the sum by numbers. So, that is something like say something like this that result is equal to sum divided by 3 in this case right and announce result. So, these 3 these steps are very well known, very clear and since these are very clear. So, I can solve them by programming. Why because programming what is programming? Programming is informing a computer as how to solve a particular problem. So, I can say let me just do it a fresh.

(Refer Slide Time: 04:53)

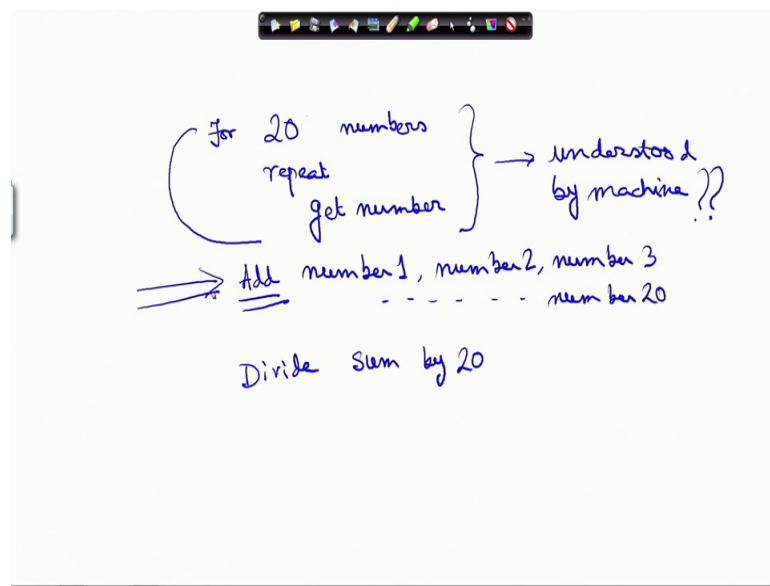


Programming is nothing but, letting the computer know how to solve a problem.

Now, there are some important aspects to this sentence this phrase. Letting the computer know is one important thing, but what do you let the computer know? How to solve a problem? So, how to means I have to tell the computer, that what are the steps to be followed to solve a problem. I have to be very clear about it and I have to let the computer know. So, there is a big problem here how I let the computer know I will come to that later.

The steps to be followed are what we have to tell the computer. So, in the earlier example we just saw that the steps for finding the mean of 3 numbers are very clearly stated, and I can inform the computer often this is informing the computer is also known as instructing the machine. The machine is nothing but the computer itself and these are the computer instructions. A programme therefore, consists of a number of instructions to be executed. Now the example that we saw was for adding three numbers now we will soon see that the thing becomes a little more complex if I wrote get first number, get second number, get third number suppose there are 20 numbers, then I will have to write them 20 times get first number, get second number, get third number like that upto get twentieth number that is boring is not it.

(Refer Slide Time: 07:31)

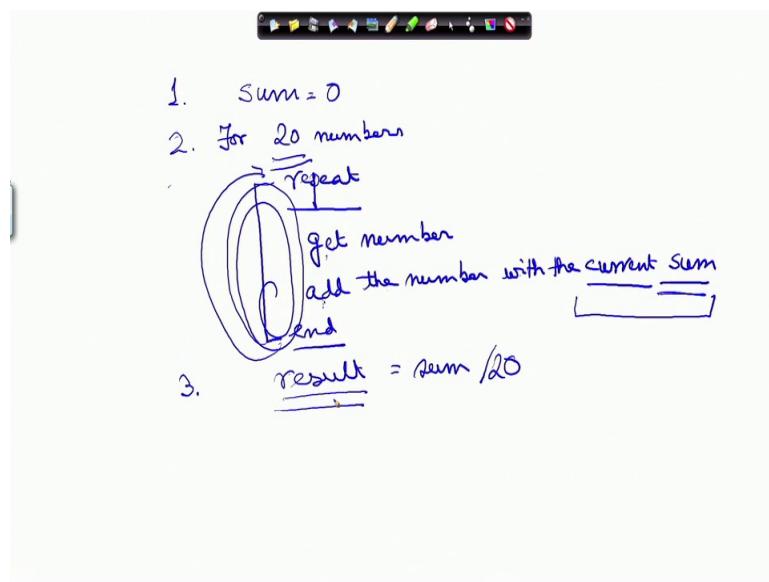


So, instead I can possibly write that for 20 numbers let me do it in a let me do a number here for 20 numbers repeatedly get number, aright that is possible.

So, for 20 numbers I will be getting the numbers. So, I do not need to write down one get first number 2 get second number 3 get third number not like that, I can straight away with one instruction I can express myself, but whether this will be understood by the computer or not is a different question. So, I am keeping that question open. Understood by machine and whenever I talk about machine, I mean a computer. So, for 20 numbers get number, in that way I can say that or say I could have written that in a little more smatter form let us let me try to do that.

Now, if I do that then what would be the next one? I have not I have to add those 20 numbers. So, one thing is that again I can write here add number 1 number 2 number 3 upto number 20, then divide sum by 20 this is one way of saying that, but this is again boring right this is very boring, I mean it is not interesting to look at this why should I write this in this dot dot dot form instead say I can write it in a smatter form say I write it in this way and you.

(Refer Slide Time: 10:07)



Just see whether you can understand what I am writing for 20 numbers, I am using my own language, it is not a computers language, it is an English and it is a version my version of English. I just need to express it to you and then whether that will be understood by the computer or not is a different question and we will address it separately ok.

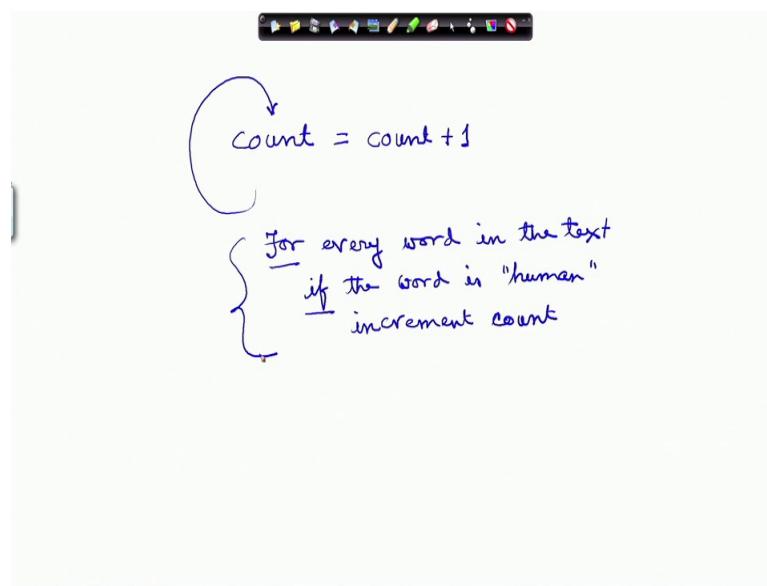
For 20 numbers get number sorry I say repeat get number or let me add the number with the current sum; that means, presently whatever is the value of sum, I add that number that I read with that. Now initially therefore, what should be the current sum? Initially when I am starting the current the sum is 0. So, I can say let us sum be 0 initially. Now I get a number at the number to the current sum and this repeat I am just using my.

So, just showing as if it is a bracket, that this part I am repeating I am repeating this part how many times I am repeating this part? This part I am repeating, how many times I am repeating? 20 times because for 20 numbers I am getting the number adding the number getting the number, adding the number and I am going on doing this and adding the number and I am keeping that number in sum. So, ultimately I do it 20 times and then this part is finished then what I have to do. So, this was step 1, this was step 2 this was step 3. So, this step 3 was repeated 20 times I am sorry let us make it sorry actually this is the second step, which I did 20 times and after that I have got the all those numbers added.

So, next I come and I say result is sum divided by 20. So, that will be my result and then I can announce the result frame the result whatever. So, here you see in the earlier example what I did is, read first number, read second number, read third number read forth number like that here I just expect do it in a smatter way that for 20 numbers do this activity repeatedly 20 times. So, that is another way I can express it. And that shows that I can specify very clear steps by which the problem can be solved and since I can specify the steps clear steps through which the problem can be solved, this problem can be solved by programming. I can programme a machine to solve this problem.

So, similarly say how many times the second problem that we had looked at, how many times a particular word human occurs in a particular text. So, I have to do something like this, I have to open the texts I have to see look at the text and read a word and see is it human? No, is it human; no is it human yes. So, I will have to have some sort of counter or a count, which I will increase every time I am sorry every time, I encounter the word human.

(Refer Slide Time: 15:13)



And this will go on as many times for every word in the text. So, I can write that down as for every word in the text if the word is human increment count. So, this will go on only if the word is human I will increment the count otherwise I will go on reading the words. So, this is the very clear step which I can express to the computer in its own way. So, that it can find out how many times the word human has occurred.

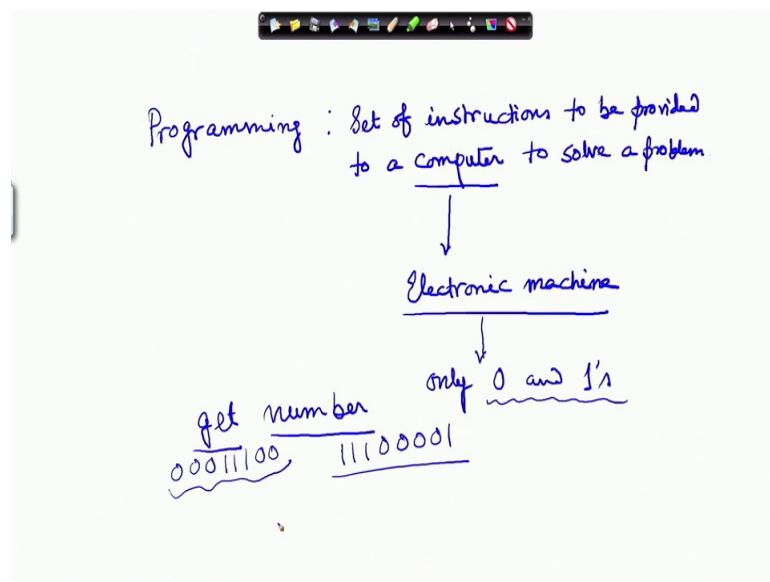
Now, there are other different variants of this that I have told you that, whether its a human being or whether it is an equivalent man, a woman boy girl whether it refers to a human being or not then; obviously, the instruction will be a little more complicated than this in these way. So, I once again come back to my state when that, what are the problems that I can solve using a computer? I can solve the problems where I can enunciate, I can express the clear steps set of steps one after another by which I can solve the problem correctly.

So, that is now for example, now if we take the example of that searching in a maze, where I was trying to use the diagonal up down whichever, there also I can express some intelligent ways by which I can instruct the computer to approach the problem. But everything as I said do not render themselves, to such enunciation of very clear steps I do not really know. I really do not know exactly that may be either the problem is not well understood, I mean a student of friend of yours is feeling depressed is not feeling well now there may be you really do not know exactly what is the reason for that. If you know the reason then you can try to solve it help him out.

Now, when the problem is not well understood then of course, we cannot solve it in such clear through such clear steps. If the information that you have you are getting all are not very reliable then also there are ways and means by which we can think of how we can get a good enough solution. Another thing is that there are some things which we do not know. So, we cannot solve that using programming or may be some cases where. So, the problem is so complicated for example, solving the problem of hunger all. Right now obviously, if you say buy food give food by that hunger will be solved, now that is too simplistic solution and that is not a realistic solution there are many angles many interacting variables, which are working towards that. So, that is not directly amenable to solving using programming.

So; however, we are been able to give an idea of the category of problems, which can be attempted to be solved by programming. And I have also said what is programming once again programming is the set of instructions.

(Refer Slide Time: 19:58)



I am rewriting it in different way, set of instructions to be provided to a computer to solve a problem. Now a computer is an electronic machine, till now what I was doing I was writing the set of the statements in the form of somehow like English. So, you could understand that, but the computer being an electronic machine will not be able to understand that. An electronic machine is made of switches. So, since it is made of switches it will understand only it understands only zeros and ones.

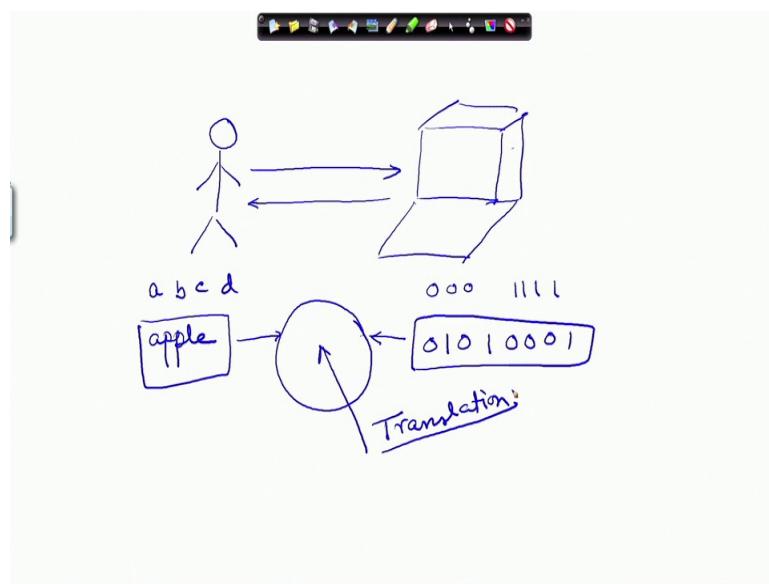
So, whatever I say get number, now each of these get a number is one simple instruction or you can always understand this you can all of you can understand this, but for a computer I will have to somehow write it in the form of some zeros and ones. So, maybe something like this 000 111 00 suppose this is representing get. A number can be this anyway be are that number can be say 111 000 101 something of that. So, that is the completely different type of expressions alphabets. So, the way we are writing it here cannot be directly understood by the computer. I think in the last lecture I had mentioned that how to let the computer know the steps.

Now, if I want to let somebody know of I want to express myself to somebody, I must do it in the form of a language that he or she understands for example, whenever I am talking to French person, I will have to talk in French otherwise he will not understand if he does not know other languages right. Similarly if I encounter an alien for example, I will have to talk in his language or I will not understand his language. Therefore, but

how do you do that if I meet a person who does not understand my language, some the natural solution is that will have somebody who will be acting as an interpreter, who understands both my language and the other parties language. So, we can understand my language and convert it to the other language.

So, what is the other parties language. So, what are the parties here let us see.

(Refer Slide Time: 24:02)



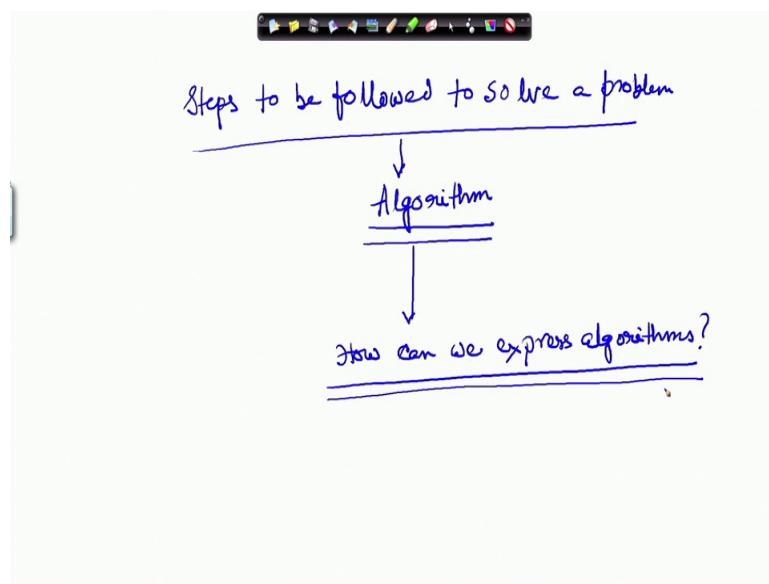
On one side it is we who want to do something, on the other side we have got this machine called computer, alright this machine which is lying in front of you.

Student: (Refer Time: 24:25).

Now, I have to communicate with this machine and this machine will have to communicate to me. And this communication, but this understands its word consists of zeros and ones and my word consists of a b c d's, and with that I can say apple, but this apple to this machine is 0 1 0 1 0 0 0 1 something like that. So, I will not understand this to be apple and he will not understand this to be apple. So, there we need some sort of mechanism that is this is that translator some sort of translation is required to. So, we will be talking about this translation process in the next class, but before that let us summarize what we you have learnt till now. We have seen that there are problems which can be solved by computers and there are problems which cannot be solved by computers.

The problems which can be solved by computers have we have to in order to solve them; we have to express the specific steps to be followed for solving that particular problem. We have to express that to the machine we have to express that to the machine in its own language somehow we have to express it in its own language and this specific steps.

(Refer Slide Time: 26:14)



The steps that to be followed to solve a problem, this is also known as algorithm: algorithm consists of the specific steps or the methods that have to be followed in order to solve a problem. In the next lecture will see how we can express these algorithms.

Student: (Refer Time: 27:07).

How can we express the algorithms, that is what will do in the next mo next module next lecture; how can we express algorithms there are different ways, even I mean when I say how can we express algorithms not to the computer, but maybe I want to communicate an algorithm to you or you want to communicate an algorithm to me, I was showing some examples in which I was writing in English. So, there are different ways of doing that we will discuss that in the next class.

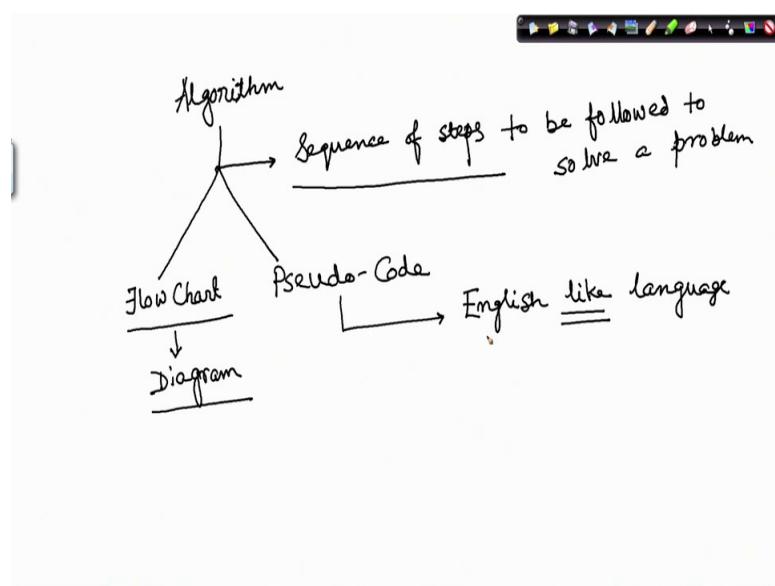
Thank you.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 03**  
**Flowchart and Pseudocode**

In the last lecture we had concluded with a term called Algorithm.

(Refer Slide Time: 00:22)



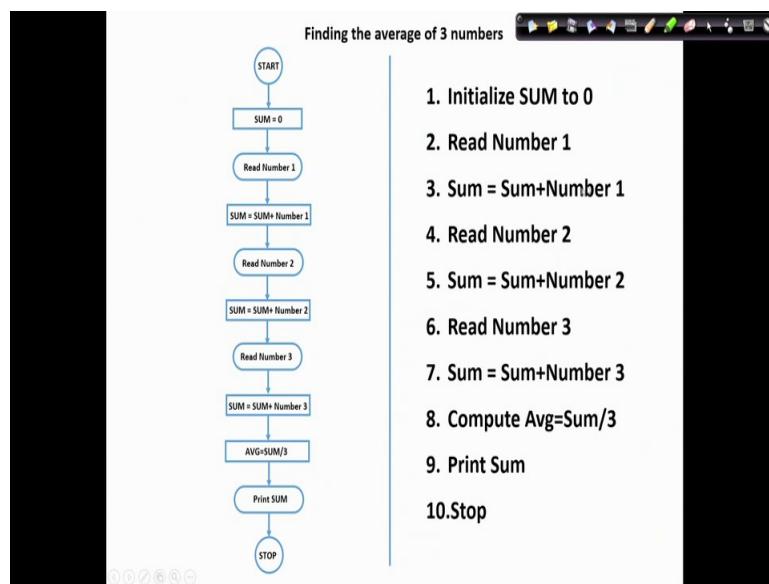
And we said that an algorithm means a sequence of steps that are followed to solve a problem; to be followed to solve a problem. Now the question is that how do we express an algorithm, how do we express the sequence of steps how do we express, that there can be different ways of expressing it right. So, usually ultimately we will have to write a program for this.

So obviously, program is the final form of expression of the sequence of steps, that we want to reach to, but even before that I mean a program is for communicating the sequence of steps to a computer, but even for our human exchange, we may like to express; what are the steps to be followed. There are 2 distinct ways by which an algorithm can be expressed one is flowchart another is pseudocode.

Now, a flowchart as the name implies is a diagrammatic representation of the sequence of steps it is a diagram. And pseudocode on the other hand is an English like English or

whatever in human language English like not exactly English, we can take a lot of liberty we will see how to when I express the sequence of steps, English like language to express the sequence of steps. Now we will show both the flowchart and pseudocode with respect to some of the problems that we had discussed earlier.

(Refer Slide Time: 03:13)

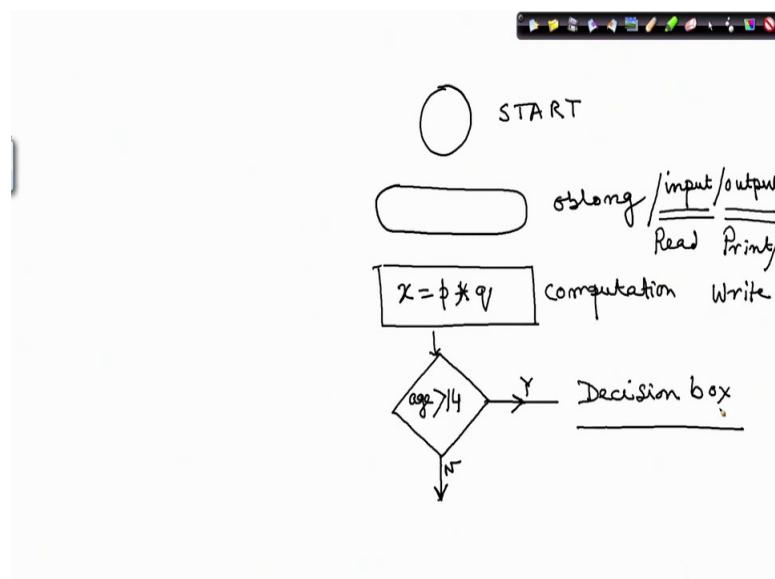


So, let us start with this finding the average of 3 numbers. This is the simplest possible way simple simplest possible problem that we can solve. So, we will start with the first we will start with the start node, we are trying to draw flowchart. So, once we draw this start. Then next will be sum assign 0. Now what all of us know that in order to find the average of any number set of numbers, we have to first add those numbers. So, the addition result is stated as sum. So, here we are when there is no number that has been we have already taken into account, we initialize the sum to be 0. So, right now sum is 0 right next what we do is we read the first number read number one.

Now, what does this read mean? Read means who is reading whom do you want to read; who ever who ever will be finding the average he or she will have to read the number. If I ask a human being to do that the human being will have to know the number. So, if I show that number on a piece of paper writing 5, he will read that number 5 and will say. So, 5 is the first number then I show the second number 7, 7 is a second number in that way that is the significance that is the meaning of this read.

Now, in our case ultimately the computer will read it, and earlier we had talked about um the input often we had discussed about this, this is also known as an input mechanism. So, it is reading the number, now here I would like to state a few things that is.

(Refer Slide Time: 05:42)



Say a flowchart has got some basic fundamental elements, this one you have already seen that is a start symbol.

Student: (Refer Time: 05:53).

Now this figure is known as oblong.

Student: (Refer Time: 05:54).

This figure is a rectangle. Now this oblong is actually used for input and output power I mean specification wherever I need to do some input or the system provides me some output let me explain. So, input I have already explained that when I am writing some number for you to read or for the; anybody will do the computation to read that is the input. And you know all of you that we have got some input devices the most standard input device is the keyboard using which we can put in the data. the output is after the computation is done the system will or system will give me the output or if it be a human being who is doing the computation, he or she will be telling me the output that is the result. So, that is known as output.

So, while drawing a flowchart sometimes we will write read for this input, and maybe print or write for output. Now this diagram is essentially the computation box. So, whenever we do some computation, that we show in this sort of a box for example, we can say  $x$  is  $p$  multiplied by  $q$  something like this. Or the other very important block is the decision blocks block which looks like this a diamond now here what we do we just take decisions depending on some conditions. Depending on some conditions I will either take the left path or the right path. So, for example, I can say whether the age of a boy is greater than 14.

Student: (Refer Time: 08:15).

If the age of the boy is not greater than 14 he is not allowed to go to a film. So, if it be no, then he cannot go to see a particular film, and if it be 14 or greater; that means, it is greater than 14, if it is yes then he can go to a film. So, we come to this particular point and make a decision looking at this out of scenario. So, this is known as the decision box.

Student: (Refer Time: 08:48).

Now, these are the basic 4 elements of any flowchart. Now, we will again come back to our flow chart. So, you can see here that sum is 0 and the number that has been read is the first number right. Now suppose that number that was read is 5; so some number. So, next what I do is, I add that particular number with the sum. So, now, if the sum was initially 0; so I add 5 with sum. So, if becomes the total sum becomes 5. So, next what should I do after I computed the sum, then I read the second number because I have to read 3 numbers and find the average. I read the second number and then I add the second number with the sum the first number was 5.

So, the value of sum is now 5 I have read the second number suppose that was 8, then the sum becomes 8 plus 5 that is 13. Now I come and read the third number suppose it is 3. So, I read the third number and then I compute the sum. So, I have computed 3 the sum to be 13 plus 3 16, next what do I do? I have to compute the average. So, what I do I find the AVG is by AVG? I am depicting the average, average is sum divided by 3. So, it was what was it 13 plus 3 16 divided by 3. So, it will be 5 point something. So, that result whatever I get that I will be printing. So, that is a flowchart these are the steps that I have to do.

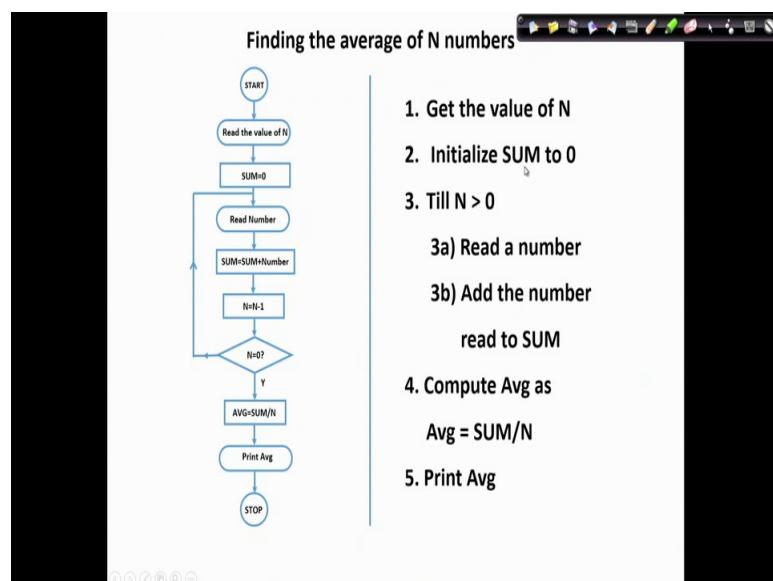
Now the same thing I can express in the form of a pseudocode or English like language let us see how it looks like. Here you just see what I have done, I have initialize I what I wrote in the first step is this one is equivalent to this initialize the sum to 0 and then I read the first number, sum is added sum equals sum plus number one exactly what I did here read the second number then I sum equals sum plus number 2, and then I read the third number and then sum is sum plus number 3.

So, these are the steps you see exactly if you just do not look at the left part of the whole thing, if you just look at this right part you can also understand what are the steps through which I must go and. So, ultimately I compute the average which is I just write. So, it is English like compute AVG equal. So, it is not exactly English, it is English like anybody who knows English will be able to understand this. So, this is known as the pseudocode.

So, this is for the very simple problem that we are doing. Now suppose let us think off for a second that suppose instead of 3 numbers I ask you to find the average of 10 numbers how would the flow chart look like? Obviously, this same thing that I have done here read number 3 read number 4 read number 5 read number 6 in that way it will go on and on and on, had it been 100 numbers there will be even larger.

Now, can we do something better in order to I do not have so much paper to waste. So, can I write it can I express it in a much better way?

(Refer Slide Time: 13:39)



So, let us go to the next problem where we want to find the average of N numbers average of N numbers I want to find out. So, the first thing will be again start that is from where I am starting next is read the value of N. So, what is this value of N? Now if I say this value of N when I read what is this N? This N is telling me how many numbers I will be considering for finding the average is it 100 is it 1000 is it 10 whatever. So, I am that is n.

So, I am reading the value of n, now I am initializing the sum to 0 again just as before I initialize the sum to 0 and then I read now you see here I have written read number I have not written read number one earlier since I had to only do number one 2 and 3 here, there I wrote down number 1, number 2, number 3, here I am just writing read number, because I do not know how many numbers I will be reading the reason will be very clear soon.

So, next what do? I do I read the number the sum is sum plus number. So, I take the sum was 0, I take the sum and at that number which number? The number that I just now read say- I have just now read number 5. I have read just now I am sorry I just read number 5. So, I read number 5 and I have added that number with sum. So, sum is now 5, next I decrement this is a new thing that is coming up here. Since I know here at this point the user or the whoever supplied the data told me that you have to look for 100 numbers say N is 100 I have learnt that. Now, here I have read one number and I have taken care of that number because I have added that with sum and. So, therefore, I now decrement that value of N what does this signify? This signifies that I now since initially it was 100, I have to read 99 more numbers 99 are yet to go.

Now, I have to check have I read all the numbers, what is the value of n? The value of a N is 99; that means, N is not 0; that means, I have not been able to read all the numbers. If I have since in a decision box as I have told you just a couple of minutes earlier that I can have 2 options, I can either have yes or no answer if N is equal to 0 what does that signify? If N is equal to 0 that signifies that I have read all the numbers I have already read all the numbers, then I can compute the average as sum whatever sum I got divided by N.

Otherwise if N is not equal to 0 then let us look at here if N is not equal to 0 no then I will go back here again look at this part I will go back here again and read number. I read

that second number read and add it to sum I will go over here decrement. So, I have read to number. Now, how many are yet go 98 N is 0 no, N is not 0 N is not 0. So, I go up again I read another number add that particular number to the sum decrement N how many to go 97 is it 0? No not yet 0.

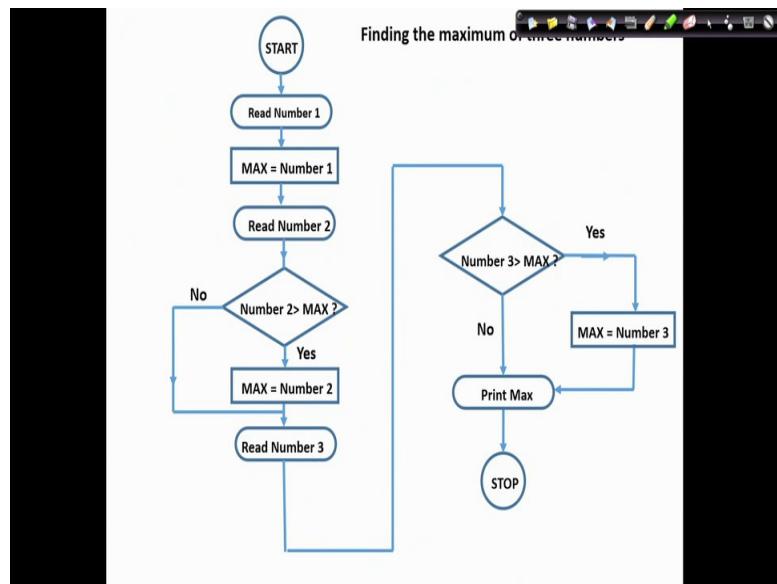
So, I go on in this way I will go on in this way ultimately after I have read the hundredth number this N will become 0. So, at that point I will come to this average and compute the average as sum divided by [noise], and then I will print the average and stop. So, see is just by this thing I am sorry just by this thing which is nothing but this is called the loop this is called a loop and using this looping repetition I could reduce the length of the flowchart the repetition of the flowchart in a very elegant way, I hope it is clear. So, the same thing can be also reflected in the flowchart.

Now compared to the with the earlier flowchart, earlier statement here when I read it only for 3 numbers, I read initialize sum to 0 read number one read number 2 and at every point I was adding it to sum, here what I do is I write it in a little bit flexible way, because when I write in a pseudocode I am not very constrained I can take the flexibility as long as I can express myself its good enough I just wrote get the value of N earlier probably I wrote read the value of N means the same thing.

So, get the value of N initialize sum to 0, and till now here I am talking of this looping till N is as long as till means as long as N is greater than 0, it will better to say as long as N is greater than 0, read a number at the number read the number that you have read to sum you read a number add the number read to sum. And one thing I missed out here and you decrement N you decrement N here you have to make N equal to N minus 1 in your note you please take it down that here it will be N n minus 1. And so this till N as long as N is greater than 0, I will be going on doing this after that I will compute average as average divided by sum average is sum divided by N I am sorry and then I will print the average. So, I can express it either in the form of a flowchart or in the form of a pseudocode.

Now, let us take a; I mean it is not very visible here, finding the maximum of 3 numbers that was another problem that we had discussed last time. What I am trying to do is I am trying to find the maximum of 3 numbers. So, how can I go about doing the flow chart?

(Refer Slide Time: 21:27)



So, I start just 3 numbers not a large set of numbers, I first read the number one. Now tell me one thing when you are first read one number; what is the maximum number. Obviously, since you have read only one number that itself is a maximum only one. So, it is a single ton. So, it is the maximum. Therefore, I say that max is nothing but the first number, the number that I have read max is number 1.

Now, I read the second number I read the second number; now which one is the maximum if the second number is greater than the first number, then the second number will be the max. Now when I read the first number when I read the first number here, I said that to be the max. Now, after reading the second number I can compare the second number with respect to the max. If the second number is greater than the max, then what will happen then number 2 will be the max. If it is yes the number to become the max and if no; that means, what number 2 is not greater than the max then I skip this part. I skip this part and go out by this way skipping this part; that means, at this point who is the max; obviously, at this point I had number one, and number 1 is still the max I hope its clear.

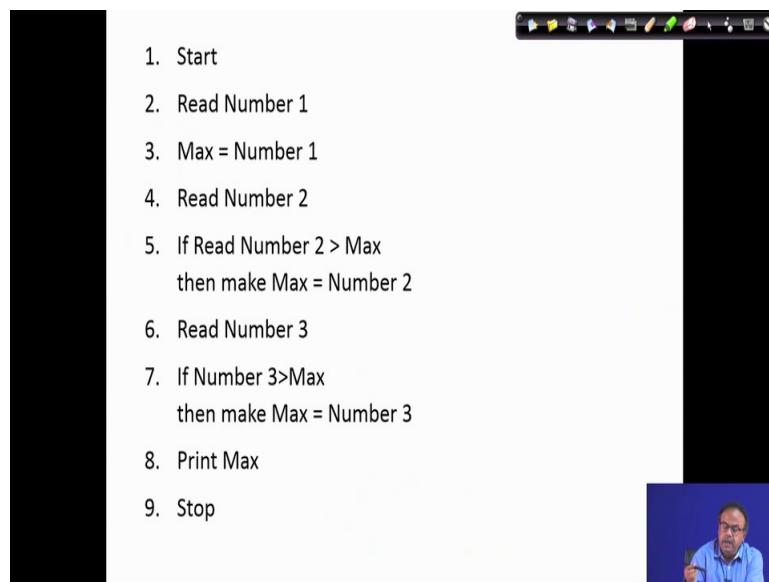
Next I read the third number, again I do the same thing after reading the third number is the third number greater than what number 2 should I compare no. I should compare with the max because till now between number 1 and number 2 whatever is the maximum I have already remember that in max. So, I read the number 3 and then again

compare is number 3 greater than max, I can have yes or no. So, if it be yes then number 3 becomes the max because number 3 is greater than the maximum number till now.

And then I will proceed and if not if number 3 is not greater than max then I will. Then obviously, whatever was the max till now, that is the number one that will remain the max think of number 1 to be 5, number 2 to be 3 and number 3 to be 1 what will happen? 3. So, max is 3 sorry this was 5. So, I read 5 next I read 3 is 5 greater than 3 no. So, I kept 5 as the maximum alright, I read number 3 that was 1, I compared is it greater than the maximum which was 5 no then I print that 5 which is the maximum, but if suppose number 3 was 7. So, it is greater than 5.

So, then I can go here and update this max to be number 3 or the 7, and then I come to the print max part. So, this is a flowchart of finding the maximum of 3 numbers and consequently you can see the pseudocode of this.

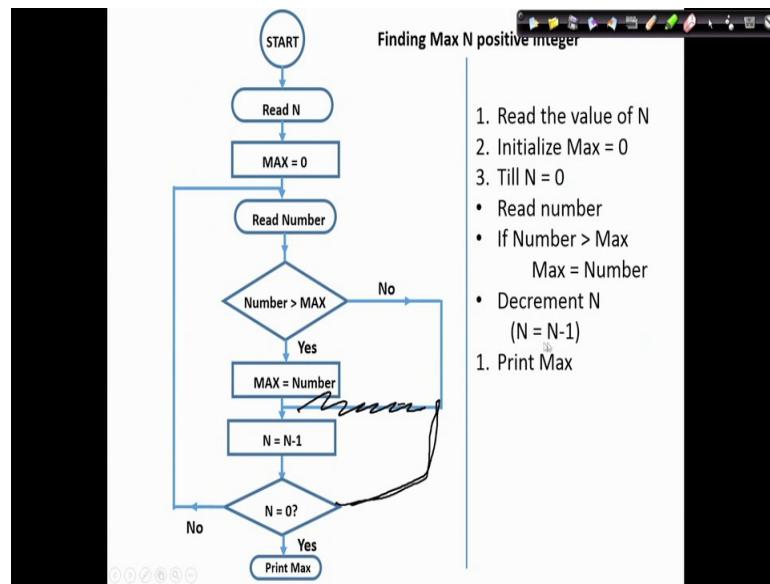
(Refer Slide Time: 25:26)



Start read first number, then I assign max to be the first number, then I read the second number assigned. Now if number read is greater than max, then I will make the max to be number 2. If the number read if the if its badly written read number it is wrong it if number 2 is greater than max then make max number 2. Then read number 3 if number 3 is greater than max, then min make max number 3 otherwise you are continuing.

So, if number 2 is not greater than max then I am reading number 3 I am not doing this part. This is another way of explaining the pseudocode. So, the same thing now we come to another variety of this finding the maximum for N positive int N positive integers, here I have taken a little liberty the finding the maximum of N positive integers, earlier I was doing it for all numbers, but just as a change. So, let us see.

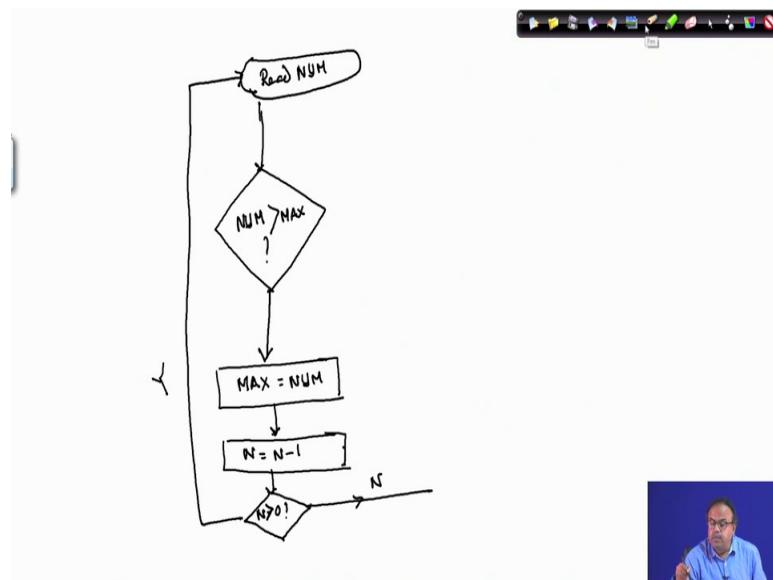
(Refer Slide Time: 26:42)



I read N again, N means the number of numbers that I want to read. Now I set max to 0 why do I set max to 0? Because I am trying to find the max of N positive integers, any positive integers will be greater than or equal to greater than 0. So, I am putting that max to the minimum value possible and then I am reading a number I read a number and if that number is greater than max, which is by default 0 I have kept it as the bottom was possibility. In all unless that number is 0 it will always be greater.

So, I will set that number to be the max, next again you remember what I did for finding the average? I reduce the value of N. Suppose I had to do find the max of ten numbers. So, N becomes now 9 and then I check. I have the max sorry this is a mistake here. This arrow, this arrow should come from this point this arrow should come from this point. So, I am just drawing it here I am drawing it here.

(Refer Slide Time: 28:21)



It should come like this. So, I have read a particular number and then if that particular number is greater than max, I am sorry I read a number and if that number is greater than max then I will if it is true, then I will make that number to be the max right max will be that particular number, and then I check N I have already read N. So, is N 0 then I have read one number and then I reduce N, just as I did in the earlier case and then I check any N greater than 0. If yes then I have to go back and again read a number right read number and this will go on as long as N is greater than 0. If N is greater than is not greater than 0 no then I will come and print the max.

So, there was a little error in that diagram which you will be able. So, you can see that this line should have come from here is N 0 yes then print max, otherwise this should have come from this point this is wrong. So, in this way we can again find the max for N numbers it is a flowchart. So, again I can have the pseudocode for this. So, a pseudocode for this is shown here, read the value of N here initialize max to 0, till and till N is 0 read the number if number is greater than max, max is equal to number decrement N, N is N minus one and then it will be not one here it will be again print the max.

So, in that way you have seen examples of some intermediate representation of the sequence of steps or the algorithms. We will encounter a number of algorithms in the course of these lectures. And the Next step we will see what is to be done in order that this sequence of steps can be transferred to a computer language so that; and can be

transferred to a computer so that computer can solve the problem, that we will see in the next part.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 04**  
**Introduction to Programming Language Concepts**

So, in the last lecture we had seen how we can represent algorithms, and we had mentioned particularly two different ways. One is flowchart which is a diagrammatic way of representing the different steps, and also we saw English like form, which is known as a pseudocode. It is not exactly why is it called a pseudocode, it is called a pseudocode because it is not exactly the code that can run on the computer, but it is a way close to that and by which you can express ourselves, and from which we can also convert to the computer language.

Now, we had given a few examples of such flowcharts for a few problems. And I strongly encourage you to take up more problems from text books or some mathematical problems that you may encounter, and try to solve them; try to draw the flowchart of those.

(Refer Slide Time: 01:27)



### Text/Reference Books

1. C Programming : Kernighan and Ritchie
2. Programming with C  
B.S. Gottfried, Schaum's Outline Series, Tata McGraw-Hill, 2006.  
OR  
Any other book on C



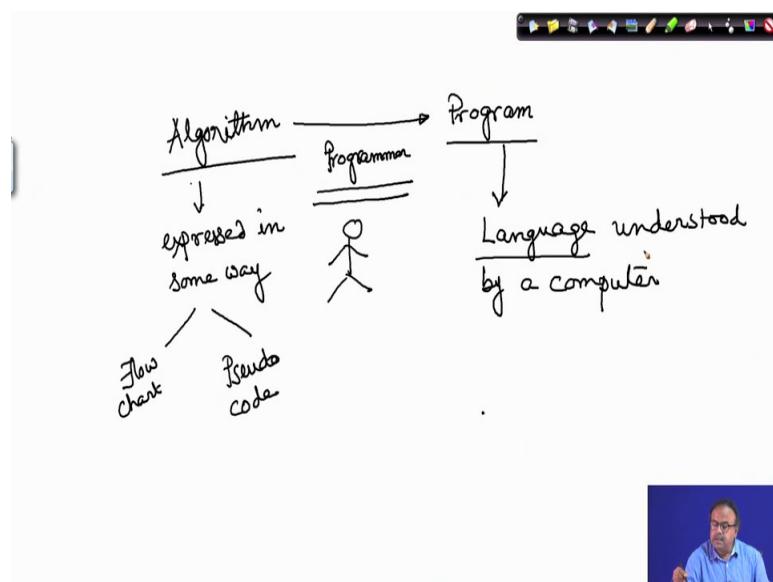
7

Now coming to the context of the books here are some of the suggested books, but I will start from the end that is any book on C will serve your purpose. You can take any good book on C language and try on that. And the number 1 that is C programming by

Kernighan and Ritchie is the most authentic book for C programming C language for that matter and, but I suspect that for some of you may find it to be a little difficult therefore, a middle path be a this second book programming with C by B.S Gottfried which is Schaum's outline series. So, and this second book has got a number of examples solved examples and examples given as exercises. So, I think that would a good very good starting point.

Now, having said that, what we have right now is we have some algorithm.

(Refer Slide Time: 02:37)



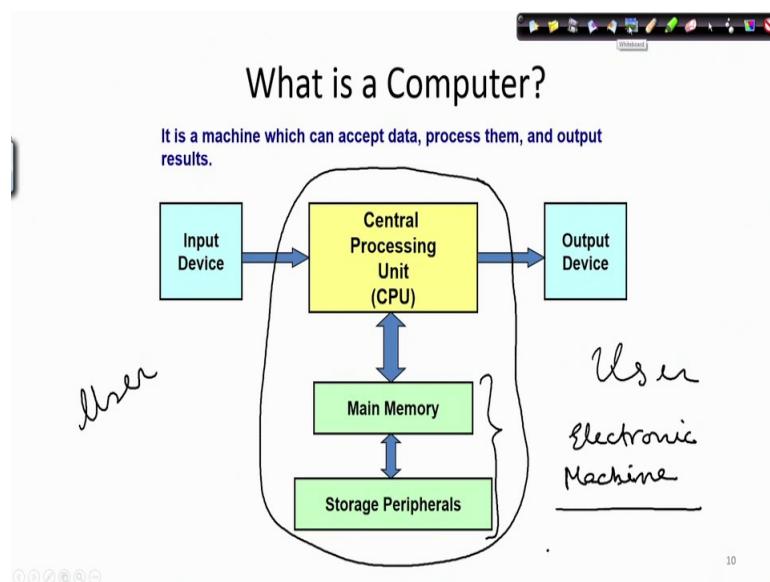
And from that algorithm, we want to come to a program. Now algorithm is a sequence of steps that might be in our mind also. So, I know how exactly I want to do that, I and I express it in some way and that is understandable. Now you know about 2 ways one is the flowchart another is a pseudo code which expresses how what are the steps that a to be taken. Now from this algorithm to a programmer program is a task of a programmer. Programmer is a person who knows how to write programs.

So, from the algorithm to the program whatever language the program might be that has to be translated. Now the key point to understand is that this algorithm I can express in any particular way; however, that program must be written in a language that is understood by a computer; alright language understood by a computer. So, a programmer is essentially a person sitting here who is translating the algorithm or the steps as expressed in some informal way in to a form, that is understood by the computer and this

language just like any other language we will have it is own vocabulary, we will have it is own grammar.

Now, here comes the question what is the language that is understood by a computer.

(Refer Slide Time: 05:13)

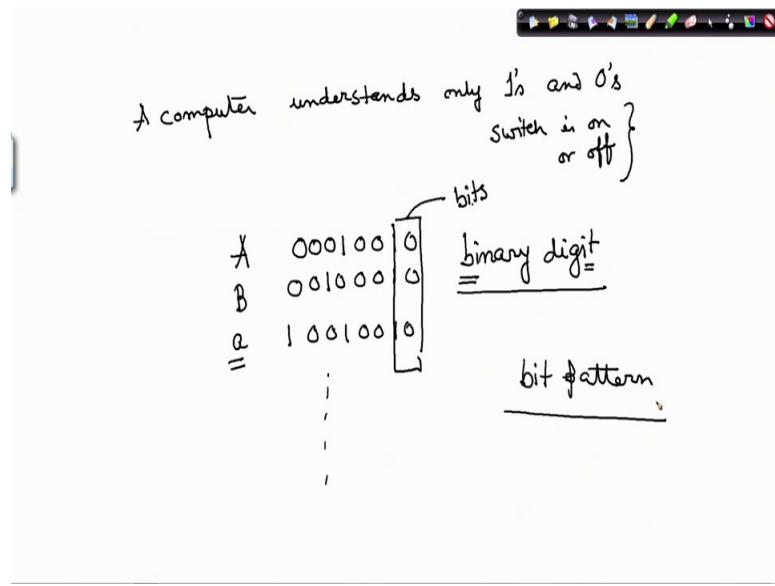


Now in order to understand that, we have to look at what is there inside a computer; now all of you many of you may be knowing it from your school days, but just it is a quick revision of it is machine computer is nothing but a machine which can accept data process them and output the results. Now there is an input device which can be a keyboard. Nowadays it can be speech, microphone, it can be a joystick it can be many things right mouse is also another one. And ultimately we have got the computer here which consists of the CPU and the main memory and storage peripherals.

So, this is the actual part of the computer. Now the data comes over here, it is processed here and the output is sent to the user. So, the user is on this side the user is also on this side. So, this side is also the user and this side is also the user. Now there are couple of important things to understand here, typically we tell the computer how a problem is to be solved and that is the sequence of steps that we do, and whatever in sequence of steps that is specify. Now whatever sequence we specify that is remembered by the computer in it is memory. Now the memory can be actually it is stored in the secondary memory I will come to that later. But the key point to think about is that this entire box is nothing but an electronic machine and consequently it only understands 1's and 0's.

So, whatever we have to express; we have to express that using 1's and 0's.

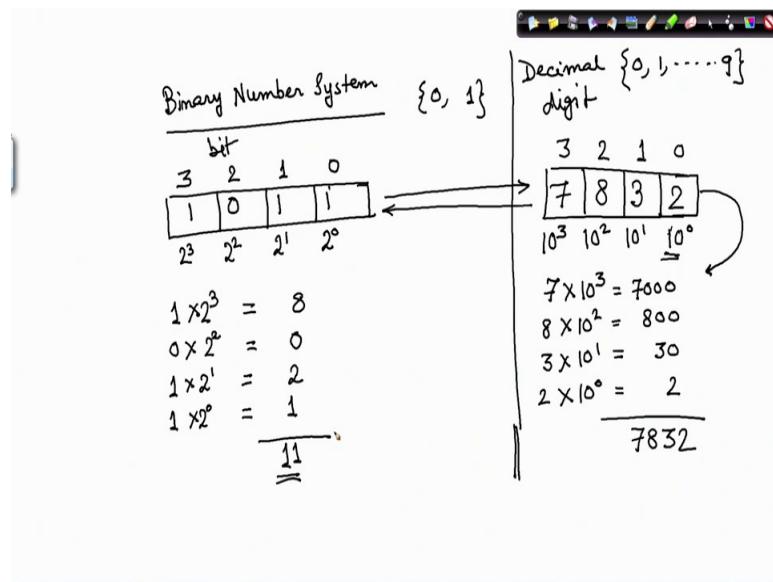
(Refer Slide Time: 07:41)



So, this is the first thing a computer being an electrical machine understands only 1's and 0's, because it is an electric machine, it only understands whether a particular switch is on or off right that is what it can at this understand. Therefore, whatever we have to express we have to express it in the form of 1's and 0's. So, consequently say A B all these alphabets small a everything will have to be represented in the form of 1's and 0's. So, maybe A just hypothetically it is not to be taken to be accurate, I am just saying that suppose 0 0 0 1, 0 0 1 0 is a pattern that represents A. So, similarly maybe 0 0 1 0, 0 0 1 0 this pattern represents b and may be 1 0 0 1 0 0 1 0 represents small a. In that way everything will be represented into a pattern of 1's and 0's and might be knowing some of you that each of these one and 0's these are known as bits, bit stands for binary digit binary digit.

So, bi for binary and bit that last t for this, binary digit from spades. So, it is a sequence of bits. So, anything, anything is represented in a computer as a sequence of bits or we often say bit pattern. Now just 1's and 0's gives rise to a new number system that is known as the binary number system.

(Refer Slide Time: 10:13)



So, that leads to the binary number system might be some of you might be aware of this. Binary number system where I have got only 2 elements, 0 and 1 I have to do. Now in decimal visa we if I just consider that regularly we use decimal number system where we have got from 0 to like that up to 9, and I whatever I express as a combination of this. Now in binary number system everything is represented just by 0 and 1.

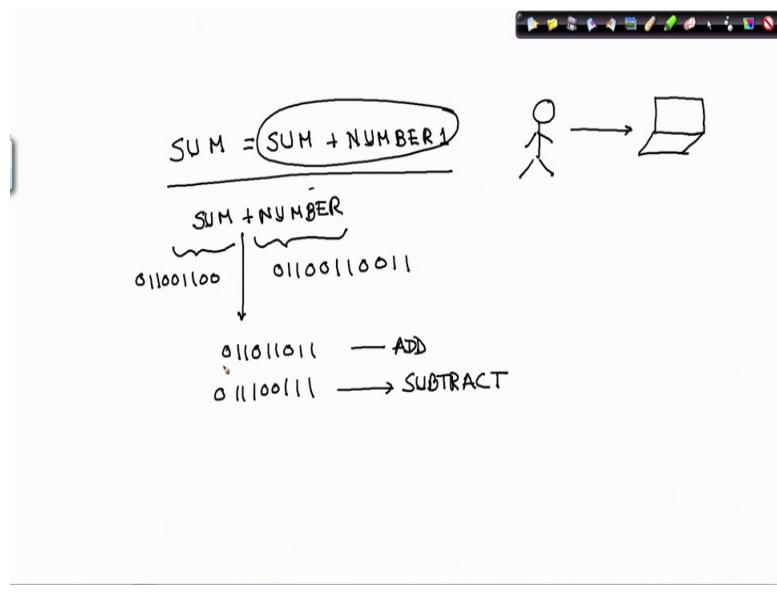
So, quickly let us look at; what are binary number system is, that suppose I have got a 4 bit number. So, there are four positions 1, 2, 3, 4 and suppose. So, con parallely let us think of a four digit number. Remember that in decimal we call it digit and in binary we call it bit right. So, if I have 7, 8, 3, 2; now all of you know what is the value of this number each of this place have got a weight, which are known as play I mean weights of the different places. So, this is 10 to the power 0, this is 10 to the power 1, this is 10 to the power 2, this is 10 to the power 3. 0 weight, 1 weight, 2 weights, 3 weight. So, this number is actually meaning 7 times 10 to the power 3 that is 7000, 8 times 10 to the power 2 that is 800. 3 times 10 to the power 1 that means, 30, and 2 times 10 to the power 0 that is 2 so that is giving us 7 8, 3, 2. It will be easier to understand this with this analogy for example, here I have got something like 1 0 1 1.

Now unlike; so here also the weights are 0, 1, 2, 3 but the base here was 10. So, here it is 2 to the power 0 is binary here. So, is 2 to the power one 2 to the power 2, 2 to the power 3, here the base is 2 here the base was 10, that is why this is a binary system this is a

decimal system. So, if I have this pattern 1 0 1 1 then I can straightway convert it to an equivalent decimal like what does this mean. So, I can take it there is a 1 here, 1 times 2 to the power 3 that is what 8, then 0 times 2 to the power 2 that is 0, 1 times 2 to the power 1 that means, 2, 1 times 2 to the power 0 that is 1. So, the value of this will be 11. So, this is the binary number system.

So, there are ways and means by which I can convert from a binary number system to a decimal system, from a decimal system to a binary system that is possible right. So, if you want to see that I do not want to go into that because many of you have done it in school. So, let us leave it for the time being. But what is more important is to know that a computer in a computer everything is represented in the binary system as a bit pattern of 1's and 0's. Since everything is expressed as binary number system. So, if I remember the say we had something like this.

(Refer Slide Time: 15:06)



Let us not take max, let us take a sum equals sum number one.

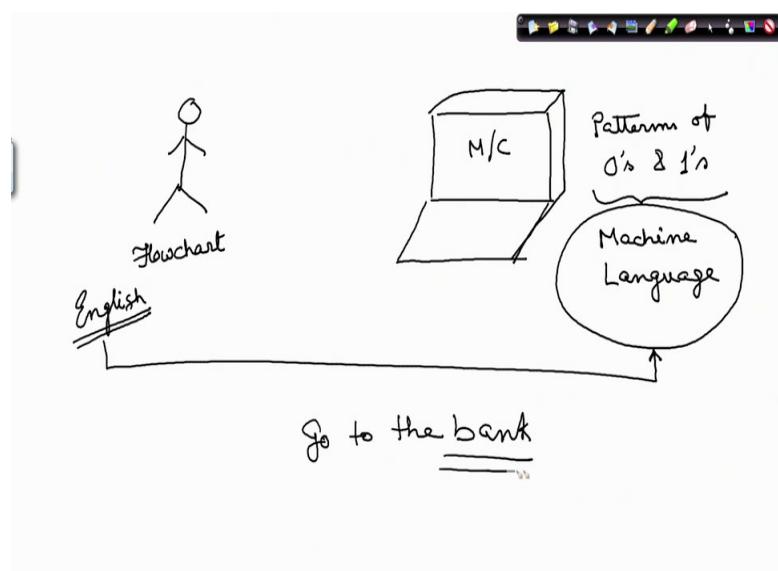
Student: (Refer Time: 15:23).

This is something we saw in our earlier lecture something like that. Now this is a statement how do I represent it to a computer. In order to represent this to a computer say I want to say this is and you have to do some addition and some assign; some you have to copy it somewhere and write it somewhere. Since or I just take this simple part add

sum with number, sum plus number how do I express that? Everything has to be expressed say this number have will be some patterns say 0's 1 0 1, 1 0 0 something of that sort sum will be something 0 1 1 0 0 1 1 0 0 whatever and class will also have some code 0 1 1 0 1 1 0 1 1 might be. Suppose this is add and suppose 0 1 1 1 0 0 1 1 1 is subtract similarly there will be a similarly there will be some code of for multiplication etcetera.

Now, since I am here, I am the user and I am also the programmer and here is my computer and I have to translate my thoughts my algorithm in a way so that a computer understands. I have to translate that algorithm or the flowchart into this pattern of 0's and 1's, because that is the only thing that it understands. The language that the computer understands is known as the machine language. So, let me once again draw another picture here.

(Refer Slide Time: 17:30)



So, here I have got my machine I have got a machine here nice looking machine and here is the programmer. The programmer understands flowchart or the programmer understands English like pseudocode, but this one only understands patterns of 0's and 1's. A pattern of 0's and 1's, and this is a language it understands and this programmer understands the language of flowchart for example. Now, this patterns of 0's and 1's that the machine understands, this is my machine M C is the machine. Now this machine

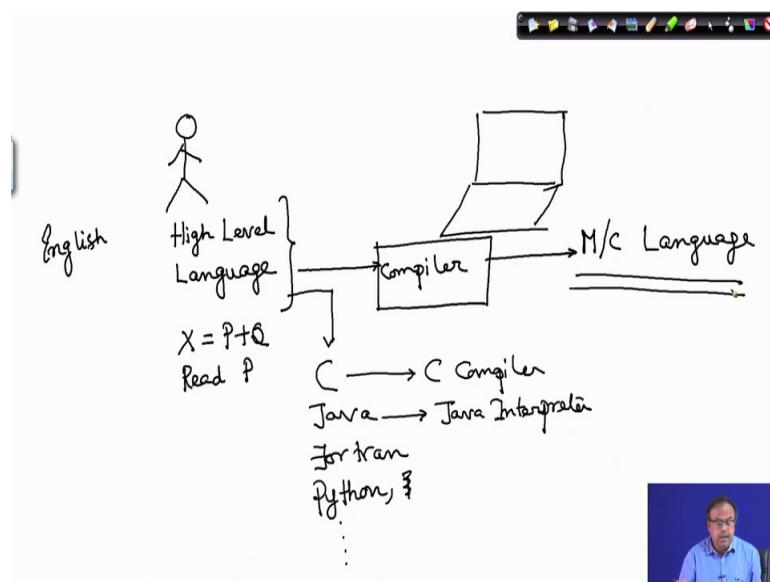
understands that language is known as the machine language, and that machine language is not understood.

If some genius can understand remember all those patterns of 0's and 1's for every possible combination, that is a different thing, but for normal people it is not it is very tedious, it is not possible it is not advisable also. Therefore, earlier people had to for small computers it was possible at the very beginning, people used to program the computer by setting up different switches. And thereby, it was being programmed. But at that time the operations and the capability of the computers say much more limited, but once it became larger we are doing much more flexible programming and everything, it is not no longer possible to remember the machine language.

So, what is the other alternative? The other alternative is the best alternative would be that on this side I have got my normal language English and if I could directly translate into machine language, but there are some problems with that because in English or any natural language that we use, many things are often ambiguous. It is not very clear it can have multiple meanings just I am giving an example go to the bank it is in English statement.

Now, what does this word bank mean? It can be many things right it can be in the bank where you deposit the money, it can mean the river bank etcetera. Therefore, English language or any natural language is not the yet suitable to be translated to the machine language. So, instead people thought if we could have some intermediate language. So, once again I come here, I again draw the machine here, the machine is here and the person is here.

(Refer Slide Time: 20:37)



So, on this side is my natural language like English, but which I am not being able to use. So, people developed some intermediate language which is close to English, but the grammar is much more stricter and we cannot allow any ambiguity no ambiguity. But it is high level language because it is easier for us to remember, it is easier for us to explain. For example, if I write this say  $X$  equal to  $P$  plus  $Q$  or I write read  $P$  now this is meaningful, this looks like English and I can understand this.

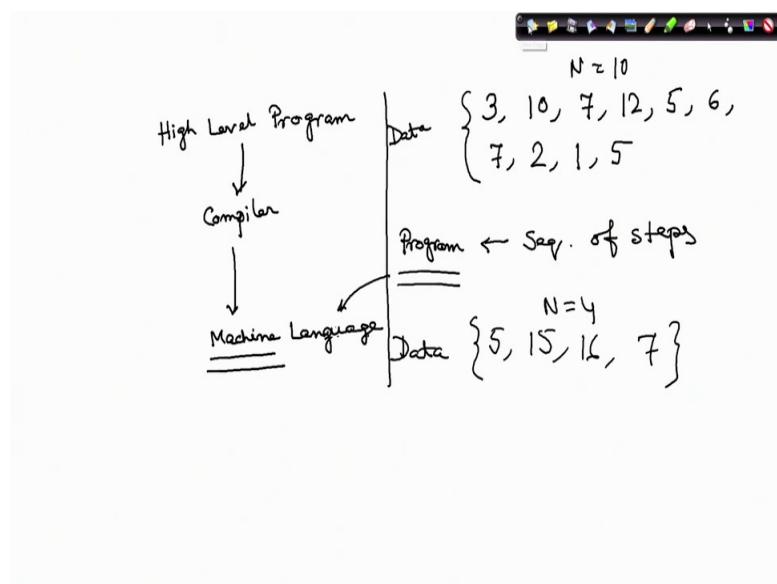
Now, given a high level language and on this side this machine can only understands machine language. So, we need to have some translator which will convert this high level language to the machine language, and this translator is known as the compiler. Now think of the situation that here is a person say just forget about the computer high level language, think of a person who knows French, and here is a person who knows maybe English.

So, a compiler or an interpreter is something which is somebody, who will translate from French to English right. So, here also depending on what language I am using as the high level language, there will be different compilers. Examples of high level languages are C which we will be discussing or taking as an example in our course; it can be Java, it can be FORTRAN, it can be Python, alright. We can have many other languages that are coming up nowadays other languages also coming up.

So, depending on the language high level language, in which we want to express our algorithm we will have to select a suitable compiler. So, for C we will need a C compiler, for Java will have to need a Java interpreter. The same thing the more or less same thing that it will translate the basic function of this is translation. So, it will translate from a high level language to the machine language. And as you had seen that here all those things I will be binary, I mean in binary system in the pattern of 0's and 1's, I have to convert if this machine has to operate with my program this have to process this my program I have to convert it into 0's and 1's.

So, now here is something more that we have to say. Now whenever we write a program what we do here is we write some high level program.

(Refer Slide Time: 24:52)



And that high level program is taken up by a compiler, and is converted into a machine language.

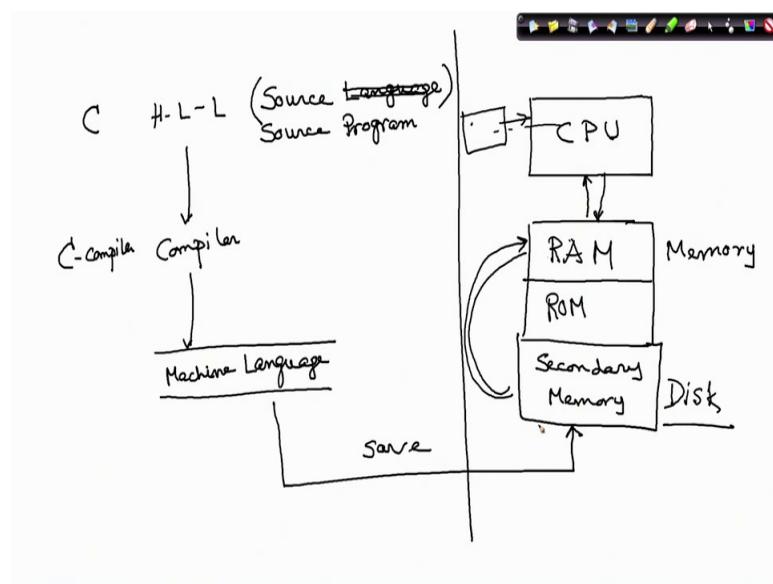
Now, here you have to be a little careful that depending on the different machines, you can have pentium machine, you can have apple machine and other things. So, depending on what machine you are using, you will you may need different compilers for that particular machine. So, for the time being let us ignored that thing, let us assume that one high level program can be translated by a compiler to a machine language. So, there is a program right. So, remember what we did in our earlier cases, we had we said find the

maximum of 10 numbers, and the numbers were 3 10 7 12 5 6 so and so forth right 1 2 3 4 7 2 1 5. So, out of these I have to find out the maximum.

Now, through the flowchart or through the pseudocode what did we specify? We specified the sequence of steps. So, those sequence of steps that have to be adopted in order to find out the maximum of this is our program. And this program will be converted to the machine language. Now this program will run on some data this program is a program that can run on this set of data or another set of data say 5, 15, 16, 7 say for 4 data if I just run it for n number of things and here the N was 10. Remember what we did earlier and here N is 4. The same programme should work on both of these therefore, the program remains the same, but here is some data set and here is another data set. So, we have got 2 different data sets.

So, in order to execute we need both the program and the data. Now when we start say our program and the program when we start with the high level program.

(Refer Slide Time: 28:03)



I am writing H L L is the high level language, now this by the in the context of passing I am saying this is also known as the source language. So, in my lecture often I will be using this term source language. Source language means or I will say the I will not I will not say source language, I will say source program. By source program I mean the program that is written in the high level language that will be translated by a compiler to a machine level language. Now when we now this is one side. So, the machine language

has been generated by the compiler. If my source language is C then I need a C compiler here, and I will get the machine language. And this machine language is independent of what the source language is, it is only dependent on the particular machine on which it is running. So, the compiler we translate into that machines machine language.

Now, in my computer diagram we had the CPU here and we had memory. Now the memory can be divided into 2 parts, one is the RAM and ROM some let me put it 3 parts RAM and ROM are 2 types of memories, which all of relatively smaller capacity I think this is not very clearly coming. So, let me write it clearly, RAM stands for random access memory and ROM stands for read only memory.

Besides that we have got the secondary memory, which is the disc. Now the machine language after compilation, when it is saved I save it and it is saved and stored in the secondary memory in the disc. Now when that program will be RAM, I have done the compilation and after compilation my program is ready to be executed is ready to run. Now when I run it, can I select some plain color; sorry. So, when I run it from the secondary memory it will go to the RAM and the CPU the program will move to the RAM only when I execute it, and the CPU you will read that program from the RAM and will execute it. So, this is a sequence you should remember I repeat once again. I write the program and type in the program in the machine, after that I do the compilation after compilation it is converted to the machine language and I save it. When I save it, it is saved in the secondary memory.

Secondary memory is typically the disc; I save it in the secondary memory. But on execution when I execute it, when I run the program then only it goes from the secondary memory to the RAM and the CPU reads from the RAM and executes it. So, we will be bothering about mostly this RAM and secondary memory, and how that is stored many variables and all those things are stored well in the next lecture we will see how we can gradually move towards encoding our pseudocode to the machine language to the high level language. So, summarizing what we discussed in this lecture, it is that we have got the pseudocode, but on the other side we have got a machine which does not understand this pseudocode, the machine only understands the patterns of 0's and 1's.

So, we have to have something which will convert the pseudocode cannot be directly converted because of lot of ambiguity. So, will have a high level language designed

which can be which is unambiguous, and which can be converted to the machine code by another system that is a compiler. And that compiler is nothing but another more sophisticated program, using which you can convert this and keep it over there take it in the machine language. And the machine can then read that machine language program and can execute it.

Now we have also seen that a program I mean if we have to run a program we need the program as well as the data both this are stored in this memory part and from which the CPU will be taking that. Sometimes the data can be fed directly from the input device at runtime. So, whenever we say read n then from the input device somebody will type in the value and that will come from to the CPU into the RAM. So, this part will see later gradually.

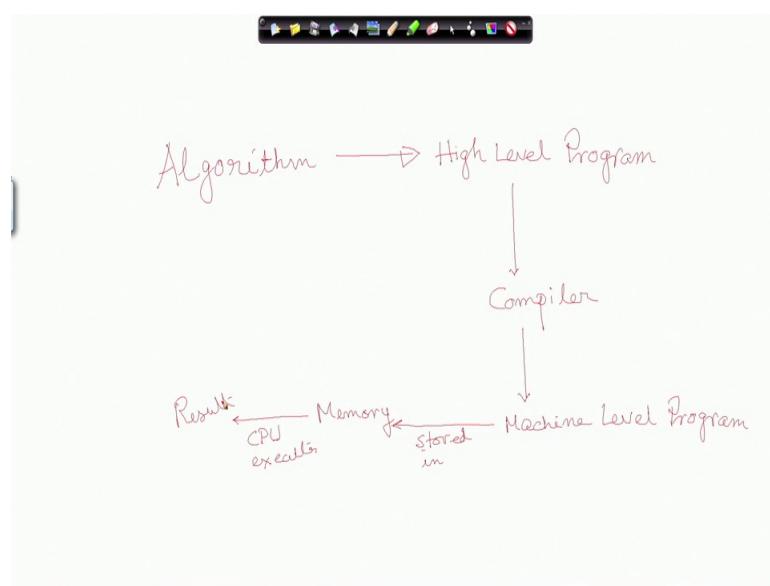
Thank you very much.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 05**  
**Variables and Memory**

In the earlier lecture we had seen that we start with an algorithm and convert that algorithm into some sort of high level program.

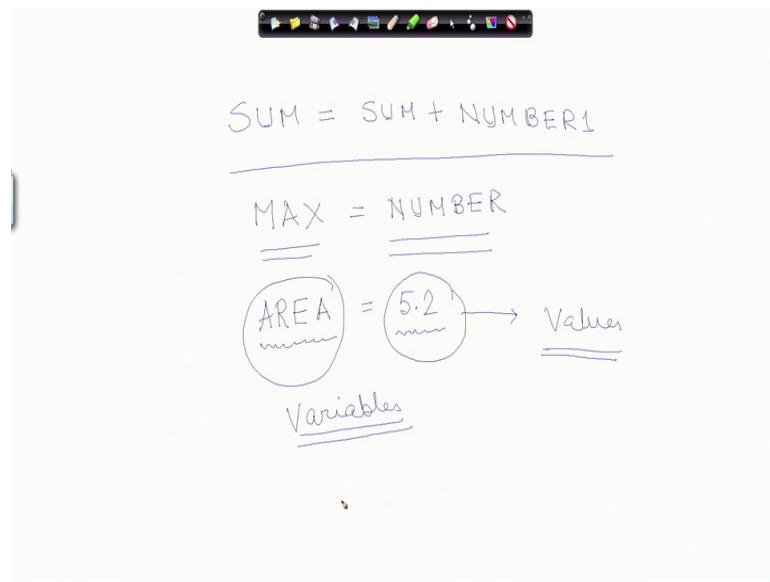
(Refer Slide Time: 00:22)



We convert it to some high level program and that high level program is fed to a compiler and the compiler prepares the machine level program. This machine level program is fed is stored in the memory of the computer and from this memory stored in the memory of the computer.

Now, from the memory the CPU execute set and we get the desired result that is the overall flow of the whole thing. Now we will now come back to some of the statements that we used in the earlier example programs will come back to that.

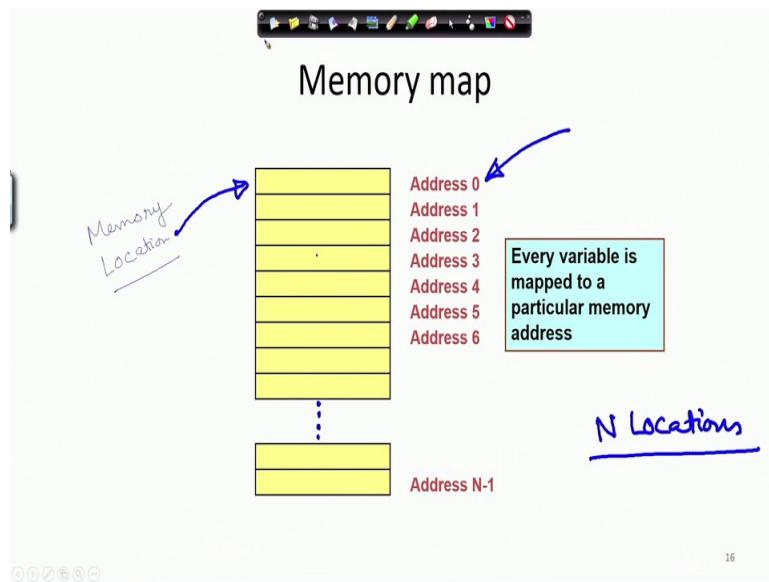
(Refer Slide Time: 02:38)



Say we had statements like SUM is equal to SUM plus number 1 right yesterday we accounted we encountered such statements SUM equal to SUM plus number 1 now also say for example, MAX assigned number 2 then the question is what are these things max number all right or for example, if I write AREA is assigned 5.2, what is this area and what is this 5.2 now this is some very fundamental concept that will be discussing today these are known as the variables and these are known as the values.

Now, we must be very clear before we start translating or writing a program in any high level language it is imperative to know very clearly what is a variable and what is a value. So, today's lecture will be devoted to explaining the difference between variables and values and what is the significance of these variables.

(Refer Slide Time: 04:43)



Now, let us look at this diagram the memory can be considered to be a cvr number of racks in a book rack sort of thing or you can think of a number of a drawers in the chest of a drawers essentially is these are specific places as is being shown here, now each of this place is known as a memory location all right.

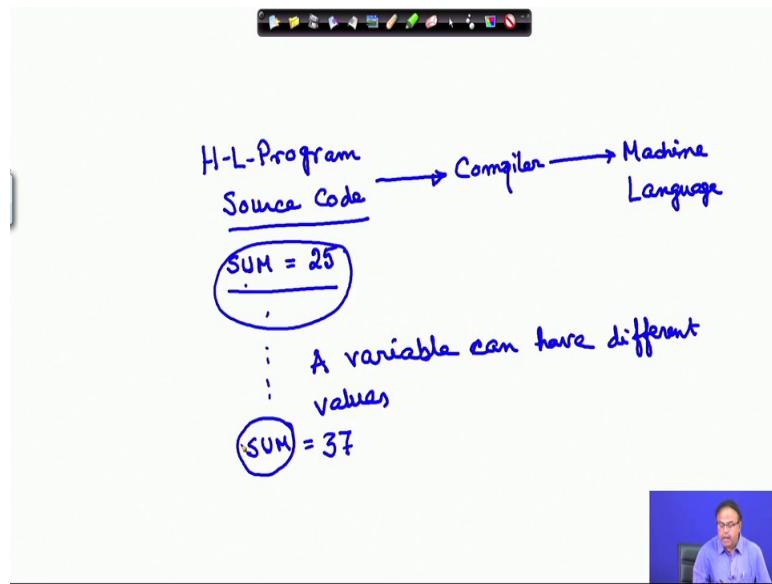
Student: (Refer Time: 05:40).

Now, here you can see that each of these locations have been marked with a particular address, these are the addresses and each address of every memory location. So, think of a scenario that in a locality there are a number of houses each of this house is a location, this is a location right, now I want to reach a particular house; that means, a particular location, how can I identify where I should go for that we need some address. So, similarly in the case of memory for every location there is an address, here we can we are showing 1, 2, 3, 4, 5, 6, 7, 8, 9, actually there are more locations which have not been shown.

We are showing  $N$  locations,  $N$  number of locations are being shown and each of these have got some address there is a peculiarity here the reason of that will be clear later let us for the time being accept that we are starting our journey with the number 0. The first address is address 0 and, the  $N$ th address will be address  $N$  minus 1 because the second address will be address 1, the  $N$ th address will be  $N$  minus 1.

Now recall that we had the scenario that we had the high level program.

(Refer Slide Time: 07:52)

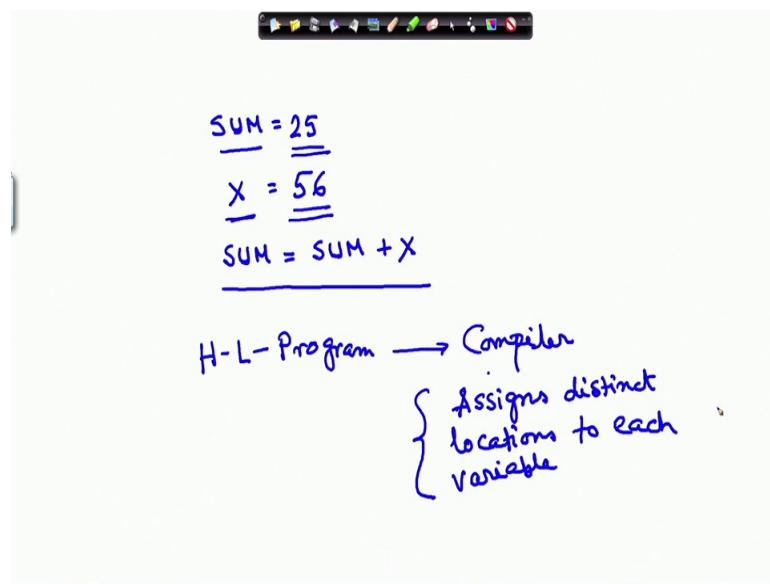


High level program or if you remember we call it also the source code and from the source code, the compiler converts them into a machine code machine language. Now in my high level source code I have a statement like SUM equals 25 say now when I say that SUM is equal to 25 the compiler will look at this and will assign these variable now this SUM is a variable means what it is a variable because it can be loaded with different values.

A variable can have different values for example, here I am making SUM equal to 25 after a while I can make SUM equal to 37 therefore, SUM is a variable and it can have different values right.

So, say in a program in a program I have got here SUM equals to 25.

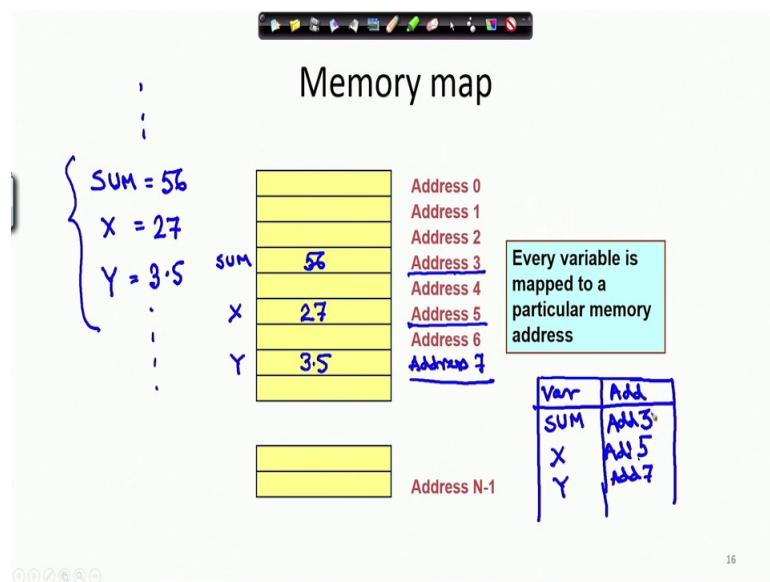
(Refer Slide Time: 09:33)



X equal to 56 and SUM equals to SUM plus X something of that sort, here how many variables do you see and how many values do you see we find that sum is a variable and X is a variable and these variables have been repeated here and what are the how many values do you see 25 is a value, 56 is another value. Now when the compiler takes the high level program and it looks at the compiler looks at all the variables and assigns or allocates distinct locations to each variable for the sake of simplicity let us say that it assigns distinct locations to each variables all right.

Sometimes advanced compilers also share variable share location, but that is not our concern right, therefore, let us come back to the slide once again you can see.

(Refer Slide Time: 11:20)



If I have a number of variables when every variable is mapped to a particular memory address for example, here if I write SUM equals 56, X equal to 27 and Y is equal to 3.5 the compiler will look at this piece of program and you will allocate and we will allocate some location to each of these variables maybe SUM can be located allocated here, X is allocated here, Y is allocated here say therefore, SUM can be identified now by the computer or the CPU when it will run the program it can look at whenever a sum appears in the program the compiler will convert it to the corresponding address of SUM. SUM has got the address 3, X has got the address 6 and Y has got the address 7, therefore, this thing is converted to a scenario where I am actually writing 56 into address 3.

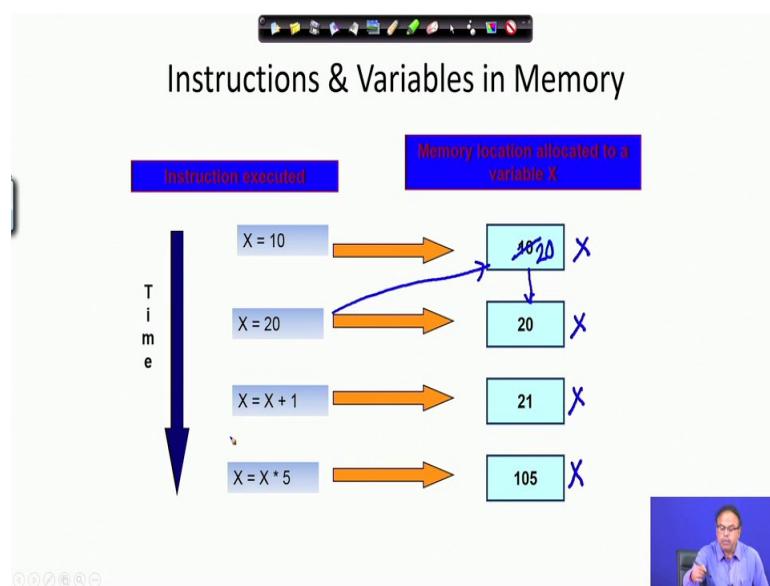
So, as if 56 is being written here I will explain it later, X is being written here, 27 is being written here and 3.5 is being written here all right because while the computer executes the program it will take the data from the memory location only and the memory locations it no longer understands whatever name you gave to this particular variable, whether it is sum or any other name, but it is identified uniquely by the address that the compiler has given it.

The compiler can depending on the availability of the memory locations allocated different addresses to different variables once that is done there is a table for example, there is a table like this a record that is kept that you can think of that we have got the

variables here the variable and the address. So, some table of that sort is prepared SUM is address 3, X is address 5 and Y is address 7.

Now, the CPU whenever it, the program in the machine language is converted in terms of these addresses this variables will no longer appear in the machine language, let us look at this in a little more detail.

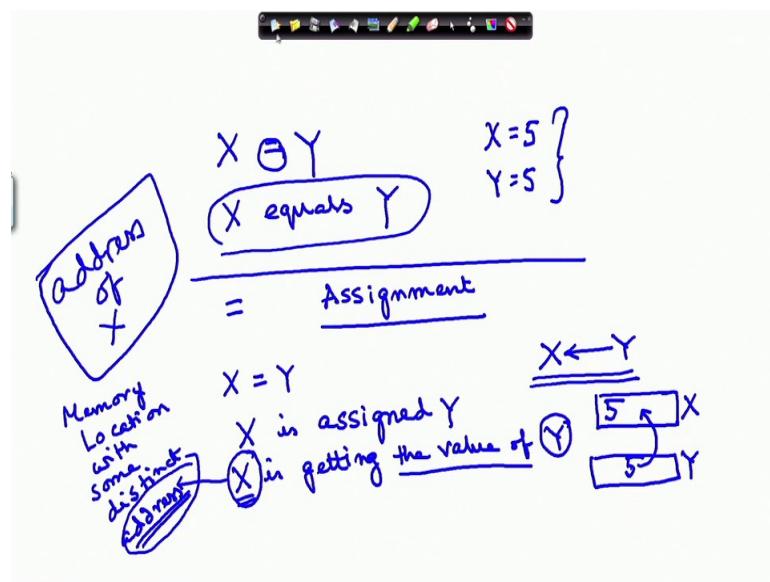
(Refer Slide Time: 15:17)



Now, suppose look at this instruction this is a program now one thing to mention here is typically a program consists of a sequence of instructions that we know that one instruction after another will be executed. So, this is usually done in a sequence where is an exception from that sequence? We had seen in the flowcharts that whenever there is a loop I am repeating an instruction time and again, we have seen when we are trying to find the maximum of n numbers or average of n numbers we are repeating some instructions time and again, in that case we will go back to the earlier instruction otherwise one instruction is executed after another right.

As time passes we have got some instruction here not doing any meaningful computation say just to illustrate X equals 10, now here is another point what does this equal mean all right.

(Refer Slide Time: 16:48)



This is something that we should understand in our school we often do say X equal to Y that means X, the value of X is equal to the value of Y. If X is 5, then Y is 5, but here this symbol whenever we are using now is this symbol actually means assignment now this is something which you should understand and this is fundamental to any programming language that we will use. When I write X here I am writing X this symbol Y I will read it as X is assigned Y what does it mean X is getting the value of Y.

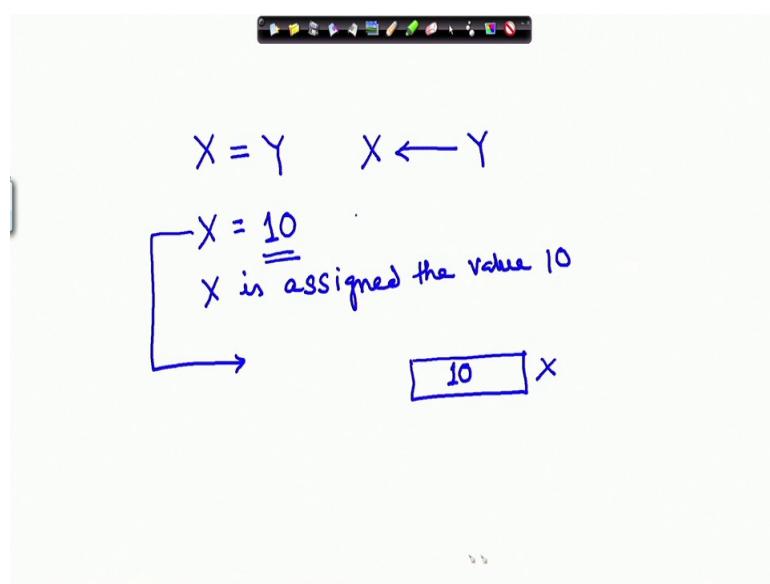
Now often is therefore, in order to avoid this confusion we write it in this way that the value of Y is assigned to X now here is something to really understand what is X, X is it a value no, X is a variable therefore, X is a memory location the compiler has allocated some memory location with a specific address, this X is nothing, but a memory location with some distinct address is it readable, memory location with some distinct address that is X and Y is also a memory location with distinct address, but this assignment means it is getting the value that is in Y.

Let us think of like this that here is a memory location that has that is X. So, X when I am writing X that is means that it is actually the address of X, that has been address of this memory location and Y is also memory location and when I am writing Y this variable name Y just for ease of our understanding otherwise I would have to write the address here and the address in a computer system in machine language will be a string of binary bits I mean a bits bit string, that will be really complicated. So, for the sake of

understanding we are just writing X and Y, but you must understand that this X or this Y means the address of X and who has allocated the address of X or the address of Y the compiler. Now when I am doing assignment this X is assigned the value of Y suppose Y had 5 all right the value was 5 now this 5 is then written over here and this also becomes 5 this is the meaning of assignment it is not exactly meaning equal to Y essentially after this what happened is the value of X is actual equal to the value of Y, but this sign typically means assignment in order to show this equality we have got other symbols will come across that later.

Let us move to this, here we are executing an instruction one instruction after another. So, here in the first instruction we find that X has been assigned the value 10 is not say once again let me come here.

(Refer Slide Time: 22:05)



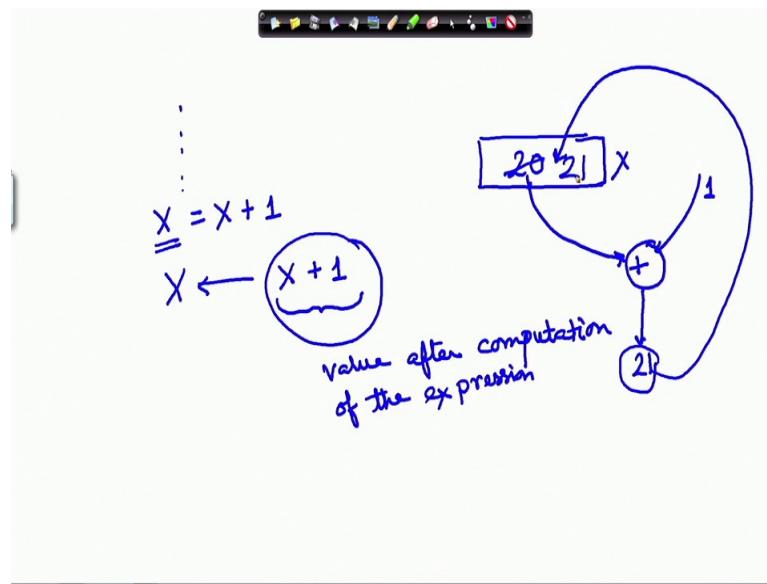
This is this means X is assigned the value of Y if I write this X assigned 10 that means, X is assigned the value 10 because here I had some value in Y whatever it is and that was being transferred to X, but here it is not this side the right hand side is no longer a variable, but a value right the right hand side is a value therefore, X is a memory location.

For this example X is a memory location which is having the value 10 after execution of this instruction this will be the situation in the memory after the execution of this instruction right. Here let us now you will understand as I assign 10 to X and this is a

same memory location this is a memory location for X the same memory location see how the picture is changing, here that memory location X is getting the value 10 on this side I am showing the memory location allocated to the variable X. So, 10 next step I assign 20, this location gets 20.

This 10 that was there same location 10 was there 10 is being over written by 20 here. So, when this is being executed then this 10 is being over written and 20 is being written, I am getting here 20. Now X has got 20 what is X now X is 20 now I do another arithmetic operation here what how do you explain this X assigned X plus 1.

(Refer Slide Time: 24:44)

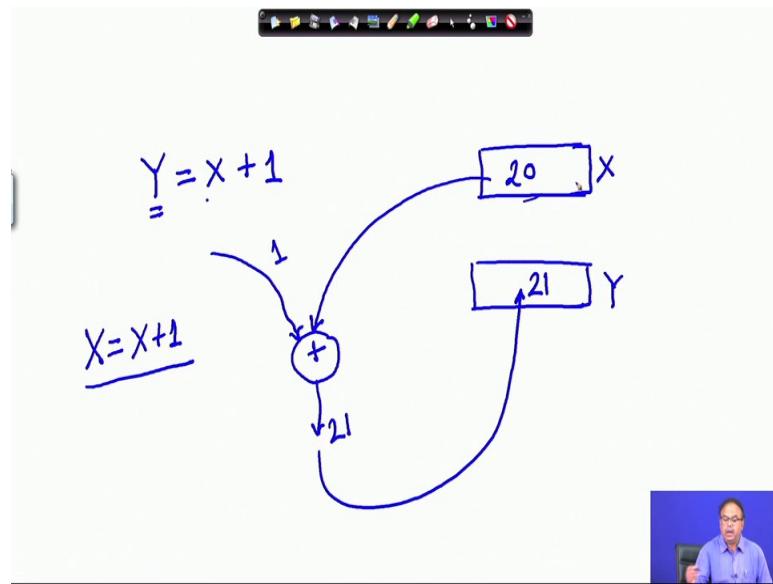


Let us go back one moment to this X was this is X it was 20 and I have come across an instruction, which says X equal to X plus 1 what does it mean? It means assign to X what? X plus 1; that means, whatever value you get after computation of the expression will be assigned to X. So, how will it be done? Right now what is the value of X 20, the CPU will have doing some addition operation, will it read this value 20 here and some incrementation by 1.

So, that these 2 will be added the value from the location X will be taken it will be added with 1 or incremented with 1. So, here I will be getting after addition I will be getting 21, now this assignment means whatever value you get after computation will be assigned to this. Why this because on the left hand side I have got the same location X, this will be changed to 21.

Now suppose if instead I had the situation like this.

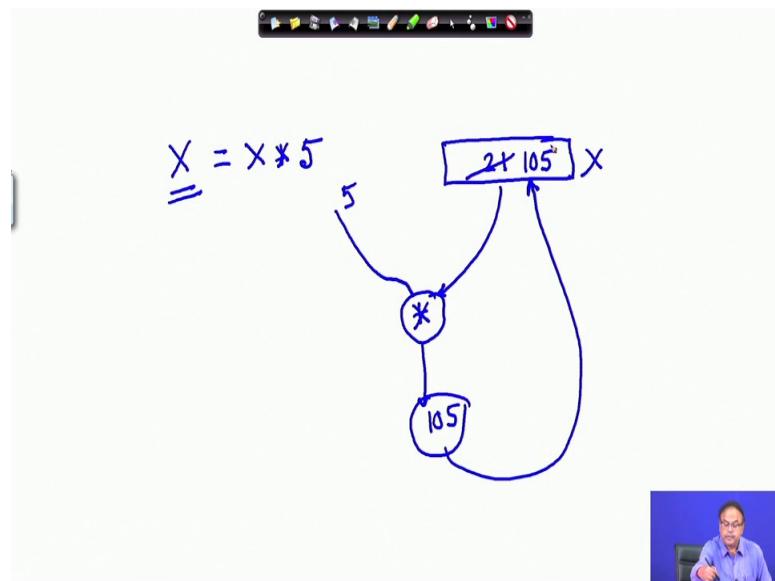
(Refer Slide Time: 26:32)



X at 20 and I write Y assigned X plus 1 what does it mean? It means that this Y is another memory location and what is been told over here, you take X take the content of X and, you will take 20, and have 1 added add them you will get 21 and assign it to Y in that case you see the picture, the main content of the memory location the value of the memory location X remains unchanged and another location gets the value 21 right unlike the earlier case. When we had written X assigned X plus 1 when I do that then in this case this value will be overwritten this 20 will be over written, but in this case it is not being over written it is remaining all right.

Now, let us go back to the slide here, when I therefore, you understand X was 20 assign 20, then next step X is assigned X plus 1, it becomes 21 the same location X is becoming 21, now I am doing X assigned X multiplied by 5.

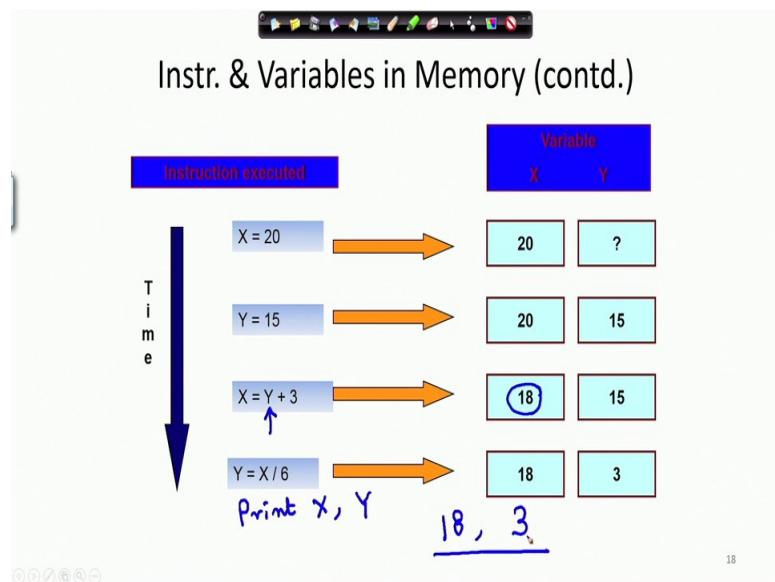
(Refer Slide Time: 28:22)



So, again what I am doing? I am the same thing X assigned X times 5, I had X and X was 21 here I took that value, and this time instead of adding the CPU is doing some multiplication and with this constant value 5 and is getting the value 105. Now this 105 is again written back to X, it is going over there and it will over write this 21 and we will write 105.

Therefore here we will have the result as 105.

(Refer Slide Time: 29:05)

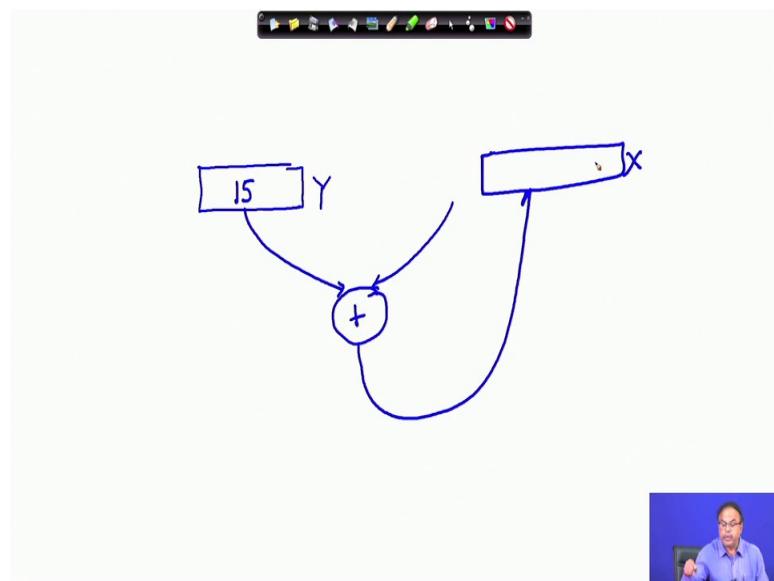


I hope you have been able to understand the difference between the variables and values here, and the variables the fact that the variables are nothing, but memory locations. I

will continue with this in the next class, quickly let us have a look at this. Let us revise here another situation another program segment the part of the program. X is assigned 20, Y is assigned 15, next step Y is assigned 15, X is assigned Y plus 3, Y is assigned X divided by 6 now; obviously, since there are 2 variables in this program, there will be 2 memory locations assigned.

Now, here in the first instruction X is being assigned 20 therefore, this location has got 20 written in it what is there in Y I do not know; anything that was earlier from the beginning was remaining that is there I do not care. Next step I am assigning 15 to Y therefore, now X remains as it is 20 and Y becomes 15. Now what am I doing in the third step? I am doing X assigned Y plus 3 just as I had explained in the last slide just now that I will be actually taking the content of Y.

(Refer Slide Time: 31:13)



Whatever Y was it was 15 or something I take it and add something to this and store it back to X all right that is what I am doing right.

X assigned Y plus 3, Y was 15, that is taken 3 is added to that the result is 18 and where is that loaded, that it loaded into X and, it becomes 18 and what happens to Y? Y is not disturbed because here at this point Y has just been read it is not been written into Y has been just read, I read from this location I did not disturb it I disturbed x. So, I added 3 and made it 18 here what I am doing? I am now disturbing Y how? I am reading the value of X dividing it by 6, 18 divided by 6 will be 3 I am writing that to Y therefore,

now I am disturbing Y and Y has got a new value, now, my result if I had written here print X and Y then my result would be 18 and 3 right.

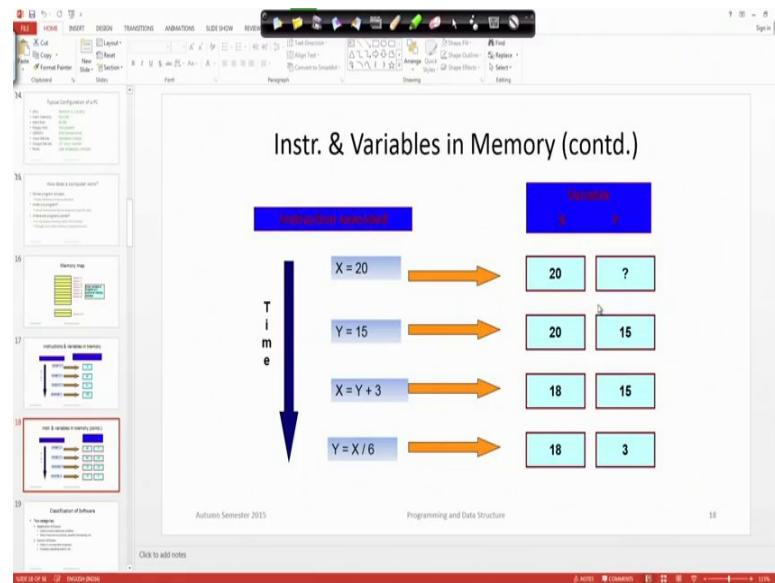
So, that is how the variables are handled by a programme and I tried to explain to you the difference between variables and values, you please look at the pseudo code of the examples that we did in the last lecture and see what are the variables and what are the values and try to draw a diagram as I have done today just to have clearer conception for that.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 06**  
**Types of Software and Compilers**

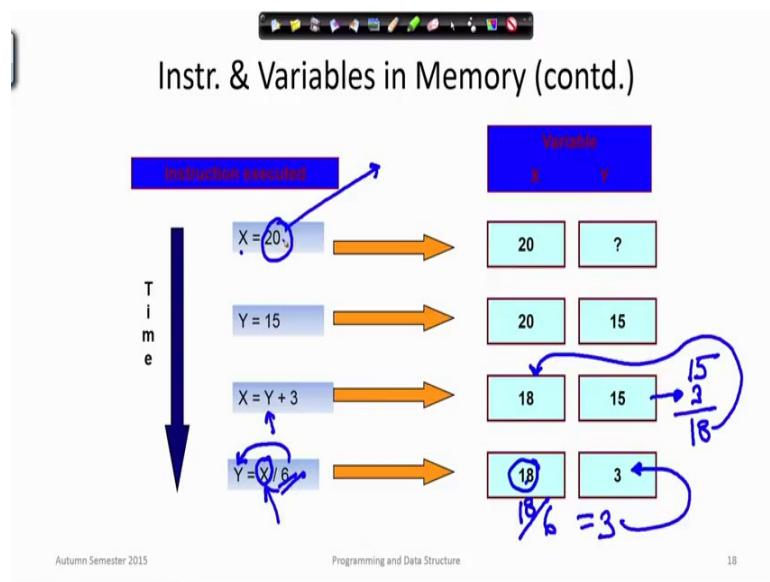
In the last class, last lecture, we were discussed about variables and values

(Refer Slide Time: 00:24)



And we explained that variables or mapped by the compiler to different memory locations. So, whenever we mention about any variable X Y or Z, each of them corresponds to a memory location, each of them corresponds to a memory location and each of them corresponds to a memory location

(Refer Slide Time: 00:48)



And whenever we assign some value like 20 to X; that means, in a particular memory location that value is written. So, we had done this example once again, we quickly go through it that X is being assigned 20. So, the memory location corresponding to X is getting the value 20 whereas, the memory location corresponding to Y can be anything, when we assign in this statement the value 15 to Y.

Then 15 is written in the location, corresponding to Y, when we do Y plus 3 and assign it to X; that means, actually we are reading this value of Y, here reading the value of Y which is 15 and we are adding 3 with that and we are getting 18 and that 18 is written into this location, all right. This may appear to be very simple, but this way of thinking or way of looking at the things will come in very handy as we will see later, during the later phases of programming.

Now, here again when we are, what is meant by this statement? That the value how will I read? I will read it like this that the value of X value stored in X that is 18 will be divided by 6 and that result will be stored in Y all right. So, X was 18, 18 has been divided by 6 and we get 3 and that 3 is written in Y. Now, here there are couple of things, that what is this 18? What is this 20? These are values and whereas, these are variables, these are also known as constants. Constants are the values which do not change during the execution of the program, next we can now, we can think of the software.

(Refer Slide Time: 03:26)

## Classification of Software

- Two categories:
  - Application Software
    - Used to solve a particular problem.
    - Editor, financial accounting, weather forecasting, etc.
  - System Software
    - Helps in running other programs.
    - Compiler, operating system, etc.

A hand-drawn diagram on the right side of the slide. It shows a bracket labeled 'HW' (Hardware) enclosing three items: 'CPU', 'Memory', and 'IO'. An arrow points from 'SW' (Software) down to a bracket labeled 'Instructions'. A blue arrow points from the 'Instructions' bracket to the 'HW' bracket, indicating that software instructions are executed by hardware.

Autumn Semester 2015      Programming and Data Structure

A small video frame showing a man with glasses and a blue shirt, likely the lecturer, speaking.

Now, you know any computer system; consist of hardware as well as software right. So, we will have some hardware as well as software. Now, the hardware is consisting of the C P U, the memory, the I O devices, all those things are hardware and the software is the instructions, that this hardware that is executed by this hardware, the instructions that are executed by this hardware constitute the software. Now, software can be of two types for example, first one is the application software; application software is the software that we write.

(Refer Slide Time: 04:16)

## Classification of Software

- Two categories:
  - Application Software
    - Used to solve a particular problem.
    - Editor, financial accounting, weather forecasting, etc.
  - System Software
    - Helps in running other programs.
    - Compiler, operating system, etc.

A hand-drawn diagram on the right side of the slide. It consists of three concentric circles. The outermost circle is labeled 'Application Software'. The middle circle is labeled 'System Software'. The innermost circle is labeled 'Hardware' (HW). A blue arrow points from the 'Application Software' label to the outermost circle. Another blue arrow points from the 'System Software' label to the middle circle. A third blue arrow points from the 'Hardware' label to the innermost circle.

Autumn Semester 2015      Programming and Data Structure

So, we can just have an idea of this, through this onion type of diagram at the core, we have got the hardware. The hardware is here and I am putting two layers around this and the user is standing somewhere here and the user is not directly interacting with the hardware, because the hardware nearly understands ones and zeros and it is very difficult for the user to write in ones and zeros. So, the user will write in some high level language H L L, in which the user writes and the system automatically converts it into a way that is understood by the hardware.

And what is that automatic way of converting it, we have seen that is compiler, is a software compiler is again a software right, a complier is software which converts before coming to application software. Let me talk about system software. So, we know that when the user has written something in high level language, that is converted by a program called compiler into the machine level language or high machine level language right, which the hardware understands.

So, the compiler is let us mark it like this. These are part of this layer, which is the system software similarly; operating system is another very important software, that is lying in this layer, internal layer which enables the user to use the computer in a much more user friendly way and in a much more efficient way. So, operating system compiler etcetera are the core, very important elements of the entire computer system, without which we cannot, we would not be able to use the computer in, as efficient way as we do it now a days.

Now, given this hardware and this layer of operating system, and compiler, and other system software, now, we are in a position to write some programs for our day to day use, for example, a company wants to find out the salary information of the people, they can use some pay roll software, here or for example, you want to design some data analysis software that will take some data and using a particular software will analyze the data statistically and give you some good insights.

So, all those things, the user is writing and they are forming the application software. So, most of the time the programmers, who are not systems programmers, not the system designers, but just users they mostly use the application software given this, we come to a very important software.

(Refer Slide Time: 08:10)

The slide has a title 'Operating Systems' at the top. Below it is a list of bullet points:

- Makes the computer easy to use.
  - Basically the computer is very difficult to use.
  - Understands only machine language.
- Operating systems make computers easy to use.
- Categories of operating systems:
  - Single user
  - Multi user
    - Time sharing
    - Multitasking
    - Real time

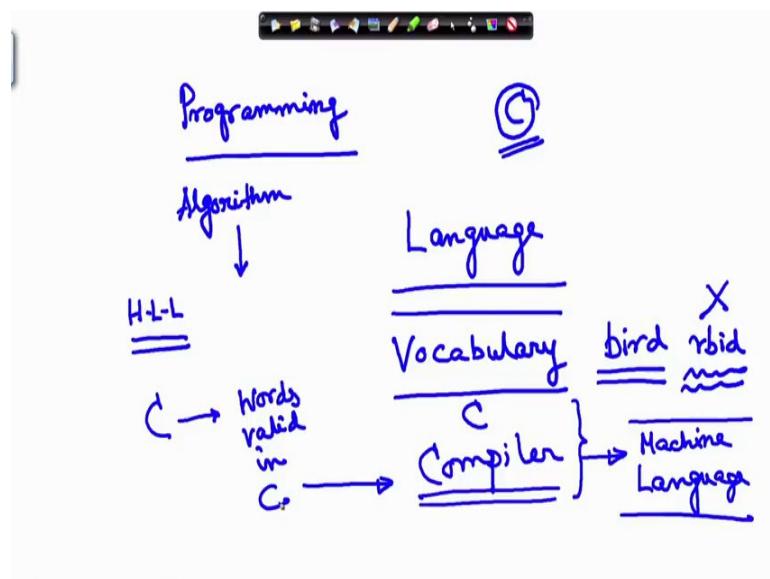
On the right side of the slide, there is a hand-drawn diagram. It shows a blue circle representing a computer system. Inside the circle, there is a smaller blue circle. A bracket is drawn around the top part of the outer circle, containing the text 'Windows', 'Linux / Unix', and 'Mac OS'. An arrow points from the word 'Compiler' written below the main circle towards the inner circle.

Autumn Semester 2015      Programming and Data Structure      20

We just now mentioned, that is an operating system. Now, you know that you are aware of typical operating systems like windows, Linux or Unix, now a day's Apple is becoming popular Mac OS, all these things are operating systems.

Now, what is the operating system? The operating system is a layer around the hardware which enables a user to use the software, use this computer system. I am sorry, use the system in a much more friendly way. Now, there are different varieties of operating system; single user operating system, multi user operating system, etcetera. Now, this operating system also activates another system software that we have talked about, that is a compiler right. The operating system will call or will activate the software system, software call the compiler, when you want to run a high level program, given this background. Let us now move to discussion on programming.

(Refer Slide Time: 09:41)



We know by now that programming means we have to express our intention of solving a problem by executing a number of steps and those number of steps, once again you know that by now, we start with the algorithm and that algorithm can be expressed in different ways like pseudo code or flow chart and then the programmer actually writes them in a high level language, some high level language.

Now, what we are going to discuss now, is a particular high level language, which is called C. We are taking C just as an example of an high level language, because we have to express the logic in the form of some high level language. We are taking C as an example and as I have mentioned earlier that the logic, the style and the philosophy, remains more or less the same across different programming languages like Java, C plus plus and others of course, C is the simplest to start with.

Let us see now, we are using the term language, think of a human language; any language is constituted of some vocabulary right. The vocabulary say for example, in English, the vocabulary consists of some words right. Different words like a bird is a valid word in a, in the English vocabulary now, but if I had written r b i d that is possibly not a valid word in the English language vocabulary all right.

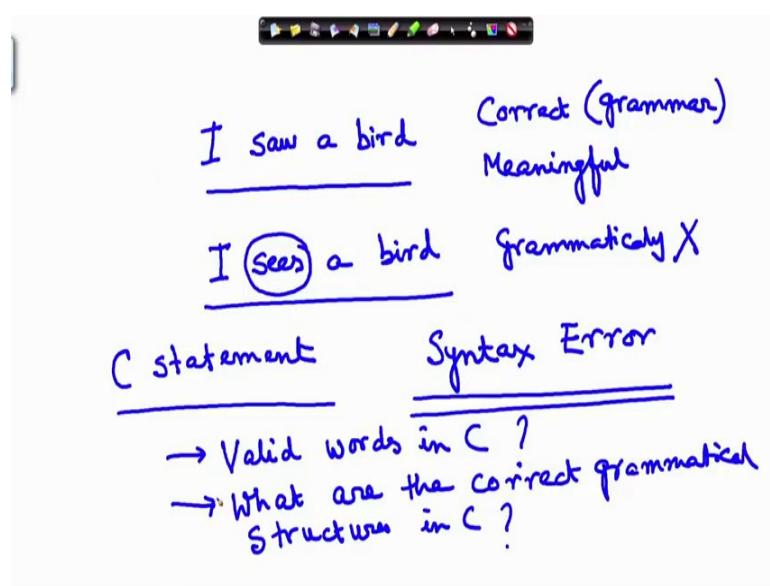
Now, we have got some valid words. Similarly, C will also have some valid words, which will see through, which we can express the basic elements of a C program, just as English sentence is built using English, valid English words, otherwise the meaning will

not be understood in the case of C programming language. Say C will have, it is own vocabulary right. The words valid in C now, if I had written some word in English, that is now some string, some patter in English like r b i d that unless this has got some special meaning, this will not be an, if this is a part of a sentence, this will not be very clearly understood by anybody.

Now, when I have written a C program, who is going to understand this, for whom I am, I writing this C program, I am writing this C program, for the compiler C, compiler all right. I am writing it for the C compiler and the C compiler is responsible to understand this and just as we understand an English sentence open the door. So, we understand the meaning of that sentence, we go and execute that we open the door.

Similarly, in C if we write something, unless the compiler understands this, it will not be able to convert it to the machine language, which will be executed by the hardware or the computer right. So, the C program must constitute of valid C words and we will see what are the valid words and what are the rules for that. Now, the next thing, if I write a particular sentence

(Refer Slide Time: 14:38)



I saw a bird that is a valid English sentence, why is it valid? Because it is grammatically correct and also it is carrying a very clear meaning. It is a meaningful sentence right, this is both correct, grammatically by grammar and also it is meaningful. Now suppose, I wrote it, wrote something like I sees a bird. Now, this is grammatically wrong; however,

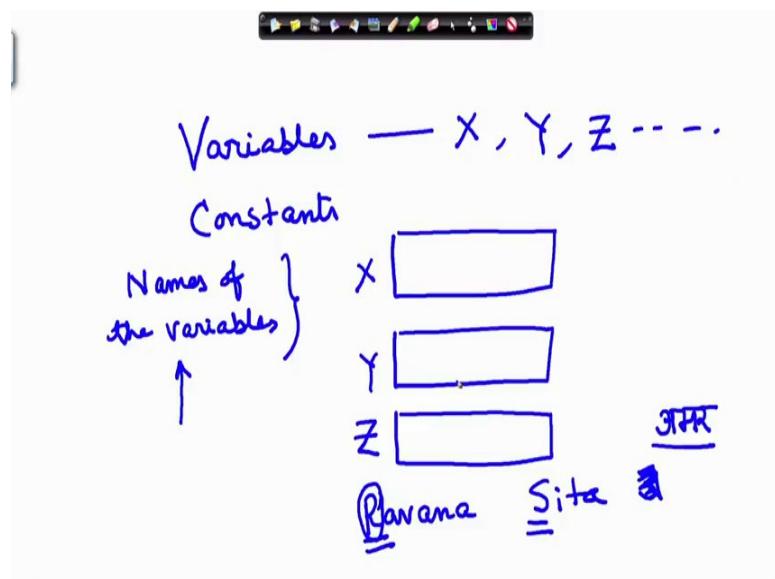
it conveys some meaning, I understand that the person who wrote this is weak in grammar, he is not very much conversant with subject, verb, agreement, but still I can make a meaning out of it.

On the other hand for a compiler, if I write a C sentence, let me call it not sentence, let me call it C statement. Now, a C statement; that means, a statement in the language C will consist of some valid words in C and also it will have to follow some grammatical rules of C. Unfortunately, here although it is grammatically wrong, I could understand the meaning of this, but a C statement, if it is grammatically wrong, grammatically according to the grammar of C, if it is not in tune with the grammar of C.

Then is grammatically wrong and since this C sentence will be interpreted not by a human being, but by a machine, computer hardware smart, it may look like is basically, not as intelligent as human beings. So, that as of now, whatever we can make out the meaning of it a compiler a C program will not the compiler, will not be able to make out correspondingly the correct machine language cannot be generated therefore, if there is something that is grammatically wrong according to C grammar that will be indicated marked by the compiler as a syntax error all right. So, unless we write something in the correct syntax, there is always a chance of it will it not there is always a chance it will; obviously, lead to a syntax error and the compiler will not produce the corresponding machine code, all right.

So, we are now supposed to learn what is therefore, we need to know what are the valid words in C and what are the correct grammatical structures in C. Now, if we learn both these, we learn the C language similarly, if you want to learn any other language computer language you have to exactly know this things what are the valid words, in that language and what is the correct grammatical structure in that language. So, given this we will start looking at C programs, even before that we once again recapitulate.

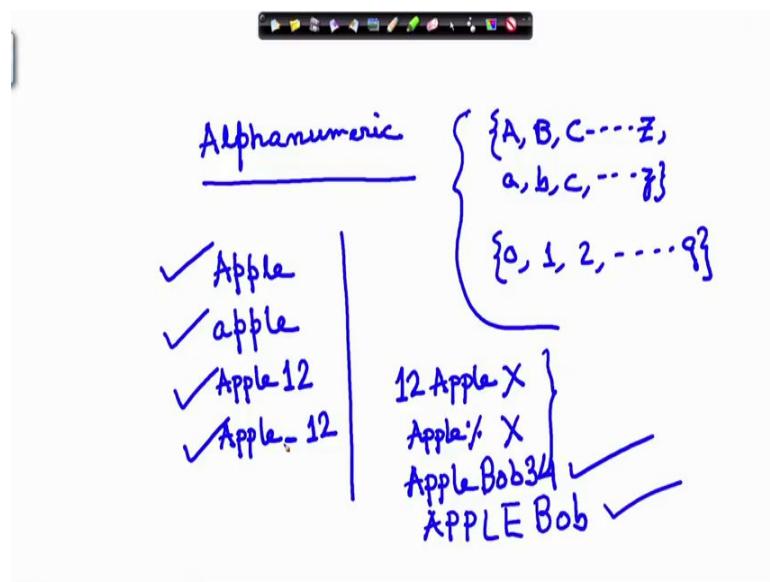
(Refer Slide Time: 19:34)



Whatever we are expressing them using some variables constants, right and the variables I have got, we have shown X Y Z, etcetera, at variables. So, each of this variables which are nothing, but memory locations are been given some names right. So, we have to give some names of the variables all right. There are some rules for naming them just as in English, we start a proper noun with a capital letter Ravana, all right Sita, we write them with the capital letter this is the rule of English; however, if you write in any Indian language, say you write anything, Amar you write that there is no question of any capital letter here, but it is a, it is a property of English, it is a rule of English that the capital first letter of a proper noun must start with the capital letter.

Similarly, for naming the variables in C there are some rules. So, here we have seen X Y Z, etcetera. These are the variables. So, there are some rules for naming the variables, we will come back to this again, but first of all any string of alphabets say, let me introduce one word; alphanumeric.

(Refer Slide Time: 21:45)



Alphanumeric means what alphabets and what are the English alphabets A B C up to Z small letter a b c up to z that is alphabet and numerals, we know 0 to 9 all right 0 1 2 3 up to 9. Now, an alphanumeric means A union of both these either alphabet or numeral. So, a variable name in C can consist of any alphanumeric character now.

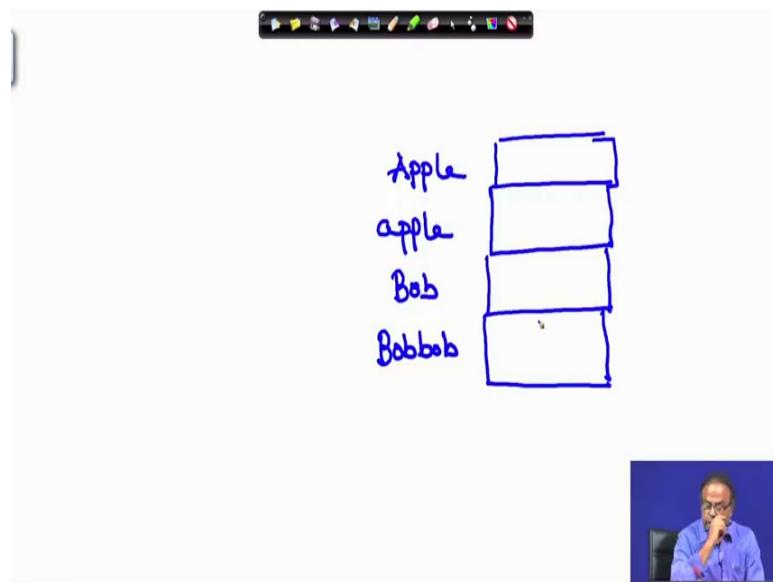
So, it can be say, Apple is a valid variable name in C, again if I start with small a this is also valid variable name in c, but although these two are same, since I have put in different characters, one capital, one small, these two will be treated as two separate variable names all right. Similarly, I can write, say Apple 1 2 that is also valid variable name. Now, there are some special characters like underscore, that is allowed like I could have written something like this, Apple underscore 1 2 that is also valid variable name.

However I cannot start a variable name with a digit or number for example, 1 2 Apple is not a variable, valid variable name other special characters like say, Apple percentage is not a valid variable name. So, these are some of the rules. So, the what are rules of naming a C variable, it can consist of any alphanumeric character, any length, but it must start with an alphabet and only some specific special characters like the underscore is allowed, others are not.

So, this is not allowed, but say Apple bob 3 4 is a valid name. So, here I can use as again a p p l e b small o, small b, there is also a valid name. So, in general what is the rule; I can have a string of alphanumeric character, starting with an alphabet and having no

special characters except for this underscore all right. So, with that, that is how we will name the variables and each variable I will again repeat, you know that a variable essentially consists of memory locations, a variable is nothing, but a memory location and we are putting the name to that particular memory location.

(Refer Slide Time: 25:34)



So, if Apple be this memory location, may be a p p l e is another memory location. Now, which variable will go to which memory location is decided by the compiler. So, might be Bob is another one, all right. So, these are separate memory locations.

So, variables and naming of variables is a fundamental step in writing a C program, because whatever we write, we have to write them through variables, now quickly once again we look at this that whenever we are writing suppose, again I am writing.

(Refer Slide Time: 26:28)

A, B, C, AVG

AVG = (A + B + C) / 3

C - Sentence

C - Statement

Now, suppose there are three numbers A B C or. So, I want to find the average of that. So, I can write it in the form of say and say average, you can see that average is the valid name in the case of a variable. So, I can say A V G is A plus B plus C, whole thing divided by 3. Now, this one I can call to be A C sentence or I am saying sentence.

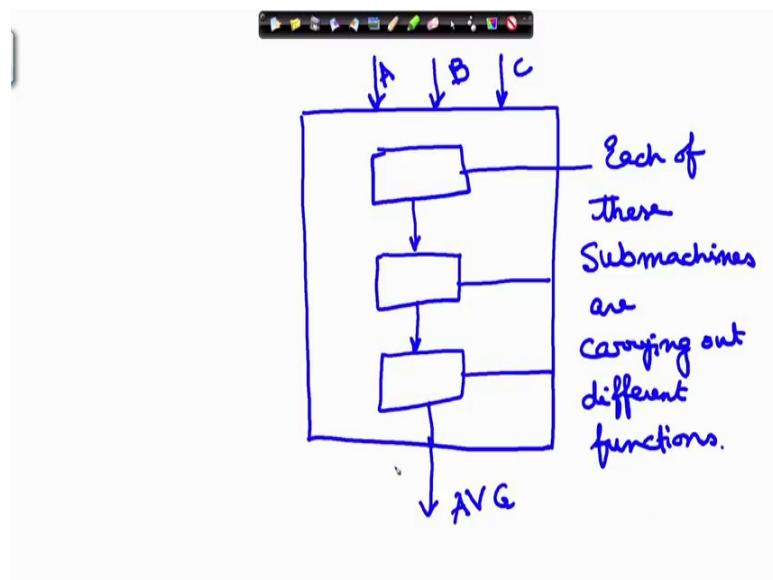
So, that you can have an analogy without English, but we will call it C statement all right and this C statement consists of several things, will come to the other things, but you can see that we are, it is consisting of 1 2 3 4 variables and one constant that is 3 all right and there are some special parenthesis and all these are also valid symbols in C and these are operators. So, we will see how was C statement is constituted, but whenever we write A C statement, it will consist of a number of variables.

(Refer Slide Time: 28:12)

A C program consists of a number of (may be one) functions.

Now, coming to A C program, A C program consists of a set of in, let me call it a number of may be one, but at least one number of functions. What are functions? There is a significance of this term functions, but for the time being let us simply try to visualize it in this way that I am trying to build a machine.

(Refer Slide Time: 29:17)



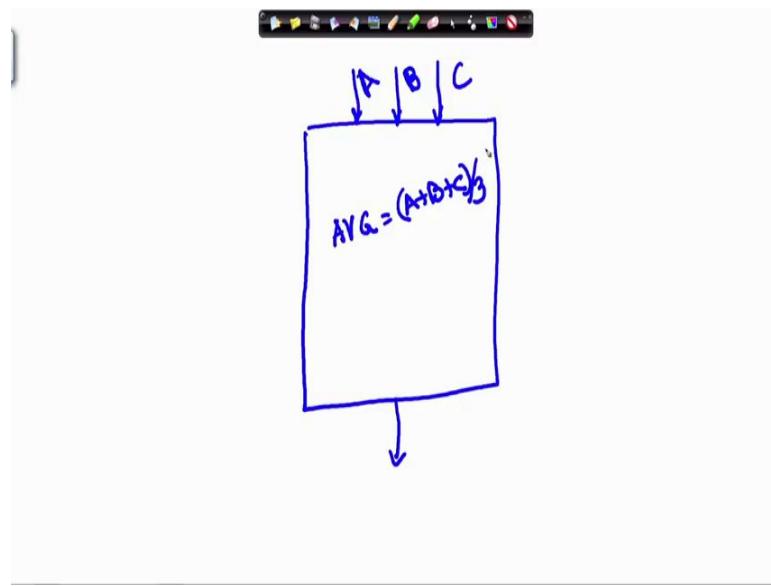
Which will do something, what will it do.

Say for example, it is a machine that will take, say 2 or 3 numbers, A B C and will produce the average. Now, this is the machine all right. It does something; takes some

input and give some output. Now, this entire machine can be built of with smaller sub machines all right and one machine, each of this sub machines can do some specific task for example, if I take a complicated, say paper rolling machine, then there are so many things to be done in order to roll out or prepare papers.

So, similarly, there can be different sub machines, which are doing different functions. So, each of them, each of these sub machines are carrying out different functions are carrying out different functions, all right. This is doing one, this is doing another and all these three together is doing something.

(Refer Slide Time: 31:05)



Now, as I said in the worst, in the, in a special case just like for a simple case of average, I may not need many sub modules, because I can very well take A B and C here, A B and C and simply in this machine I can write a program like A V G is equal to A plus B plus C divided by 3 and that will be the output right.

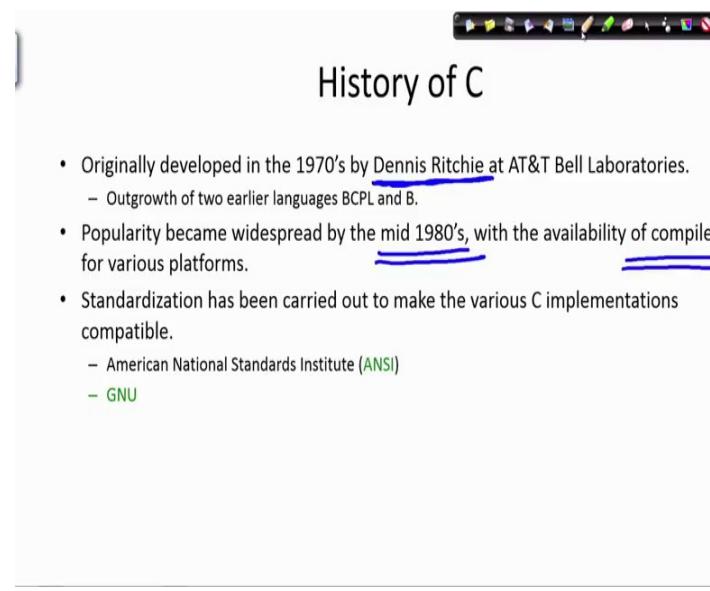
So, here I do not need any sub machines, because it is simple problem, but never the less, I need at least one function, that is what is this task is being done. So, any C program will require at least one function and in many cases we will see, it will require more number of functions. We will come to this in the next lecture.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 07**  
**Introduction to C Programming Language**

So, we have started our discussion on a specific programming language; that is C and once again I repeat, that a C program will be constituted of some valid words of the C language, just as the an English sentence must be constituted with valid English words, similarly for C and must be every language has got a grammar. So, C has got a very strict grammar and any statement that is written in C that is not adhering to that grammar will not be accepted by the compiler. The compiler will reject it, saying that it is a syntax error or grammatical error. So, before going further into discussions on C briefly, let us look at the history of C, it was originally developed in 1970 by Dennis Ritchie all right and his book.

(Refer Slide Time: 01:17)



The slide has a title 'History of C' and a list of bullet points:

- Originally developed in the 1970's by Dennis Ritchie at AT&T Bell Laboratories.
  - Outgrowth of two earlier languages BCPL and B.
- Popularity became widespread by the mid 1980's, with the availability of compilers for various platforms.
- Standardization has been carried out to make the various C implementations compatible.
  - American National Standards Institute (ANSI)
  - GNU

I have also referred to you at AT and T, Bell labs and become very popular by the mid 1980's, because it was the compilers. Now, a language, a computer language cannot be popular, cannot be used unless there is a compiler for it. So, it took some time, although it was developed in seventies, but many compilers in, for different platforms, for

workstations, for PC's, for different platforms were made available by mid 1980's and this became very popular and there were some standardizations that also took place.

Now, even that we, let us look at the structure of a C program. Let us look at the structure of a C program, here every C program will consist of

(Refer Slide Time: 02:15)

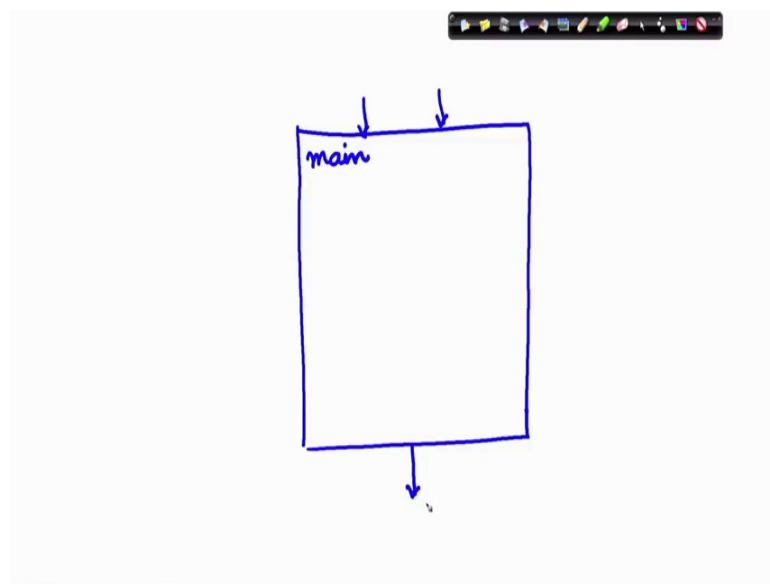
## Structure of a C program

- Every C program consists of one or more functions.
  - One of the functions must be called main.
  - The program will always begin by executing the main function.
- Each function must contain:
  - A function *heading*, which consists of the function *name*, followed by an optional list of *arguments* enclosed in parentheses.
  - A list of argument *declarations*.
  - A *compound statement*, which comprises the remainder of the function.

4

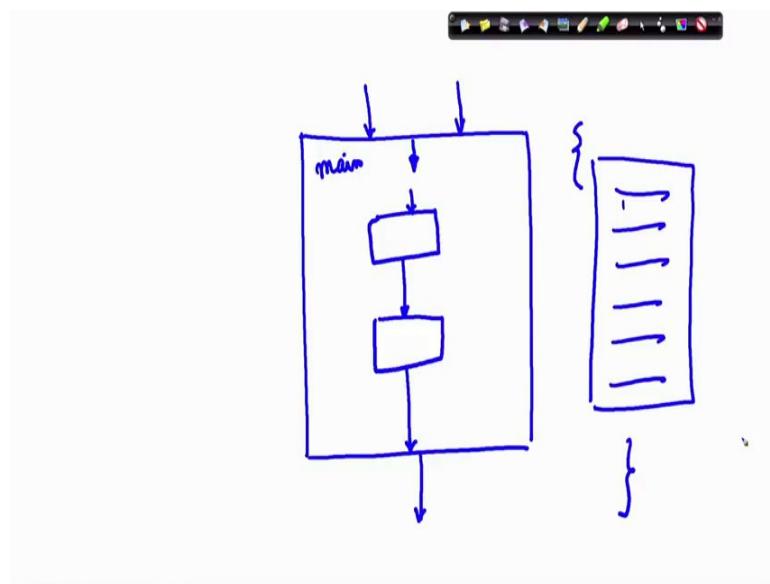
One or more functions one or more functions. One of the functions will be given the named main, which is the function. Now, I in the last lecture, I said that there should be at least one function, if nothing else is there, if the even, if the program is. So, simple that I do not need any sub machines, in that case I can do it with only one simple function and that function has to be named main and if there be a number of say for example, sub func, number of functions or sub machines, in that case also there has to be one program, which has to be called the main and the rest can be given some other names, the program will always begin by executing the main function. So, once again let us go to the diagram that we had used in the last lecture.

(Refer Slide Time: 03:17)



Say I have got a program, which has got only one function, very simple then this function will be called main and no other sub functions are there it will take some inputs, whatever the inputs are, it will process that and the output will be made available. Now, suppose I have got two sub machines, required for this one here and after that another one.

(Refer Slide Time: 03:46)



So, this sub machine will do something, then this sub machine will do and then will come out and this is the output and there is some input coming in. Here these are the

inputs and they are some sub machines, but I have got one machine, which is the main. This is again the main. So, although I am taking help of the sub machines, I have to first enter the main function and then from the main function I can enter here and go somewhere else and ultimately, I will have to come out through the main function. So, we cannot escape the function.

So, the main will be there and typically we write the main any C body, within two curly braces, here whatever I am drawing as a diagram that is equivalent to these two curly braces. Inside this, whatever is written is a C program, whatever I write here, whatever C program I want to write, I have to write within these two curly braces, I will come back to this in a moment. So, so there

(Refer Slide Time: 05:10)

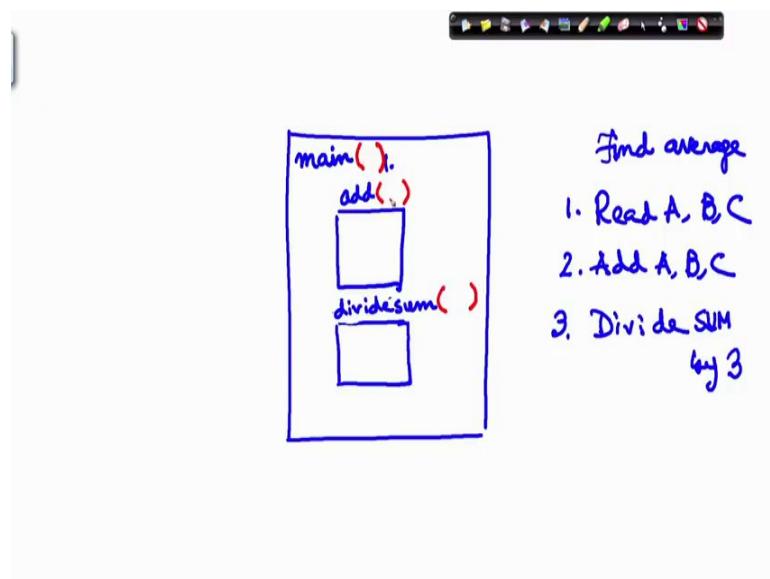
Structure of a C program

- Every C program consists of one or more functions.
  - One of the functions must be called *main*.
  - The program will always begin by executing the main function.
- Each function must contain:
  - A function *heading*, which consists of the function *name*, followed by an optional list of *arguments* enclosed in parentheses. <      >
  - A list of argument *declarations*. ←
  - A *compound statement*, which comprises the remainder of the function.

4

has to be one function called the main and the program will always begin by executing the main function, each function must contain a heading, which consists of the function, name followed by something application, say; followed by an optional list of arguments, I will explain that later.

(Refer Slide Time: 05:49)



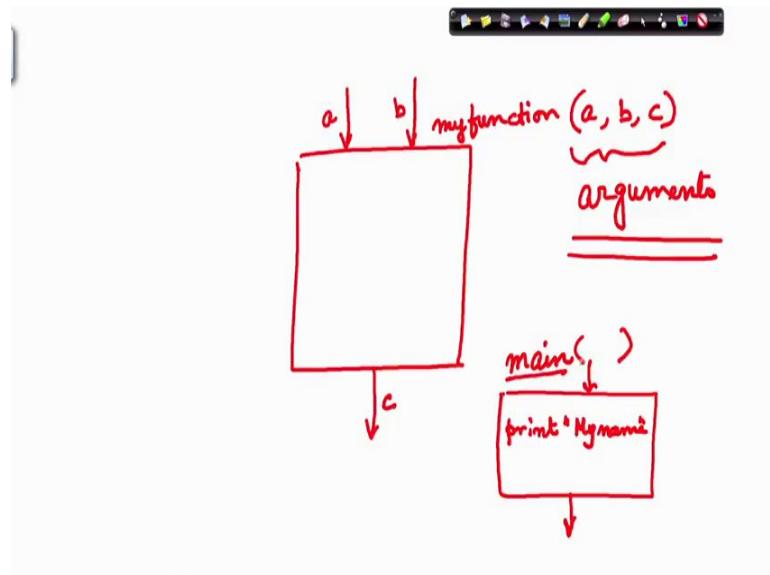
Let us again come back to this, that I have got a function, I have got a program which has got one function, whose name is main and I have got another function, whose name is say add and there is another function whose name is find. Is it readable? Find average or let me, this is a little confusing, let me rename this, I name it as divide sum, take the example of our simple finding average all right. I want to find average.

So, what are the sub tasks, one is I have to read the numbers, say read say ABC that reading suppose, I am doing here this read ABC is here, one after that I am doing, add ABC, say adding ABC is being done by this sub machine say, and therefore, I have to give a name to this, sub machine that this sub machine is inside the body of this sub machine, I will actually do the addition, but the name of the sub machine should be expressive of what.

This sub machine of the function is supposed to do similarly, after I do that then I divide sum right I will divide sum by 3 by whatever 3 here in this case. So, that is being done by this sub machine this is too simple these are two simple sub machines, but this is being done by this submachine. So, they each of them has to be given some name each of the functions including main has to be given some name. Now, along with that there is another point that has been mentioned here, there is the argument each of these functions must have a place, where I can write the arguments right. Now, I am not writing anything

on main, but say when I say what are these arguments, if I just consider this separately all right, any function separately any function

(Refer Slide Time: 09:05)



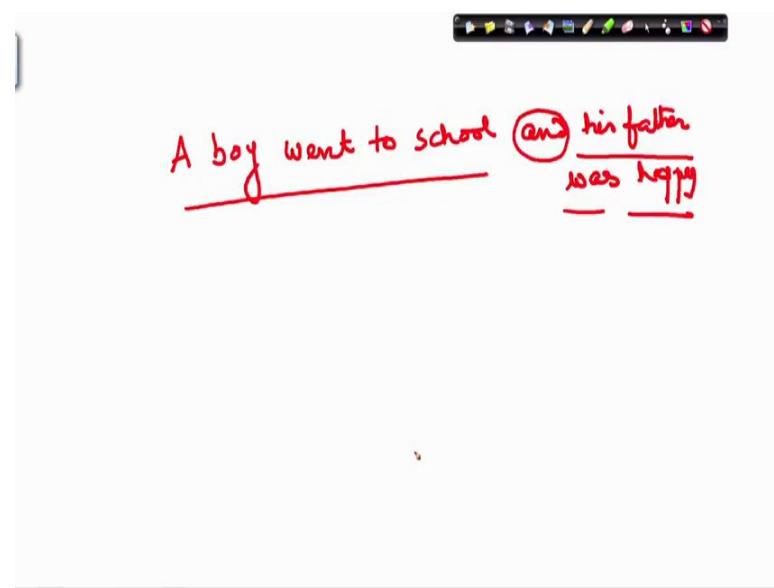
If I consider separately, now this function will have some inputs and some outputs. What are the inputs and what are the outputs? Suppose, the name of this function is my function, then this function will have some arguments, which are given in within this parenthesis and suppose, these are A and B are two variables, which are read by this and C is a variable that is output by this.

So, the name of this function should be associated with the names of the variables, which are taken as input or is supplied as output by this function these are called arguments, we will come to this when we discuss some other very important properties of functions, but for the time being just remember that a function must have a name and there should be some place for writing the arguments, often you will find that a function like main.

Suppose, I am writing a function which will simply print a particular line, then inside the body of the function I just want to print, I will say print my name, is something whatever. So, my name just say my name. Now, this function whenever it is entered, it will just print my name it is not requiring any values to be passed, but still then I will have to put this parenthesis with the name although, I may keep the inside of this parenthesis gland. So, a function in order to be a valid function in C, these are a C rule that in order to have a valid function. A function must have a name and a place for the

arguments. So, a function heading is the function name all right, followed by an optional list of arguments enclosed in parenthesis like this is optional, because it can be blank and there are these are the things that there will be some argument declarations, etcetera, will come to this particular part later, but before that we have to understand a compound statement. What is a compound sentence? Say, simply in English if we consider, you know that a simple sentence has got only one principal, verb principal finite verb; however, I can have a compound sentence like a boy went to school and his father was happy say.

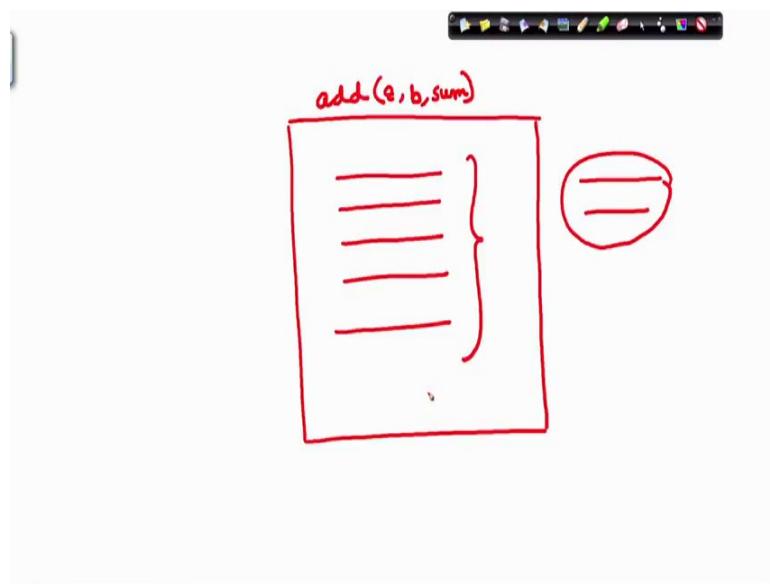
(Refer Slide Time: 12:41)



So, here there are two sentences his father was happy and a boy went to school. These are two simple sentences and we are connecting them with an end here, all right. So, these are compound sentence. Similarly, we can have complex sentence and all those.

Here the idea is similar, but not exactly the same, I am saying that.

(Refer Slide Time: 13:23)



A function is a machine, which has got a name like say, add and sum parameters A B C A B and the third parameter is sum; that means, it will take A as input B as input and will produce sum, but here I write a number of C statements, a number of C statements are written over here. Each of this C statement is a statement and all these together are together forms are compound statement. Whenever, we find more than one C statement working together that will be a compound statement. So, here the entire function add will be if it consists of a number of statements, then it is a compound statement, in the default case, in the very trivial case, when the program consists of only one statement then also we can call it a compound statement, but with only one statement.

(Refer Slide Time: 14:55

Contd.

- Each compound statement is enclosed within a pair of braces:  
'{' and '}'
  - The braces may contain combinations of elementary statements and other compound statements.
- Comments may appear anywhere in a program, enclosed within delimiters '/\*' and '\*/'.
  - Example:  
 $a = b + c; /* ADD TWO NUMBERS */$

( ) { }

So, next, each compound statement is enclosed within a sorry, is enclosed within a pair of braces like this, this is called braces, this is called parentheses and this is called braces right.

The braces may contain combinations of elementary statements, a single one or other compound statements. Now, once again let us see till now, I was drawing the machine as this

(Refer Slide Time: 15:39)

It rains.

{

$a = 20 ;$

$b = 15 ;$

$c = a+b ;$

}

$a=20, b=15;$

$a=20 \quad b=15$

??

I was drawing it as a rectangle. Now, I will move towards, more towards C. So, I will say that this boundary will be specified in C as this sort of boundary, whatever is here is within this parenthesis. Now, each of them, each of this braces, I mean inside these braces, we will have some statements. It can be an elementary statement like say, A assign 20 or maybe more number of statements. Now each of the C statements are delimited by a semicolon.

This is very important, this you must remember, each of these are delimited by semicolon, if I do not write the semicolon, the sentence is not completed, just like in English we have to write a sentence and we have to complete it by a full stop right. It rains very simple sentence, it rains and then there is a full stop here.

Similarly, in C, the end of one single elementary statement or a set of statements is pointed out by semicolon all right. Otherwise, it will be ambiguous, there can be a problem like for example, if I write A 20 B 15 and I forget the semicolon. If I can, I am writing one after another, because that is nice to write one after another, but if I write them side by side that is also equivalent, but if I put semicolons here, the compiler will understand that this is a one statement.

This is another statement, but if I do not give the semicolon here, for example, I write A 20 B 15, then the compiler will be in a problem, because it does not know what it is supposed to assign 20 or 20 B or whatever is it possible to assign 20 B many things will come. So, we must be very careful about completing the statements with semicolon.

So, the braces may contain combinations of elementary statements and other compound statements. Now, there is another very important thing called comments, whenever you write programs, you must be generous of writing comments. Now, what are these comments? Comments are statements, which are not, which are not converted by the compiler to machine language, but then why do we write it? The reason is that whenever we write a complicated or a large enough programs, the presence of the comments enables us or enables suppose, I have written a program, it will enable somebody else to understand the program. So, if I write something like say

(Refer Slide Time: 19:45)

The slide shows a presentation interface with a toolbar at the top. The main content area contains handwritten C code:

```
a = 20;  
b = 15;  
sum = a+b; /*  
c = 25;  
sum = sum + c;
```

In the bottom right corner of the slide area, there is a small video window showing a man in a blue shirt speaking.

I am again, I am not going to a complicated example as yet I am remaining with that finding the average of the numbers, but suppose, I am doing it in two steps all right. Say, I do, I am writing, I am just doing something like A assigned 20, B assigned 15, sum assigned A plus B and then C assigned 25, sum assigned sum plus C. Now, when I write this, somebody may this rather too simple, but somebody may say what are; what am I doing here at both the places? I am doing sum. So, that somebody else does not get confused, I can write something like this. Let me just check the syntax here, yes I was right. So, what is happening?

(Refer Slide Time: 21:07)

The slide shows a presentation interface with a toolbar at the top. The main content area contains handwritten C code with annotations:

```
a = 20;  
b = 15;  
sum = a+b; /* adding first 2 numbers */  
/* _____ */  
c = 25;  
sum = sum+c; /* adding 3rd no. to sum */
```

The annotations explain the code: the first line of the sum assignment is followed by a comment /\* adding first 2 numbers \*/. Below that, there is a line of asterisks /\* \_\_\_\_\_ \*/. The final line of the sum assignment is followed by a comment /\* adding 3rd no. to sum \*/.

So, I can write say, A assign 20, B assign 15, note the semicolon that I am putting after every statement, sum is equal to A plus B. A plus B and here I can put in a comment, say adding first two numbers I am ending this with this symbol, this is again I am doing a bad job here. Say, I write numbers and then I put this sort of symbol.

So, here you see I put here this and here reverse this, whatever I write in between that is assumed by the compiler to be a comment statement. So, the compiler need not convert this into machine code. So, here again I can write later say CS sorry, C assigned 25 and sum equals some plus C. Now, I can explain that adding whatever I want I can write in any form, adding third number to sum and put this I am sorry, here always I am making, I am not being able to manage the space, it should be, there should be a space in between. So, I will have to rank the third number to sum and I put this end of comment. Now, this part will not be compiled.

So, for any program, when we write as the program, becomes more and more complicated. We should be generous about writing the comments that will also help us. Say, you have written a program today and he want to look at it after, say one month and see what you did? Such comments will be helping you to understand what you did or somebody else of course, to understand what you did right. So, for examples let us see here,

(Refer Slide Time: 24:10)

The slide has a title 'Contd.' and a list of bullet points:

- Each compound statement is enclosed within a pair of braces: '{' and '}'
  - The braces may contain combinations of elementary statements and other compound statements.
- Comments may appear anywhere in a program, enclosed within delimiters '/\*' and '\*/'.
  - Example:  
a = b + c; /\* ADD TWO NUMBERS \*/

Handwritten annotations in red highlight the text '/\* ADD TWO NUMBERS \*/' and draw a bracket underneath it, connecting from the start of the line to the end of the comment. There are also small red asterisks placed above the opening and closing slashes of the comment delimiters.

I have got a simple thing A assign B plus C. So, here you see, add two numbers between this to this and the end of comment is this, all right.

(Refer Slide Time: 24:30)

```
#include <stdio.h>
main()
{
    printf ("\n Our first look at a C program \n");
}
```

stdio  
standard input, output  
Keyboard → Screen / Printer / file

Now, let us look at a simple C program there for now, this part I will be explaining separately, but before that let us look at this part of the code is a very simple program, simplest possible program. What it does? It simply prints our first look at C program, this line as is will be printed as is will be printed, where will it be printed this? Will be printed on the screen or on the printer or on some other file, where will it be printed here, we are writing another statement hash include stdio dot h.

This stdio dot h means standard input output std for standard. So, it is stdio is standard input output. Now, if nothing is specified, then the standard input is our keyboard, by default. It is the keyboard and the output by default, is the screen. So, when I write this then, when I write print f; that means, this line will be printed on the screen that is the meaning of a stdio dot h.

Now, this single piece of simple program has got many things to illustrate. This term hash include is an instruction to the compiler that you need not convert it to the machine language, but before you convert the program to the machine language, please do this. What is doing this?

(Refer Slide Time: 27:17).

The slide shows a sample C program:

```
#include <stdio.h>
main()
{
    printf ("\n Our first look at a C program \n");
}
```

A handwritten note above the code area says stdio. h. In the bottom right corner, there is a small video window showing a person speaking.

Please, include stdio the standard i o dot h. Now, what is this stdio dot h? In order to understand this, you have to know what a library is now? Any programming environment, any programming environment provides you with a library of functions C, provides you a library of functions.

(Refer Slide Time: 27:36)

The slide has a title Library of functions / code.

Below it, a hand-drawn diagram illustrates the compilation process:

```
# include <stdio.h>
{
}

```

The code is shown on the left. To its right, a vertical line connects to three boxes on the right labeled "stdio.h", "machine code", and another empty box. A red arrow points from the code block to the "stdio.h" box, and another arrow points from the "stdio.h" box to the "machine code" box.

In the bottom right corner, there is a small video window showing a person speaking.

Now, this library of functions and library of other packages, which are library functions or codes, let us see that is already there now. So, I am going to write a program here and here there is a library and in that library there are many things here, all right and one

such thing is stdio and all these libraries files have got an extension dot h. Why that is? That we will see later. Now, when I am writing this part of the program, if I say hash include within this corner brackets stdio dot h; that means, whenever I am writing this program, whatever program I have written, that will be converted to the machine code that will be ultimately converted to the machine code.

While doing that, this before, doing that this stdio part must be included or must be included some of the information, must be included, that will tell me that well you see whenever you are doing this print f that print f will be, will have to be done on standard output and what is that standard output? The printer, it is a printer, so, that is the purpose of hash include. There is a preprocessing statement, compiler a preprocessing statement. Now, I had said that every language has got, some of it is own vocabulary. Now, in English when you understand, whenever I write print or write this words, carry meaning to you in English in C, we write it as print f.

(Refer Slide Time: 29:47)

Sample C program #1

```
#include <stdio.h>
main()
{
    printf ("\n Our first look at a C program \n");
}
```

Print Write

This is a special word of C, why they say if there is a meaning for that. That is in C, wherever we write, whatever we write, we consider that as if we are writing in a file, say in your office or in your work desk, whenever you write something, you write it on a piece of paper and put it in a file in C. We consider every input or every output device to be a file and this f stands for file as if I am printing on the file, which file? The printer

file, then the next class, we will start with this and will describe some of the other he brew like statements, here which will from, which we will go ahead.

Thank you.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 08**  
**Variables and Variable Types in C**

So, we are looking at a sample C program.

(Refer Slide Time: 00:18)

The screenshot shows a presentation slide with the title "Sample C program #1". The code is displayed in a code editor window:

```
#include <stdio.h>
main()
{
    printf ("\n Our first look at a C program \n");
}
```

Annotations explain various parts of the code:

- "Header file includes functions for input/output" points to the `#include <stdio.h>` line.
- "Main function is executed when you run the program. (Later we will see how to pass its parameters)" points to the `main()` line.
- "Curly braces within which statements are executed one after another." points to the brace `{`.
- "Statement for printing the sentence within double quotes (""). '\n' denotes end of line." points to the `printf` statement.
- A handwritten note "W" is visible on the right side of the slide.

At the bottom of the slide, there is a footer with the text "Our first look at a C program" and a navigation bar with icons.

And as we had discussed in the last lecture, we have a structure as is shown here. You can see that there is a main function here a main function is there, and that main function is covered by two parentheses. Now, we can see that we start with the header file that includes stdio dot h. An stdio dot h stands for standard IO.

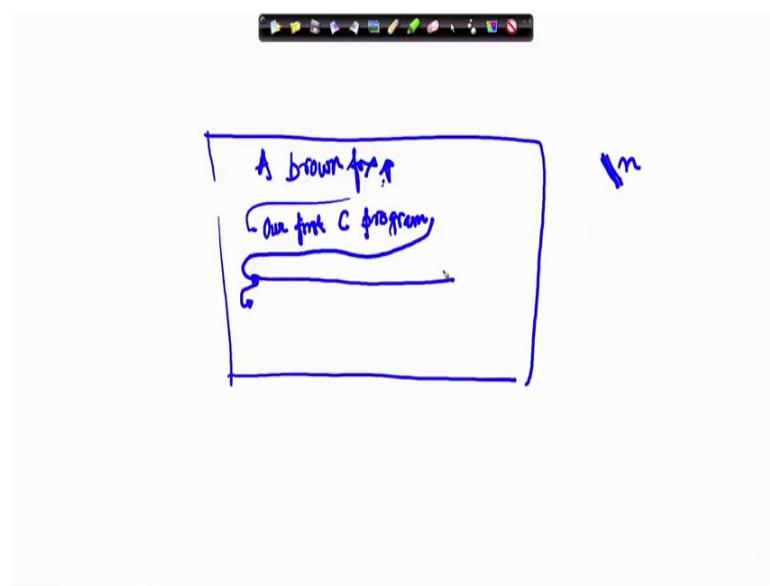
That means, whenever we will get some instructions to carry out input or output that will be in reference to or with respect to the standard input that is a keyboard; that means, the data will be taken from the keyboard. And if anything is to be printed it will be printed on the screen. So, that is a default thing for any C program we have to put in stdio dot h include hash include, this is known as this is read as hash include stdio dot h. Unless you want to take the file do not want to take it from the keyboard or do not want to print on the screen.

Next thing is that the main function which as I said in the last class that there must be a main function. And the main function will have a place for the parameters, you can see here for the parameters here which may or may not be empty. The third point is the structure the overall boundary of the main function of for that matter for any function there should be a boundary specified, and that boundary specified with by this to curly brackets.

Next we come to the statement. As I had said that just like any other language C also has got some words which are understood by any by the compiler; print if is one such word which is the statement for printing the sentence that is given within these double quotes. You can see that double quotes here and so, what will it print? It will print our first look at the C program here as his shown here that will be printed.

Print f within quote here there is a quote n quote and here is a start quote our first look at the C program. In addition there is I am sorry let me go up, we can see two special symbols which are these backslash ns right let me just clear it out, here you can see this backslash n something of the sort. This means go to a new line or end of line now suppose.

(Refer Slide Time: 04:09)

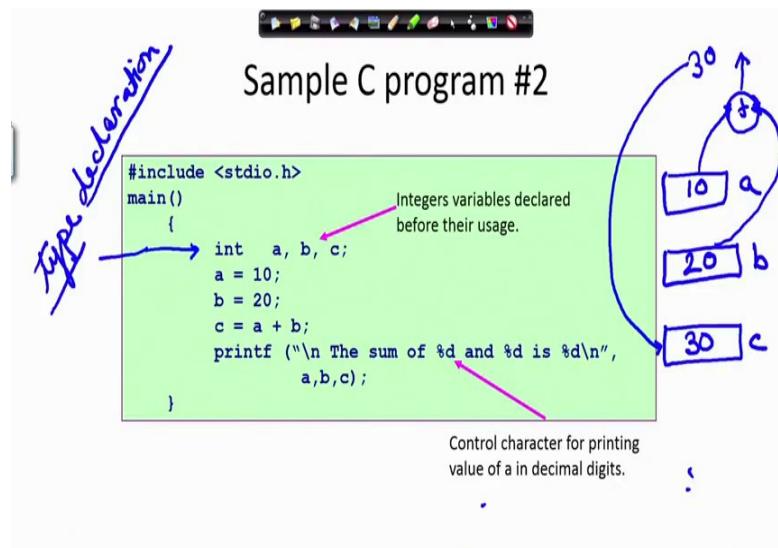


Let us see suppose I am I had something like this I had printed something all right I had printed something this is a screen, I had printed a brown fox. Now after I printed; that means, a computer printer you can see where my pen is my pen is lying here right my

pen is here. Now if I say backslash n this is no radar backslash, backslash n; that means, my pen will come to the beginning of the new line and here I will write our first C program, and by default my pen is here, but since I have given another backslash n the pen will come here.

So, next time if I again start with the backslash n it will come to the new line, and if I do not give a backslash n it will continue from here. You will understand this more when we look at more number of programs, it will be much more clear to you. So, this the structure of a very simple C program we will see more of this.

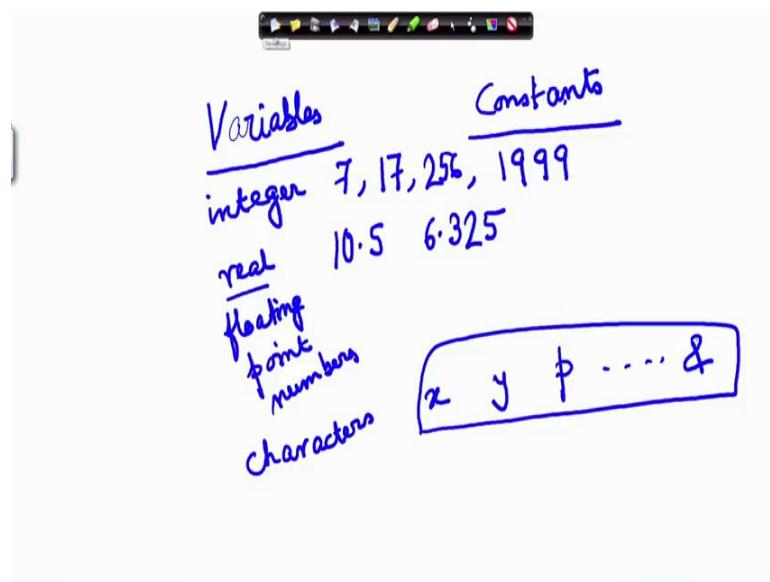
(Refer Slide Time: 05:39)



Now, here is a another program. Again now it is easier to understand you have got an includes t d i o dot h which has to be there for any C program, there will be a main function as is being shown here, and there are parenthesis between these two the program should be written, there is a boundary of the main function. Now here there are some more new things which are being shown to you in the form of example.

So, you see the first line is int a, b, c what is meant by that is that I am I will be using in this program 3 variables a b and c and each of them is of type integer. Now you are already acquainted with the term variables, but you are probably not acquainted with the term type of a variable.

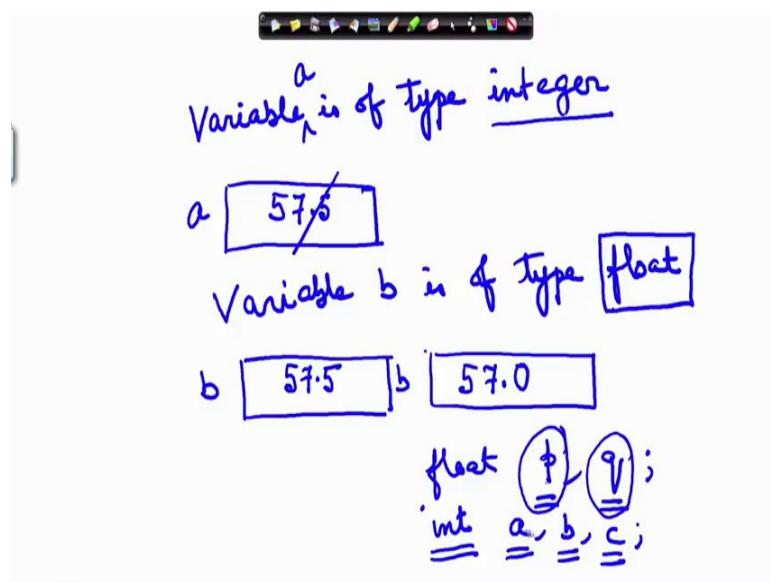
(Refer Slide Time: 06:50)



So, quickly let us go to that, we have got variables and the variables can be of different type, also constants can be of different type for example, the variables can be of type integer for example, 7, 17 all these things 256, 1999 all these are integers. Another type of variable can be real numbers which are say 10.5, 6.325 fractional numbers right these are real.

This is also known as in C as floating point numbers. Now integers are these reals of flow floating point numbers of these. Similarly I can have characters like say x y p whatever these are different characters or maybe and is a character. So, each of any of these alphanumeric and all those can be characters now. So, we know what is an integer, what is a real what is a floating point, what is a floating point what is a character.

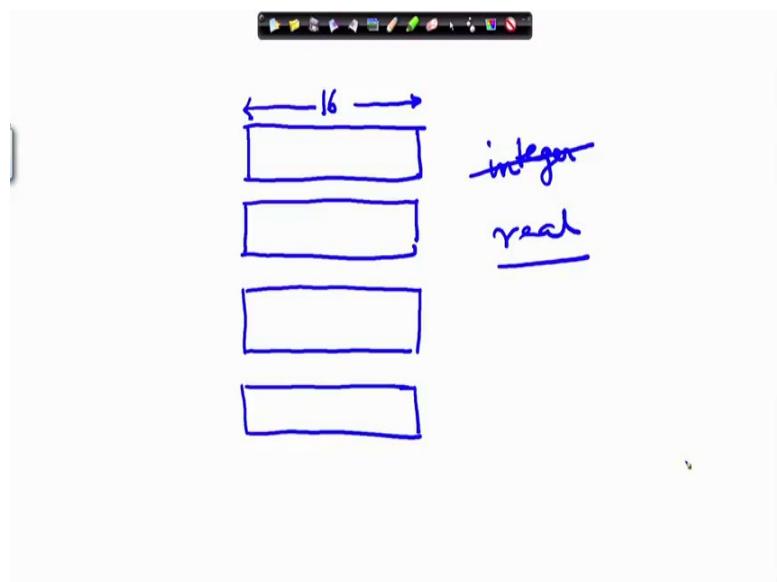
(Refer Slide Time: 08:51)



Now, a variable is of type integer, what does it mean? As you know a variable is nothing, but a memory location right it means and suppose this variable I say variable a is of type integer. So, there is a memory location corresponding to a and this memory location can only hold values which are integers. So, 57 I can store here, but if I try to store 57.5 here, it will not store 57.5, because it has been told to be an integer.

Similarly if I say variable b is of type floating point, I am in short I am writing float; that means, this variable b can store a floating point number. So, 57.5 can be stored here. Even if I try to store in b 57 in b if I store 57 just 57 then that will be stored as 57.0 all. Similarly there can be types of characters. Now depending on the type of the variable the compiler assigns different sizes of memory for the different variables.

(Refer Slide Time: 10:50)



For example in standard c compilers and it is a convention that for an integer two memory locations are allocated.

Now, how big this will be say sometimes it is my each of these memory locations can be 16 bits, in that case I am using 2 16 bits; that means, 32 bits to store an integer whereas, for a real number. Now this is a convention, 4 locations are used for storing a real number. So, 64 bits 16, 16, 16, 16. So, that will be for real numbers now. So, depending on what I write, what how I define the type of the variable when I say a b c I also say what type of variability; is it an integer, is it a real or what it is.

Now, if you come here we will see that here I have written float and I did not write floating point number. Now in c in order to specify variable to be real we declare that to be float say p q; that means, semi colon; that means, p and q are two variables which are of type floating point number right and if I write int a b c; that means, a b and c are 3 variables, which all of type floating integer type int. So, we do not write integer we just write in c int. Now obviously, then for p how many bytes how many depending on how many bits will be given how many memory locations will be given for p may be 4, q it will be 4, but for a b c it will be only 2 all right.

Now, let us therefore, go back to the program here, here we find int a b c now this statement this is called a type declaration all right this is known as a type declaration this is the first statement. So, I have declared the variables now in C program before the

variable is used it should be declared about its type. The other thing is you can see this. So, here you can see a has been assigned 10, b has been assigned 20; that means, what.

That means in the memory location a corresponding to a memory location corresponding to a 10 has been written, and to the memory location corresponding to b 20 has been written all right and this statement c assigned a plus b, I had explained in another lecture that; that means, this data this value 10, and these value 20 will be taken out on the from the left hand side from the corresponding memory locations they will be added and the result will be 30 all right; the result will be 30 and the 30 will be written into the location c. So, this will be 30 this much is clear.

Now what about this line? I know that a print if statement just tells me that I have to print whatever is there in the quote. So, how will the print look like? Please note the sum of percentage d and percentage d is percentage d this is equivalent to writing this.

(Refer Slide Time: 15:38)

```
#include <stdio.h>
main()
{
    int a, b, c;
    a = 10;
    b = 20;
    c = a + b;
    printf ("\n The sum of %d and %d is %d\n",
           -a+b+c);
}
```

The sum of 20 in 30

Control character for printing value of a in decimal digits.

integer

The sum of dash and dash is dash there are 3 gaps, one here, one here and one here now how will these gaps we filled? They will be filled by the values of a, b and c respectively each of these dashes will be filled up by the respective values of a b c. Now this percentage d is a format statement is saying that this gap can be filled by a digit or by an integer.

This gap can be filled only with an integer. Now here I a is an integer therefore, this gap will be filled with the integer value 10 and percentage d, and it is being printed as it is because it is within this double quote and dash. this dash will be filled with another digit, and what is the digit? The second space b and b is 20. So, it will be 20 is what is a plus b is c and what is the value of c c the value of c is 30.

So, this is what will be printed all right. So, this is how we print a sentence, where I want I have got some places for different variables and these places will be filled up by the values of the variables, whose names are being specified here. I repeat this gaps will be filled by the values of the variables whose names or whose identifier are been specified here and the type of this variable, and the specification of this dash should match. Now, it will be printed sum of 10 and 20 is 30; now coming to the third sample program.

(Refer Slide Time: 18:13)

```
#include <stdio.h>
/* FIND THE LARGEST OF THREE NUMBERS */

main()
{
    int a, b, c;
    scanf ("%d %d %d", &a, &b, &c);
    if ((a>b) && (a>c)) /* Composite condition check */
        printf ("\n Largest is %d", a);
    else
        if (b>c) /* Simple condition check */
            printf ("\n Largest is %d", b);
        else
            printf ("\n Largest is %d", c);
}
```

Now, this is a little more complicated, and this is trying to find out the largest of 3 numbers; that means, what we had done in a flowchart exercise finding the max of 3 numbers. Now that was discussed using flowcharts and pseudo code and here we are discussing that using a program a C program how is that idea translated into C program. Again let us start with even before that I would like to point out something, again let us revise the structure of the program, we start with header stdio dot h, now we start we put in a comment what is this? We saw in the last class it is a comment this comment is just

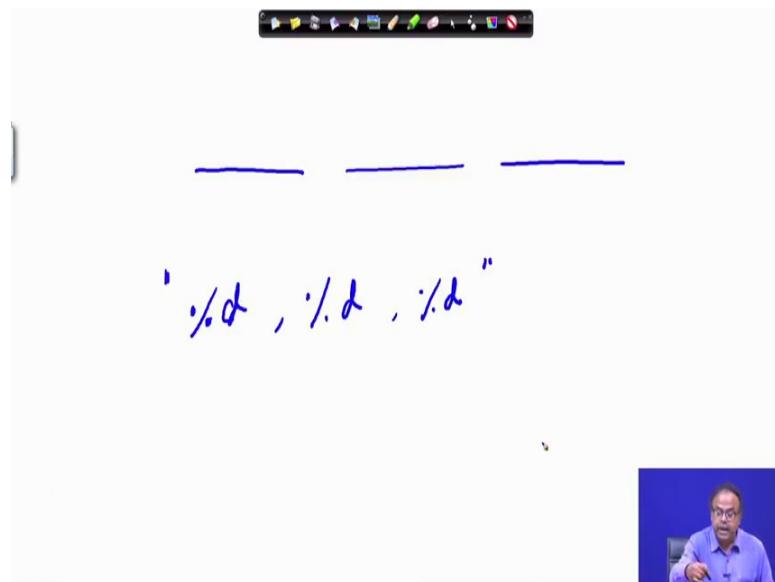
telling us the compiler has got nothing to do with, it is just telling us that this program finds the largest of 3 numbers.

Then as usual we have got a main function which must be there and there should be a parenthesis which is delineating the boundary of this program. Now for and also we have seen this, I am using 3 variables a b and c and I have put in the type of that. The next new thing that is coming up is here scan f, this is an input statement for reading 3 variables from the keyboard. Now recall that I said that for every language there are there is a vocabulary, there is the set of words that the language understands.

So in c, in an earlier one we have seen in an earlier program we have seen the statement print f right we had seen a program print f. So, print f is a particular word, now we now encounter another word int is also another word; int is another word which specific to c and the meaning of this every word will have a meaning. So, this means that whatever follows are variable names of type integer. Whatever follows this word int are variable names of type integer.

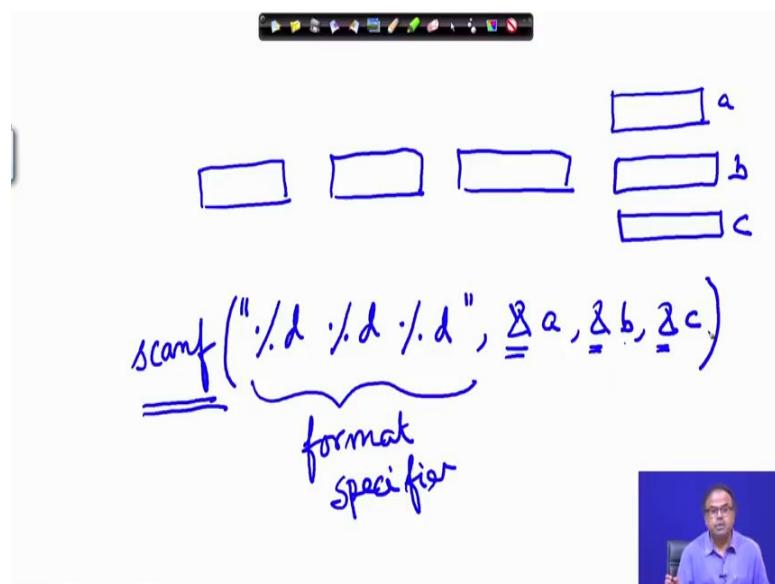
Next, new word that we are getting is scan f. Just like we had seen print f earlier all right print f is a word, that tells us that something is being printed here scan f is an int is a word that tells that whatever is inside this parenthesis is a input statement. Now let us study this a little bit here. Here again you will see that there is a percentage d percentage, d percentage 3 percentage ds within quote and that percentage d means it is a specified for an integer, some integer is being will be that scan f means say assume it is really.

(Refer Slide Time: 21:47)



So, therefore, it something like this, I am going I am creating as if 3 places, because corresponding to each percentage d, percentage d, percentage d, percentage d within the quote, I am creating 3 spaces and followed by that let us go back to this, I am sorry here there should be no comma all right; I am just giving 3 percentage ds.

(Refer Slide Time: 22:21)



So, let me go back here. So, it should be within the court here is scan f, within quote percentage d blank percentage d blank, percentage d and the quote is closed comma; that means, as if just to understand I am creating 3 spaces each of which I have ready to

accept an integer and where will those be stored? Here I am writing and a, and b, and c. Now here I will request you to just forget about this ampersand sign forget about this for the time being this will explain a little later.

Just assume that a, b and c, but before that for any read statement will have to put an ampersand before the variable names there is a reason for that. What is the meaning of this sentence? The meaning of this sentence is that there are 3 places which are ready to hold 3 variables, which will come to the locations a b c; when I am reading as if I am reading from the user 3 variables a b c ok.

Now, why I put that ampersand will explain a little later. So, 3 space have been created and what are the spaces? These spaces are nothing, but these 3 memory locations which 3 memory locations whose names are being specified here all right. So, this is something that you have to be a little careful and practice a little bit it will be very easy later on. So, within the quote I specify the format specify that is integer in this case percentage d and this other variable names.

Now, let us go back to that anything that is new here. Now a new word we are encountering here, if is a conditional statement you did not bother about it, if you recall we had in the flowchart we had a thing called diamond right where we are taking some decisions based on some conditions yes and no right. We are doing that here we are looking at some condition here if a is greater than b and a is also greater than c, if that is yes then I am printing that a is the largest number. Try to apply simple logic here, if is a conditional word condition is a conditional word and this entire statement starting from if to this semi colon is one statement, and where I check the condition whether a is greater than b and a is greater than c.

If that is so, that is if the diamond comes out with that yes, then I am printing that the largest number is a. Otherwise, that means if the answer to this is no otherwise I am again checking again if again I am checking another condition, I am checking another condition here what to do I what my checking? Is b greater than c yes if yes then I am printing the largest number is b otherwise is yes, otherwise if it is no then I am printing the largest number is c.

So, we are encountered in some new words if and else. If means if in the diameter result of this diamond box is yes then this statement will be executed otherwise; that means, if

the answer to this diamond box is no, then this part will be executed. Now here again the second block I am checking if b is greater than c, again if this is true for this condition this will be executed when this condition is true or yes if it is false then this statement will be executed. So, it is a little more complicated program, but a very useful example.

(Refer Slide Time: 27:30)

So, we see the comments we have already mentioned, now you are coming to another program.

(Refer Slide Time: 27:32)

```
#include <stdio.h>
#define PI 3.1415926
/* Compute the area of a circle */
main()
{
    float radius, area;
    float myfunc (float radius);

    scanf ("%f", &radius);
    area = myfunc (radius);
    printf ("\n Area is %f \n", area);
}

float myfunc (float r)
{
    float a;
    a = PI * r * r;
    return (a); /* return result */
}
```

This is using what we earlier had said that a big machine can we divide into sub machines. So, big program this one is not a big program, but any programs can you broken down into different functions. Here is the main function this one is main let me draw it a little nicely. Here we have got a main function and inside there is another function whose name is my function I am sorry.

Now, let us look at this, now first here there is another new thing we are introducing and that you should understand that is define PI to be 3.1415926; that means, this verified in this program this PI will appear, there will replace it with this value 3.1415926, but inside the program I will not right PI, I am just defining it once for all and this means I am replacing pi by this value before the compilation is done. So, this is again a p processor statement.

We will continue with this example in the next lecture.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 09**  
**Introducing Functions**

So, we were discussing a little more complicated a little more different program that is using a function.

(Refer Slide Time: 00:19)

The slide has a title 'Sample C program #4' at the top. It contains two code snippets and some explanatory text:

- Preprocessor statement.**  
Replace PI by 3.1415926  
before compilation.  
Handwritten note: *#define PI 3.1415926*
- Example of a function**  
Called as per need from  
Main programme.  
Handwritten note: *# define resistance 10.2*

```
#include <stdio.h>
#define PI 3.1415926

/* Compute the area of a circle */
main()
{
    float radius, area;
    float myfunc (float radius);

    scanf ("%f", &radius);
    area = myfunc (radius);
    printf ("\n Area is %f \n", area);
}
```

```
float myfunc (float r)
{
    float a;
    a = PI * r * r;
    return (a); /* return result */
}
```

A small video player window in the bottom right corner shows a professor speaking.

Here, what we had discussed in the last lecture is, that we are using a new type of statement that is hash define this is a preprocessor statement; this is called a preprocessor statement which is replacing PI with the value that is specified.

Now, anywhere I can do I can define other things like this also for example, I can write see hash define resistance is 10 ohm; 10.2 ohm whatever. That means, in my program wherever I get this variable name resistance, that will be replaced by this constant value 10.2 even before the program is compiled; that is the first new thing that we saw.

(Refer Slide Time: 01:35)

```
Sample C program #4

Preprocessor statement.  
Replace PI by 3.1415926  
before compilation.

Example of a function  
Called as per need from  
Main programme.

#include <stdio.h>
#define PI 3.1415926

/* Compute the area of a circle */
main()
{
    float radius, area;
    float myfunc (float radius);

    scanf ("%f", &radius);
    area = myfunc (radius);
    printf ("\n Area is %f \n", area);
}

float myfunc (float r)
{
    float a;
    a = PI * r * r;
    return (a); /* return result */
}

Function called.

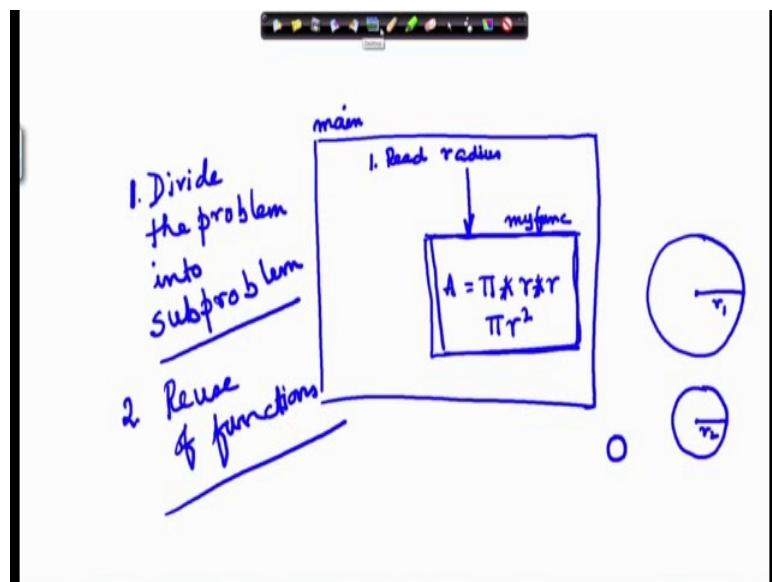
15.25 radius
```

The second thing is look at this, let us study this program. First time putting an a comment and the comment tells me what this program is going to do. The comment is compute the area of a circle, and as usual I start with a main sorry as usual I start with a main an inside main let us see what we have done, we have done something different.

Now all of you will be able to say what is this. This is a type declaration for two variables radius and area. At what sort of type declaration is that it is float; that means, radius and area are two variables which will hold real numbers or floating point numbers right. Now I also declare myfunc this is another function, and the style in which I wrote tells a compiler that this is a function and this function is also of type float.

What is the type of a function when we will be studying function in detail, we will be understanding there. Now what have we done here? Scan f you know that by now scan f is reading the value of some variable. How many here I look at this point? I find that it is 1 percentage f. So, only one variable is being read and what is that variable that is that is radius alright. So, I will read the value of the radius and radius is of type float right. So, it will be say 15.25 that is being read now computing the area the main function is not doing itself. So, let us quickly go back to our old diagram.

(Refer Slide Time: 04:09)



So, this is the main machine of the main function. So, here what have you done? We have read I am writing in a pseudo code I have read the value of radius and there is a function called myfunc; my function. Now after I read the radius I do not want to take the botheration of computing the area, this machine submachine knows how to compute the area it knows that area will be computed as pi times r times r; that means, pi r square right area of a circle.

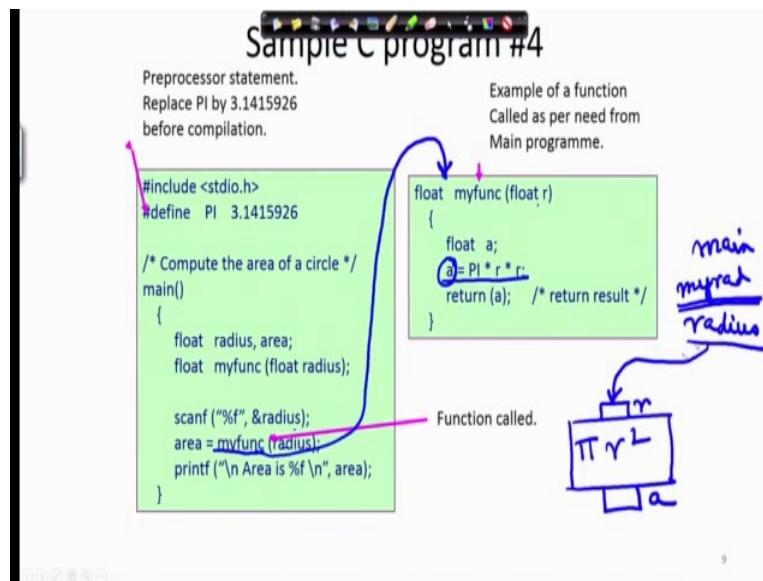
But this is a function that can compute area of any circle it has been design, because by pi r square you are not finding the area of only this circle, you can find the area of the circle or might be this circle only thing that is differing is the radius right; here it is one radius, here there is another radius, here is another radius. So, this machine or this function can compute any (Refer Time: 05:51) any area of any circle provided it gets the radius told to him told to this.

So, this machine is expecting some value to be passed on to this. Please note the terms I am using. This function once some value required some value to be passed on to this. Now you can ask that well be such a simple program, I could have written it here itself yes certainly you could have written it here itself, but this is just an example. Actually in a very complicated scenario, there are many complicated tasks and if we want to solve the entire task by yourself by through single program, there is a chance of error that is problem number one.

So, we want to divide the problem into sub problems; that is a general programming philosophy that we want to divide the problem into sub problems and solve each of the sub problems separately because they are more manageable, I can find out what are the errors whether they are working properly or not and then I combine them together and solve the overall problem that is advantage number 1.

Advantage number 2 is that suppose let us just for the sake of argument, assume that my function this finding the area is really a complicated program very complicated program. So, somebody has really taken the effort to develop this complicated program. Now whenever me or you or anybody any programmer wants to use this facility, then they do not need to reinvent the wheel and write that complicated program once again it is available and I can reuse it, and only thing that I need to do while using it is just to pass the parameter. Therefore, the second big advantage is reuse of functions that is why the concept of function is so important in programming alright. You will encounter this irrespective of programming languages. So, let us go back to this.

(Refer Slide Time: 08:41)



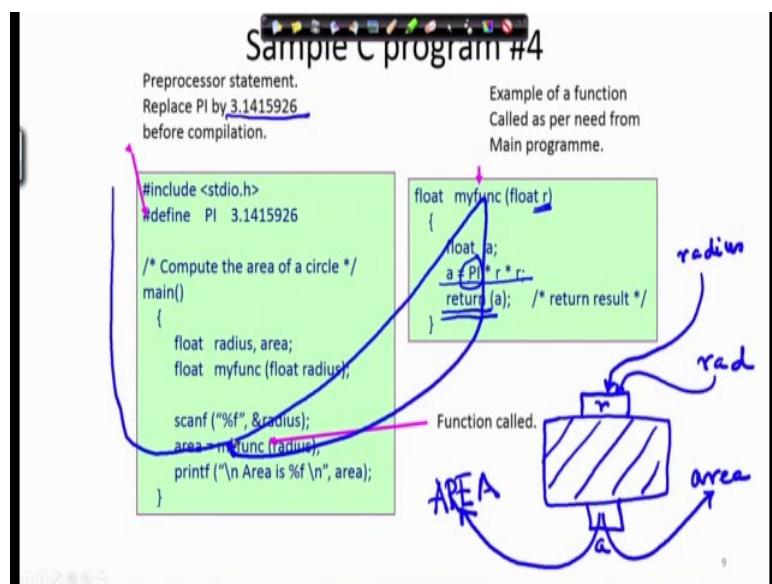
So, here you see I have I am not computing the area, I means the main is not computing the area what the main is doing? It is just here it is calling my function.

So, immediately this one is being called, and what will my function do? My function is ready my function is here and it is just expecting something which is  $r$  to come in because it will compute  $\pi r^2$ . This expecting  $r$  and this one has got the variable

that radius in the variable radius. So, this is being passed on to this. So, over here from the main function here is the main, from the main this radius is being connected to this r alright and this function my function is reading is getting the value of r, and inside that it is computing the area.

Now, there are some details which I will mention later, it is computing the area and it is finding the value in its own variable a. A is a variable now for any program, if this function is used then instead of radius you may have my radius or any other value any other sorry any other variable name can be there. Now any this will work for all of them let me clarify it once again.

(Refer Slide Time: 10:30)



See here there is a function and this this function accepts r and produces a.

Now from, as I said that it can be reused from my program, I am using it with the variable radius it is been connected here. Maybe from your program you have not used the variable name radius, you have used the variable name say rad so that will that when your program is using this function, this rad will be connected to r. Now suppose here in my program I have used the variable name area for noting the area. So, this output a will connect to area for example, for your program you said area.

Now, recall that in c small and capital makes difference to this is another variable name is a different variable. So, for you this a will connect over here, now this person this my

function is only concerned with r and a and the job of connecting them is up to the function that is using it. So, here what happened I called the main function my function computed a how here I wrote PI r r know how do I know PI; because pi has already been defined here. So, this pi will be replaced by this value. So, 3.1415926 times r times r and what is this r? Since it has been call when it has been called with radius the value of radius will come here and return a. Now this return term means where will it return, this is another new word that you are coming across. This return means it is returning to the point from where you are called. The programmers going on like this from here it called my function.

So, after my function is executed, it will return back to the same point.

(Refer Slide Time: 12:59)

**Sample C program #4**

Preprocessor statement.  
Replace PI by 3.1415926 before compilation.

```
#include <stdio.h>
#define PI 3.1415926

/* Compute the area of a circle */
main()
{
    float radius, area;
    float myfunc (float radius);

    scanf ("%f", &radius);
    area = myfunc (radius);
    printf ("\n Area is %f\n", area);
}
```

Example of a function  
Called as per need from Main programme.

```
float myfunc (float r)
{
    float a;
    a = PI * r * r;
    return (a); /* return result */
}
```

Function called.

*Area is 255.72*

So, we have come across a new word return and return is well returning the value a, and that is being assigned to my variable area all right and then print f you can now understand it will what will be printed? Area is dash and this one will take some floating point number maybe 255.72 whatever it is and this value of area will be filling up this place.

So, this is another example of a C program. So, through these examples, what we tried to do is to introduce you to some of the very common names, common words and common features of the C programming language.

(Refer Slide Time: 14:05)

main() is also a function

```
#include <stdio.h>
main()
{
    int a, b, c;
    {
        a = 10;
        b = 20;
        c = a + b;
        printf ("\n The sum of %d and %d is %d\n",
               a,b,c);
    }
}
```

Next we move to this that main is also a function I have already said that that main is also a function this is clear to you now, but just quickly have a look at this. Here I have identified some variables, have identified some variables a b c which are integers and the same old program and is also a function fine.

Now, here are some words of advice, when we are writing a program it has got a couple of things we have to keep in mind, first it must be correct so that it can be executed by a compiler. It should not it should adhere to the rules of the game the syntax in the grammar of the C language. The other thing is also this program has to be understood by others, because if they you have written program and someone else wants to extend this program. You are working in a big company and there are number of programmers and each of them are writing a small segment of the program and everybody must be able to understand everybody's program.

Therefore the programs should be written in such a way so that it is understandable more better understandable. Just like if we write in very bad hand writing write something here, something there, something there I not understand it, but if you write it nicely in a sequence it is much with proper paragraphs etcetera it is much better readable and better understandable. So, here we will see some of the requirements for desirable programming style one is of course, clarity.

(Refer Slide Time: 15:55)

The slide has a title 'Desirable Programming Style' and a bulleted list:

- Clarity
  - The program should be clearly written.
  - It should be easy to follow the program logic.
- Meaningful variable names
  - Make variable/constant names meaningful to enhance program clarity.

Handwritten notes on the right side of the slide:

- $\text{sum} = a + b$
- $\underline{\text{sum}} = \underline{\text{num1}} + \underline{\text{num2}}$
- $\text{pqr} = \boxed{(x+y+z)/3}$
- $\cancel{\text{avg}} = t * t - x$
- $\checkmark \text{avg} = (x+y+z)/3 ;$
- $\checkmark \text{average} = - -$
- $\checkmark \text{AVG} = - -$

A small video thumbnail of a person speaking is visible in the bottom right corner.

It must be very clear; the program should be clearly written it should be easy to follow the program logic. Now here is something that is very important I insist on that there should be meaningful variable names. For example, I want to add two numbers and when I say sum assigned a plus b or sum assigned num 1 plus num 2 it is good semi colon please do not forget the semi colon and listen.

Now, the sum is a meaningful variable name, it immediately tells me what this variable is meant for it is holding the sum. But suppose if I had written something like tt is equal to ttt multiplied by t minus x, then from there its not very clear what is this tt why are you using this tt. If say for example, average I am computing average as avg and that is x plus y plus z divided by 3 is very understanding semi colon it is understandable, I could have written the full thing average equals to this or I could have written avg all these things are fine.

But if I had written pqr is equal to x plus y plus z divided by 3, then its not very clear although I can look at this side and understand that therefore, it is pqr is the average, but this not a very good practice. Now so, make variable and constant names meaningful for example, pi should be written as pi is not good that 3, 4 3.1415 all those that value I (Refer Time: 18:21) chi because that is not very clear what this value standing for.

(Refer Slide Time: 18:28)

## Desirable Programming Style

- Clarity
  - The program should be clearly written.
  - It should be easy to follow the program logic.
- Meaningful variable names
  - Make variable/constant names meaningful to enhance program clarity.
    - 'area' instead of 'a'
    - 'radius' instead of 'r'
- Program documentation
  - Insert comments in the program to make it easy to understand.
  - Never use too many comments.



So, here are some examples use area instead of a, radius instead of r and program documentation. We had said that we should be very generous about writing comments, but if that too many comments then that is also not very desirable.

(Refer Slide Time: 18:47)

## Contd.

- Program indentation
  - Use proper indentation.
  - Structure of the program should be immediately visible.



Now you are coming to a very important point that is program indentation what is that? The structure of the program should be immediately feasible. We will give you some examples.

(Refer Slide Time: 18:58)

## Indentation Example #1 :: Good Style

```
#include <stdio.h>
#define PI 3.1415926
/* Compute the area of a circle */

main()
{
    float radius, area;
    float myfunc (float radius);

    scanf ("%f", &radius);
    area = myfunc (radius);
    printf ("\n Area is %f \n", area);
}

float myfunc (float r)
{
    float a;
    a = PI * r * r;
    return (a); /* return result */
}
```

Here is a good example, here this is simple I am writing in straight line, here there is a function that is fine let us come to another example which will be better illustrating this this is a bad style you see here compare these 2.

(Refer Slide Time: 19:13)

## Indentation Example #1 :: Bad Style

```
#include <stdio.h>
#define PI 3.1415926
/* Compute the area of a circle */
main()
{
    float radius, area;
    float myfunc (float radius);
    scanf ("%f", &radius);
    area = myfunc (radius);
    printf ("\n Area is %f \n", area);
}

float myfunc (float r)
{
    float a;
    a = PI * r * r;
    return (a); /* return result */
}
```

Here there is a declaration then I leave some blank, and then I put the actual code here these are the declaration. So, I understand this a declaration. Compared to this which is easy to understand this is something this is something else. Compared to that here why I where I do not put a gap it is difficult to understand, where the declaration ends and

where the code starts. So, here is another good example of finding the largest of three numbers look at this.

(Refer Slide Time: 19:50)

```
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
{
    int a, b, c;
    scanf ("%d %d %d", &a, &b, &c);
    if ((a>b) && (a>c))           /* Composite condition check */
        printf ("\n Largest is %d", a);
    else
        if (b>c)                  /* Simple condition check */
            printf ("\n Largest is %d", b);
        else
            printf ("\n Largest is %d", c);
}
```

As I said that if I have read this, but one bad thing is here, there should be a gap here there should have been a gap here. Now here if a and b. So, that is the first diamond a decision box if a is greater than b and a is greater than c then I am doing something alright then I am doing this this part of the program, otherwise I am doing this. So, here you can quickly looking at this indentation, that this print f is a under this and this part is under this, that is very clear from this indentation I have shifted it a little bit.

And so immediately you can understand that this print f is if this condition is true. That means, this part is for this diamond box to be true like that; is a good indentation good style now the same thing.

(Refer Slide Time: 21:06)

### Indentation Example #2 :: Good Style

```
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
{
    int a, b, c;
    scanf ("%d %d %d", &a, &b, &c);
    if ((a>b) && (a>c)) /* Composite condition check */
        printf ("\n Largest is %d", a);
    else
        if (b>c) /* Simple condition check */
            printf ("\n Largest is %d", b);
        else
            printf ("\n Largest is %d", c);
}
```

Same thing would be bad style if I write it in this way.

(Refer Slide Time: 21:19)

### Indentation Example #2 :: Bad Style

```
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
{
    int a, b, c;
    scanf ("%d %d %d", &a, &b, &c);
    if ((a>b) && (a>c)) /* Composite condition check */
        printf ("\n Largest is %d", a);
    else
        if (b>c) /* Simple condition check */
            printf ("\n Largest is %d", b);
        else
            printf ("\n Largest is %d", c);
}
```

16

From here it is not at all clear which one is a corresponding to which one, not very clear if I study it closely I will be able to understand sorry, I will be able to understand that this print f this print f corresponds to this condition.

But immediately when I look at it is not very visible. So, this is a bad style alright. So, indentation is something that is very much prescribed in good programming, this becomes very important when we actually go for more complicated programs.

(Refer Slide Time: 21:58)

The C Character Set

- The C language alphabet:
  - Uppercase letters 'A' to 'Z'
  - Lowercase letters 'a' to 'z'
  - Digits '0' to '9'
  - Certain special characters:

# include  
%.d  
%.f

!	#	%	^	&	*	(	)
-	_	=	~	[	]	\	
	;	:	'	"	{	}	,
.	<	>	/	?	blank		

A small video window in the bottom right corner shows a teacher speaking.

Now coming to the C character set. In an earlier lecture I had briefly talked about this just as in English we have got a character set a to z capital A to Z. Similarly C language has got a character set with which we make the words we have already seen them.

So, the first thing is of course, the upper case letters, the lower case letters a to z digit 0 to 9 and some special characters like you can see this hash we have already encountered this when we said hash include that is a part of the c symbol set percentage we have seen that when we say percentage d, percentage f you are seen that and this sort of symbols this have already encounter all these are some certain special characters, that are allowed in C.

(Refer Slide Time: 23:09)

The slide has a title 'Identifiers and Keywords' and a list of bullet points defining identifiers:

- Identifiers
  - Names given to various program elements (variables, constants, functions, etc.)
  - May consist of *letters*, *digits* and the *underscore* ('\_') character, with no space between.
  - First character must be a letter.
  - An identifier can be arbitrary long.
    - Some C compilers recognize only the first few characters of the name (16 or 31).
  - Case sensitive
    - 'area', 'AREA' and 'Area' are all different.

A small video thumbnail in the bottom right corner shows a person speaking.

Now the identifiers and keywords we have also talked about that. The names are given to very different programming elements for example, variables we already know, constants we already know, functions we have already seen each of them are given some names.

(Refer Slide Time: 23:32)

The slide displays three handwritten examples of identifiers:

- myfunc: A function identifier.
- avg: A variable identifier.
- pi: A constant identifier.

A small video thumbnail in the bottom right corner shows a person speaking.

For example when we wrote `myfunc` that was the name given to a particular function. When I wrote `avg` that was the name given to a particular variable that was the name given to a particular variable. When I wrote `pi` that was the name given to a particular constant right. So, in that we have encountered this. So, now, the how there is some

restriction on the names that I have also mention, may consist of letters digits 0 to 9 and this underscore character with no space in between. Whenever I want to put space, I will put in this underscore character is very useful is very useful to put this underscore character hear what is happening here.

The first character must be a letter, this character must be a letter an identifier can be orbital is long, but that depends on some c compilers some c compilers recognize only the first few characters 16 or 31. And another very important thing that you had talked about that it is case sensitive. A small area written in small letters and area written in capital letters or an area with a mix of small and capital are all different these are not the same this we had mentioned in the last class.

(Refer Slide Time: 25:14)

Contd.

- **Keywords**
  - Reserved words that have standard, predefined meanings in C.
  - Cannot be used as identifiers.
  - OK within comments.
  - Standard C keywords:

auto	break	case	char	<u>const</u>	continue	default	do
double	<u>else</u>	enum	extern	<u>float</u>	for	goto	<u>if</u>
<u>int</u>	long	register	<u>return</u>	short	signed	sizeof	<u>static</u>
struct	switch	typedef	union	unsigned	void	volatile	while

A video player interface is visible at the bottom right of the slide.

There is some keywords some words which are reserved for and have got some predefined meanings in C for example, auto, brake, constant, float you know float has got a specific meaning int got a specific meaning return your seniors got a specific meaning etcetera. If it has got a specific meaning, I c f seen it has got a specific meaning, but within comments you are free because it never being so, close the looked at by the compiler the compiler simply text it and print it out.

So, this is another thing that you have to remember, whatever keywords you face during programming you will gradually remember that these are the keywords and that should not be used as a variable name.

(Refer Slide Time: 26:34)

The slide has a title 'Valid and Invalid Identifiers'. It contains two lists: 'Valid identifiers' and 'Invalid identifiers'. Handwritten annotations in blue ink are present on the right side of the slide, pointing to specific examples in both lists.

- Valid identifiers
  - X
  - abc
  - simple\_interest
  - a123
  - LIST
  - stud\_name
  - Empl\_1
  - Empl\_2
  - avg\_empl\_salary
- Invalid identifiers
  - 10abc
  - my-name
  - "hello"
  - simpleinterest
  - (area)
  - private
  - simple\_interest

Annotations include:

- A blue bracket groups '10abc' and 'my-name'.
- 'my-name' is underlined.
- '"hello"' is underlined.
- 'simpleinterest' is underlined.
- '(area)' is underlined.
- 'private' is underlined.
- 'simple\_interest' is underlined.

So, here we will conclude this lecture with some examples, you can see some valid identifier x is a valid identifier, a b c of now simple interest I put in an underscore here, a 123 is fine because it starting with a, and alphabet list stud name (Refer Time: 26:55) student name very clear understood employee 1, employee 2 average employee salary see I wanted to write a big thing. And I could take to that buy instead of blank I just put underscore alright invalid identifier studies and why because I cannot start with sorry.

Because, I cannot start with blank sorry I cannot start with the numeral, for this is wrong this is wrong because there is a special character that is been put in here, I am in hide could have written my underscore name. My thing is also is also this not looking nice I could have done this my name, but (Refer Time: 27:41) is not allowed hello is not allowed because here I put in some special characters, simple interest is not allowed because hes a blank I cannot put in a blank, but I could have rewritten simple underscore interest that is quite for light. Area is not valid because it has got a parenthesis, and percentage is using special characters here. So, these are invalid identifier examples of invalid identifier. So, you should keep that in mind. We will continuity with R programming lectures.

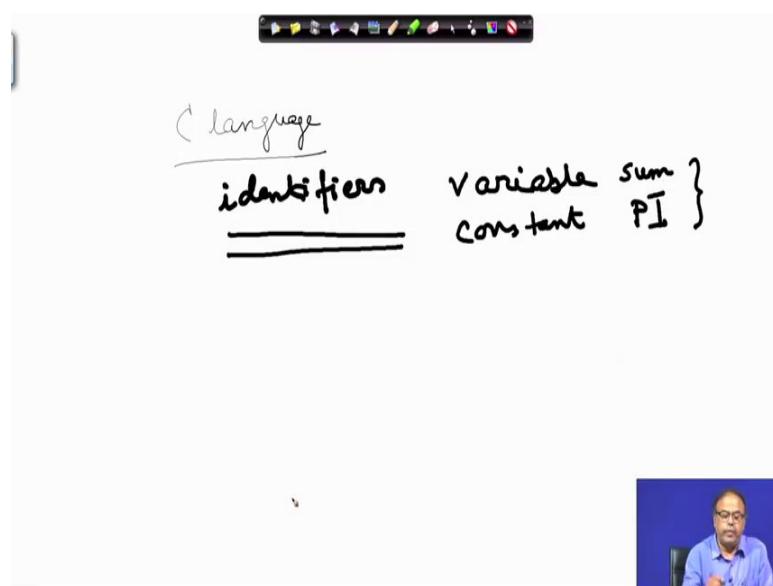
So, till now what we have done is we have looked at some of the rules for writing the program and how the variables and the constant should be named; we will continue with this.

**Problem Solving Through Programming In C**  
**Prof. Anupam Basu**  
**Department Of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 10**  
**Address and Content of Variables and Types**

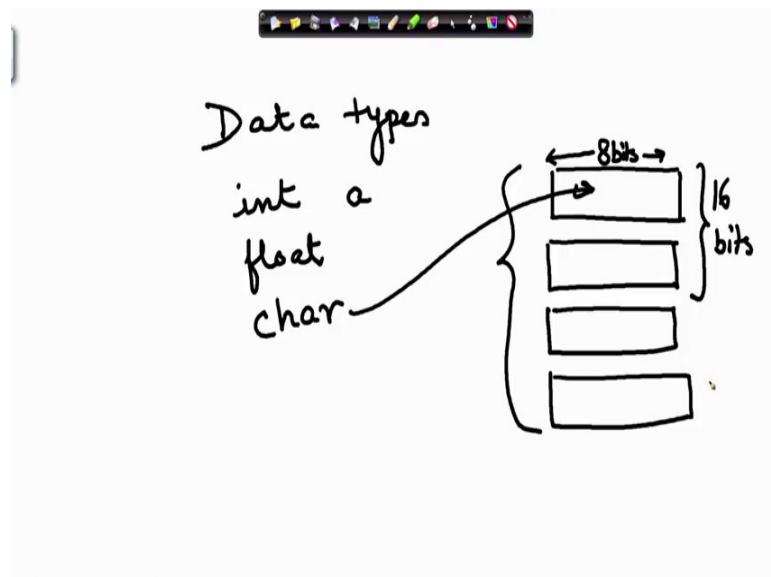
So, till now we have looked at how in C language.

(Refer Slide Time: 00:29)



We can write the identifier, this is becoming a little too thick, which identifiers are used for used for writing the variable names and the constant name. So, for example, PI or some variable name suggest sum all those things. Now we have seen what are the rules that the language c imposes on writing the names of variables or constants, that is how we can write the identifier alright.

(Refer Slide Time: 01:30)

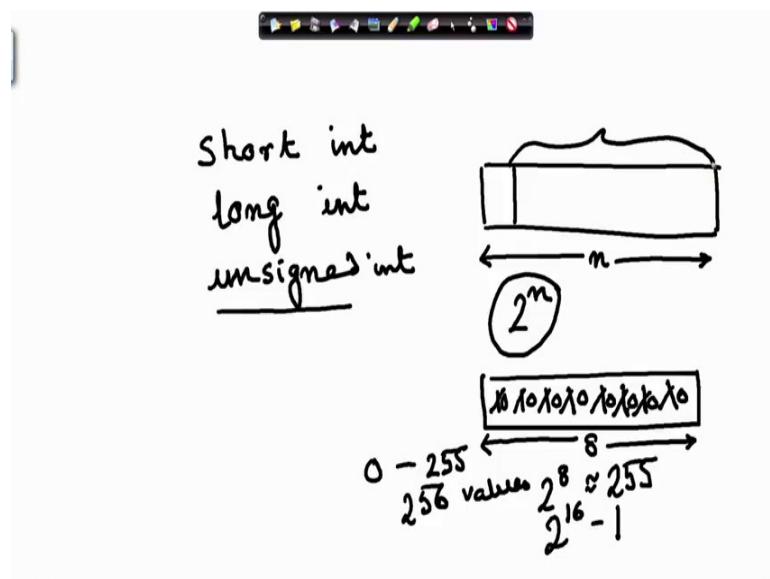


Also we have seen that there are different data types like int with stands for integer, and float which represents floating point numbers or real numbers, and char is used to specify some data which is of type character. Will see some more examples of this in the course of these lectures, now we also know that int means integer and whenever a particular variable a is declared to be an integer, then typically it also varies from machine to machine and compiler to compiler, 2 bits or 16 bits are located for storing one integer alright.

Each of these boxes are that I am showing here are 8 bits wide. So, there are 2 such. So, 16 bits, for float we will have 2 more real numbers are stored using 32 bits whereas, characters are typical is stored in 8 bits. Now this is not so sacrosanct as in some machine which is much more powerful and much more accurate having high resolution we can have 32 bits for storing integers 64 bits for storing floating point numbers and characters can be a bit 16 bits.

However depending on what is the type declaration, the amount of storage the amount of memory that is allocated to a particular variable varies ok.

(Refer Slide Time: 03:37)



We just like int float char we also have got some more like short int alright, long int or unsigned int. These are also different data types, will come across his in the course of this lecture. Short int means just if in an integer takes 2 bits or short int will take one bit if a long int if a int take 16 by 16 bits; that means, 2 bytes or long int can be made to consume 4 such bytes or 32 bits, but still that will be an integer. So, in this 4 bytes an integer that that will be stored; that means, a integer accuracy will be much larger.

So, depending on the number of bits I allocate to a particular variable, depending on the number of bits suppose I allocate n bits the range of values that I can represent varies for example, if there be n bits then I can go from I can have  $2^n$  distinct values stored for example, if there be 8 bits, then the maximum value that I can store is when all these 8 bits are ones and that is your knowledge of binary arithmetic will tell you that this will be  $2^8$  right that is 256 it will be actually 255 alright. So, 255 and if I make everything 0 if each of them are made 0 all zeros will be 0 right, I can have the range from 0 to 255; that means, total 0 to 255; that means, I can store any of 256 values 2 out of 256 values distinct values I can store any one of them. Now if this 8 would have become 16, then my maximum range would be  $2^{16}$  minus one right that is the maximum value that I can store it.

Now signed and unsigned mean sometimes in our presentation we keep one bit for the sign part in that case of course, the range decreases, but if I go for unsigned, then we remain within say 16 bits to bytes.

I can have a larger representation we will go will encounter these details as and when we need them.

(Refer Slide Time: 06:46)

Some Examples of Data Types

- int  
0, 25, -156, 12345, -99820
- char  
'a', 'A', '\*', '/', //
- float  
23.54, -0.00345, 25.0  
2.5E12, 1.234e-5

int yourvar  
char myvar

myvar = "a"  
yourvar = 10

E or e means "10 to the power of"

' ' myvar = ''

Now, let us come to some examples of the data types. You can see integers 0, 25 minus 156 all these are examples of integers. Now here I am for the first time showing some characters. Now the character values are you see the character values have got something special say I am declaring some variable char as type char my variable, I name that my var my var is of type character and I assign I want to assign to my var I want to assign that my variable will hold the character a.

Now, when I am assigning a character, then I have to put a single quote around this. For example, I had another variable int yourvar. Yourvar is another variable which is of type int. So, if I assign into that in yourvar then I can stay to say an integer value 10, but when I right onto a character a character constant has to be always in cap, I mean encapsulated with into single quotes right as here.

Now, this single quote within the single quote is a character slash, what about this this is just single quote single quote; that means, we win that there is a blank so; that means, I

am see if I say myvar is assigned this; that means, myvar will be assigned a blank character. Now you should remember that each of these characters, that we type in have got an ASCII value each of these characters have got an ASCII value.

(Refer Slide Time: 09:02)

The slide has a title 'Some Examples of Data Types'. To the left is a list of data types and their examples:

- int  
0, 25, -156, 12345, -99820
- char  
'a', 'A', '\*', '/', ''
- float  
23.54, -0.00345, 25.0  
2.5E12, 1.234e-5

To the right is a table titled 'ASCII' with two rows of characters. The first row contains lowercase letters a, b, c. The second row contains uppercase letters A, B, C, followed by a note: 'E or e means "10 to the power of"'. Below the table are digits 1, 2, and 9. In the bottom right corner of the slide area, there is a small video thumbnail showing a person speaking.

What is ASCII? ASCII stands for American standard code for information interchange. Now according to this table for every English character a b c d and capital A B C D and 1 2 two everything up to 9 or all of them have got some particular code American standard code and that is accepted in all the computers. So, whenever I type an a, when I strike the key on the keyboard when I strike the key a right whenever a strike a then actually when I as I press a what goes inside the computer is an ASCII code of a alright.

Now, this ASCII code of a will be store therefore, and the code for b the quote for capital A are all distinct. So, whenever I type in a character from the keyboard a particular ASCII code goes in whenever I assign some value to a variable for example, as I did right now myvar signed a this means that myvar will now, have done the ASCII code of a alright it will have the ASCII code of a.

(Refer Slide Time: 10:32)

## Some Examples of Data Types

- int  
0, 25, -156, 12345, -99820
- char  
'a', 'A', '\*', '/', ''
- float  
23.54, -0.00345, 25.0  
 $2.5E12$ ,  $1.234e-5$

$myVar = 'a'$

E or e means "10 to the power of"

$2.5 \times 10^{12}$

$1.234 \times 10^{-5}$



Now the third variety that is these three are very common float for example, 23.54 or minus 0.00345 25.0 or I can also write it in this way.  $2.5 E 12$  what does this mean?

This means, it is 2.5 times 10 to the power 12, what does this mean? This means 1.234 times 10 to the power minus 5, because it is E minus 5 here it is E 12 I can use capital E or small e that really does not matter these are the examples of floating point constant ok.

(Refer Slide Time: 11:44)

## Some Examples of Data Types

- int  
0, 25, -156, 12345, -99820
- char  
'a', 'A', '\*', '/', ''
- float  
23.54, -0.00345, 25.0  
 $2.5E12$ ,  $1.234e-5$

$float x, y, z;$

$x = 23.54;$

E or e means "10 to the power of"

$y = 2.5e12$

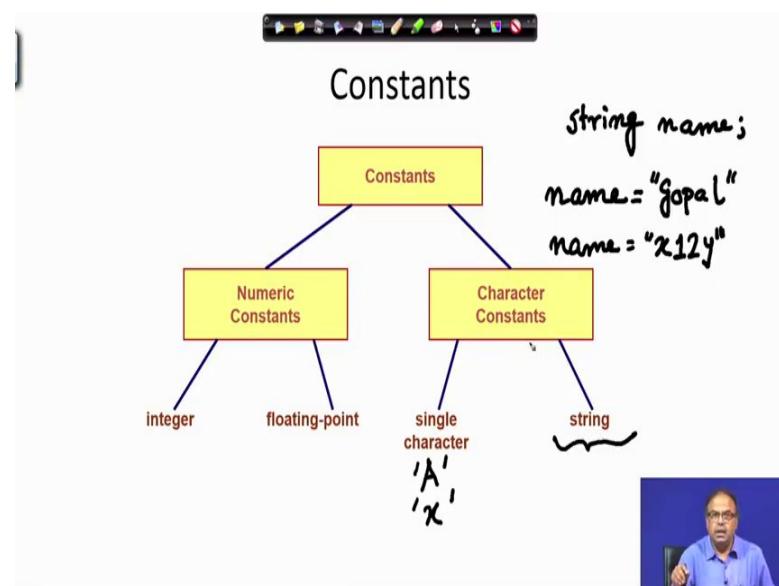
$z = 1.234E-5$



So, if I have a variable like float x and I assign I can assign x to be say x is 1, y is 1, z is another one. So, I can assign x to be 23.54 semi colon or I can assign y to be 2.5 E 12 and z to be 1.234 E minus 5 alright.

That means now z will have the value 1.234 times 10 to the power minus 5 that is how we represent the floating point numbers given this. So, this part is clear that is how we write the variables now and the these are the examples of data types.

(Refer Slide Time: 12:47)



Now, coming to constants the constants can give integer constants on floating point constants just the once that we I was showing right now. But there is another type of constant character constant so on. We have already seen example of character constants of single character like we had say a sorry what is happening here; can have a single character like A or say x all these within a single quote a single characters, and there is another type of character constant which is a string.

For example I can have a string which is another type of character constant which is not a single character, but a string of characters. For example, I want to write down my name the name of a person. So, string type variable name to name and I can assign some value to the string like say g o p a l alright. So, this is a number of characters taken together is forming a string of characters, it could also be named to be x 1 2 y double quote. Now note that in this case I am using double quote where is for single character I was using

single quote. Now these are some of the rules of defining character constants on numeric constants in C.

(Refer Slide Time: 14:55)

The slide has a title 'Integer Constants' and a bulleted list:

- Consists of a sequence of digits, with possibly a plus or a minus sign before it.
  - Embedded spaces, commas and non-digit characters are not permitted between digits.
- Maximum and minimum values (for 32-bit representations)
  - Maximum :: 2147483647
  - Minimum :: -2147483648

Handwritten notes above the list:  
32-bit representations  
 $2^{31} - 1$

Diagram below the list:  
Binary representation of integers:  
Positive side:  $2^{31}$  (with a circled '32' above it) followed by 31 zeros.  
Middle: 0  
Negative side:  $-2^{31}$  followed by 31 ones.

25

Next we move to you have seen the integer constant. Now couple of things to be just mentioned that the maximum and minimum number of values, that can be stored as an integer constant is dependent on how many bits are allocated for the presentation. For example, as I said that for 32 bit representation, I can have  $2$  to the power 32 different combination alright. So, if you compute this you will find that on one side here is 0 when everything.

If I take one bit to designate positive or negative, then I will be left with that 31 bits. So, the maximum I can have on the positive side is  $2$  to the power of 31 minus the 1 middle one is 0 alright and I can go up to this and on the other side I can go up to  $2$  to the power minus  $2$  to the power 31 right. So, this is the range, there is a maximum integers and the minimum integer that I can represent.

But; obviously, we need not be so concerned about it, because that varies with the number of bit representation in the machine. So, for a 64 bit representation; obviously, this size will be doubled; will be much larger I am sorry it will it will be much larger.

(Refer Slide Time: 16:32)

The slide has a title 'Floating-point Constants' and a bulleted list of facts about floating-point numbers. It also shows examples of scientific notation and a diagram illustrating the components of a floating-point number.

- Can contain fractional parts.
- Very large or very small numbers can be represented.  
23000000 can be represented as 2.3e7
- Two different notations:
  1. Decimal notation  
25.0, 0.0034, .84, -2.234
  2. Exponential (scientific) notation  
3.45e23, 0.123e-12, 123E2

Diagram illustrating floating-point representation:

0.123 × 10<sup>-15</sup>

1 2 3

e means "10 to the power of"

A small video player window in the bottom right corner shows a person speaking.

So, we have also seen floating point numbers just now. So, I do not need to repeat this, and where it why are we going for this this type of why are you going for exponential type of representation, because that enables us to represent much larger numbers and very small numbers also using less number of bits because I can always write 0.123 into 10 to the power minus whatever.

So, here minus 12 I could have written minus 15. So, it has got 2 num 2 parts one is the mantis apart that is this part I just put the decimal part 1 2 3 be present in binary somewhere and on this side I put some bits for the exponents. So, it can be minus 15 plus 15. So, using less number of bits I can increase the range and can go for a much larger range of a presentation.

(Refer Slide Time: 17:48)

The slide has a title 'Single Character Constants' and a list of bullet points:

- Contains a single character enclosed within a pair of single quote marks.
  - Examples :: '2', '+', 'Z'
- Some special backslash characters
  - '\n' new line
  - '\t' horizontal tab
  - '\' single quote
  - '\\' double quote
  - '\\' backslash
  - '\0' null

Handwritten notes on the slide include:

- A box around the character 'n' with a note: 'm'.
- Below the code 'printf("abc\n");' there is a handwritten 'abc' with a curved arrow pointing from the 'n' in the code to the 'c' in the output.

A small video thumbnail of a teacher is visible in the bottom right corner.

This one we have already explained that single character constants, now here of course, you can see that this operator plus also has gotten ASCII quote every character has gotten ASCII representation whatever we have we find on the keyboard has gotten ASCII representation. Therefore, I can also have capital z or plus as a character now here is something that is a little new to you we have already encounter one of these as a friend earlier, here you can see that we are using a special character like backslash.

This backslash means that whatever is following a backsplash is not the normal nature of that for example, if I write n, it really does not mean an a character n, but backslash n has got a different meaning alright. For example, suppose I was writing something printf say I write you have seen that example earlier printf suppose I am just writing a b c and then I put backslash n; that means, I am I will be painting a b c, but after that I will not print n, but since its backslash n, its a some other information it is telling us that go to the new line. So, immediately we go to the new line. Similarly we can see that backslash t this one is the horizontal tab.

(Refer Slide Time: 19:37)

The slide has a title 'Single Character Constants'. Below the title is a bulleted list:

- Contains a single character enclosed within a pair of single quote marks.
  - Examples :: '2', '+', 'Z'
- Some special backslash characters

'\n'	new line	
'\t'	horizontal tab	←
'\'	single quote	←
'\"	double quote	
'\\'	backslash	
'\0'	null	

To the right of the table, there is a drawing of a rectangular box with several characters inside: a dot, a horizontal tab, a single quote, a double quote, a backslash, and a null character. Arrows point from the text descriptions to their corresponding characters in the box.

In the bottom right corner of the slide, there is a small video frame showing a person speaking.

So, if I have backslash t my cursor will move from here to some fixed tabular distances right.

Backslash now you know single quote or double quote; single quote if I put a charac if I just want to print the characters single quote how do I do it? I will do it because any character have to do it in this quote. Now if I put single quote here then it will be confused it will take these 2 and will take a blank character in between, because a blank character is represented as n is blank with into single quotes, but I really want that here not blank, but I want to print the single quote.

(Refer Slide Time: 20:29)

The slide has a title 'Single Character Constants' and a list of bullet points. It also contains a table showing escape sequences and their meanings, with arrows pointing from the table to the corresponding characters in a sample string. A small video player window in the bottom right corner shows a person speaking.

{'\n': 'new line', '\t': 'horizontal tab', '\"': 'single quote', '\"\"': 'double quote', '\\': 'backslash', '\\0': 'null'}	<pre>'\\' '\"' '\"\"' '\\\\' '\\0'</pre>
--	--

So, in that case what should I do? For this should take the signal quote and then backslash single quote back I mean single quote; that means, this single quote is different from these 2 single quotes. So, these are the boundaries of the character representation and what is the character? That is single quote similarly for double quote you can now very easily reason that I must enclose it with in single quote, and then backslash double quote followed by single quote .

Similarly, if I want to print backslash what should I do? Single quote then backslash; that means, it is something different, backslash single quote similarly backslash null is backslash 0 alright.

(Refer Slide Time: 21:38)

The slide has a title 'String Constants' and a list of bullet points. The list includes: 'Sequence of characters enclosed in double quotes.', 'The characters may be letters, numbers, special characters and blank spaces.', 'Examples: "nice", "Good Morning", "3+6", "3", "C"', 'Differences from character constants: - 'C' and "C" are not equivalent. - 'C' has an equivalent integer value while "C" does not.' There is also a diagram showing the string "3+6" with the characters '3', '+', and '6' underlined, and the double quotes at the beginning and end also underlined.

- Sequence of characters enclosed in double quotes.
  - The characters may be letters, numbers, special characters and blank spaces.
- Examples:  
"nice", "Good Morning", "3+6", "3", "C"
- Differences from character constants:
  - 'C' and "C" are not equivalent.
  - 'C' has an equivalent integer value while "C" does not.

So, these are some special character constants that we may encounter during our programming practice. The other new things that we have learnt is string constant; now string constants are us are sequence of characters its a sequence of characters enclosed within double quotes.

Just like we wrote that the characters may be the characters within the double quote may be letters, numbers, special characters bank blank spaces like that for example, nice good morning this is a blank here, now what will happen with this. When I put this as a string do not think that it will be computed and printed as 9. It is just a string that will be printed. So, if I write in this way within double quote if I write three plus six then just 3 plus 6 that string will be printed. The difference between the with character constants is that, backslash I mean the single quote c this is a character this is a string and they are not equivalent.

Because their representations we will see we will be internally there will be represented in a different way this one has got an equivalent integer value that is ASCII code whereas, this does not have an ASCII code this is something different where they will be c and something more which will see later. So, string constants thing only thing to remember is, string constants are a sequence of characters which can be letters numbers expressions whatever this sort of operator special characters enclosed within double quotes alright that is a string character .

(Refer Slide Time: 23:40)

The slide has a title 'Variables' at the top. Below the title is a bulleted list of four points:

- It is a data name that can be used to store a data value.
- Unlike constants, a variable may take different values in memory during execution.
- Variable names follow the naming convention for identifiers.
- Examples :: temp, speed, name2, current

At the bottom right of the slide, there is a small number '29'.

Now, we already know what variables are. So, we do not need to repeat that.

(Refer Slide Time: 23:47)

The slide has a title 'Example' at the top. Below the title is a code block with annotations:

```
int a, b, c;
char x;

a = 3;
b = 50;
c = a - b;
x = 'd';

b = 20;
a = a + 1;
x = 'G';
```

Two pink boxes on the right side of the slide are labeled 'Variables' and 'Constants'. Arrows point from specific lines of code to these boxes. The lines 'a = 3;', 'b = 50;', 'c = a - b;', 'x = 'd''; and 'x = 'G'' are grouped under the 'Variables' box. The lines 'b = 20;' and 'a = a + 1;' are grouped under the 'Constants' box.

At the bottom right of the slide, there is a small number '30'.

And we have seen the variables.

(Refer Slide Time: 23:51)

The slide has a title 'Declaration of Variables' and contains the following bullet points:

- There are two purposes:
  1. It tells the compiler what the variable name is.
  2. It specifies what type of data the variable will hold.
- General syntax:  
data-type variable-list;
- Examples:  
`int velocity, distance;  
int a, b, c, d;  
float temp;  
char flag, option;`



We know that the variables are to be declared and the general syntax is a particular data type.

Sorry it will be a particular data type followed by variable list right. So, like examples we have already seen, int velocity distance int a b c d, a b c d velocity distance all integer variables temperature is a float temp is a floating point variable, flag option these are character type of variables we have already seen them right.

(Refer Slide Time: 24:29)

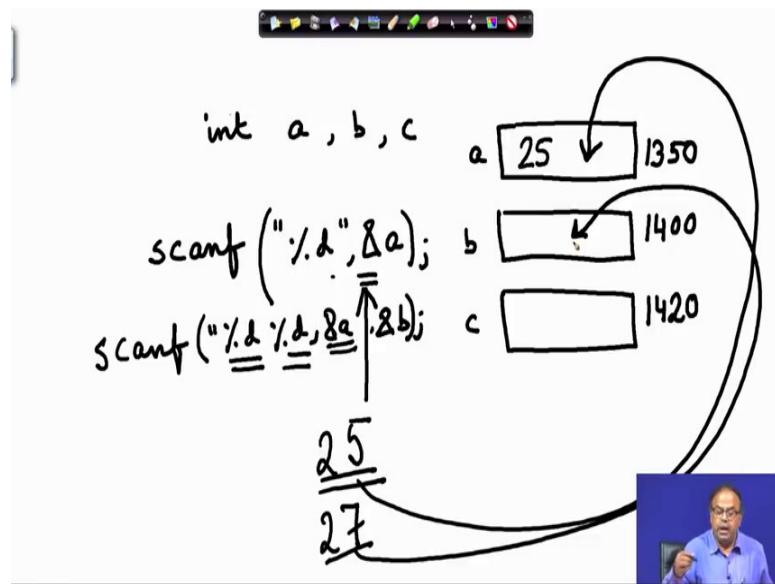
The slide has a title 'A First Look at Pointers' and contains the following bullet points:

- A variable is assigned a specific memory location.
  - For example, a variable **speed** is assigned memory location **1350**.
  - Also assume that the memory location contains the data value **100**.
  - When we use the name **speed** in an expression, it refers to the value **100** stored in the memory location.  
$$\text{distance} = \text{speed} * \text{time};$$
- Thus every variable has an **address** (in memory), and its **contents**.



Now, we come to something that we evaded till now. Pointers have got big role in C programming, but we will just have a very simple look at the pointers. Here pointer means basically address alright. So, you please forget about the title for the time being, a variable is assigned a specific memory location that we know and that memory location is assigned by the compiler. So, if we have some variable say when we find out int a b c.

(Refer Slide Time: 25:14)



Then as we have discussed earlier a b c are three memory location, which are the sign by the compiler which of these memory locations actually have got an address right. So, the address can be say this is 1350 is an address just like our houses have an address just like your rooms have got some numbers, just as your drawers may have some levels. So, similarly might be this is 1400, this is say 1450 or 1420 suppose a b c has got this 3 addresses are right.

Now, when I read when I try to read something, we know that I need to scanf. Now in scanf what I did is percentage d and a; that means, I am trying to read the variable a, but I did not explain to you earlier why I put this. And this and means that and you know what is this percentage d. So, I have got some space some space to hold an integer and that space is the in the a variable, but when the you from the keyboard type in say the value 25.

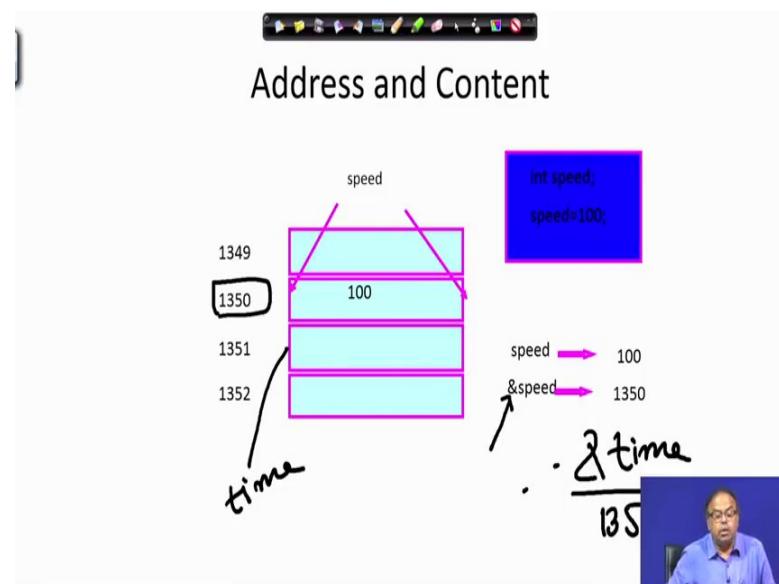
When will that value go? The value will go to the address of the variable a, what is the address of the variable a? 1350. So, it will go to 1350, 25 will come here similarly when

we write say scanf percentage d, percentage d comma and a and b then I am going to read 2 values and 2 integers, and the address of the first once upon suppose type in 25 and 27. So, for the add 25 will go to the address of a that is a 1350 and 27 will go to the address of b that is 1400.

Given this lets now read this a variable is a sign the specific memory location we know that. For example, if variable speed is assigned memory location 1350 and assume that the memory location contains the data value 100. So, when we use the name speed in an expression it refers to the value 100. So, for example, when we write distance is speed into time, then it will take this speed from this location 1350. Every variable has an address and its contents. So, we have seen a has got an address a is a variable a has got an address 1350.

And when I write the 25 into that 25 is a content. So, address and content we had earlier discussed also.

(Refer Slide Time: 29:06)



But you see here integers speed I think you can read it and. So, speed is this particular location that is in 1350 and when I right when I write speed equals speed assigned 100, then 100 is written over here alright when I assign it. So, speed is getting the value 100 whenever, but when I say what is and speed when I am asking the question what is the address of the variable speed, what is the address of the variable speed and the answer would be 1350.

So, this and sorry this and operation this and operation is nothing, but asking for the pointer to speed or the address to the variable speed. So, this should be the answer. So, and of suppose here time is given here, if I just say and time what will that be returned what is and time? And time will be 1351 something of this sort alright. So, that is another thing that we needed to understand what is the purpose of this and.

(Refer Slide Time: 30:38)

The screenshot shows a presentation slide with a title 'Contd.' and two bullet points. The first point discusses C terminology, mentioning 'speed' and '&speed'. The second point provides examples of printf and scanf statements. To the right of the text, there is a hand-drawn diagram. It features a whiteboard with a box containing the number '25' and the word 'speed' written above it. Two arrows point from the text 'speed' in the list to the 'speed' label on the whiteboard. Below the whiteboard, there is a small video frame showing a person speaking.

- In C terminology, in an expression
  - `speed` refers to the contents of the memory location.
  - `&speed` refers to the address of the memory location.
- Examples:

```
printf ("%f %f %f", speed, time, distance);
scanf ("%f %f", &speed, &time);
```

So, here in C terminology speed refers to the contents of the memory location, and speed refers to the address of the memory location corresponding to the variable speed. So, let us come to this example printf percentage f percentage f percentage f; that means, I am going to print three floating point numbers and what are the variables c floating point values speed time and distance; that means, what am I going to print look here I am going to print the contents of the memory location speed, the content of the memory location time, the content of the memory location distance.

And when I am reading percentage f percentage f and speed and time; that means, what that I am reading where I am reading in the address of the variable speed, I am reading in the address of the variable time. So, this is required to be understood. So, basically when I have say speed I once again repeat, suppose speed is 25 and I print speed; that means, I am printing the content of this location speed, but whenever I am reading into speed where am I reading the value? I am reading into the address of speed that is this location.

That is the main difference between these 2 alright. Let us stop here in the next lecture, we will straight way move head to write some c expressions, because till now whatever we have learnt are the bits and pieces the tools of c that is how the how just like in the language how the word are written what are the some of the simple rules, but then we will have to learn writing the real sentences in a language. So, that it we will start from the next lecture

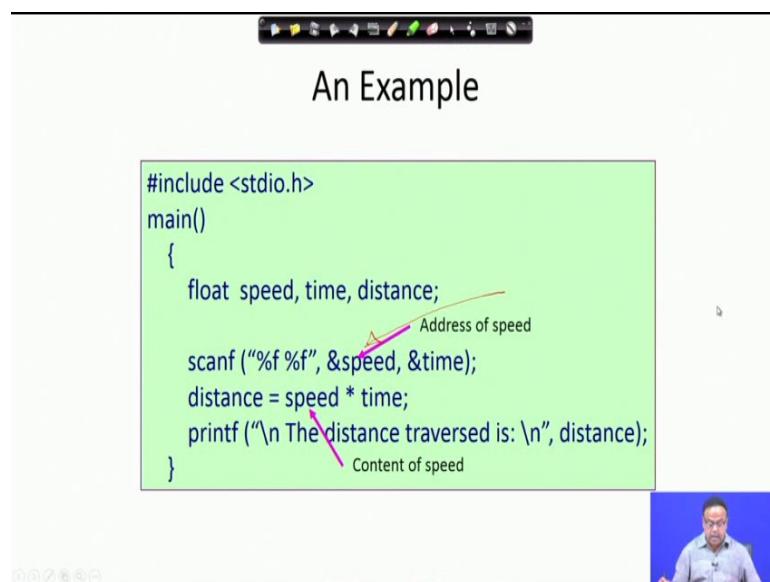
Thank you

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 11**  
**Assignment Statement and Operators in C**

In the last lecture, we encountered a special notation as if like this where we can put in ampersand and a variable name to denote the address of the particular variable right.

(Refer Slide Time: 00:24)



An Example

```
#include <stdio.h>
main()
{
    float speed, time, distance;
    scanf ("%f %f", &speed, &time);
    distance = speed * time;
    printf ("\n The distance traversed is: \n", distance);
}
```

So, that is what we encountered in the earlier class. Next we will be moving ahead towards some statements which are absolutely essential for writing any C program.

(Refer Slide Time: 00:54)

The slide has a title 'Assignment Statement' and a bulleted list of points. Handwritten annotations are present: 'variable\_name = expression' with a red box around 'expression' and 'speed' written next to it; several C code examples are shown in green, with 'speed = 500;' having 'speed' underlined and '500' written next to it; and a small video player window in the bottom right corner showing a person speaking.

- Used to assign values to variables, using the assignment operator (=).
- General syntax:  
variable\_name = expression;
- Examples:  
velocity = 20;  
b = 15; temp = 15;  
A = A + 10;  
v = u + f \* t;  
s = u \* t + 0.5 \* f \* t \* t;

Just as in English, we write different types of sentences, similarly in the language C just as it is a language there are again different types of statements that we can write, and through which we can express what we want to do. The simplest type of statement that we have already encountered about is assignment statement. Assignment statement means I have got some contents in some resistor; some resistor say X some memory location say variable x and has got some value 50, now how does this value come over here? Initially it was not there. So, suppose I have got memory location variable say speed, and I want to put in some value into this location alright say 500. So, what we do we write speed assigned 500 semicolon.

So, the general syntax for dot is a variable name followed by the expression; as you have written here speed is a variable name and the expression here is nothing, but the constant alright it could be something else.

(Refer Slide Time: 02:50)

The slide has a title 'Assignment Statement' and a bulleted list of points:

- Used to assign values to variables, using the assignment operator (=).
- General syntax:  
`variable_name = expression;`
- Examples:  
`velocity = 20;  
b = 15; temp = 12.5;  
A = A + 10;  
v = u + f * t;  
s = u * t + 0.5 * f * t * t;`

Handwritten annotations in red:

- Red arrows point from the words 'velocity', 'b', and 'temp' to their respective assignments.
- A red bracket groups the first three assignments: `velocity = 20;`, `b = 15;`, and `temp = 12.5;`
- Handwritten text on the right side of the slide reads: `speed = 2 * 500;` and `speed = v * q;`

A small video thumbnail of a person speaking is located in the bottom right corner of the slide area.

For example, I could have written speed assigned 2 times 500 or may be v times q whatever speed is assigned v times q. I and here are some examples of such assignments. As you can see here velocity is a variable right on the left hand side the variable name is there, on this side is an expression now what type of expression is this? 20 is nothing, but a constant value. Please note that all the statements must be ended with a semicolon you get the second. B assigned 15 again a constant. Look at third one temp assigned 12.5. So, what type of variable is temp? You will immediately answer temp must be a floating point variable of real number.

Here you see a different type of expression a variable A is being assigned the variable A plus 10 what does it mean. So, A assigned A plus 10. So, A being a variable, A is a memory location and suppose it has got some value 25.7, if A is a floating point variable.

(Refer Slide Time: 04:22)

## Assignment Statement

- Used to assign values to variables, using the assignment operator (=).
- General syntax:  
`variable_name = expression;`
- Examples:  
`velocity = 20;  
b = 15; temp = 12.5;  
A = A + 10;  
v = u + f * t;  
s = u * t + 0.5 * f * t * t;`

The diagram illustrates the assignment statement `A = A + 10;`. It shows a variable `A` with the value `25.7` in a box. An arrow points from this value to a plus sign, which then points to the value `10.0` in another box. Another arrow points from this sum to a final value `35.7` in a third box. The variable `A` is also labeled with the value `35.7` above it, indicating the updated state.

Now if I do `A` assigned `A` plus 10; that means, whatever is there in the as the value of the variable `A` is taken out. So, we take out we read 25.7 add 10 with that and we get 35.7 and then this 35.7 comes in here and I get a new look of `A` the same location, will now hold 35.7. So, you see the left hand side if the destination, where the new value occurred computation of the expression will go. And the left side sorry the right side can have the same variable of the source or might be the sum of a variable could be sum of the variable also like here this expression you see, I am using 3 variables here.

(Refer Slide Time: 05:47)

## Assignment Statement

- Used to assign values to variables, using the assignment operator (=).
- General syntax:  
`variable_name = expression;`
- Examples:  
`velocity = 20;  
b = 15; temp = 12.5;  
A = A + 10;  
v = u + f * t;  
s = u * t + 0.5 * f * t * t;`

The diagram illustrates the assignment statement `v = u + f * t;`. It shows three variables `u`, `f`, and `t` with values `20.0`, `0.5`, and `2.0` respectively. These values are added together (`20.0 + 0.5 * 2.0`) to produce a new value `21.0`, which is assigned to the variable `v`.

V is one variable, u is one variable I am sorry I am using 4 variables here, f is another variable and t is another variable.

Now, suppose you have got some value 20 f have got some value 0.5, and t has got some value 2. Then v is being computed as u is been taken 20 plus the product of these 2 2 and 5. So, 2 and 0.5 will be 1 and 1 is being added to 20. So, these 2 being added is becoming 21, and this 21 is filling up this variable v alright. So, here you see I can use more number of variables, I am sorry this is must be a little nice. So, it should be 21.0. Similarly here you see if they mix up variables how many are here s if a variable u is a variable t is a variable f t t, f t. So, 1 2 3 are there again u f t and s alright and there is a constant 0.5. So, this side entirely this side is the expression.

Similarly, this is again an expression, this is an expression this is also an expression will see more of these expressions in a moment alright.

(Refer Slide Time: 07:56)

The slide contains handwritten text in red ink:

- A value can be assigned to a variable at the time the variable is declared.

Handwritten code:

```
accel      [red box]  
float: accel; }  
        accel = 2.5;  
float accel = 2.5; ✓
```

A small video player window in the bottom right corner shows a person speaking.

So, a value can be assigned to a variable when the variable is declared for example, when I am declaring a variable, just starting I want to declare a particular variable and. So, I want to declare a variable may be acceleration. Acceleration alright now when and I say that acceleration is a real number. So, when I declare it I will write float acceleration, and then at some point later on I can say acceleration assigned say 2.5, this is one way. The other way is that I could have written it when I declared it a float acceleration assigned or initialised to 2.5 this is also allowed in c. So, a value can be assigned when it is declared.

(Refer Slide Time: 09:13)

Contd.

- A value can be assigned to a variable at the time the variable is declared.  
int speed = 30;  
char flag = 'y';  
← char flag;  
← flag = 'y' ;
- Several variables can be assigned the same value using multiple assignment operators.  
a = b = c = 5;  
flag1 = flag2 = 'y';  
speed = flow = 0.0;  
float speed = flow = 0.0;

*char flag;  
← flag = 'y' ;*

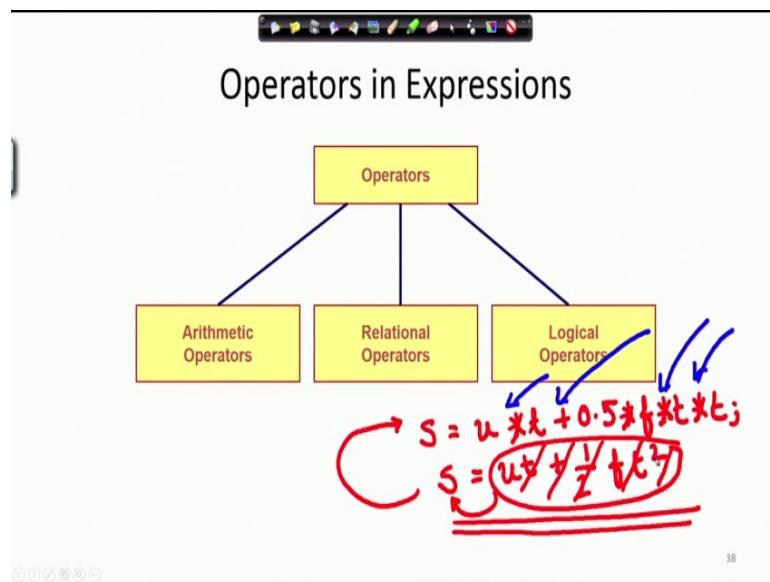
*float speed = flow = 0.0;*



For example here speed is an integer, which is being declared as an integer and along with that it is being assigned the value 30. Here you see the char is a type of another variable flag and when I am saying that it is a flag is a character, I am along with that I am also assigning it to value y.

Now, you understand you remember that within the single quote means it is a did the character string. So, I could have done it also like this char flag, and then alter on flag assigned y that is equivalent to what i did here. Several values can be assigned the same value using multiple assignments operative what does it mean? As soon as we see the example it will be clear for example, a b c all these are being assigned the value 5 flag ne and flag 2 both are being assigned the value y, speed and flow both are being assigned the value 0.0. So, here say for example, the correct thing would be to say float, speed flow are assigned 0.0. So, that is the simplest opposable way we go about it.

(Refer Slide Time: 11:05)



Now, when we write an expression, first thing we have seen now is the assignment statement. Now in the assignment statement if you have seen the earlier slide we were using some operative. For example,  $u$  assigned say  $v$  minus  $f$  times  $t$  right. So, these are an; this is an assignment statement here is an assignment, but on this side I have written an expression.

So, how can we write an expression in C. We have seen quite a few examples of that expressions in the earlier slides right, we have seen different types of expressions like  $s$  assigned  $u$  times  $t$  plus 0.5 times  $f$  times  $t$  times  $t$ . So, this is you will immediately recall that this is our standard school formula  $f$  is equal to  $u t$  plus half  $f t$  square right. Now this expression has got 2 components, this well-known expression in school has got 2 components. One side is the expression here which has to be computed and then that has to be assigned to another result of other variable. So, that is being written here. Now this is the way an expression is written in C you cannot just write it in this way as we are doing in our school, this is not possible we have to write it in this way ok.

Now, here when we write this expression you please observe a couple of things, look at these this, this, this. These are known as the operators you have this is meaning multiplication this is meaning addition these are again multiplication symbols. Now these they are sets of allowed operators in C. Just as every language allow some contrast to form sentence and some of them are not some other contrasts are not valid in a

sentence formation, similar to that in the programming language c or for that matter for any programming language there are some allowed operators by which we can form expressions. So, we have got 3 types of operators the type of operators; that we have encountered till now we have seen till now are arithmetic operators, but besides arithmetic operators there are 2 other types of operators called relational operators and logical operators ok.

(Refer Slide Time: 14:30)

**Arithmetic Operators**

- Addition :: + ←
- Subtraction :: - ←
- Division :: / ←
- Multiplication :: \* ←
- Modulus :: % ←

$$\begin{array}{r} 15 \% 3 \\ 3 ) 15 (5 \\ \hline 0 \end{array}$$

So, let us see a little more of this arithmetic operators, these are the some of the very familiar arithmetic operators you know that this is, this means addition normal subtraction, this is division, remember unlike this sort of division we use in schools we use here this symbol for division. Unlike this symbol that we use for multiplication, here we use this symbol and new symbol that we are introducing here is this symbol, like the percentage sign. Now this does not mean percent computing percentage, it means modulus what does modulus mean? Say the modulus means finding the remainder for example, if I compute 15 modulus 3; that means, I am dividing 15 by 3 and whatever is the remainder is my result. So, what is my remainder here? Remainder is 0. So, modulus is basically the remainder operator. So, another example let us look at.

(Refer Slide Time: 15:56)

The slide has a title 'Arithmetic Operators' and a list of operators:

- Addition :: +
- Subtraction :: -
- Division :: /
- Multiplication :: \*
- Modulus :: %

Handwritten calculations on the right:

$$27 \% .2 = 1 \quad \text{quotient} = 13$$
$$26 \% .2 = 0 \quad \text{remainder} = 1$$
$$26 / 2 = 13$$

A small video player window in the bottom right corner shows a person speaking.

Say I have got 27 modulus 2 what would be the result be? 27 divided by 2 the quotient is 13 right quotient is 13 and the remainder is 1. So, this modulus would be 1 whereas, if I had done 26 modulus 2 that would be 0.

Now, again I have got if I instead of modulus operator if I had done 26 divided by 2 the result would be 13. So, this gives you the quotient whereas, this gives you the remainder alright. So, this is a new operator that we are coming across and you should keep that in mind next let us proceed.

(Refer Slide Time: 17:05)

The slide has a title 'Examples' and several assignment statements:

```
distance = rate * time;
netIncome = income - tax;
speed = distance / time;
area = PI * radius * radius;
y = a * x * x + b*x + c;
quotient = dividend / divisor;
remain = dividend % divisor;
```

Handwritten annotations on the right:

- #define PI 3.14159;
- $\text{Area} = \pi r^2$

A small video player window in the bottom right corner shows a person speaking.

And here some examples distance is rate or velocity or speed multiplied by time. Please note again as I have told earlier also that any expression must end with a semicolon as is being done here what does it this mean? Can you read this variable, can you read this variable it is meaningful net income is income minus tax. So, operator is minus this is the arithmetic expression and this is the arithmetic operator. Speed is distance divided by time again ended with a semicolon speed is distance divided by time how do we find the area of a circle? Pi you remember pi we can define pi we had seen this example earlier has defined pi 3.1415 etcetera, etcetera, we could have done that. So, pi is a constant times radius times radius. Here basically what I am computing is area is being assigned is the assignment operator and the expression is pi r square. Now this expression I am writing in this way pi times r radius times radius.

(Refer Slide Time: 18:50)

The slide is titled "Examples". It contains several pseudocode assignments and their corresponding mathematical formulas:

- `distance = rate * time ;`
- `netIncome = income - tax ;`
- `speed = distance / time ;`
- `area = PI * radius * radius;`
- $y = ax^2 + bx + c;$
- $\text{quotient} = \text{dividend} / \text{divisor};$
- $\text{remain} = \text{dividend \% divisor};$

A blue arrow points from the handwritten formula  $y = ax^2 + bx + c;$  down to the pseudocode line `y = a * x * x + b*x + c;`. A blue arrow also points from the handwritten formula  $\text{remain} = \text{dividend \% divisor};$  down to the pseudocode line `remain = dividend % divisor;`. In the bottom right corner of the slide, there is a small video player window showing a person speaking.

Here is another expression y assigned a x square what does it, how does it how do we write it typically in school. This is expression which is a x square plus b x plus c alright this is a very familiar expression of a quadratic expression. Now when we write it again here you see how many operators I have got 2 multiplication operations 3 multiplication operations and 2 addition operations right and one assignment operation. So, here again quotient is divided by divisor. Now this is exactly what I was telling a couple of moments back, that this operator is actually turning you the quotient alright of a division operation and this is actually giving you the remainder of a division operation. So, here are some examples of arithmetic expression.

(Refer Slide Time: 20:06)

Contd.

- Suppose  $x$  and  $y$  are two integer variables whose values are 13 and 5 respectively. (13) 5

$x + y$	18
$x - y$	8
$x * y$	65
$x / y$	2
$x \% y$	3

→ (13) 5  $x$   
(5)  $y$

$\underline{z} \leftarrow x * y;$   
 $\underline{z} \leftarrow x * y;$

Suppose  $x$  and  $y$  are 2 integer variables, and the values are we know 13 and 15. When added to  $y$   $x$  plus  $y$  arithmetic operator plus will give me 18  $x$  minus  $y$  13 minus 5 will give me 8. Now the point to note here is that here again always try to think in terms of our memory location diagram  $x$  and  $y$  are 2 variables,  $x$  is 13 and  $y$  is 5. So,  $x$  plus  $y$  means the content of the location  $x$  plus the content of the location  $y$ ;  $x$  minus  $y$  is the content of the location  $x$  minus the content of the location  $y$ .

Similarly, when we multiply it is 65, now if I add an assignment with this operation with this operation I just add a assignment for example, I write I am giving 2 variations  $z$  assigned  $x$  times  $y$ ; that means, what? There is another location with where the content of  $x$  and the content of  $y$  are taken and multiplied and 13 times 5 is 65 that is stored there and that is possible because I have assigned it here. Again I could have assigned alternative could have done  $x$  assigned  $x$  times  $y$  what would have happened in this case? The content of  $x$  would have been taken 13 multiplied with a content of  $y$  that is 5 13 and 5 would be 65, and this product 65 where would that be written? It would be written in  $x$  why because it is being assigned to  $x$ . So, then this would be over written with 65.

Similarly, we will recall now  $x$  divided by 2 is a division from expecting the quotient 13 divided by 5 what will be the quotient 2, but  $x$  modulus  $y$  would be the remainder of when I divided 13 by 5. So, that modulus is 3.

(Refer Slide Time: 22:50)

Operator Precedence

- In decreasing order of priority
- 1. Parentheses :: ()

$$x = \overbrace{(p+q)*z}^A - \overbrace{x/(l+m)}^B;$$

A small video player window in the bottom right corner shows a person speaking.

Now, in an expression we can have different operators. If more than one different operator occurs in a particular expression how will that expression be evaluated. So, now, we are concerned about how we will evaluate or find the result of computing an expression k. Now here is a list in decreasing order of priority. So, if I have something like this say x assigned p plus q times z minus x divided by l plus m. Now here you can see that I have got different operators what are those? Our well known operators are plus multiplication minus division plus again here is another operator that is parentheses. Now as we learnt in school algebra that the parentheses has got the highest priority alright. So, I will first compute the elements which are within the parentheses. So, first p plus q will be computed then l plus and l plus m will be computed now out of this p plus q and l plus m which one will be computed first? Whenever these 2 that are these are of the same priority this parentheses, now if there will be more than one operator of the same priority they will be computed left to right.

So, first we will have p plus q computed alright suppose that is something, say let us call it a some value A times Z minus X suppose this is computed to be some constant B. So, in that way it will be computed. So, the parentheses has got the highest priority next after parentheses is unary minus.

(Refer Slide Time: 25:20)

The slide has a title 'Operator Precedence' and a bulleted list:

- In decreasing order of priority
  1. Parentheses :: ()
  2. Unary minus :: -5

Handwritten notes explain the evaluation of the expression  $-x + 7.5;$  given  $x = 2.5;$

The expression is shown with a box around  $-2.5$  and a circle around  $x.$  Arrows point from the circled  $x$  to the circled  $-$  sign and then down to the result  $5.0.$

A small video player icon in the bottom right corner shows a person speaking.

Unary minus means usually when we write something like  $x$  minus  $y,$  then I have got 2 variables on which I am carrying out this subtraction. This is binary operator in the sense that I am leaving 2 variables of 2 constants 2 elements and which I am carrying out this computation. Unary minus means that particular variable is being operated on for example, if I had something like this minus  $x$  plus 7.5 semicolon and suppose  $x$  was 2.5 n. So, what will be the value?  $X$  is 2.5. So, for this will be done. So, it will be minus 2.5 plus 7.5. So, the result will be 10 5; it is not that I will first compute this 2.5 plus 7.5 and then do negation alright.

So, the unary minus. So, here I have shown a variable, it could be something constant also minus 2 plus 6 that means, you all know that 6 subtracted by 2 although it is plus because this has got the higher precedence.

(Refer Slide Time: 27:17)

## Operator Precedence

- In decreasing order of priority
  1. Parentheses :: ()
  2. Unary minus :: -5
  3. Multiplication, Division, and Modulus

42

The next one is multiplication division and modulus. These 3 will have the same priority multiplication division and modulus these 3 operators will have same priority therefore, if I have an expression like say x division y multiplied by z modulus q how will that be which one will be done first? This, this and this have got the same priority therefore, we will carrying it out left to right alright, but if the expression was something like this minus.

(Refer Slide Time: 28:06)

## Operator Precedence

- In decreasing order of priority
  1. Parentheses :: ()
  2. Unary minus :: -5
  3. Multiplication, Division, and Modulus

42

X divided by y multiplied by z modulus q then which would be done first? First is unary minus should be done first followed by these 3 candidates which will be done left to right left to right.

(Refer Slide Time: 28:42)

Operator Precedence

- In decreasing order of priority
  1. Parentheses :: ()
  2. Unary minus :: -5
  3. Multiplication, Division, and Modulus
  4. Addition and Subtraction

$$\begin{array}{c}
 p + x * y / z - q \\
 \downarrow \\
 p + A / z - q \\
 \circled{A} \downarrow \\
 p + B - q \\
 \downarrow \\
 c - q
 \end{array}$$

Next comes addition and subtraction. So, if I have got an expression again like say x times y even before that if we put it, p plus x times y divided by z minus q then in which order would it be computed? First plus will not be computed multiplication has got an higher priority this also has got an higher priority. So, out of these 2 which one will be done first left to right? So, first I will do this suppose this is yielding a result A. So, it will be p plus A then divided by Z minus q and then this one will be done because left to right of the same precedence suppose it is B then it turns out to be p, it turns out to be p plus b minus q alright. Now out of this plus and minus have got the same priority then which one will be done first this part will be done first, suppose that is c, c minus q this is how this entire operation will be done.

(Refer Slide Time: 30:12)

The screenshot shows a presentation slide titled "Operator Precedence". The slide content is as follows:

- In decreasing order of priority
  1. Parentheses :: ()
  2. Unary minus :: -5
  3. Multiplication, Division, and Modulus
  4. Addition and Subtraction
- For operators of the *same priority*, evaluation is from *left to right* as they appear.

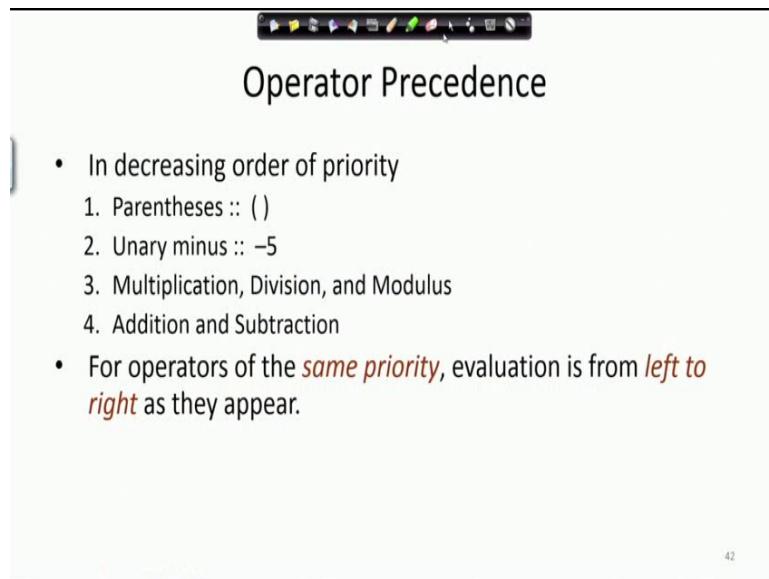
A small video thumbnail of a person speaking is visible in the bottom right corner of the slide area.

For operators of the same priority the evaluation is from left to right. In the next class we will see some more examples of this and we will proceed further.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 12**  
**Arithmetic Expressions and Relational Expressions**

(Refer Slide Time: 00:20)



The slide has a title 'Operator Precedence' and a bulleted list:

- In decreasing order of priority
  1. Parentheses :: ()
  2. Unary minus :: -5
  3. Multiplication, Division, and Modulus
  4. Addition and Subtraction
- For operators of the *same priority*, evaluation is from *left to right* as they appear.

42

In the last lecture, we were introduced to the precedence among the operators, and we had seen that parentheses has got the highest precedence followed by unary minus then multiplication division module, and modulus have got the same priority addition and subtraction are having the next priority. And for operators of the same priority if an expression there are more than one operators which are of the priority then evaluation is done from left to right.

Only one thing we did not mention and that is if we want to change the precedence of the evaluation, then we can always do it using parentheses because we know parentheses is the highest priority one or so, over riding scenario. So, let us look at some examples for this which will make the idea clear.

(Refer Slide Time: 01:30)

Examples: Arithmetic expressions

$a + b * c - (d / e)$	$\rightarrow a + (b * c) - (d / e)$	$b * c - x$
$a * -b + d \% e - f$	$\rightarrow a * (-b) + (d \% e) - f$	$d \% e - y$
$a - b + c + d$	$\rightarrow ((a - b) + c) + d$	$a + x - y$
$x * y * z$	$\rightarrow ((x * y) * z)$	$x * y - z$
$a + b + c * d * e$	$\rightarrow (a + b) + ((c * d) * e)$	$a + b + c * d * e$

Here are some examples arithmetic expressions; here you can see a plus b times c minus d divided by e. Now according to my precedence rule, this is equivalent a plus as if I have if I have done it hand computation. If I had done by hand computation how would I have done there is a chance of confusion, somebody could have done a plus b and the whole thing multiplied by c etcetera the you know even in school level in order to avoid such confusions we use parentheses. And this is equivalent to this scenario that a; this b c are parenthesized and d divided by e are parenthesized. So, first this will be done then this will be done why b into b times c first, because it is left and this is right, right.

So, first this then this then addition and sub. So, let us see how you will break it down. Again we will have first b times c will be done followed by b times e then we have suppose this result is x and this result is y. Then my expression is a plus x minus y now out of these again these 2 they are the same precedence. So, first this will be done suppose this is z minus y that is how the computation will go on.

Here what is the significance of this? The critical point to note here this part here there is an unary operator. Therefore, during computation what is it means the computer will automatically assume that this parentheses are there alright. If you had not wanted that then you better write in some other way put it some other way. So, that the confusion is not there. The computer will not allow the compiler will generate the code in a way that it will be treated as this a times first minus b will be done, then this one is the highest

precedence next highest precedence. So, this will be done, then multiplication between these 2 will be done now I am sorry I am sorry here normally if I had done this what would have happened.

(Refer Slide Time: 04:07)

$a + b * c - d / e$	$\rightarrow a + (b * c) - (d / e)$
$a * \underline{-b} + \underline{d \% e} - f$	$\rightarrow a * \underline{(-b)} + \underline{(d \% e)} - f$
$a - b + c + d$	$\rightarrow ((a - b) + c) + d$
$x * y * z$	$\rightarrow ((x * y) * z)$
$a + b + c * d * e$	$\rightarrow (a + b) + ((c * d) * e)$

First of all I will be doing minus b and by capital letter I am writing some constant value alright. So, b is right now is 5. So, first after computing this it would be a times minus 5 because b was 5 plus suppose d was 100 and e was 9. So, what is my modulus 1 right. So, then I am I will be now at this point, I have got b modulus e minus f.

Now, at this point which one has got the higher priority? This and this, now since now I have put a parentheses here therefore, if I put a parentheses here then this will be done first otherwise this would be would have been done first; because this multiplication has got the same priority as this and put it in a place to write now. So, similarly here a minus b plus c plus d.

Now, here these are having the same priority left to right, now I want it would to be done in a particular order therefore, I have put the parentheses in such a way that a minus b should be done first, then a minus b plus c will be done first then d will be added to that. That would have been done anyway why because since these are of the same priority that would be done left to right. So, first this would be done, then this would have been done then this would have been done either way.

Here also the same I can write in this way, but that is the precedence is being shown in the form of bracket.

(Refer Slide Time: 06:23)

Examples: Arithmetic expressions

$a + b * c - d / e$	$\rightarrow a + (b * c) - (d / e)$
$a * -b + d \% e - f$	$\rightarrow a * (-b) + (d \% e) - f$
$a - b + c + d$	$\rightarrow ((a - b) + c) + d$
$x * y * z$	$\rightarrow ((x * y) * z)$
$a + b + c * d * e$	$\rightarrow (a + b) + ((c * d) * e)$

A small video player window in the bottom right corner shows a person speaking.

But here this is an example, where I use parentheses to over write the precedence. The precedence between this multiplication and this modulus was the same, but just by putting this parentheses there, I have forced this to be computed first. Similarly here now normally given this, this would be done first, then this would have been done, then this would have then then additions would have been done.

But I have over written that by saying that wait first you do it a plus b, then you multiply c and d multiplied e that is there anyway, and then you do the addition, but this is actually over writing the normal precedence. So, you should I mean with a little bit of practice we will initially we will make some mistakes, but then gradually with practice that will go away n and all of you will be able to write it fine.

(Refer Slide Time: 07:32)

The slide has a title 'Integer Arithmetic' and two bullet points:

- When the operands in an arithmetic expression are integers, the expression is called *integer expression*, and the operation is called *integer arithmetic*.
- Integer arithmetic always yields integer values.

There is a small number '44' at the bottom right corner.

Now, we are coming to a very important concept of this arithmetic expression evaluation integer arithmetic.

What does it mean? It means that when the operands in an arithmetic expression are integers, then the expression is called an integer expression and the operation is called the integer arithmetic. Integer arithmetic always deals integer values, it will always yield integer values. So, it will be clearer when we take some examples.

(Refer Slide Time: 08:20)

The slide has a title 'Real Arithmetic' and three bullet points:

- Arithmetic operations involving only real or floating-point operands.
- Since floating-point values are rounded to the number of significant digits permissible, the final value is an approximation of the final result.  
 $1.0 / 3.0 * 3.0$  will have the value  $0.99999$  and not  $1.0$
- The modulus operator cannot be used with real operands.

A handwritten note is overlaid on the slide, showing the expression  $a + b$  with 'operator' written above the '+' sign and 'operand' written below each 'a' and 'b'. There is also a small number '45' at the bottom right corner.

On the other hand real arithmetic is arithmetic operation involving only real or floating point values.

For example here you can see this expression  $1.0 / 3.0 * 3.0$ . Now all these are operands now here is I would like to introduce a term, when I say a plus b then this plus I call it as an operator, and these I call as operand on which the operation is being done. Now in real arithmetic all the operands are floating point numbers.

Now, sometimes the floating point values are rounded to the number of significant digits because it may be 7.5976 like that you can go on, its often rounded to the number of significant digits permissible in the particular machine, we get an approximation of the results. Now one thing that we have to remember is that the modulus operation is not defined over real operands.

(Refer Slide Time: 09:55)

Real Arithmetic

- Arithmetic operations involving only real or floating-point operands.
- Since floating-point values are rounded to the number of significant digits permissible, the final value is an approximation of the final result.  
 $1.0 / 3.0 * 3.0$  will have the value  $0.99999$  and not  $1.0$
- The modulus operator cannot be used with real operands.

3.5 % 2.1 X

So, I am not allowed to do  $3.5$  modulus  $2.1$  that is not allowed. The modulus operation is allowed only for integer arithmetic.

(Refer Slide Time: 10:17)

Mixed-mode Arithmetic

- When one of the operands is integer and the other is real, the expression is called a *mixed-mode* arithmetic expression.
- If either operand is of the real type, then only real arithmetic is performed, and the result is a real number.

Some more issues will be considered later.

$25 / 10 \rightarrow 2$      $25 / 10.0 \rightarrow 2.5$

$25.0 / 10.0 = 2.5$

$25 \% 10 = 5$      $25 / 10 = 2$

Now, we can also have mixed mode arithmetic, where some of the operands are integers and some are float or real in the case we call it a mixed mode arithmetic expression. If any of the operand is of real type then only real arithmetic is performed and the result is a real number this is this example will make it clear.

Now, you see here look at the first one 25 divided by 10, both the operands this operand and this operand both these operands are integer therefore, when I divide by divide them I get an integer quotient or integer result whereas, if one of the operands was integer and one was real if at least if even one is real, then the result will be real. So, what is happening here? This is becoming like 25.0 divided by 10.0 so that will become 2.5 alright.

Now, here on the other hand since both are integers the result will be integer. So, let me just put in one twist on this. Suppose now here 25 divided by 10 is actually what? Is a if I divide it, it is also 2.5 right 2.5, but since this operator is nothing, but an integer division. So, it will in out the quotient and the quotient is an integer the quotient is 2, if I had done 25 modulus 10 what would my result be? The result would be 5 as a integer right. So, we could understand what is meant by mixed mode real mode, and integer mode of arithmetic we will see some more things later.

(Refer Slide Time: 12:44)

Problem of value assignment

- Assignment operation  
variable= **expression\_value**;  
or  
variable1=**variable2**;

Data type of the RHS should be compatible with that of LHS.

v = u + f \* t ;      *value*

v = u ;      *value*

Now, there are some problems of value assignment for example, in assignment operation we actually take the expression value and assigned it to a variable or we assign a particular variable to another variable. For example, we could have as we have done say something v is u plus f times t, this is an expression value is going first it will be computed and the value will go to v. So, the value is going there right on the other hand I could have done v assigned u; that means, the value of u is going to the this variable here also, this value is going here the value of an expression here just a value of a variable. But the most important point is that the data type of the right hand side should be compatible with that of the left hand side ok.

(Refer Slide Time: 13:54)

Problem of value assignment `int v;`  
float `u;`  
 $v = u;$

- Assignment operation  
`variable = expression_value;`  
or  
`variable1 = variable2;`

Data type of the RHS should be compatible with that of LHS.

$u = 25.7$

$v = 25$

A small video player window in the bottom right corner shows a person speaking.

Now, if for example, I say `v` assigned `u` and `u` is a float and `v` is an int, then may I may have problem what type of problem can I have? Suppose `u` is 25.7, I assigned this to `v`. So, `v` can only take an integer value. So, in most of the cases we will get an error why? Because you know that depending on the computers, depending on the different architectures integers are often 2 bytes and floats are given 4 bytes.

So, you are trying to track in a large number I mean a 4 byte into a 2 byte space the space is not there. So, the errors can come, in some cases may be they will cut it out and represent it as ignore this part and you will not understand it will just assign 25 to `v`, that is also an error that is skipping in now this is this error will go unnoticed. If the compiler catches it which is mostly the case, in that case you will you will the compilation error will occur and we will understand that it is not working alright.

(Refer Slide Time: 15:40)

The slide has a title 'Problem of value assignment' and a bullet point 'Assignment operation'. It shows two code snippets: 'variable=expression\_value;' and 'variable1=variable2;'. A note states 'Data type of the RHS should be compatible with that of LHS.' Below this, an example says 'e.g. four byte floating point number is not allowed to be assigned to a two byte integer variable.'

For example as I said the 4 bytes floating point number is not allowed to be assigned to a 2 byte integer variable.

(Refer Slide Time: 15:46)

The slide has a title 'Type Casting'. It shows a code snippet: 'int x; float r=3.0; x=(int)(2\*r);'. To the right, there is a diagram showing memory structures for variables x and r. Variable x is shown as a 2-byte integer, and variable r is shown as a 4-byte floating-point number. A callout box contains the code: 'double perimeter; float pi=3.14; int r=3; perimeter=2.0\* (double) pi \* (double) r;'. A small video window in the bottom right corner shows a speaker.

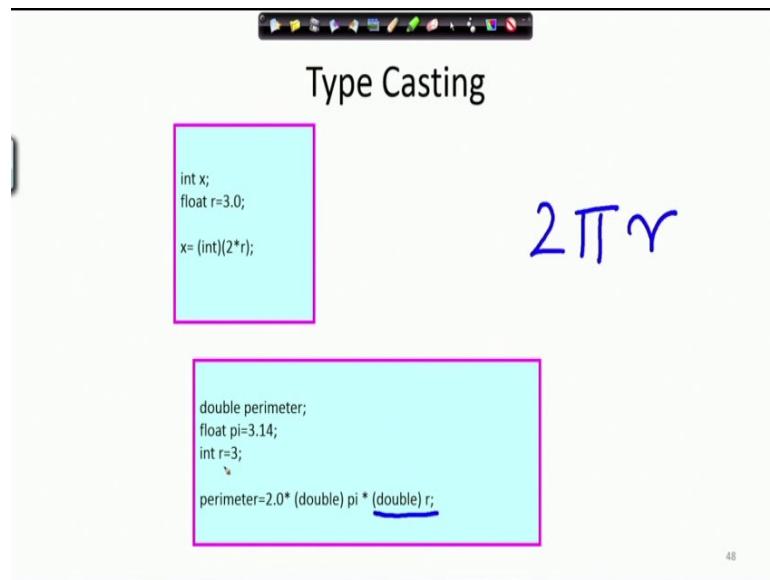
So, sometimes what we do this is a new concept called type casting, look at this example. Here we have defined x to be integer. So, x have got 2 bytes, suppose that I show 2 bytes for x and r is floating number which has been represented through 4 bytes. Now sometimes what we do we type cast this thing 2 times r will be what 2 times r will be float? Because the r is real and. So, if one of the operators are real, then it this part

will be a real arithmetic, but then that result and casting back to int. So, this operation is called the type casting operation; that means, whatever I have I have casting that off it in a way. So, that it can fit in as an integer ok.

Here is another example say perimeter has been defined as double; that means, it can have twice the floating point space. Pi is a float three point one 4 now you see this is pi we have encountered that pi earlier, but here this pi is not the constant which has defined as the pi, but pi is a variable, which has been initialised at the time of declaration and r is an integer.

Therefore when I am computing perimeter is twice pi r. So, what am I actually giving?

(Refer Slide Time: 17:31)



I am trying to compute  $2\pi r$  right the circumference of a circle. So, twice 2 is real now  $\pi \cdot r$  perimeter is doubled. So, I want to have perimeter with a larger accuracy therefore, I first make r to be doubled, I type cast this integer to a double retype, I type cast this float to a double type then I multiply and get the perimeter. So, this is another very powerful operation, which we often need for scientific computation.

(Refer Slide Time: 18:18)

The slide has a title "Type Casting". It contains two code snippets. The top snippet is:

```
int x;  
float r=3.0;  
x= (int)(2*r);
```

A callout points to the line `x= (int)(2*r);` with the text "Type casting of a floating point expression to an integer variable."

The bottom snippet is:

```
double perimeter;  
float pi=3.14;  
int r=3;  
perimeter=2.0* (double) pi * (double) r;
```

A callout points to the lines `perimeter=2.0* (double) pi * (double) r;` with the text "Type casting to double".

In the bottom right corner, there is a small video player window showing a person speaking.

So, type casting of a floating point expression to an integer as is shown here, and the type casting to double has been shown here we have already explained that.

(Refer Slide Time: 18:29)

The slide has a title "Relational Operators". It contains a list of relational operators:

- Used to compare two quantities.
- < is less than
- > is greater than
- <= is less than or equal to
- >= is greater than or equal to

To the right of the list, there are four blue hand-drawn symbols: <, <=, >, and >=.

In the bottom right corner, there is a small video player window showing a person speaking.

Now, we have till now looked at arithmetic expressions, how to form arithmetic expressions, and arithmetic expressions are formed using arithmetic operators. Besides arithmetic operators there is another very important type of operator called the relational operator. A relational operator is used to compare 2 quantities let us see for example, this symbol say very familiar denotes is less than, this symbol which is also familiar shows is

greater than, this symbol is less than or equal to this, sometimes in our normal course we write it in this way, but in a computer we cannot write it in this way we have to write it less than equal to alright.

Similarly, greater than equal to unlike the greater than equal to that we used to write in this way, we have to write here greater than or equal to now this means greater than or equal to that is well known to us.

(Refer Slide Time: 19:58)

Relational Operators

- Used to compare two quantities.

<	is less than	$x = y$
>	is greater than	$x \leftarrow y$
<=	is less than or equal to	$x == y$
>=	is greater than or equal to	$x == y$
==	is equal to	

Next here is another operator this requires some discussion is equal to. Now earlier this is very important to know than this is a very source of a very common error and common mistake. Typically when we say in our normal arithmetic  $x$  equals  $y$  we meant that the value of  $x$  and the value of  $y$  are the same right, but we have seen that in c language this actually means the variable  $x$  is being assigned the value of  $y$  right.

So, how do I compare whether these 2 variables are equal or not, for that c provides this technique of using 2 consecutive equal it is find to show  $x$  whether  $x$  is equal to  $y$  alright we will come to this in a moment, there is another operator that is not equal to again.

(Refer Slide Time: 21:11).

The slide has a title 'Relational Operators' and a list of operators with their meanings:

- Used to compare two quantities.
- < is less than
- > is greater than
- <= is less than or equal to
- >= is greater than or equal to
- == is equal to
- != is not equal to

Handwritten annotations include:  
A circled ' $x < y$ ' with arrows pointing from 5 to 7 and from y to x.  
 $x = 5; y = 7;$   
 $x + y = 12$   
 $\neq \quad !=$

Let us compare to what we had in school, we used to write something like this in C we used to write it with this exclamation mark followed by equality, this means it is not equal to alright.

Now, a very important thing comes into the play whenever I am using this relational operators what do I get? If I write  $x$  less than  $y$  in an now we know that what are these  $x$  and  $y$ ? These are 2 operands and what is this? This is one operator. So, when I carry out some operation I will be getting some result. So, if I had done  $x$  plus  $y$  and suppose  $x$  was 5 and  $y$  was 7, then  $x$  plus  $y$  would have given me what 12 would have resulted into 12, 5 plus 7.

Now, if I do this  $x$  less than  $y$ ; that means, 5 less than 7 what will this give me? It will any relational operator gives me only true or false 1 or 0. So, when I say  $x$  less than  $y$  and  $x$  is 5  $y$  is 7 it is true therefore,  $x$  less than  $y$  in this case will return 1.

(Refer Slide Time: 23:07)

Relational Operators

- Used to compare two quantities.

$x > 10$	< is less than	$y = 7; x = 5;$
	> is greater than	$y < x \rightarrow 0$
	$\leq$ is less than or equal to	$y > x \rightarrow 1 / \text{true}$
	$\geq$ is greater than or equal to	$y \leq x \rightarrow 0$
	$=$ is equal to	$y \geq x \rightarrow 1$
	$\neq$ is not equal to	$y == x \rightarrow 0$
		$y != x \rightarrow 1$

If I write  $y$  less than  $x$ , it will return me false because  $y$  is 7 and  $x$  is 5, then this will be false this will return 0 ok.

Similarly, for greater than  $y$  greater than  $x$  will return me in this case is for this particular value set,  $y$  greater than  $x$  will give me true one or it conceptually we can say true that this true cannot be represented as true in a computer it is this in c it is represented as 1. Less than equal to in this case what would we have what would have happened?  $Y$  less than equal to  $x$  is it true or false less than or equal to in this case is  $y$  less than  $x$ ? No is  $y$  equal to  $x$ ? No therefore, it will be false.

$Y$  greater than or equal to  $x$  what would have happened?  $Y$  is not greater than is greater  $x$  not equal to  $x$  therefore, it immediately becomes true  $y$  is  $y$  equal to  $x$  in this case no. So, it will return false  $y$  is not equal to  $x$  is it true or false it is true therefore, it will be one. So, what we could have observed is we can form expressions using relational operators, but these expressions only return the values true and false ok.

We will see that these relational operators if you recall in our flowchart discussions, we often had some decision box is shown as rhombus and in that decision box we used to say  $x$  greater than 10 or say.

(Refer Slide Time: 25:22)

## Relational Operators

- Used to compare two quantities.
  - < is less than
  - > is greater than
  - <= is less than or equal to
  - >= is greater than or equal to
  - == is equal to
  - != is not equal to

49

We used to write something like N equal to 3 right yes or no right we have taken one path for yes one path for no and that can be captured very easily using such relational operators as we will see.

(Refer Slide Time: 25:50)

## Examples

$10 > 20$	is false
$25 < 35.5$	is true
$12 > (7 + 5)$	is false

12

- When arithmetic expressions are used on either side of a relational operator, the arithmetic expressions will be evaluated first and then the results compared.

$a + b > c - d$  is the same as  $(a+b) > (c-d)$

10 > 17 → 0

Now, here are some other examples or I have already given you enough examples 10 greater than 20 true or false, it is false 25 less than 35.5 yes it is true, 12 greater than 7 plus 5 now here if one of the operands is an expression that will search the evaluated. So,

the arithmetic operators and expressions have higher priority over relational operators alright.

So, here what will happen 7 plus 5 will be first computed. So, that is 12 if 12 greater than 12 no. So, it is false. So, when arithmetic expressions are used, on either side of a relational operator the arithmetic expressions will be evaluated first and the arithmetic expressions will be evaluated in accordance with a precedence that we have already defined. So, if there will be something like this a plus b greater than c minus d that is equivalent to, first I will compute a plus then I will compute c plus d and then suppose a plus b is 10 and c plus d is 17, then this will lead to false first these will be computed then the relational operator will be computed.

(Refer Slide Time: 27:29)

Examples

---

- Sample code segment in C

```
x{  
    if (x > y)  
        printf ("%d is larger\n", x);  
    else  
        printf ("%d is larger\n");}  
15 in larger
```

```
graph TD; A((x > y?)) -- x --> B["15  
print x is larger"]; A -- y --> C["15  
print y is larger"]; A -- N --> D["15"]
```

---

So, here is a quick look at an application of this, suppose I want to implement the flowchart something like this, I have read x and y and I want to do x greater than y if so, yes I will say I will print x is larger otherwise I will print y is small otherwise I will print y is larger yes. So, that sort of situation how will that translate into c language here you see this part we have not discussed, but is very intuitive you can understand. If x is greater than y, I print what is larger dash is larger then what will come here the value of x. So, I will print say x is 10 and y is 15 n this is x and this is y then what will be printed here? 10 is not greater than 15. So, this part will not be executed will come to here

because x is not greater than y. So, we are taking no path, and we will print dash is larger and in dash what will come the value of y that is 15.

So, what will be printed is 15 is larger. So, that is how we can implement this sort of decision box and when I have that the choice between path based of the decision, that is being a condition is being computed using relational operators which this conditions are telling me which path I will take if the is the situation true if. So, I will take this path otherwise I will take this path this is known as these are (Refer Time: 30:06) of the relational operators.

(Refer Slide Time: 30:13)

The image shows a presentation slide titled "Logical Operators". At the top, there is a toolbar with various icons. Below the title, there is a bulleted list:

- There are two logical operators in C (also called logical connectives).
- What they do?

At the bottom of the slide, there is a navigation bar with icons for back, forward, and search, along with the number "52" indicating the slide number.

So, next class we will start with logical operators, today we have discussed about arithmetic expression how arithmetic expressions are formed using arithmetic operators and what are the very important thing that we discussed today is, you know in the earlier lecture also lecture that what is the precedence till now, why what I have discussed is that what is the precedence of the operators in an arithmetic expression and what is the relational operators. And we have seen that relational operators have the lower priority than arithmetic expressions. So, if I am either side of the relational operators the arithmetic expressions those will be evaluated first, in accordance to the arithmetic operation priority. Next we will come to another very important concept that is logical operators.

Thanks.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 13**  
**Logical Operators and Change in Control Flow**

(Refer Slide Time: 00:21)

The slide has a title 'Logical Operators' and contains the following text and handwritten annotations:

- There are two logical operators in C (also called logical connectives).  
     $\&\&$  → Logical AND  
     $\|$  → Logical OR
- What they do?
  - They act upon operands that are themselves logical expressions.

Handwritten notes:  
Temp > 50 — ①  
a\*b+c >= 25 — ②  
Temp > 50 && a\*b+c >= 25  
↓      ↓      ↓  
01      00      1  
↓  
01

In the last lecture we had talked about two different types of operators, relational operator and arithmetic operators. Today we will be discussing about the third type of operator which is known as a logical operator.

Logical operators are also known as logical connectives. So, there are two essentially there are two logical operators in C, one is logical AND and the other one is a logical OR. Now, what do they do? They act upon the operands themselves which are logical expressions. For example, let us say I am writing a logical expression temperature is greater than 50, now this will this statement suppose the temperature now is 40 degree centigrade then temperature greater than 40 greater than 50 will result in false value, because the relational operator will always generate either true or false. And suppose there is another logical expression a times b plus c is greater than equal to 25, now this is another logical expression sorry this is another relational expression. On the left hand side of this expression I have got an arithmetic expression and on the right hand side I have got a constant and I am connecting them with a relational operator greater than

equal to. Now, if a times b plus c is greater than 25 or equal to 25 then this will result in true.

Now, I can connect these two this one and this two together and write another expression like temperature greater than 50 and a times b plus c is greater than equal to 25. Now, this expression is a combination of two relational expressions and a logical operator a logical connective. This logical AND means that this entire thing expression will be true or will result in a 1 if both of them are true. So, if the temperature is 40 then this will become false or 0 and if this is 25 then this is true, but 0 and 1 both are not true therefore, this 0 and 1 will result in 0. But suppose if the temperature was 50 and a times b plus c is equal to 25 or greater than 25 then this is true and also this is true in that case these two together and because they are ANDed then this will be true.

So, the logical AND operator what it does is it turns a true value or 1 if all the components of the expression logical expression connected by the AND operator, logical AND operator is true.

(Refer Slide Time: 05:37)

## Logical Operators

- There are two logical operators in C (also called logical connectives).  
 $\&&$  → Logical AND  
 $\|$  → Logical OR
- What they do?
  - They act upon operands that are themselves logical expressions.
  - The individual logical expressions get combined into more complex conditions that are true or false.

$\&$        $\& \text{scanf}$

Now, one thing you can note here that since this ampersand is a character and we have already use this ampersand in expressions like AND scanf and we have discussed that this AND actually means we are trying to get the address of a particular I am sorry I am sorry absolutely sorry.

(Refer Slide Time: 06:10)

The slide has a title 'Logical Operators' at the top. Below it is a bulleted list:

- There are two logical operators in C (also called logical connectives).
  - && → Logical AND
  - || → Logical OR
- What they do?
  - They act upon operands that are themselves logical expressions.
  - The individual logical expressions get combined into more complex conditions that are true or false.

A red handwritten note on the right side of the slide says "scanf (%d, &velocity)" with an arrow pointing to the address operator "&". A small video player window in the bottom right corner shows a person speaking.

I actually what I write is scanf etcetera percentage d and velocity say where velocity is a variable alright. So, this AND in that case you is used to mean the address of this variable velocity.

So, in order to differentiate between this usage of AND and the logical operator logical AND is denoted as two ampersands, two ANDs.

(Refer Slide Time: 07:16)

The slide has a title 'Logical Operators' at the top. Below it is a bulleted list:

- There are two logical operators in C (also called logical connectives).
  - && → Logical AND
  - || → Logical OR
- What they do?
  - They act upon operands that are themselves logical expressions.
  - The individual logical expressions get combined into more complex conditions that are true or false.

A red handwritten note on the right side of the slide says "exp1 || exp2 || exp3" and below it "exp1& exp2 & exp3". A small video player window in the bottom right corner shows a person speaking.

Similarly logical OR means that some expressions say, I have I write it in an abstract way say expression 1 or expression 2 or expression 3. Now, this composite expression

will be true if any one of them either expression 1 or expression 2 or expression 3 is true. If any one of them is true then this entire thing will be true, if two of them are true then also it will return true, if all the three are true then also it will be true, but if none of them are true if none of them are true then it will not be true, then none of them will be true.

So, what is the difference between this logical OR and logical AND therefore? In logical AND if instead of this it was written like if, instead of this it was written like expression 1 and expression 2 and expression 3 this composite and expression would be true only if all these three expressions are true alright. So, that is logical OR.

Now, what do they do? They act up on the operands that are themselves logical expressions why logical expressions where from did I get logical expressions I got the logical expressions from relational operators.

(Refer Slide Time: 09:32)

The slide has a title 'Logical Operators' and a list of bullet points:

- There are two logical operators in C (also called logical connectives).  
    && → Logical AND  
    || → Logical OR
- What they do?
  - They act upon operands that are themselves logical expressions.
  - The individual logical expressions get combined into more complex conditions that are true or false.

A handwritten note is overlaid on the slide, showing a box around the text "t > 20" with an arrow pointing to the result "1/0".



For example, now I am writing some time is greater than 20 is a logical expression. What is this? This is a relational operator, but this expression is a logical expression. Why it is a logical expression? Because this will return only true or false nothing in between, so if t is greater than 20 if t is time or whatever value t might be if that is greater than 20 then it will return 1 or it will return 0. So, the logical connectives or the logical operators they are acting upon the operands themselves and connecting them. The individual logical expressions get combined into a more complex condition that are either true or false. We will see some examples.

(Refer Slide Time: 10:31)

- Logical AND

- Result is true if both the operands are true.

- Logical OR

- Result is true if at least one of the operands are true.

X	Y	X && Y	X    Y
FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	TRUE	TRUE	TRUE

So, logical AND the result is true if both the operands are true had or for two operands. If a three operands if all the operands are true, it should be connected as if all operands are true and logical OR the result is true if at least one of the operands are true, if at least one this is most important.

So, let us look at the truth table here X and Y any of them can have the value false or true. Accordingly we can have four combinations X false, Y false, X false, Y true, X true Y false and both X and Y are true. Now, if I carry out the logical AND then for all these cases say false false the result will be false, so 0. False and one true still it will be false because here I want all operands should be true. One is X is true Y is false the result will be false if both of them are true the result will be true. While in the case of OR X or Y will result in false if X is false and Y is false, but if X is false and Y is true will get a true because I am interested in getting at least one to be true if this is true and this is false then also true if both of them are true then also it is true.

So, I think it is clear to you what is meant by the logical operators and how we can combine logical expressions based on that.

(Refer Slide Time: 12:45)

The slide has a title 'Input / Output' at the top. Below it, there is a bulleted list about the `printf` function:

- `printf`
  - Performs output to the standard output device (typically defined to be the screen).
  - It requires a format string in which we can specify:
    - The text to be printed out.
    - Specifications on how to print the values.  
        `printf ("The number is %d\n", num);` The number is int
    - The format specification `%d` causes the value listed after the format string to be embedded in the output as a decimal number in place of `%d`.

A small video thumbnail of a teacher is visible in the bottom right corner of the slide area.

Now, we have seen three types of operations.

(Refer Slide Time: 12:55)

The slide contains a hand-drawn diagram with a brace grouping three items: 1) Arithmetic, 2) Relational, and 3) Logical operators. Red arrows point from each group to their respective outcomes: 1) leads to 'Arithmetic expressions', 2) leads to 'Logical exp', and 3) leads to 'Composite logical expressions'.

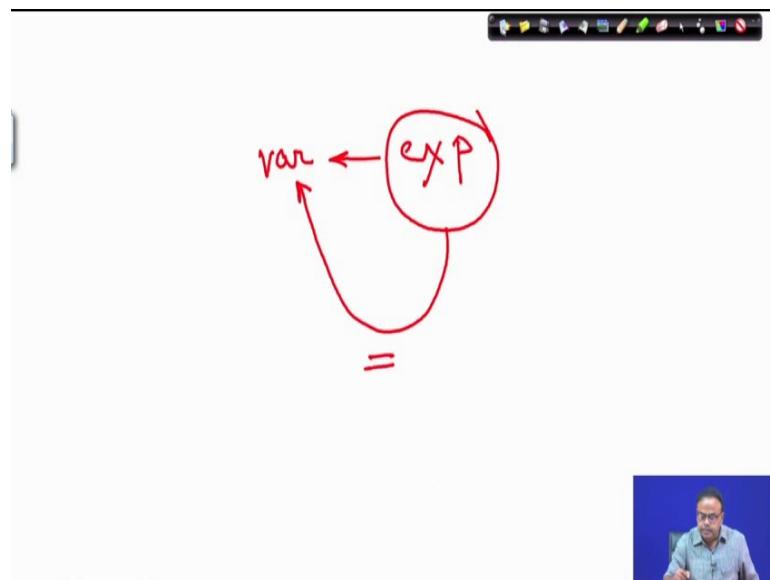
A small video thumbnail of a teacher is visible in the bottom right corner of the slide area.

So, actually operators, one is the arithmetic operators, the next is relational operators and the third one is logical operators. Now, arithmetic operations use of arithmetic operations lead to arithmetic expressions. The use of relational operators lead to logical expressions why logical expressions, they lead to true or false value right logical expressions and use of logical operators will combine and get more complex logical expressions complex or let me not write complex, let me write composite that communicates the meaning better

composite logical expressions, composite logical expressions. So, will see the use of this pretty soon when we will be looking at the control operators right.

Next, just to wrap up the things let us come to the input output statements.

(Refer Slide Time: 15:46)



By the way besides this besides the arithmetic expressions, logical expressions we had seen another type of expression those are assignment expressions or assignment operators. By that means, the left hand side is an expression and right hand side is a variable we assigned the result of the computation of the right hand sorry I just said the opposite the right hand side is an expression on this side is an expression. Here on the left side is a variable and we compute the expression and assign the value of that computation to this variable and this is the assignment operator.

(Refer Slide Time: 16:13)

The slide has a title 'Input / Output' and a bulleted list under it. The list describes the printf function, its purpose, requirements, and an example code snippet. To the right of the list is a video player showing a person speaking.

- printf
  - Performs output to the standard output device (typically defined to be the screen).
  - It requires a format string in which we can specify:
    - The text to be printed out.
    - Specifications on how to print the values.  
        printf ("The number is %d.\n", num);
    - The format specification %d causes the value listed after the format string to be embedded in the output as a decimal number in place of %d.
    - Output will appear as: The number is 125.

Now, besides that we have seen two other statements one is a printf statement we have seen that performs the output to the standard output device typically when we declare s t d i o dot h then by default it is taken as a screen. The other one and it requires a format string in which we can specify the text we printed out and the specifications on how to print the values like printf number is dash and that dash can be filled up by percentage d for; that means, the specification is that an integer can come here. And then you remember what this means this means, I am going to the new line alright and then followed by the number. The format specification causes the value listed to be embedded here I have discussed that that you can consider this format to be a place holder alright, the number is dash and how can this dash be filled out the dash since its percentage d some integer value can come and fill it up right we have seen that.

The other statement that we saw is scanf that is for reading the values.

(Refer Slide Time: 17:41)

The slide shows a list of bullet points about the `scanf` function, followed by four lines of C code. Red arrows and boxes have been added to the code to illustrate how it works. The first line `scanf ("%d", &size);` has an arrow pointing from the variable name `size` to a red box containing the address of `size`. The second line `scanf ("%c", &nextchar);` has an arrow pointing from the variable name `nextchar` to a red box containing the address of `nextchar`. The third line `scanf ("%f", &length);` has an arrow pointing from the variable name `length` to a red box containing the address of `length`. The fourth line `scanf ("%d %d", &a, &b);` has arrows pointing from both variable names `a` and `b` to a single red box containing the addresses of `a` and `b`.

- `scanf`
  - Performs input from the standard input device, which is the keyboard by default.
  - It requires a format string and a list of variables into which the value received from the input device will be stored.
  - It is required to put an ampersand (&) before the names of the variables.

```
scanf ("%d", &size);
scanf ("%c", &nextchar);
scanf ("%f", &length);
scanf ("%d %d", &a, &b);
```

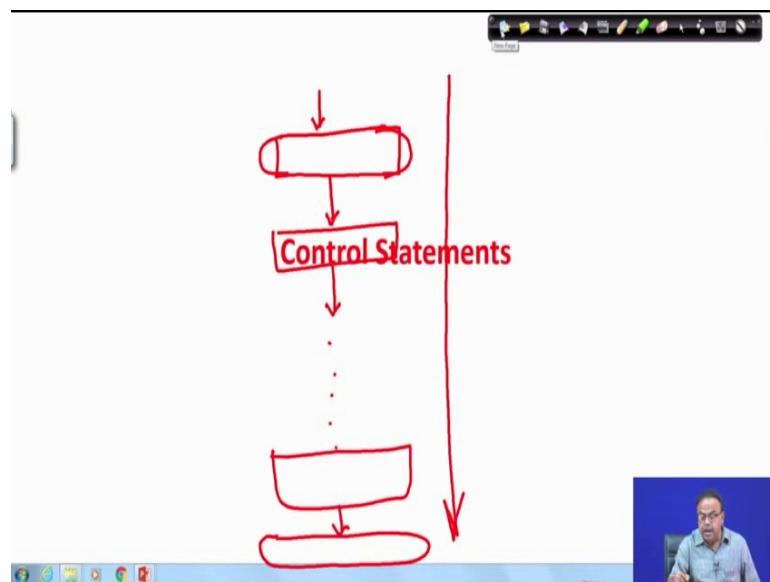
55

So, it performs input from standard in input device, normally by default it is a keyboard and then it also requires a format string and list of variables like it is required to put an ampersand before the names of the variables, we have also explained why that is so. The reason is that this ampersand essentially means the address of that variable where the value that is being read will be put.

So, here are some examples, `scanf` percentage d and size; that means, what that I am reading in a variable size, size is a name of a variable and in which I am putting in some integer value alright. Similarly next char say is a say this one is a character variable. So, I am specifying that only a character can come in here and that is why I have put in the specification percentage C. Percentage f means some floating point number will come here, but in all these cases this ampersand means the address of the corresponding variables alright. Here percentage d, percentage d means sequentially I am going to read two integer variables a and b. So, all these we have seen and you will be best learning this by practicing it time and again and we will see a number of examples and in this course there will be quite a few assignments which you will have to do.

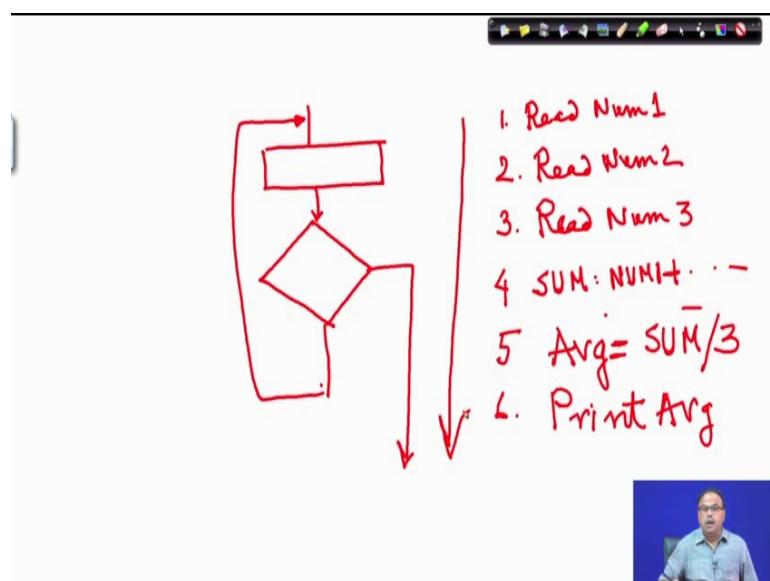
Next, we will move to, next we will move to a new topic which is a control structures and control statements. We have seen if we recall in a flowchart, let us go back to the flowchart where we have got some computation statements where we are doing some computations.

(Refer Slide Time: 20:27)



And we usually carry out one statement after another right that is how we do and in that way we will go on till the end of the program. For example, read number one, read number two etcetera divide add the numbers and divide the numbers to get the average. So, when we computed the average it was something like this and at the end we did some printf and in the meanwhile they were some reading the numbers these were some of them were input some were computations right. But it was a complete sequential thing.

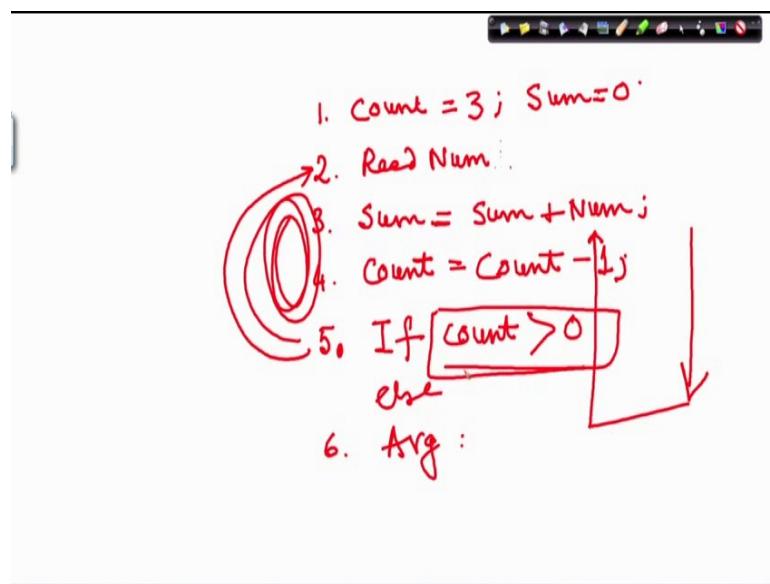
(Refer Slide Time: 21:45)



But if you recall the other type of, the other type of flowcharts that we had seen there we had started we did some computation and then we took some decisions and based on the decisions I have sometimes gone back to the earlier operation that earlier thing I had done and otherwise I would have followed this path. Typically in the examples that I we had seen we just see the pseudo code if I write say for computing the average of three numbers I will be read num 1, read num 2, read num 3 and then sum equals to num 1 plus num 2 like that I add them and then I compute the average to be sum divided by 3 right. So, and then we print the sum print the average.

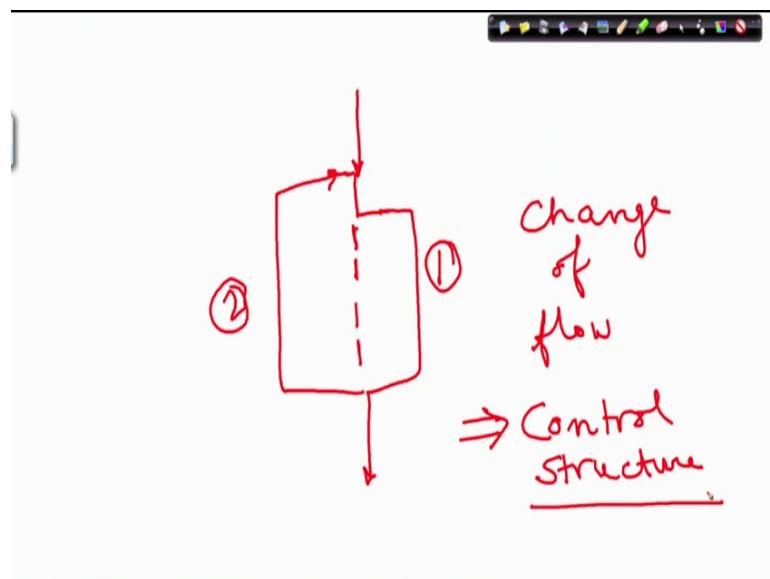
So, this entire thing is going in a sequence alright. Just one after another one after another no change in the path.

(Refer Slide Time: 23:37)



But in this case for example when we try to if you recall when we are not writing the this program in this form of pseudo code instead I initialise a count to be 3, then read number 1, sum and initially count here and say sum was 0, assign 0 and then sum is let me just make it make it just num I am reading one number and sum plus num and then count I decrement. So, I may count to be count minus 1 because I have, one I have already read. Now, I check if count is greater than 0 then what I do in my flowchart, I go back to reading the number again. Otherwise, else I come to the computation of the average right. Otherwise, if as long as count is greater than 0 I am continuously doing this thing right this steps.

(Refer Slide Time: 26:11)



So, here at this point you check that I am looking at the value of count and depending on the value of count I am deciding whether I will be going in this direction or I will go back and change the direction of the flow. So that means, as if the at this point the execution is undergoing a decision making to decide which part it will go through this path or this path.

As if you are therefore, you are controlling the flow of the program either in this way or you change the path, we will see that we can change the path in two different ways. One is that sequentially I am coming and I may go ahead I can skip some of the operations and I can go jump forward that is also a change of the path. This dotted line is showing the normal sequential execution or could be that some here I can go back to another path. So, these are two types of change of flow, change of flow that is that can be resulted in and that can be resulted in using the control structure.

So, we will soon go into the details of the control structure and see how such control structures or change of the sequential flow can be achieved in any programming will exemplify as we are doing for all others examples using the constructs of C, but that does not mean that it is restricted only to C. It can be, there are similar control constructs for other languages as well. We will come to that in the next lecture.

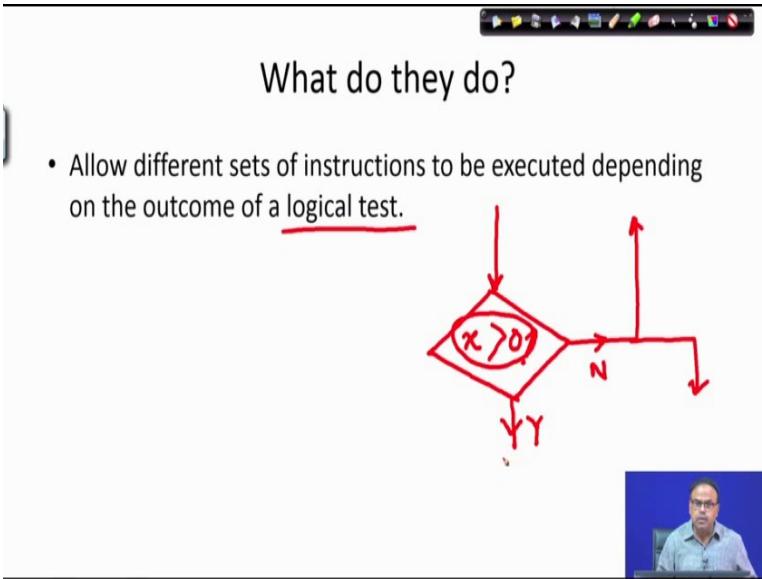
**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 14**  
**Use of Logical Operators in Branching**

In the last lecture, we had an idea of what control structures are. Control structures are some language structures which by which we can change the flow of control which normally is sequential that means, one executed after another.

Now, in the language we just as we have statements arithmetic expressions logical expressions all those are statements, in our language c language we also have some control statements for achieving the control structures.

(Refer Slide Time: 01:02)



What do they do?

- Allow different sets of instructions to be executed depending on the outcome of a logical test.



What they do? As I have said allow different sets of instructions to be executed depending on the outcome of a logical test. So, if we go back to the flowchart, we are coming in a sequential flow, at this point we encounter decision point and at this decision point we carry out some logical tests for example, is  $x$  greater than 0 yes or no. If it be yes there will be some flow and if it be no there will be some other flow all right some of not necessarily backward it can go somewhere else etcetera all right. So, based on the logical test; that means this test which will result in a logical value and what are the logical values? Logical values are either 0 and 1 either true or false.

(Refer Slide Time: 02:09)

What do they do?

- Allow different sets of instructions to be executed depending on the outcome of a logical test.
  - Whether TRUE or FALSE.
  - This is called branching.



So, whether true or false that is a; this is called branching. So, again whenever I am coming to a decision box and from there I am branching out, either in this direction or in some other direction all right. So, this is a branching branch point. So, that is change of the control flow.

(Refer Slide Time: 02:37)

What do they do?

- Allow different sets of instructions to be executed depending on the outcome of a logical test.
  - Whether TRUE or FALSE.
  - This is called branching.
- Some applications may also require that a set of instructions be executed repeatedly, possibly again based on some condition.
  - This is called looping.



Some applications they also require that a set of instructions be executed repeatedly, they will be going they will be executing as in the case reading the numbers c numbers. I will be repeating reading a number adding that with some I mean decrementing count

checking whether count is equal to 0, and then again go back and read the number and this slope will continue based on some condition. How long in the earlier example how long I will where we doing that based on what condition? The condition was whether count is 0 or not. As long as the count is not zero we are going on doing this. So, this is also known as looping doing the same repeatedly, this is also known as looping. So, branching or if we go back to the earlier one then it is called a loop.

(Refer Slide Time: 03:48)

The slide has a title 'How do we specify the conditions?' and a list of operators:

- Using relational operators.
  - Four relation operators:  $<$ ,  $\leq$ ,  $>$ ,  $\geq$  (L.E.)
  - Two equality operations:  $=$ ,  $\neq$  (LE) || (LE)  $\neq$  (LE)
- Using logical operators / connectives.

At the bottom right of the slide area, there is a small video player showing a person speaking.

Now, do we specify the conditions? By now I am sure you have get how we do that. We specify the conditions using relational operators or there are four relational operators you have seen; less than, less than equal to, greater than, greater than equal to. I have said not equal to also, but not equal to can also be seen as the negation of the equality operator. The equality operator is this double equal to sign and not equal to is exclamation mark for the equal to now. So, using this relational operators, we will get using one single any of these one single relational operator, we will get one logical expression and using logical operators and connectives, we will be able to connect a number of logical operations logical expressions and logical another logical expression. In that way we can carry out we can generate the condition using this.

(Refer Slide Time: 05:32)

How do we specify the conditions?

- Using relational operators.
  - Four relation operators:  $<$ ,  $\leq$ ,  $>$ ,  $\geq$
  - Two equality operations:  $\equiv$ ,  $\neq$
- Using logical operators / connectives.
  - Two logical connectives:  $\&\&$ ,  $\mid\mid$
  - Unary negation operator:  $!$

$x = 5;$   
 $(x > 5) \rightarrow 0$   
 $!(x > 5) \rightarrow 1$

$! \xrightarrow{\text{NOT}}$   
 $!(L E) \rightarrow X0$

Now, two logical operators we have seen and or. Another unary negation operator we did not discuss earlier, but we are doing it now that is this means not all right. For example, suppose  $x$  is 5 and at some point I compute  $x$  greater than 5, the result of this logical expression will be false right this is false, but if I take the negative of this then this false will be negated and the negation of that will be taken. So, that will be one two; that means, what is the meaning of this, is it the case I can read it in this way, is it the case that  $x$  is greater than 5 is false is not true yes that statement is true. I can state it in another way is it the case that  $x$  is not greater than 5? Yes that is the case again I tell the first one that is more complicated is it the case that the statement is greater than 5 is not true that statement is true? That it is not true is true. So, that is not means whatever logical expression I have after this that will be negated. So, it is equivalent to not of the logical expression.

So, if this logical expression if this logical expression leads to this logical expression is only 1, then not of that will invert it and make it 0 or if this logical expression was 0 then this not of this will make it 1. So, this is the unary negation operation all right.

(Refer Slide Time: 08:08)

The slide has a title 'Examples' at the top. Below it, there are four logical expressions:

- count <= 100
- (math+phys+chem)/3 >= 60
- (sex=='M') && (age>=21) ←
- (marks>=80) && (marks<90)

A red arrow points from the text '(sex=='M') && (age>=21)' to a red underline that spans the numbers '80 — 89'. To the right of this underline is a small video thumbnail of a person speaking.

Let us see some examples count less than equal to 0 this is the logical expression it will be either true or false maths plus physics; that means, here maths is the marks obtained by a student in maths plus physics, plus chemistry divided by 3 is greater than or equal to 60 that means, the average of these 3 marks is greater than or equal to 60.

Either the person the sex of that person is male sorry and let us look at this if I am using a logical connective the person this will be true if the person is a male and he age is greater than or equal to 21. So, for a female of age 22 this will be false, for a male age 20 it will be false. A male age 25 it will true. Here again another one with this connective marks is greater than 80 and marks is less than 90 sorry marks is greater than equal to 80 and marks is less than 90; that means, what anything starting from 80 to 89, if the marks is between this then this will be true .

(Refer Slide Time: 09:39)

Examples

```
count <= 100
(math+phys+chem)/3 >= 60
(sex=='M') && (age>=21)
(marks>=80) && (marks<90)
(balance>5000) || (no_of_trans>25)
! (grade=='A')
```

Another one with an or operator balance is greater than 5000 or the number of transaction is greater than 25 say in a bank see situation for example, you are making a lot of transactions what is the transaction when you deposit a money that is a transaction, when you withdraw the money that is also a transaction. So, there are different transactions. So, what we are saying here either I have got 5000 rupees greater than sorry greater than 5000 rupees in my balance or I have done more than 25 transactions. If any one of them true suppose you have got 6000 rupees in your balance and number of transactions is 26, then also this is true. If the balance is 6000 and number of transactions you have done is only 20 then also it is true, we have already explained the role of or operator earlier ok.

Now, this if the grade of the student is a if a student has got grade a what will be the value of this logical expression true or false if the student has actually got the a grade all right. So, this one this part will be true, but my actual logical expression is this one which is a negation of that. So, it will be false because the person has grade a what does it means? It means that is it the case that the student has not got grade a.

(Refer Slide Time: 11:35)

count <= 100  
(math+phys+chem)/3 >= 60  
(sex=='M') && (age>=21)  
(marks>=80) && (marks<90)  
(balance>5000) | | (no\_of\_trans>25)  
! (grade=='A')  
! ((x>20) && (y<16))

The truth table below shows the values for the variables x and y:

x	y	x > 20	y < 16	x > 20 & y < 16	Not (x > 20 & y < 16)
0	0	0	1	0	1
0	1	0	0	0	1
1	0	1	1	1	0
1	1	1	0	0	1

Here is another little more complicated  $x$  greater than 20 and  $y$  less than 16, but the way we should evaluate this is, I will evaluate each of them separately suppose  $x$  is greater than 20 suppose 21. So, this is true and  $y$  is 15. So, this is true. So, both these together is true then I take this one with this and so, the result will become false let me clear it again suppose  $x$  is 21. So, this part is true then I compute this  $y$  is 17, this part is false therefore, these two are connected by this ampersand; that means, logics are not I am sorry very sorry it should it is not ampersand double and; that means, and operator by logical and. So, these two will result in though the logical and it will result in 0, but my actual expression is the negation of that. So, the result will be 1 clear.

(Refer Slide Time: 13:09).

The conditions evaluate to ...

- Zero
- Non-zero

```
graph TD; Test{Test} --> Top[ ]; Test --> Bottom[ ];
```

Video thumbnail of a teacher.

So, that is how we carry these are the examples of evaluation of logical expressions and logical expressions. So, each of them each of these logical expressions can serve as a condition in our control structure in our this loop in our this diamond box, decision box where we test or let us call it the test box in that any of these sort of logical expressions can reside and the result of that can be either 0 or non-zero false or true ok.

Accordingly we will branch out in either this direction or in some other direction.

(Refer Slide Time: 14:00)

The conditions evaluate to ...

- Zero
  - Indicates FALSE.
- Non-zero
  - Indicates TRUE.
  - Typically the condition TRUE is represented by the value '1'.

Video thumbnail of a teacher.

So, 0 indicates false and non 0 indicates true typically the true is reprinted by the value one in most of the machines.

(Refer Slide Time: 14:14)

The slide has a title 'Branching: The if Statement' and a bulleted list of points about the diamond symbol (decision symbol). The list includes:

- Diamond symbol (decision symbol) - indicates decision is to be made.
  - Contains an expression that can be TRUE or FALSE.
  - Test the condition, and follow appropriate path.
- Single-entry / single-exit structure.

Below the list is a flowchart diagram showing a diamond shape with two exits. A red arrow enters the diamond from the bottom-left. It has two exits: one going up-right and one going down-right. Both of these exits lead to a merge point, which then leads to a final exit at the top. To the right of the flowchart is a small video window showing a person speaking.

This is something that we have already seen, the decision symbol in the flowchart indicates the decision to be made it contains an expression that can be true or false, test the condition and follow the appropriate path that is how we will do is a single entry single exit structure; that means in this diamond box we will enter through any one point and we will exit from only one point either this or this, simultaneously we cannot come out of this because the thing cannot be true or false at the same time therefore, any one exit will take.

(Refer Slide Time: 15:09)

Branching: The if Statement

- Diamond symbol (decision symbol) - indicates decision is to be made.
  - Contains an expression that can be TRUE or FALSE.
  - Test the condition, and follow appropriate path.
- Single-entry / single-exit structure.
- General syntax:  
`if (condition) { ..... }`

*if ( $x \geq y$ )  
printf ("x is large")*

The general syntax, syntax means what? Syntax means the grammar; the exact grammatical structure that we should write is if condition then I carry out some operations. You have seen in an earlier lecture when we were comparing two numbers x and y and would be printing x is large is x is greater than y then what did we write we wrote something like this, we wrote if x is greater than equal to y, print f what we printed y x is greater than equal to y, x is large or not in this form we wrote down that x is large here all right. So, if this condition is true then I will be doing this print operation, x is large then black slash end I am not being able to write very clearly here.

(Refer Slide Time: 16:48)

Branching: The if Statement

- Diamond symbol (decision symbol) - indicates decision is to be made.
  - Contains an expression that can be TRUE or FALSE.
  - Test the condition, and follow appropriate path.
- Single-entry / single-exit structure.
- General syntax:  
`if (condition) { ..... }`

*if ( $x \geq y$ )  
{ printf ("x is large\n"), }*

So, I try again if x is greater than y or greater than equal to y. So, this is the condition you see this is this is the reserved word in c end the condition. If the condition is true then this is followed by a statement. Now if it be a single statement I do not need to give this curly bracket, but just I am giving it just to keep the parity, here I write print f x is large then black slash end right and we end this statement ok.

So, if this condition is true, then this statement will be executed that is my syntax otherwise some other statement will be executed. Now this is something which takes some times to settle down. So, we will explain it repeatedly to some extent.

(Refer Slide Time: 18:15)

Branching: The if Statement

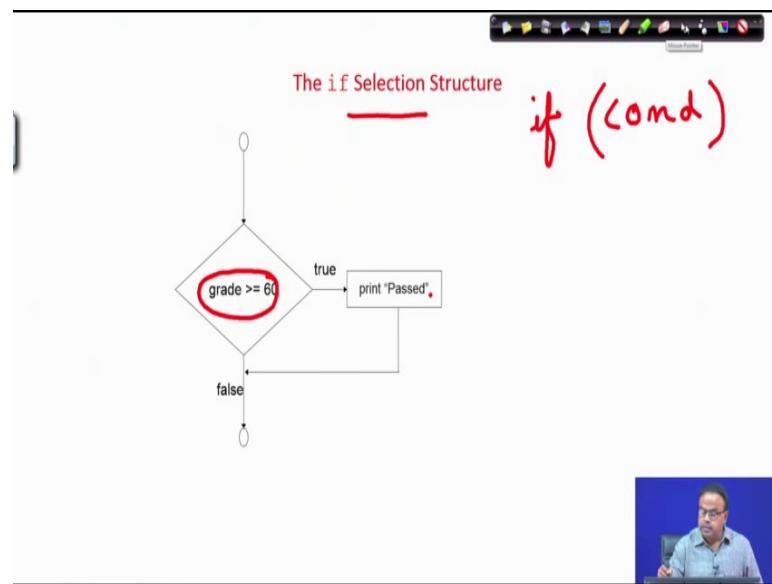
- Diamond symbol (decision symbol) - indicates decision is to be made.
  - Contains an expression that can be TRUE or FALSE.
  - Test the condition, and follow appropriate path.
- Single-entry / single-exit structure.
- General syntax:

```
if (condition) { ..... }
```

  - If there is a single statement in the block, the braces can be omitted.

If there is a single statement in the block this braces can be omitted I am sorry this braces can be omitted.

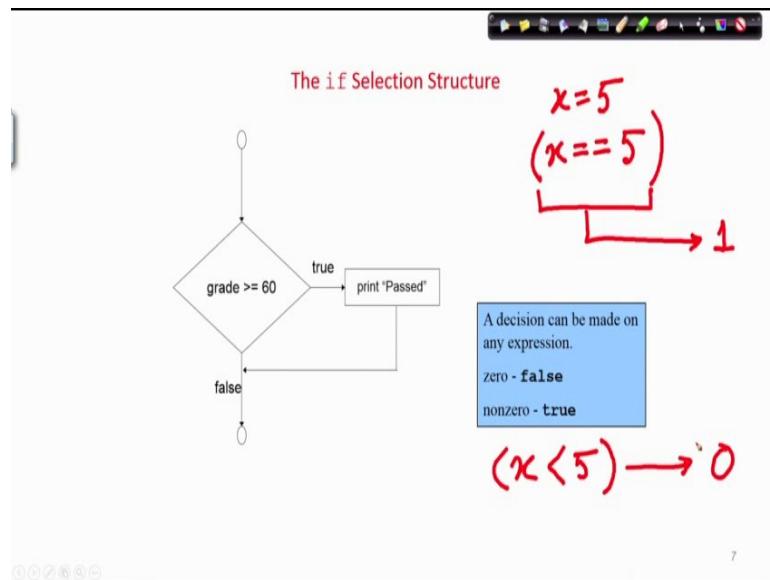
(Refer Slide Time: 18:19)



Now, let us come to this structure. If statement first of all we are looking at the first and very important control statement in C that is the if statement, if some condition. So, we come here the structure is we carry out some tests on some condition grade is greater than equal to 60 if it is true, then we do something I print past otherwise I would write something else. So, this is a selection structure what is it selecting? It is selecting which path should I go all right.

Now, you see here we have come we are coming to this path, if this here I carried out a test if this test is true I do this do this execute this statement, and come back and continue not come back and then continue to this part. If this statement is not true; that means, if this test fails then I will not take this I will not select this path, I will continue in the path which in which I was going right. So, that is why this is often known as a selection structure.

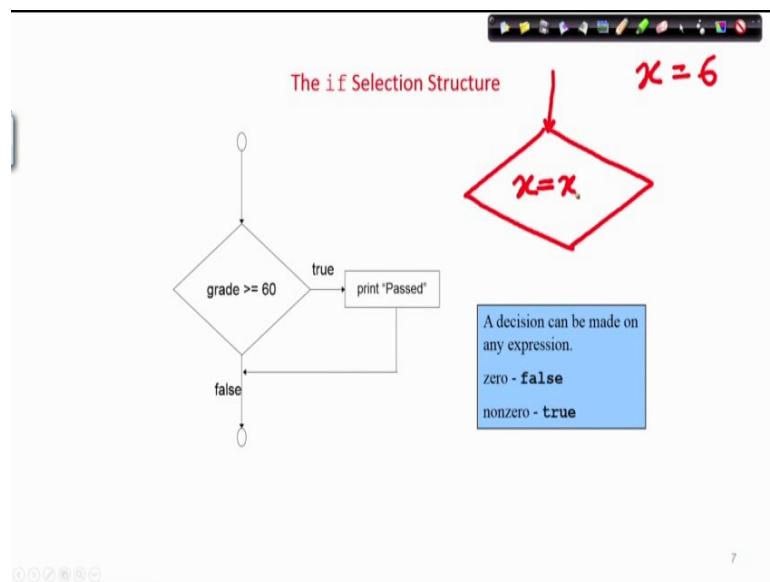
(Refer Slide Time: 20:06)



So, a decision can be made on any expression. Here you see only thing that we are trying to do is test whether some condition is true or false. Now we if the result of an expression is 0 then we take it to be false and if it be non 0 then we will take it to be true. If I evaluate something to be true like say for example, if  $x$  was 5 and I carry out  $x$  is  $x$  equal to 5, this logical expression we will evaluate to true; that means 1 that means, it is non zero.

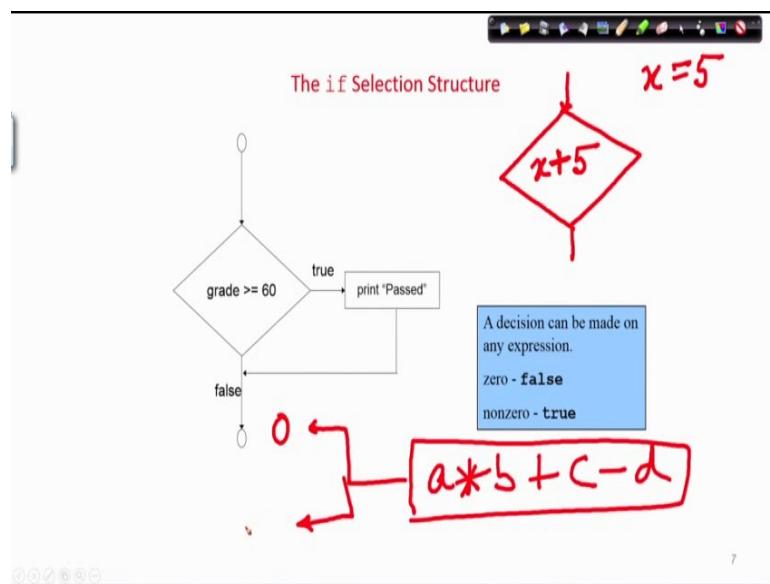
If I had written  $x$  less than 5 then it would have resulted in false it would be 0. Now suppose I simply do something different I write in this decision box I am coming like this.

(Refer Slide Time: 21:40)



And inside the decision box I write and  $x$  was 6 say I do not write a logical expression here I write an arithmetic expression  $x$  or let me  $x$  was 5, I carry out some expression  $x$  plus 5.

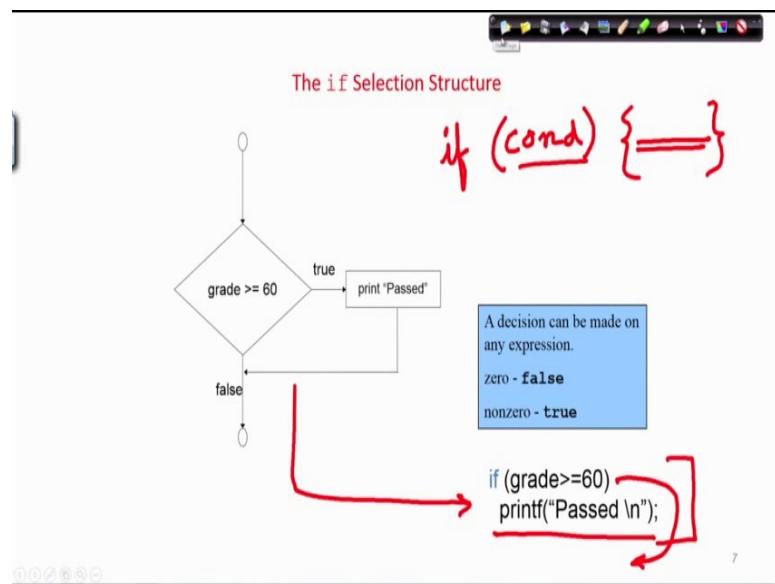
(Refer Slide Time: 22:13)



What will be the result of this expression be? It will be ten that is it is non 0 then also it will be taken as true although we will we do not need it all the time you need not bother about it, but in general a decision can be made on any expression. Typically and most conveniently we will be using logical expressions, but if I carry out an arithmetic

expression something a times b plus c minus d, then also this will give me a value and this value gives 0 means it is false and not 0 then it will be taken as true however, this is just a matter of detail you need not get too much bothered about it right now, we will try to keep as much as possible relational expressions or logical expressions inside this text blocks.

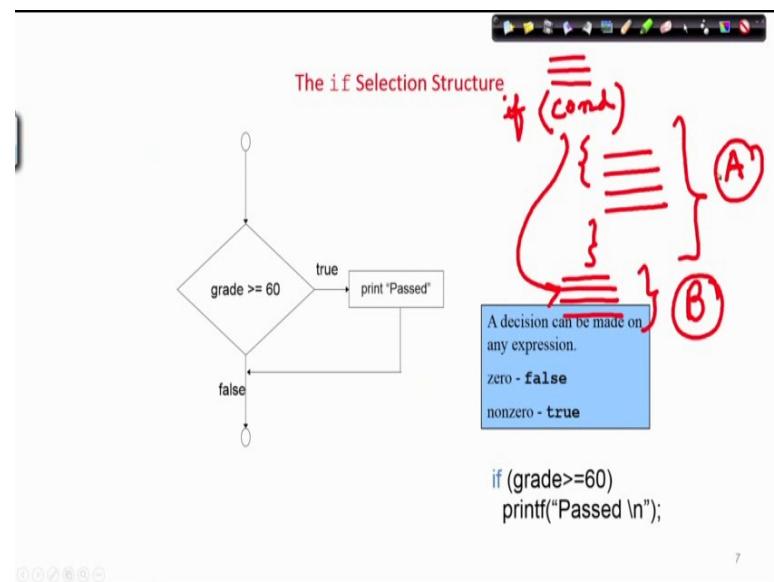
(Refer Slide Time: 23:45)



So, here if grade is greater than or equal to 60 being passed 10; that means, this is the equivalent of this structure that we have shown here. This flowchart can be simply written as this statement I think probably you will be able to read it now, if has been given in a different colour because this is what we have introduced here this if is a use of if the selection structure followed by a condition if the condition is true then this statement will be executed.

If the condition is false, then this statement will not be executed we will bypass this statement. Now please remember this, when I write if some conditions some statements this statements will be executed only if this condition is true, I write it in a different way now.

(Refer Slide Time: 24:49)

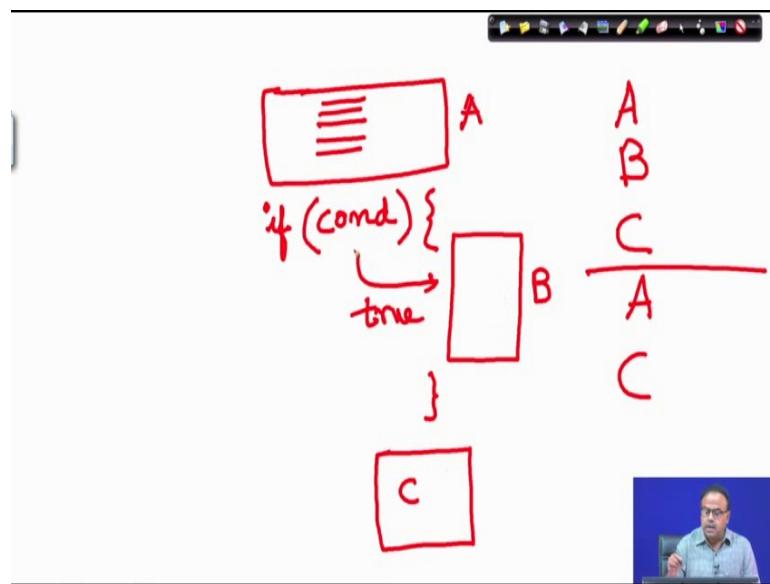


Say for example, I have got a number of statements, some statements are there and then here there is an if statement. If condition then some statements here may be more than one statements are in a brace. Now if the condition and after that there are other statements.

Now, if the condition is true, then only these statements will be executed lets name this statements to be a then this block a will be executed this block will be executed. If this condition is false then this block A will be bypassed and on false I will come here and come to so, this block b. So, if the condition is false then the block B will be executed, if the condition is true then this block A will be executed all right.

Now, suppose let me explain it in a little different way I am drawing another diagram.

(Refer Slide Time: 25:59)



Now, here is some block of statements, let us call the block of statements to be A and then I have one statement, if condition n whatever the condition is, then within parentheses there is a block of statements B. And then there is a block of statements C. So, the flow of I mean the flow will be that first A will be executed then if the condition is true, then if the condition is true then B will be executed after that what will happen? After that again we will come here and C will be executed.

Now, if the condition were false condition were false, I just computed A after that what would have computed condition will be checked B will not be evaluated not be executed because condition is false after that C will be executed. So, please note that in this structure I am computing I am executing C either way, now the choice of whether b is being done or not is dependent on the condition. So, this gives you an idea of procedure if the condition is false then also this part will be executed. So, here we will conclude today with this example, small example program lets try to understand each line of them.

(Refer Slide Time: 28:07)

```
#include <stdio.h>
main()
{
    int a,b,c;
    scanf ("%d %d %d", &a, &b, &c);
    if ((a>=b) && (a>=c))
        printf ("\n The largest number is: %d", a);
    if ((b>=a) && (b>=c)) X
        printf ("\n The largest number is: %d", b); X
    if ((c>=a) && (c>=b)) X
        printf ("\n The largest number is: %d", c); X
}
```

We start with include stdio.h I hope you have not forgotten that any C program must have a main program main function and here is the main function. Inside the body of the main function what is being done I have declared what is being done here, I have declared 3 variables a b and c. Then what am I doing here observe this line scan f percentage d percentage d percentage d and a and b and c. So, I am needing 3 variables a b c ok.

Now, what am I doing here, I am evaluating an if condition if a is greater than b and a is greater than c. So, you understand that if a is a number b is a number c is a number a is greater than b and a is also greater than c; that means, a is the largest number. So, then I change this, I change this only if this condition is true otherwise I will do something else. Now suppose a is greater than b suppose a is 20, b is 10 and c is 5. So, this one will be true then the largest number is a that we printed, next statement will come to is another if statement.

What is being done here? If b is greater than equal to a and b is greater than or equal to c b is not greater than equal to a therefore, this does not evaluate it to be true therefore, this statement it does not execute. Next I come to this statement I am showing this I am following this flow, but only branch that I took is if this condition was true, then I printed this in this particular case of data in this case of data for some other case should be very

different now if c is greater than a not true. So, this is false this will not be executed. So, the only thing that we printed is a largest number is a.

(Refer Slide Time: 30:43)

The slide shows a C program with handwritten annotations. The code is:

```
#include <stdio.h>
main()
{
    int a,b,c;
    scanf ("%d %d %d", &a, &b, &c);
    if ((a>=b) && (a>=c))
        printf ("\n The largest number is: %d", a);
    if ((b>=a) && (b>=c))
        printf ("\n The largest number is: %d", b);
    if ((c>=a) && (c>=b))
        printf ("\n The largest number is: %d", c);
}
```

Handwritten annotations on the right side of the code indicate variable values:

- $a = 10$
- $b = 20$
- $c = 5$

Red arrows point from the annotations to the corresponding lines of code where conditions are checked. The first condition  $(a \geq b) \&& (a \geq c)$  has a red X below it, indicating it is false. The second condition  $(b \geq a) \&& (b \geq c)$  has a checkmark below it, indicating it is true. The third condition  $(c \geq a) \&& (c \geq b)$  has a red X below it, indicating it is false.

Now, if for example, a was 10, b was 20 and c was 5 then what would have happened let us see here a is greater than or equal to b I try to test this fail. So, this will not be executed. So, I come here right and then I test this b is greater than a right and b is greater than c that is also right. So, this condition passes.

So, then I execute this I print larger number is b. I print that and then I come to the next statement what happens in this statement? C is greater than or equal to a false. So, the only thing that is printed is the largest number is b all right. So, we stop here we will continue our explanations in the next lecture further, we will have we will need to discuss it and also try it a little bit more, because from this point onwards our logic and logical constructs are becoming started becoming just a little more complicated, but is very interesting and it is very easy to understand, if you just try to think about it logically.

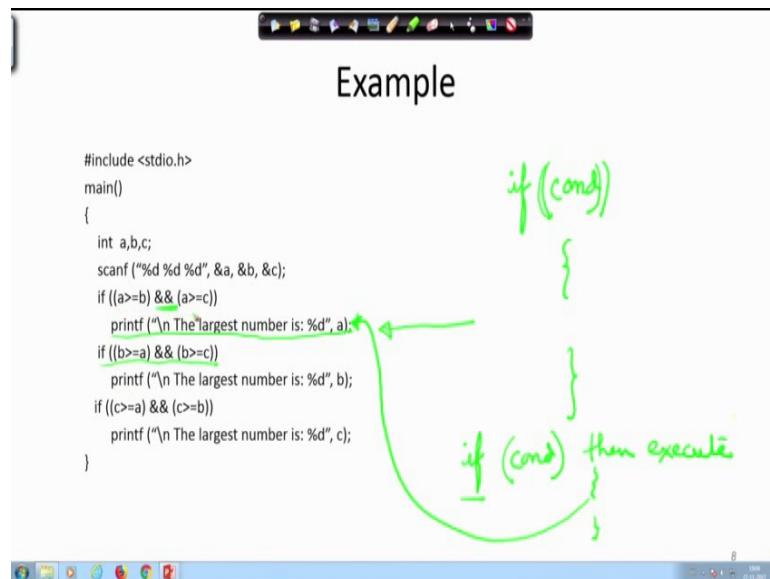
Thank you very much.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 15**  
**Branching : IF – ELSE Statement**

We are looking at the IF selection structure where the structure was something like if some condition and then there were some statements, and here is an example of what we are looking at.

(Refer Slide Time: 00:24)



The screenshot shows a Windows desktop environment with a Notepad window open. The title bar of the window says "Example". Inside the window, there is a C program:

```
#include <stdio.h>
main()
{
    int a,b,c;
    scanf("%d %d %d", &a, &b, &c);
    if ((a>=b) && (a>=c))
        printf ("\n The largest number is: %d", a);
    if ((b>=a) && (b>=c))
        printf ("\n The largest number is: %d", b);
    if ((c>=a) && (c>=b))
        printf ("\n The largest number is: %d", c);
}
```

Handwritten annotations in green ink have been added to the code:

- A bracket on the right side of the first if block is labeled "if (cond)".
- A bracket on the right side of the entire if-else structure is labeled "if (cond) then execute".
- An arrow points from the handwritten "if (cond)" label to the opening brace of the first if block.
- An arrow points from the handwritten "if (cond) then execute" label to the closing brace of the entire if-else structure.

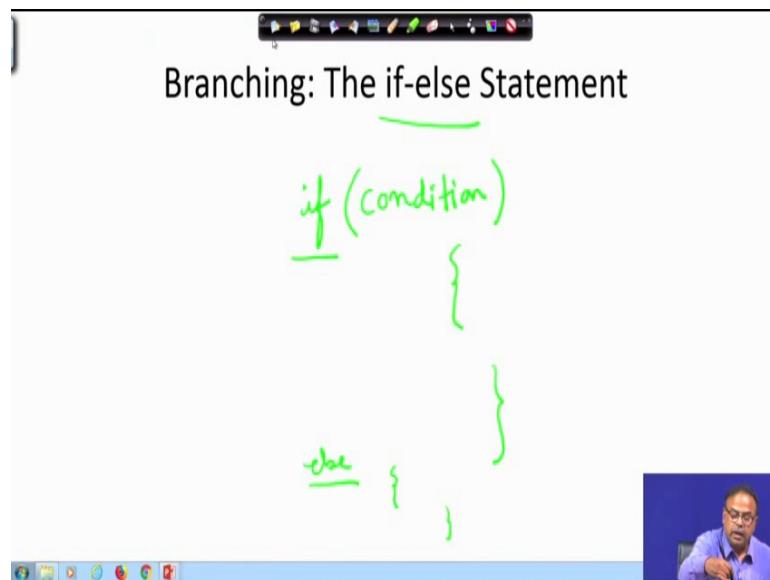
So, here you can see that if a is greater than b and also greater than c here is a logical connective. So, then this statement will be executed. So, in English we can just call it like if the condition is true then if the condition is true, then execute a set of statements. So, in this case the set of statements is just only this printf. Otherwise if this condition is true then this statement will be printed. So, this is what we discussed in the last class.

Now, one thing for those who will be writing programs in C you should remember that this condition is always within a parentheses and within that parentheses there can be a composite expression which is here you can see there is a logical expression, here is another logical expression and here is a composite logical expression joining them by AND or it could be by OR etcetera. So, in that way we can form the condition statement

and if the condition statement is true then these things will be executed. We will see more examples as we go ahead.

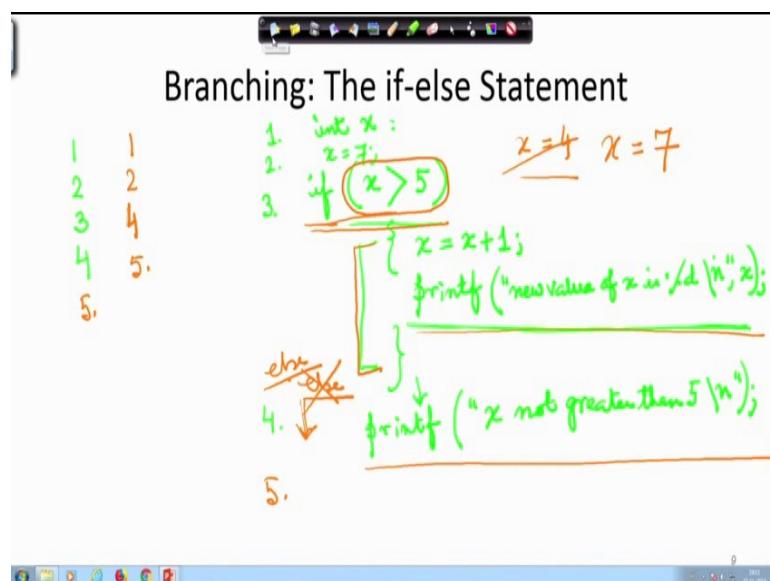
So, let us look at a little more different, a little more extensions of the structure.

(Refer Slide Time: 02:08)



Here we are going to see the if-else structure what we actually want to mean is if a condition is true then will be doing these things and if the condition is false then I will do these things. Let me clarify it a little bit more.

(Refer Slide Time: 02:43)



Suppose I write if x is greater than 5 increment x by 1 and printf “new value of x is percentage d back slash n x” alright. That means, if x is greater than 5 suppose x is an integer I am assuming that x is an integer here. So, somehow somewhere up here I have written int x equal to for int x just int x and later on I have assigned x to be 7 alright.

Now, when I encounter this condition this condition always true then I come to this set of statements because this condition is true. So, what will be the value of x now? x was 7 it is greater than 5 therefore, x will be 8 and what will be printed new value of x is 8 that is what will be printed. Now, suppose I write printf x not greater than 5. What I intent to do is if this condition is true then this will be printed otherwise this will be printed that is what my intention is, that is what I want to do. But the way I have written it will actually do something different. You see when the execution will be done you know the execution is normally done in a sequential manner. So, here we will come x is greater than 5, x is 7, so x is greater than 5 this thing will you printed and will come out and come to this next statement. Let me number these statements say this statement was 1, this was 2, this was 3. Now, see I am calling this entire if statement to be a single statement; so 3 and then 4.

So, normally 1 will be executed, then 2 then 3 now 3 will be executed because x is greater than 5. So, this condition is true and then what will be executed 4. So, what will be the output what will be written? Here new value of x is 8 will be printed and here when it 4 is executed again you printed x is not, x not greater than 5, but that was not my intention. So, in order to avoid that I can write a new statement here there was an if, if I write here else this. That means what? If this condition is true then print this otherwise; that means, if this condition is false then do this and then number 5 can be something else and that will be executed. So, what will be the execution? Suppose x is equal to 4, let us try to write down what is execution sequence then 1, then 2.

Now, x is equal to 4 therefore, this condition will be false this condition will be false therefore, this statement 3 will not be executed, this part of 3 will not be executed then we will come to this else statement; that means, if the condition is false then I am coming over here. And so 4 will be executed 4 will be executed, 3 will not be executed completely and then 5 will be executed.

But if I had not put this else then what would have happened in this case if x is 4, in that case this will not be this will not be true only this statement will be printed fine there will not be much problem, but suppose x is equal to 7. And if this else is not there then what will you print it x is 8 x not greater than 5. But if I put the else here if I put the else here I am fairly repeating a little bit more because you lot of students face difficulty in this structure. So, if I put else here in that case this will be x is 7. So, this will be printed and at this else this condition was true. So, this is this part will not be executed, this part will be executed only if the condition was false. So, this contradictory situation would not occur alright.

(Refer Slide Time: 08:43)

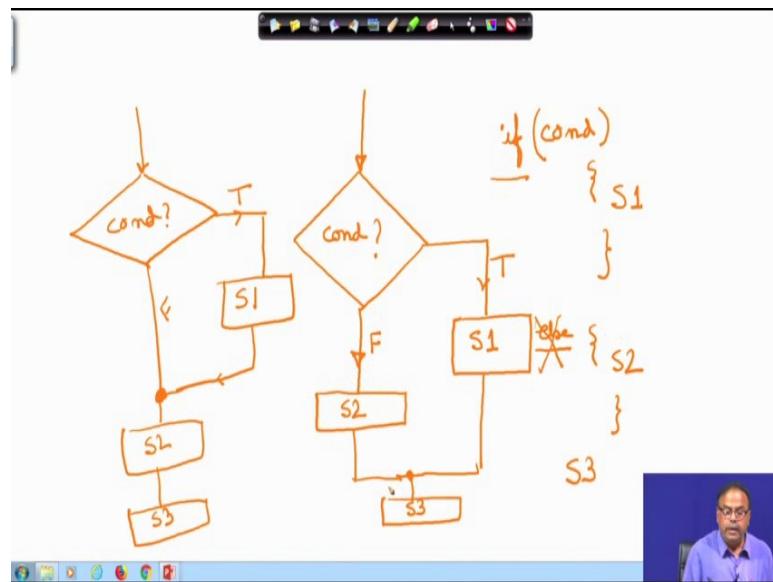
The screenshot shows a presentation slide with a title and a bulleted list. The title is "Branching: The if-else Statement". The list contains four items:

- Also a single-entry / single-exit structure.
- Allows us to specify two alternate blocks of statements, one of which is executed depending on the outcome of the condition.
- General syntax:



So, let us look at some examples of this if statement if-else. It is also a single entry single exit structure it allows us to specify to alternate blocks of statement one of which, one of the which is executed any one of them depending on the outcome of the condition.

(Refer Slide Time: 09:16)



Essentially what we are trying to say is that I will have a single entry point to a decision box and I will check for some condition I will test that particular condition if that condition is true then true then I will do some state set of statements let us call that S 1, otherwise if it is false then I will carry out some S 2 alright and then I will be meeting may be at some other point which is S 3.

So, this structure I can write as if condition then I do I have name I am not writing the statements I am just writing S 1 else S 2 and then S 3. So, my flow will be either this way and S 3 or this way and S 3. If I had not put this else how would this diagram look like? You have seen this diagram, now if I had not put this else how would the diagram look like. The diagram would look like if here I have got the condition block decision box true then I do S 1 and then I do S 2 and S 3. So, will come at this point if it false then I will also come at this point if it is true then also I will join at this point if this else was not there. But since this else is there then I just separate the out their point of joining alright will see more examples of this.

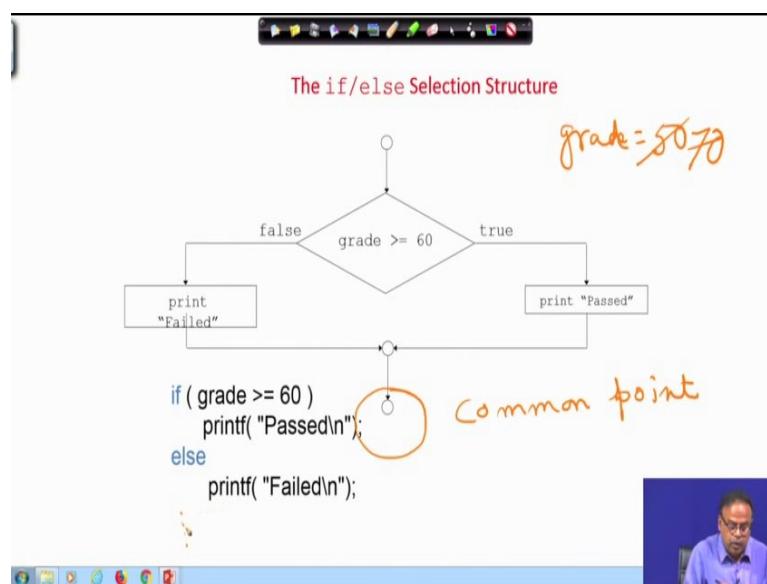
(Refer Slide Time: 11:53)

The slide has a title 'Branching: The if-else Statement' and a bulleted list of points:

- Also a single-entry / single-exit structure.
- Allows us to specify two alternate blocks of statements, one of which is executed depending on the outcome of the condition.
- General syntax:  
if (condition) { ..... block 1 ..... }  
else { ..... block 2 ..... }  
– If a block contains a single statement, the braces can be deleted.

So, the general syntax of this is if condition then block on 1, e block 1 is what I was calling S 1 the set of statements else block 2 the other statements alright. If a block contains a single statement then the braces can be deleted I am sorry if the block contains a single statement then this a start essential that I put them, but just for the sake of generality in most of the cases I will have more than one statement therefore, I am putting that in place alright.

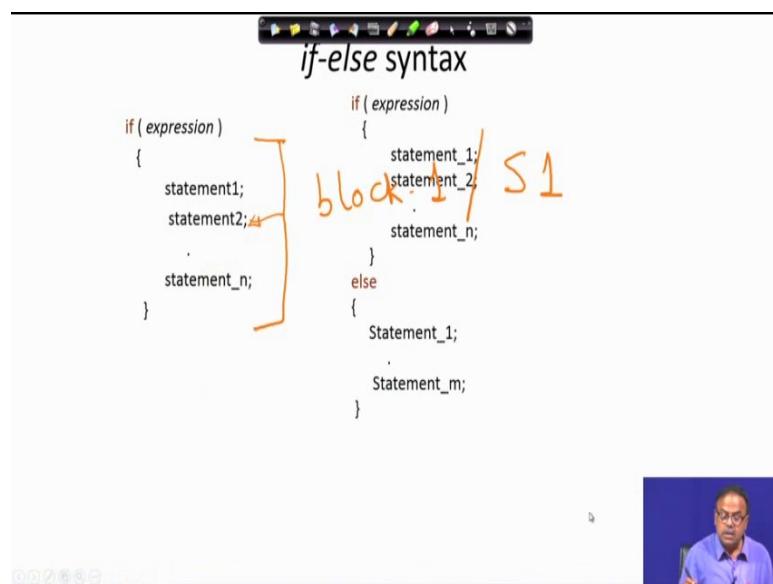
(Refer Slide Time: 12:40)



So, the diagram that I was looking at I was explaining that sort of diagram. Suppose the grade is 60, if the grade is greater than equal to 60 suppose somebody's grade is or marks is 50 then at this point the condition is false this part will be executed, if the grade is 70 that is this condition is true this part will be executed. Then I will come to this common point, this is a common point which will come.

So, this I can write as if grade is 60 see here there is only one statement. So, I did not need not put the braces here print sorry printf passed else printf failed and then whatever I had to do I will do it here and then whatever was following that will follow here. So, let us come to. So, I hope this is clear.

(Refer Slide Time: 14:03)



So, the syntax of if-else let us look at little bit more here I have said block 1, block 1 means or S 1 whatever you name it. This is block 1 or I had name it as S 1 the same set of statements where their number of statements each separated by a semicolon. Please note suppose there n statements here separated by a semicolon.

(Refer Slide Time: 14:50)

The diagram shows a computer screen with a presentation slide titled "if-else syntax". The slide contains pseudocode for an if-else statement and handwritten annotations in orange. The pseudocode is:

```
if ( expression )
{
    statement1;
    statement2;
    .
    .
    .
    statement_n;
}
else
{
    Statement_1;
    Statement_m;
}
```

Handwritten annotations include curly braces around the code blocks and labels "block 1" and "block 2" next to them. A small video window in the bottom right corner shows a person speaking.

Now, next I come to this if I want to do else also if expression statements 1 to n, else state these statements 1 to m. So, this is my block 2 and this is block 1. So, here we can see that based on this expression which is a condition evaluation if this condition is true I will carry out these statements else I carry out these statements. Note that these individual statements have got semicolon and this is a whole body of the else statement this is the whole body of the if statement. So, composite if this entire thing we call an if-else structure.

(Refer Slide Time: 15:39)

The diagram shows a computer screen with a presentation slide titled "if-else syntax". The slide contains pseudocode for an if-else statement and handwritten annotations in orange. The pseudocode is:

```
if ( expression )
{
    statement1;
    statement2;
    .
    .
    .
    statement_n;
}
else
{
    Statement_1;
    .
    .
    .
    Statement_m;
}
```

Below this, there is an example of C code:

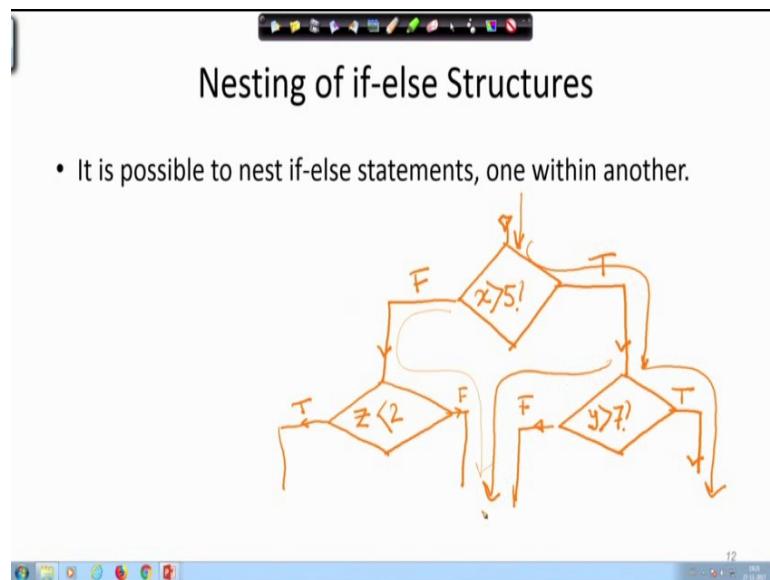
```
if (grade>=60)
printf("Passed \n");
else
printf("Failed\n");
```

A small video window in the bottom right corner shows a person speaking.

So, if grade is 60 printf passed else print failed.

Now, this next we are coming to a little more complication of the same thing it is the next thing of if-else structure.

(Refer Slide Time: 15:44)



Now, why this if-else what are we trying to achieve we are achieving branching. That means, we are carrying out a sequential flow we are carrying out a sequential flow and from there we make a decision box and then we branched either in this direction or in this direction.

Now, suppose here in this direction it is true and this is false again I come and make a decision. It can be, if x is greater than 5 then I come here and y is greater than 7 then I do something here it can be x is greater than 5 I follow this path, but y is not greater than 7. So, I follow this is true this is false I follow this false path. If x is not greater than 5 I come here and suppose I check z less than 2 true or false there can be 2 outcomes. So, suppose x is 4, not greater than 5 I will come through this path, now z is 3, z is not greater than 2, less than 2 therefore, I will follow this path. So, you see the path that we follow that is a branching that we do can have the path that we follow can have more than one decision box say x is greater than x is 7. So, here I was coming in this path then I follow this path because x is 7 this condition is true and I find that y is 8 so I follow this path alright, if y was 6 then I would have followed this path.

So, in the path that the program follows there can be more than 1 decision box. So, that is what we mean by nesting of if-else structures.

(Refer Slide Time: 18:05)

The slide has a title 'Nesting of if-else Structures' and a list of bullet points:

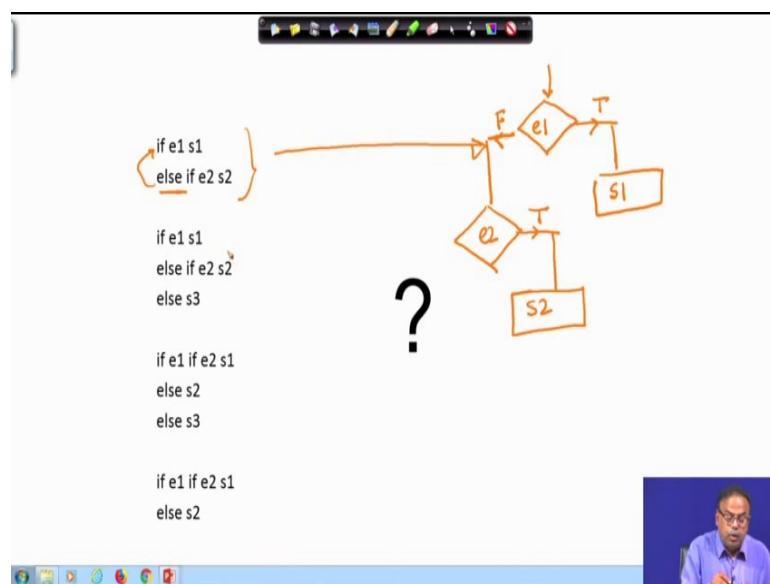
- It is possible to nest if-else statements, one within another.
- All if statements may not be having the "else" part.
  - Confusion??
- Rule to be remembered:
  - An "else" clause is associated with the closest preceding unmatched "if".

A small video window in the bottom right corner shows a speaker.

That means nest one if-else statement within another. Now, all statements may not have the else part. Now, rule to be remembered is an else clause is associated with the closest preceding unmatched if really confusing what do I mean by this.

Let us look at this how will this be executed.

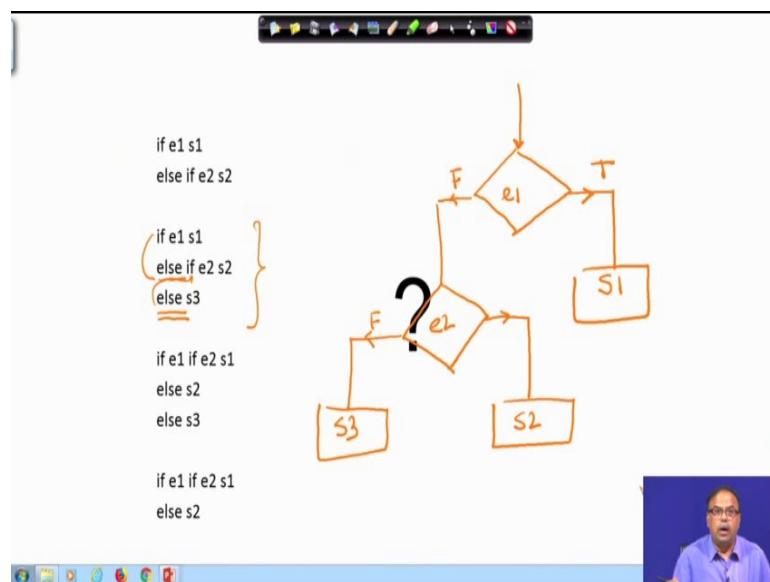
(Refer Slide Time: 18:30)



Here  $e_1$  means some expression, if  $e_1$  is true then  $s_1$  else if  $e_2$  then  $s_2$ . Now, with whom does this else go? This else certainly goes with this preceding if. If I just quickly draw this that flowchart corresponding to this how does it look like? I come here and x evaluation, evaluate the expression  $e_1$  on true I do  $s_1$ , on true I do  $s_1$ , on false here is false alright I come here and there is another if here. So, I again check  $e_2$ , if  $e_2$  is true then I do  $s_2$ . This statement actually means this graph. If  $e_1$  is true then I will carry out  $s_1$  otherwise else is there. So, this else is linked to this if then I do this one.

What about this? Let us erase this for a while and how would we interpret this one. Here you can see the two elses. With whom is this else paired? This else is paired with I have written it in a bad way, this else is paired with this if.

(Refer Slide Time: 20:31)



So, this means again I check  $e_1$ ,  $e_1$  is true so I carry out  $s_1$  else false; that means, this else is for this decision box I check here  $e_2$ ,  $e_2$  is true so I carry out  $s_2$  and else I carry out  $s_3$  this is what I am trying to do.

Now, here I would have preferred to write it in a better way. That is why if you recall in a earlier lecture we had talked about a good program writing practice that is indentation.

(Refer Slide Time: 21:35)

The slide contains four code snippets on the left and a hand-drawn diagram on the right.

Code Snippets:

- if e1 s1  
else if e2 s2
- if e1 s1  
else if e2 s2  
else s3
- if e1 if e2 s1  
else s2  
else s3
- if e1 if e2 s1  
else s2

Hand-drawn diagram (in orange):

- A large question mark is positioned between the first two code snippets.
- Arrows point from the first snippet to the first part of the second snippet (e1 s1) and from the second snippet to the third snippet (else s3).
- Brackets above the first snippet group it with the second, and brackets above the second snippet group it with the third.
- Handwritten text above the first snippet reads: 'if e1 s1' and 'else if e2 s2'.
- Handwritten text above the second snippet reads: 'if e1 s1' and 'else if e2 s2'.
- Handwritten text above the third snippet reads: 'else s3'.

I would have like to write it in this way if e1 then s1 else if e2 then s2 else s3 which makes a very clear that this else is corresponding to this if and this if this else is corresponding to this if right.

Let us look at the third scenario what does it mean. If e1 if e2 s1 how would I draw that?

(Refer Slide Time: 22:13)

The slide contains five code snippets on the left and a hand-drawn diagram on the right.

Code Snippets:

- if e1 s1  
else if e2 s2
- {  
    if e1 s1  
    else if e2 s2  
    else s3  
}
- if e1 if e2 s1  
else s2  
else s3
- if e1 if e2 s1  
else s2

Hand-drawn diagram (in orange):

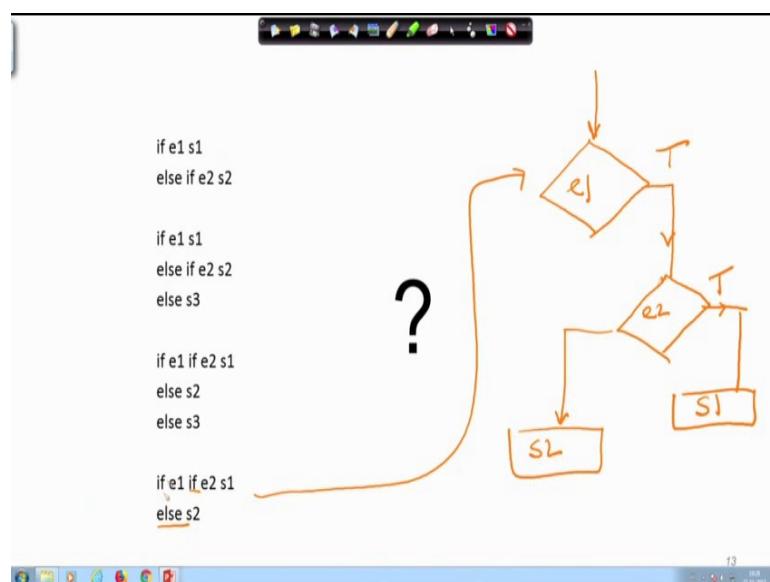
- A flowchart representing a nested if structure.
- The outer loop starts with a decision diamond labeled 'e1'. If 'e1' is False (F), it leads to box 'S3'. If 'e1' is True (T), it leads to another decision diamond labeled 'e2'.
- The inner loop starts with decision diamond 'e2'. If 'e2' is False (F), it leads to box 'S1'. If 'e2' is True (T), it leads to box 'S2'.
- There is a question mark placed near the entry point of the inner loop.

I draw that like that I take e1 if e1; that means, if e1 is true I immediately come here and the statement starts with another if. A nested if, this is known as the nested if one if

within another there is one if statement there is another if statement within this, this is known as nesting alright.

So, here sorry over here if e2 s1. So, immediately I am there is no else here by the way. So, if e1 then I come here then if statement is again there therefore, there is another decision box where I am checking the condition e2 and if e2 is true then I will follow then I will carry out s1 else s2, which else should it be here or there. Now, this is something this else is pairing with a preceding if, so it should be false will be s2. Now this else is therefore, covered this path is covered. So, this else can only mean this one and s3 will be because this is already covered right. So, this is the nearest one to this clear. So, if we go to the earlier slide. Let me go back to the presentation.

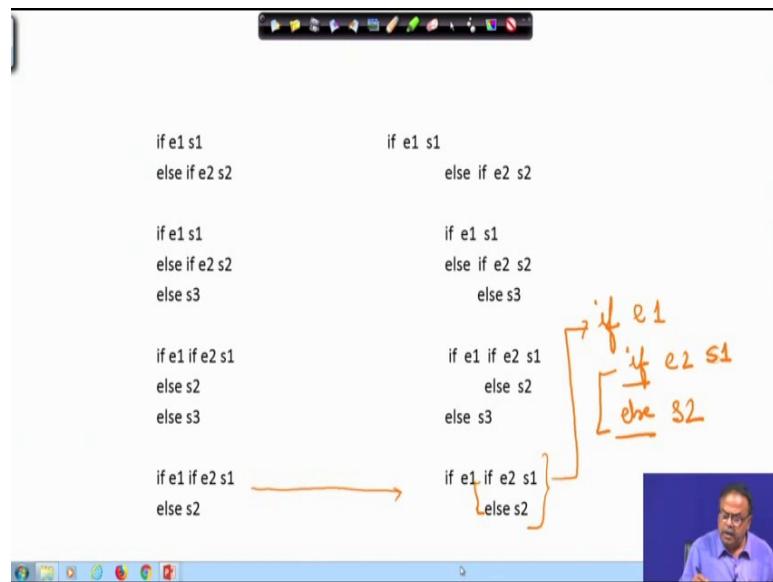
(Refer Slide Time: 24:17)



So, here this one how would this one actually look like? How will this one look like? There is something still there. So, how will this one look like; coming straight here if e1 if the condition is met then I again test here if e2 this is true this is true then I come and do it here s1 and else this else is again with this if. So, it will be here s2 and I have not specified this one here alright.

So, if you put a little bit of mind here. So, you can see many different other scenarios that can come and we can proceed accordingly.

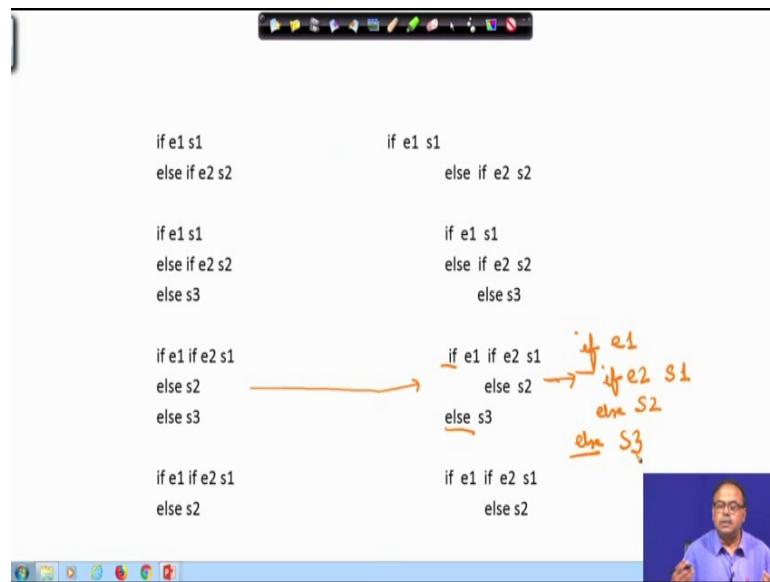
(Refer Slide Time: 25:16)



Let us see here. This one as I was saying it will be nicer to write it in this way because here you may feel there two elses two if who goes with whom. Now, here if I write it in this way if e1 s1 else; that means, this if-else else if e2 s2. Here if e1 s1 else if e2 s2 else s3. So, this else is therefore, with I am sorry I think I will have to go back to a couple of slides you see and else clause I would like to be corrected a little bit and else clause is associated with the closest preceding unmatched if, any if that is not there that should be it should match with that. So, that is what I explained in these examples.

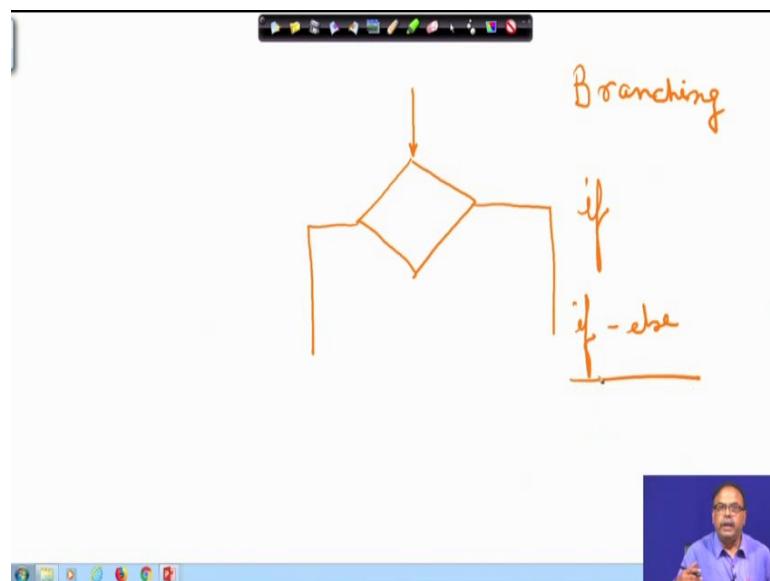
So, it is a much better for example, just look at this how would I like to like write this. If e1 I would not prefer this also, I would prefer that I write it in this way. If e1 then if e2 s1 else s2 that becomes very clear this if-else and else are very clear. Say for example, here also this is better, but still I would prefer to write this one as this one, this is one way I can write if e1 if then I mean this is the part of if, if e2 s1 else s2 and under this if that is what is been done here else s3.

(Refer Slide Time: 27:17)



So, what we have learnt today is a nesting of another structure. What we are discussing now is how we can have a better or more versatile flow.

(Refer Slide Time: 27:55)



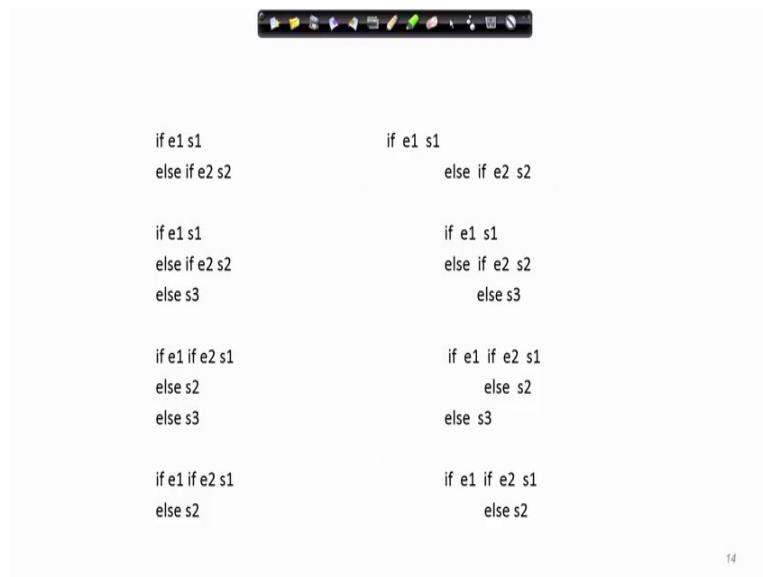
Normally it is the sequential flow, but based on the decision points I may have to take different paths which is known as branching right and for branching or selecting the proper path the structure is if. Along with that we have learnt today if-else structure and we have seen how if-else structure can be nested to give rise to some better more versatile scenarios. We will continue with this in the next lecture.

Thank you.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

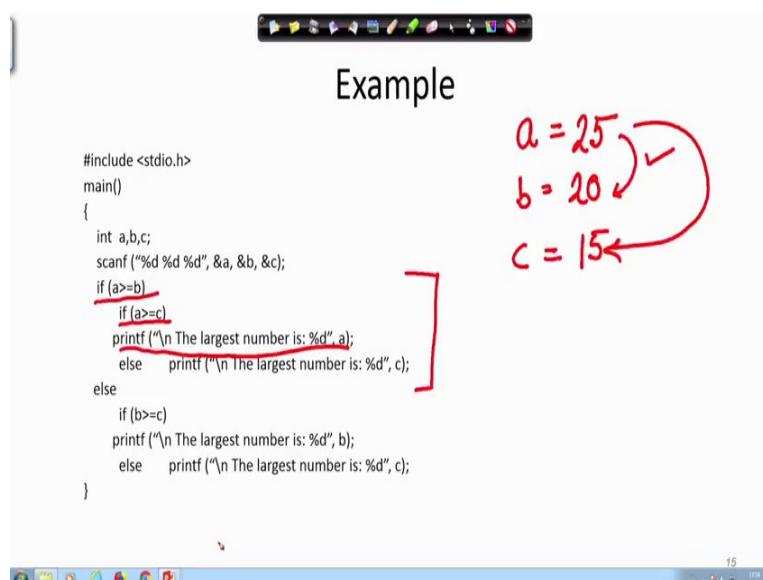
**Lecture – 16**  
**IF-ELSE Statement (Contd.)**

(Refer Slide Time: 00:21)



Welcome. In the last lecture we had looked at the nested if-else structure. Today we will continue with that discussion.

(Refer Slide Time: 00:30)



So, here is an example that uses the nested if-else structure. Here you can see again let us come to the programming fundamentals in C, we have got just for revision we start with an include stdio dot h and then we have got the main function. Inside the main function we declare 3 variables a, b and c and probably you might have guessed now that we are again trying to find out our, I mean the maximum of the 3 integers a b and c.

So, what are we doing next? We are first reading the 3 numbers from the keyboard right, scanf, percentage d, percentage d, percentage d, then a, b and c preceded with an ampersand and you know why this ampersand is used that is the address of the location a b and c.

Now, comes the main logic here if a is greater than or equal to b, a is 25 b is 20. So, if a is greater than equal to b then I check whether a is greater than c. So, this is passed then I check this. So, if a is greater than b then I proceed to do if a is greater than equal to c if that is also true, then we print the largest number is a; otherwise what we do not know as yet.

(Refer Slide Time: 02:47)

```
#include <stdio.h>
main()
{
    int a,b,c;
    scanf ("%d %d %d", &a, &b, &c);
    if (a>=b)
        if (a>=c)
            printf ("\n The largest number is: %d", a);
        else    printf ("\n The largest number is: %d", c);
    else
        if (b>=c)
            printf ("\n The largest number is: %d", b);
        else    printf ("\n The largest number is: %d", c);
}
```

So, if a is greater than b, a is 25, b is 20, c is 15 in that case a is greater. Let us suppose c is thirty then what happens a is greater than b true then I come and compare a is greater than c this part, no false. Therefore this else is with respect to this nearest if that we learnt in the earlier lecture. So, if this fails then we print the largest number is c because a was greater than b, but a is not greater than c therefore, c must be the largest.

(Refer Slide Time: 03:40)

```
#include <stdio.h>
main()
{
    int a,b,c;
    scanf ("%d %d %d", &a, &b, &c);
    if (a>=b)
        if (a>=c)
            printf ("\n The largest number is: %d", a);
        else    printf ("\n The largest number is: %d", c);
    else
        if (b>=c)
            printf ("\n The largest number is: %d", b); ✓
        else    printf ("\n The largest number is: %d", c);
}
```

a = 20  
b = 25  
c = 30



Now, if a is not greater than b. Suppose a is 20 and b is 25, then a greater than b this fails, this fails then I am not entering this block at all I am straightway going to this else then I am checking a is not greater than b, but therefore, b is must be greater than or equal to a or not even equal to b must be greater, greater than a then I check if b is greater than c suppose b is greater than c then the largest number is b otherwise if that is not. So, b is not greater than c suppose c was thirty then this else will come because this if we will fail therefore, I will print gain the largest number is c.

(Refer Slide Time: 04:46)

```
#include <stdio.h>
main()
{
    int a,b,c;
    scanf ("%d %d %d", &a, &b, &c);
    if (a>=b)
        if (a>=c)
            printf ("\n The largest number is: %d", a);
        else    printf ("\n The largest number is: %d", c);
    else
        if (b>=c)
            printf ("\n The largest number is: %d", b);
        else    printf ("\n The largest number is: %d", c);
}
```

$\left. \begin{array}{l} \text{if } (a \geq b) \& \text{if } (a \geq c) \\ \text{if } (b \geq c) \end{array} \right\} \begin{array}{l} \text{printf} (\dots, a); \\ \text{else printf} (\dots, c); \end{array}$

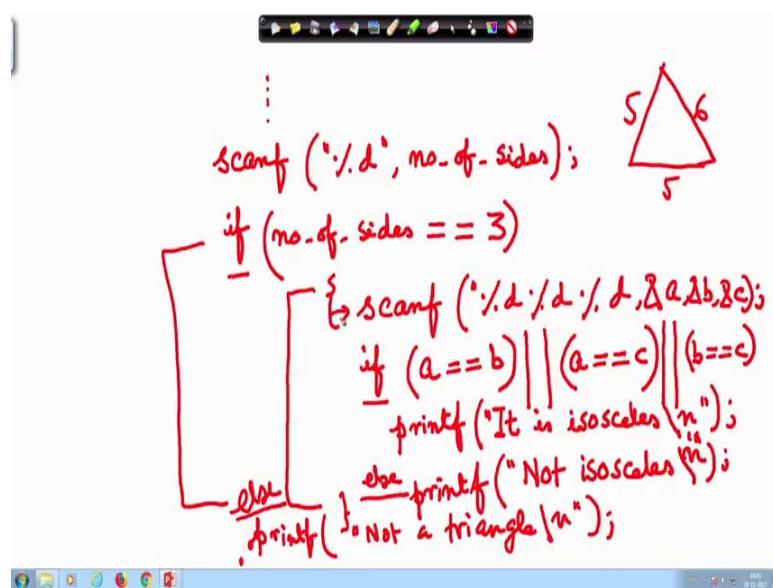


So, I would suggest you to work it out yourself and I can also write the same thing. If you consider that this if a is greater than b then if a is greater than equal to c, tell me if I write it in this way if a is greater than equal to c and and; that means, and a is sorry here I right a is greater than equal to b and a is greater than equal to c printf this line largest is a else printf largest is c. So, do you think that these two are equivalent? These two nested ifs and this logical operator.

Now, here I should have put a parenthesis here. Please note I need a parenthesis here it is better that if I put it it becomes much clearer. Are these two equivalent? If you just think a while you will find yes they are equivalent because I am computing this and then I am computing this if this any one of this is false I will not be executing this statement right. So, this is an example of nested if.

We can have many such examples. Let me give you one example. Let us workout right now.

(Refer Slide Time: 06:45)



Suppose I am trying to see if a figure is a triangle or a square or a rectangle or rectal I mean 4 sided figure whether it is a triangular figure or a quadric quadrilateral figure. And if it is a triangular figure then I want to see whether it is isosceles; that means, two sides are equal or it is not isosceles. Suppose that is what I want to do. So, how can I proceed to do that; how would I write the logic? Let me just only write the relevant part. I have done some the declarations are here and then I have done scanf percentage d, number of

sides all right number of sides. So, number of sides are variable. And then I also then; what I can do? I check if number of sides is equal to 3 then I will be doing something all right because in that case it will be a triangle.

Then what do I do I? Then read that is scanf the 3 sides of the triangle percentage d, first and suppose all the sides are integer has got all the sides have integer values all right 5, 6, 5 something of that sort. That is why I am putting percentage d then I am reading the 3 sides and a and b and c semicolon.

If a is same as b or you remember this is the logical OR, or a is the same equal to c or b is equal to c then what can I say, printf “it is isosceles backslash n”, as our common practices. If any of these conditions are true either this is equal to this or this is equal to this or this is equal to this then is isosceles. Else I will printf say I am writing in brief “not isosceles”. You can write in a much better way. Remember this double quotes we complete this. Now, that is the, I complete this. Now, this is the if part. If the number of sides is 3 if the number of sides is not 3 then I do not do anything I will come at this point because this if you will fail I will come at this point. I can say here can I write here printf not a triangle, can I do this, just think because here I have checked that the number of sites is 3. If it is number of sites is 3 I will do this and print and otherwise I write it here is it all right. If you look at it you will see that there is a problem here, that if the number of sites is 3 and then it checks whether it is isosceles or not and then it comes out and again prints is not a triangle I do not want that. So, what should I do, how can I avoid this problem?

Here you must have guessed now, here I have to put an else that goes with this if all right and then if this is not true then only this will be executed otherwise some point down the line here will be executed. Another thing that I would like to mention in this case look at the use of this parenthesis why was this parenthesis required because I had more than one statements for this if condition if this condition is true then there are more than one statements.

Now, here is a puzzle. How many statements are there inside this block? How many statements are there? 1, 2, 3, 4, let us see. Here is one statement scanf so one and then this if statement you see goes up to this. So, that is one statement, but here I am using I said that if felt is a structure. So, this entire thing is a statement. So, actually I have got

two statements. Since I have got more than one statement I had to put these braces. Now, suppose of course, then the program will not work here, suppose I had not written this statement this statement was not there all right this statement was not there in that case I could have done away with these braces because then it is within this if statement there is one statement only this brace was not essential. So, you could see this and you can practice and in the assignment we will also give few programs that you have to do using this sort of if-else type of structure.

(Refer Slide Time: 14:52)

The screenshot shows a presentation slide with a title 'Example' centered above a code block. The code is a C program that reads three integers from the user and prints the largest one. It uses nested if statements and printf for output. The slide has a standard Windows-style toolbar at the top and a navigation bar at the bottom.

```
#include <stdio.h>
main()
{
    int a,b,c;
    scanf ("%d %d %d", &a, &b, &c);
    if ((a>=b) && (a>=c))
        printf ("\n The largest number is: %d", a);
    else if (b>c)
        printf ("\n The largest number is: %d", b);
    else
        printf ("\n The largest number is: %d", c);
}
```

So now, here is the example that we did.

(Refer Slide Time: 15:02)

Confusing Equality (==) and Assignment (=) Operators

- Dangerous error
  - Does not ordinarily cause syntax errors
  - Any expression that produces a value can be used in control structures
  - Nonzero values are `true`, zero values are `false`

The hand-drawn diagram shows two `if` statements. The first is `if (a == b)` with a red arrow pointing to it. The second is `if (a = b)`. Below these, there is a red annotation  $a = b$  with  $a \leftarrow b$  underneath, indicating assignment.

Now, there is a problem here there is a danger I would rather say. You have seen earlier when I was writing I was writing if a is equal to b, then I was doing something right. Now, this is very important that is a common point of source of, common source of error that often because we have learnt to say equal to like this it will we can write in this way. Now, the problem is if I write it in this way when a compiler looks at this what will it assume it will see that is an if often you will not find that it will cause an error because what the compiler will think is well, what is the meaning of this; a this means a is assigned the value of b. So, it will try to successfully, it will try to transfer copy the value of b to a all right and this operation will be executed successfully.

Now, if x in any expression is computed successfully; what does it return? If you recall it returns a one and one means true; that means, this condition will evaluate to true. I once again repeat. Now, we can use any expression that produces a value in the control structure. Now, if it be a nonzero value then is true and 0 value is false.

(Refer Slide Time: 17:03)

Confusing Equality (==) and Assignment (=) Operators

- Dangerous error
  - Does not ordinarily cause syntax errors
  - Any expression that produces a value can be used in control structures
  - Nonzero values are true, zero values are false

if(z=a+b+c) > if(1)  
printf("A");  
else printf("B");

So, if I write if a plus b plus c printf "A" else printf "B" what will happen? a plus b plus c is if they will be just be added, but there will be no assignment suppose I make it more meaningful. Suppose I make it z sorry z assigned (Refer Time: 17:53) I write z assigned a plus b plus c all right then this there is a problem here because there is no semicolon here right. So, this will become suppose I do this.

Now, this will be computed and this assignment will be done and this assignment will be done successfully. So, anything that is done successfully will return this will be equivalent to if 1, if 1 means it is true and so what will be printed, printf A will be executed, this will be printed.

(Refer Slide Time: 18:48)

Confusing Equality (==) and Assignment (=) Operators

- Dangerous error
  - Does not ordinarily cause syntax errors
  - Any expression that produces a value can be used in control structures
  - Nonzero values are `true`, zero values are `false`

Now, similarly I can say let us see I just write if this and something. What does this mean? Semicolon nothing before this we call it a null statement; that means, I have not state anything meaningful, but it is still a statement. Remaining silent is sometimes a set statement. Now, this means that this is always true this will always be true. So, this is equivalent to if 1 this; that means, always you do this statement this, whatever is here you read do it always all right. So, this is a interesting thing in C.

(Refer Slide Time: 19:43)

Confusing Equality (==) and Assignment (=) Operators

- Dangerous error
  - Does not ordinarily cause syntax errors
  - Any expression that produces a value can be used in control structures
  - Nonzero values are `true`, zero values are `false`
- Example:

```
if ( payCode == 4 )
    printf( "You get a bonus!\n" );
```

Checks paycode, if it is 4 then a bonus is awarded

However sometimes you know it varies all with the compilers also some compilers take care of such situations and may give you a warning some very few compilers can also give any error. Now, let us look at this example sorry this example.

If pay code is equal to 4, what does this mean? This means if the pay code is 4 this is a logical equality if this is true then will print you will get a bonus. Now, instead of, I want to do like this, this means that I am coming here I am checking pay code in non C language you speak out 4 yes I do something no I do something else. But if I had written it in this way what would it mean?

If pay code assigned 4. Now, pay code being out suppose pay code is an integer type of variable then this can be assigned the value 4 successfully and anything that is done successfully this will result in if done successfully one do this. That means, it will be successfully executed it is not doing what you are intending. So, sometimes we are saved by the compiler when it points out that note here, here is something some syntax error that you are committing all right, but there are situations when we unintentionally can do such mistakes which will go unnoticed by the compiler and the result that will be getting may not be what we desired.

So, this is a very critical point you should keep it in mind for all programming languages we will find there such nuances some specialties which you will have to keep in mind. We are just mentioning here the ones for C. But if I had done it in this way it would be ok.

(Refer Slide Time: 22:15)

The slide has a title bar with icons and the text "Confusing Equality (==) and Assignment (=) Operators". Below the title is a bulleted list:

- Dangerous error
  - Does not ordinarily cause syntax errors
  - Any expression that produces a value can be used in control structures
  - Nonzero values are `true`, zero values are `false`
- Example:

```
if ( payCode == 4 )
printf( "You get a bonus!\n" );
```

Checks paycode, if it is 4 then a bonus is awarded

```
Equality check proper
if ( payCode == 4 )
printf("You get a bonus!\n");
```

So, generalization of expression evaluation in C is assignment operated is also a part of expression that we know.

(Refer Slide Time: 22:24)

The slide has a title bar with icons and the text "Generalization of expression evaluation in C". Below the title is a bulleted list:

- Assignment (=) operation is also a part of expression.

Below the list, there is a code snippet with annotations:

```
i=3;
```

*>Returns the value 3 after assigning it to i.*

Handwritten notes next to the code:

*if (~~10/maz~~)  
if (1)  
if (3)*



Now, let us see here `i` is assigned 3. So, typically some compilers what they do till now, I was saying that if an instruction is executed correctly it will return 1, but generally speaking it will return the value that has been assigned.

So, typically here `i` assign 3 will return the value 3 after assigning it to it, but for us when we do what we are bothered about if the condition this part is open sorry I would rather.

Now, say is part is 0 or nonzero. If it is nonzero then this true I was saying if 1 it is true, if 3 that is also true the only falsity will come if it is 0. So, this returns the value 3 after assigning it to i.

(Refer Slide Time: 23:48)

Generalization of expression evaluation in C

- Assignment (=) operation is also a part of expression.

`i=3;`      Returns the value 3 after assigning it to i.

`int i=4, j;`

`if(i==3)`

`j=0;`

`else`

`j=1;`

Annotations: A red box highlights `i=3;` with a note "Returns the value 3 after assigning it to i.". Red arrows point from the condition `i==3` in the if-statement to the value 3 in the assignment `i=3;`. Another red arrow points from the `i` in `i=3;` to the `i` in the variable declaration `i=4, j;`. A red bracket groups the `i` and `j` declarations, with a red arrow pointing from the `i` in the declaration to the `i` in the assignment. A red arrow also points from the `i` in the assignment to the `i` in the if-condition.

And then, so let us look at this code. I give you 1 minute to look at this code and think what this code will do, what will be the result of this. Look at this code carefully then we will analyze it.

You can see that here we have initialized a variable i to 4 and j has just been declared, but not initialized. So, here the picture is something like this, i has been initialized to 4 and j can have something, but I have not initialized it. Now, here if this statement, what will this statement do? First the condition part; that means, within the bracket this part will be evaluated and what will happen, what is this i is being assigned 3 so this will be 3 and the value that will be returned by this execution of this statement will be 3, then j will be 0 then j will be 0. Else j equal to 1 that will never come because this will be done successfully, this will never come. So, the output will be, output I have not given any printf here, but this j, value of j will be 0.

(Refer Slide Time: 25:41)

The diagram illustrates the generalization of expression evaluation in C. It shows two code snippets and their execution flow:

**Code 1:**

```
int i=4, j;  
if(i==3)  
    j=0;  
else  
    j=1;
```

**Code 2:**

```
int i=4, j;  
if(i==3)  
    j=0;  
else  
    j=1;
```

A callout box labeled "j?" points from the condition in both if statements to a note: "Whatever be the value of i, j is always 0." Another callout box labeled "i=3;" points to a note: "Returns the value 3 after assigning it to i."

19

So, what will be the value of j? Whatever be the value of i, j will be 0 right. Now, instead if I had written it correctly, correctly means if my intention was that if i is equal to 3, j should be 0 otherwise j should be 1 then I should have written it in this way. So, that if i is equal to 3 then j will be 0 otherwise j will be 1. So, this is a very important point that you have to keep in mind and gradually with practice they will be ok with this.

(Refer Slide Time: 26:34)

The diagram illustrates more about expressions, specifically Increment (++ ) and Decrement (--) Operations. It shows the state of variable i as 5, and the effect of the assignment statement  $i = i + 1;$  which results in  $i = 6;$  and the effect of the assignment statement  $i = i - 1;$  which results in  $i = 5;$  with annotations  $i++;$  and  $i--;$  respectively.

20

So, here, more about expression. Now, here we introduce two new expressions typically when we write suppose there is a variable i and the variable i has got a value 5. Now, if I

want to implement the value 5, if I want to implement the value of i I will write i assigned i plus 1 or if I want to decrement the value of i then I can write i assign i minus 1, but C allows us or gives us one special implemented another special decrement operation.

So, this thing I could have written also as instead of this, this part I could have simply written i plus plus, i plus plus means i is assigned i plus 1 and this 1 will be i minus minus all right. So, this is known as the decrement operator, this is known as the increment operator. Sometimes it becomes handy to write it in this way, but if you are new in programming i personally would discourage you because you should not get ground in the special features of a language initially better you go by the standard features because these features have got some more complications which will come to later. But you may know that this is an increment operator which is equivalent to this and this is a decrement operator which is equivalent to this all right.

(Refer Slide Time: 28:55)

More about expressions

- Increment (++) and Decrement (--)Operations

Prefix operation       $\boxed{++i;}$

Postfix operation       $i \boxed{++}$

So, this is I can write it in two ways sorry i plus plus or plus plus i. The reason why I were I discourage you to get into the usage of this initially will be evident soon. If I write i plus plus it means this is called post operation this means pre operation of prefix operation. We will soon see what that means. So, I can have plus plus i or minus minus i that is a prefix operation and postfix operation is i plus plus and i minus minus.

(Refer Slide Time: 29:55)

The slide has a title 'More about expressions' and a bullet point '• Increment (++) and Decrement (--)Operations'. Below this is a diagram comparing Prefix and Postfix operations:

Prefix operation	$++i;$	$--i;$
First increment / decrement and then used in evaluation		
Postfix operation	$i++;$	$i--;$

Handwritten notes on the slide show the evaluation of expressions:

$$\begin{aligned} z &= 5; \\ x &= \underline{\underline{++i}} + z; \rightarrow x = 11 \\ x &= \underline{\underline{i++}} + z; \rightarrow \end{aligned}$$

86

Now, plus plus i means first increment or decrement depending on whether it is plus or minus and then use it in the evaluation. For example, suppose I have got the value i here as 5 and I write x as signed plus plus i plus z or let me make it a constant, let us say z is 5. Then what will happen, this is what, this is pre increment i first increment i and then do the operation.

Now, this has got precedence of course, over this. So, first i will be, i will be incremented, so i will be 6 and z was 5. So, the result will be x will be 11, but if I had done written it x assigned i plus plus plus z here also I will first increment this 6 and then z will be added to that. But there are situations where we will do it after computation of the whole thing. I think I will need a separate session for you to explain this clearly. We will do that in a more careful manner.

(Refer Slide Time: 31:50)

The slide has a title 'More about expressions' and a bullet point: '• Increment (++) and Decrement (--)Operations'. It compares Prefix operation (++i;) and Postfix operation (i++). A note says 'First increment / decrement and then used in evaluation' for prefix and 'increment / decrement operation after being used in evaluation' for postfix. Below, two code snippets are shown: `int t,m=1;` followed by `t=++m;` and `int t,m=1;` followed by `t=m++;`. The first snippet results in `m=2;` and `t=2;` with `t=` underlined and a red arrow pointing to 1. The second snippet results in `m=2;` and `t=1,` with `t=` underlined and a red checkmark.

Let us look at this here int t is an integer and the value of m is 1. t assigned plus plus m; that means, what m was 1 and m is first incremented and assigned to t therefore, t will be 2 all right. Again if we do it like this here then yes, here it makes the difference very clear I am sorry t sorry here you see m was 1 and what I have done I have put the assignment of I have done a post operation, post fix operation; that means, first I will be doing the assignment, first I will be doing the assignment and then I will increment. So, what will happen? m was 1 here, so first this 1 will go over here. So, t will become 1 right and then before I complete this operation I increment m so m becomes 2 all right. So, we will see more examples of such things in the subsequent lectures.

Thank you.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 17**  
**Switch Statement**

(Refer Slide Time: 00:23)

The slide has a title 'More about expressions' and contains the following text and diagrams:

- Increment (++) and Decrement (--)Operations

Diagram illustrating the evaluation of increment and decrement operations:

- Prefix operation:** Shows `++i;` and `--i;`. A callout box states: "First increment / decrement and then used in evaluation".
- Postfix operation:** Shows `i++;` and `i--;`. A callout box states: "increment / decrement operation after being used in evaluation".
- Code examples:**
  - `int t,m=1;` followed by `t=++m;` leads to state `m=2;` `t=2;`
  - `int t,m=1;` followed by `t=m++;` leads to state `m=2;` `t=1;`

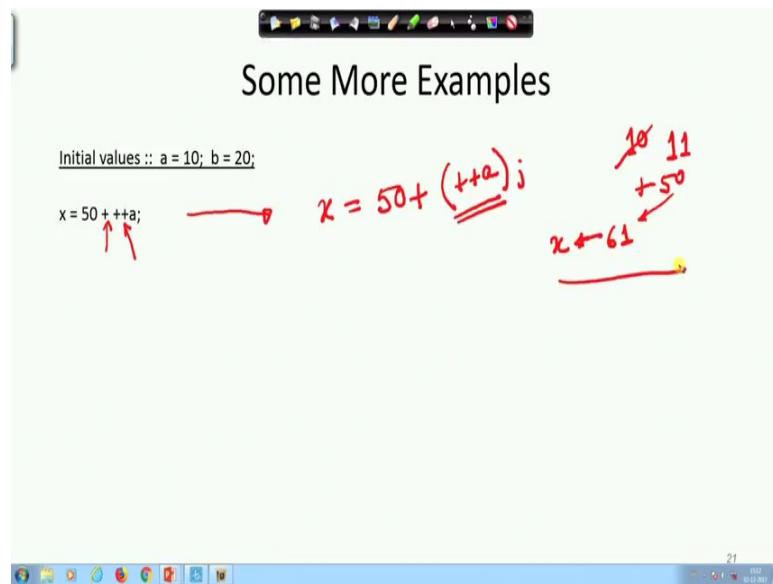
In the last lecture we had discussed about 2 new operations, that is post increment and pre increment. Here you can see that we are calling them as prefix operations because the operator is before the operand here you can see also the operator is before operand and what is the operator? Operator is incrementation, but that is being denoted as plus plus twice the addition operation or twice the subtraction operation. And in this here we are talking off we are calling it to be postfix operation, here you can see that the operand is coming first and the operators are following right. So, the operators are following them. So, that is why this is known as postfix here also you can see the operated operand to be I followed by the operator minus minus right. So, that is why these are called postfix or prefix. The significance of these as we discussed in the last class is that a particular variable.

Will be first incremented here in this case it will be first incremented and then used in operation, here it will be first decremented then will be used in the operation. While here it will be first the operation will be done and then the decrementation will be done here

first the operation will be done then the incrementation will be done. So, we can, that is what we have written here first increment or decrement and then use it in the evaluation and here increment or decrement is done after being used in evaluation. So, it will be clearer when we come with some examples, see about the example that we have shown in the last class t and m are 2 variables, m is initialized to 1 and when I am saying plus plus m; that means, first I increment m.

So, m will be here you can see as soon as I do that m will become 2 and then that is being assigned to t. So, t becomes 2 whereas, here you can see that m is 1 and we have done post increment. So, first the assignment operation will be done; that means, m was 1. So, sorry m was to here after this I am doing this. So, m will be. So, t will be one. So, what will happen is m is 1. So, that m will be assigned to t. So, t becomes 1 and after that it will be incremented. So, ultimately m will be to, but t will be 1. So, if you look at these 2 you will find the difference between these.

(Refer Slide Time: 03:32)

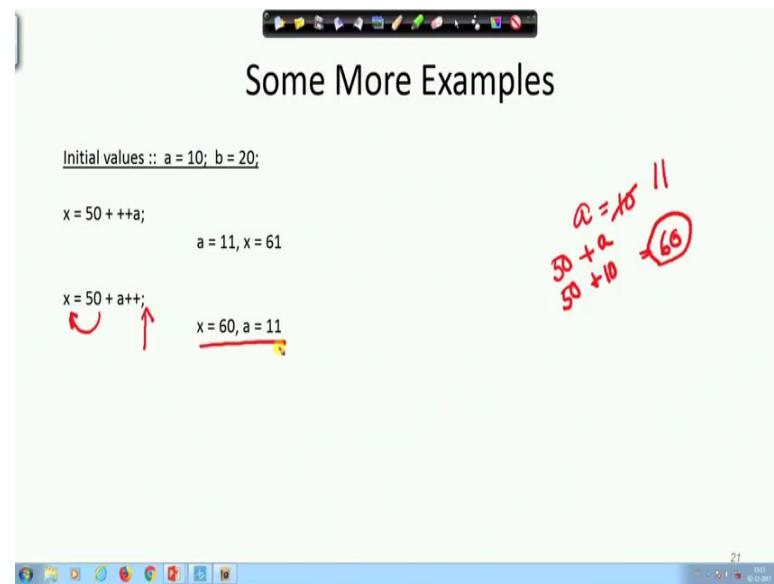


Let us have a few more examples, suppose we have got 2 variables a and b whose initial values are 10 and 20. They are initialized to 10 and 20 next we carry out look at this operation very carefully, it looks a little clumsy here there are 2 operators one is this one another one is this one. So, this is pre increment.

So, this is equivalent to x is assigned 50 plus as if I have put a parenthesis, but that is not required that first a will be incremented and then that will be added to 50. So, what result

do we expect a was 10. So, first that will be incremented. So, that will be 11 and that will be added to 50 and. So, x will be x we get the value 61 all right that is what will happen in this case. So, a is 11 and x is 61 as we have shown.

(Refer Slide Time: 04:54)



Now, let us take this again this is the; this is post increment. So, what do we expect here let us see what will happen in this case, first here after the operation is done then a will be incremented. So, what will happen? X was 50 sorry sorry I am sorry a was 10 and I am adding 50 and 10 and a; that means, 50 plus 10. So, that becomes 60 and that 60 is being assigned to x.

Because, the assignment operation, is taking precedence over this post increment. So, we get x to be 60, but I have my job is not it clear because after is participation in this operation, it will be incremented by 1. So, a was 10 then we will be made incremented by 1. So, it will become 11. So, consequently this is the result what we will get.

(Refer Slide Time: 06:21)

The slide title is "Some More Examples". The code shown is:

```
Initial values :: a = 10; b = 20;  
x = 50 + ++a;  
a = 11, x = 61  
x = 50 + a++;  
x = 60, a = 11  
x = a++ + -b;
```

Handwritten annotations in red:

- $a = 10$
- $b = 20$
- $x = 61$
- $a = 11$
- $x = 60$
- $a = 11$
- $x = 29$
- $x = 29$

Now, let us see again here; what do you expect to happen. A is 10 as usual, a is 10 and b is 20 what is being said here? A is post increment, but b is pre increment pre decrement. So, first b will be decremented. So, b will become 19 decrement means by 1. So, it will be decremented it will be nineteen it will be added to a; what was a? 10.

So, these 2 will be added and what is that 29 and that 29 will go to x. So, x will be 29 and then my job is not it clear not it over. So, this will be incremented and a will be 11 all right. So, this is how the thing will happen.

(Refer Slide Time: 07:36)

The slide title is "Some More Examples". The code shown is:

```
Initial values :: a = 10; b = 20;  
x = 50 + ++a;  
a = 11, x = 61  
x = 50 + a++;  
x = 60, a = 11  
x = a++ + --b;  
b = 19, x = 29, a = 11  
x = a++ - ++a;
```

Handwritten annotations in red:

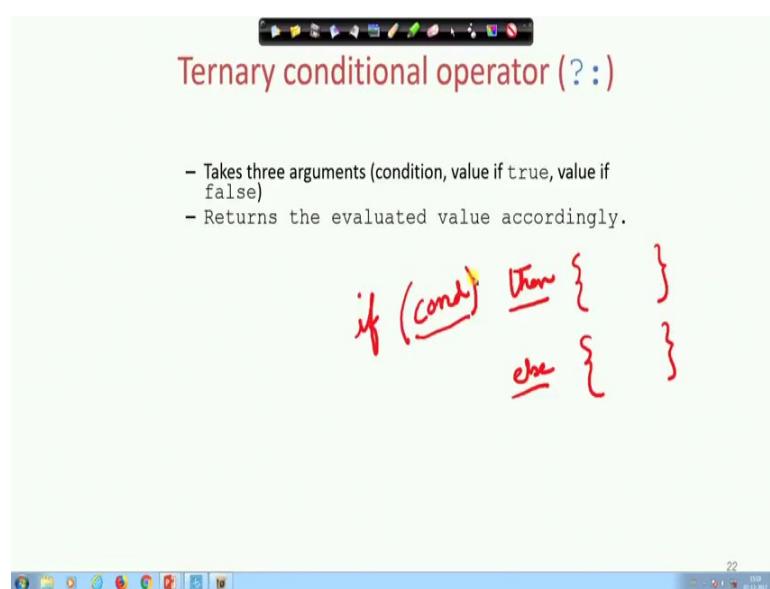
- $a = 10$
- $b = 20$
- $x = 61$
- $a = 11$
- $x = 60$
- $a = 11$
- $b = 19$
- $x = 29$
- $a = 11$
- $x = 29$
- $x = 29$

A video feed of the speaker is visible in the bottom right corner.

So, the results that we get in this case will be b will be 19, because b has been decremented x will be 29 and a will be 11. I hope you have understood the flow. So, now this is another one, here we really do not know what to do because here I am incrementing pre incrementing a post incrementing a here now I will take this value of a now what will happen. Suppose the problem here is suppose a is let us take the case here a to be 10 now first.

I will decrementing I will incrementing. So, a will be 11, but a was this a is not incremented. So, a minus 11. So, 11 minus 11, will it be 0 and then this will be a will be 12 or which one will be done first this is a particular scenario, for the same variable in the post increment the post decrement is given often that is implementation dependent and the compiler does not take it well. So, it often results in undefined value. So, you should try to avoid this as much as possible actually you should not use this you should use you can use the pre increment post increment, but as I said in the last class I had strongly suggest that at the initial phase of programming you avoid using this pre increment post increment operations, instead although it will be a little laborious you write x assigned x plus 1 or x assigned x minus 1 for pre increment and pre decrement, but it is good to know that c allows us with this facilities.

(Refer Slide Time: 09:47)



Now, here is another one which at the initial phase you should not try to use unless you are very confident. We know if conditions then we do something else something. So,

there are three things, if some condition then although we do not write them, then we do some operations else we do some other operation. Then the same thing can be written, it call it a ternary conditional operation here we write it in a different way.

(Refer Slide Time: 10:45)

The slide has a title "Ternary conditional operator (?:)" in red. Below the title is a list of two bullet points:

- Takes three arguments (condition, value if true, value if false)
- Returns the evaluated value accordingly.

Below the list is a C code snippet: `grade >= 60 ? printf("Passed\n") : printf("Failed\n")`. This code is annotated with red circles and arrows. One circle covers the condition `grade >= 60`, another covers the part `printf("Passed\n")`, and a third covers the part `printf("Failed\n")`. To the right of the code, there is handwritten text in red:  
`if (grade >= 60)  
 printf("Passed\n");  
else printf("Failed\n");`

Just to save space on I mean the size of the program what we do here say grade greater than equal to 60 printf passed otherwise printf failed. So, this means here this is the this part is the condition part and this part is the part that will be computed if the condition is true, and this part will be computed if the condition is false all right. So, this is equivalent to writing if grade greater than equal to 60 printf, past else printf failed all right this is equivalent to that. So, what is happening here in this example? So, the general syntax is if there is an expression there is a condition expression then we carry out if that is true.

(Refer Slide Time: 12:21)

The slide title is "Ternary conditional operator (?:)". Below the title, there is a list of two points:

- Takes three arguments (condition, value if true, value if false)
- Returns the evaluated value accordingly.

Below the list, there is a code example:

```
grade >= 60 ? printf("Passed\n") : printf("Failed\n");
```

Below the code, there is a hand-drawn diagram illustrating the ternary operator. It shows a box with a question mark '?' at the top-left and a colon ':' at the bottom-right. A red arrow points from the question mark to the condition 'grade >= 60'. Another red arrow points from the colon to the 'Failed' part of the code. A yellow arrow points from the colon to the 'Passed' part of the code. The letter 'F' is written below the box.

Then we carry out expression true otherwise if it is false, then we carry out if it is true then we carry out this part this part, that is fall following interrogation mark, and we carry out the expression three that is the expression that is following the colon mark for the else case or if the condition is false. So, this is again writing that if then else in a cryptic shorter way this is known as ternary conditional operation, but again I would say that this is a specialty of c it is good to know that, but initial phase at the initial phase of programming I discourage you to use this .

(Refer Slide Time: 13:11)

The slide title is "Ternary conditional operator (?:)". Below the title, there is a list of two points:

- Takes three arguments (condition, value if true, value if false)
- Returns the evaluated value accordingly.

Below the list, there is a code example:

```
grade >= 60 ? printf("Passed\n") : printf("Failed\n");
```

Below the code, there is another code example:

```
(expr1)? (expr2): (expr3);
```

Below the second code example, there is an "Example:" section:

interest = (balance>5000) ? balance\*0.2 : balance\*0.1;

A pink curly brace is placed under the entire line of code in the example section, with the text "Returns a value" written below it.

So, here is an example what will this v can anyone tell me? We are computing the interest on the amount of money you have got in bank, this means here let us look at this part this is a ternary operator. If the balance is greater than 5000 then you compute balance times 0.2 that means, 20 percent of the balance or compute 10 percent of the balance. If somebody has got a balance 6000 then have 20 percent of that computed and assign it to the interest part. So, you see a bigger statement if balance is greater than 500 then interested equals balance multiplied by 0.02 else interest equal to balance multiplied by 0.01 all these things can be done in one single sentence or one single statement using c all right. So, that is why some people prefer to use such ternary operators and you will also certainly use it, but when you are very confident about your programming. So, this.

(Refer Slide Time: 14:48)

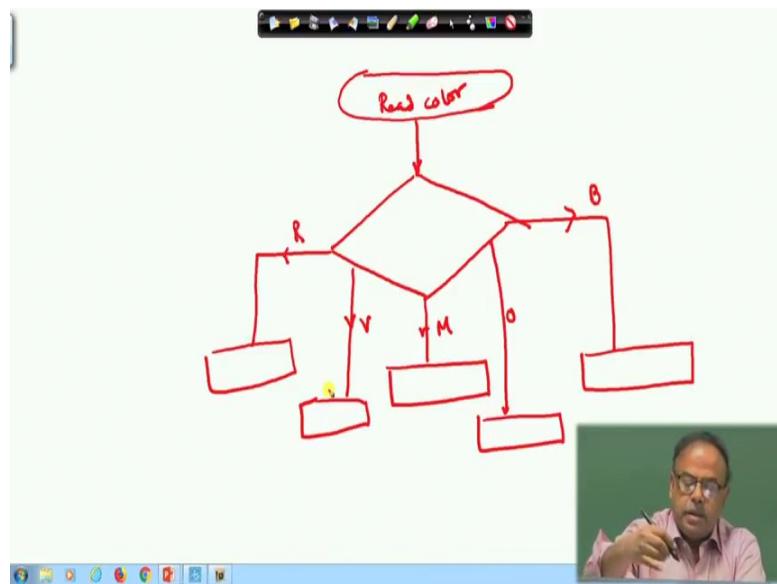
The slide has a light green background with a dark green header bar containing icons. The title 'The switch Statement' is centered in a large, bold, black font. Below the title is a bulleted list of points. At the bottom of the slide, there is a code snippet for the 'switch' statement syntax, followed by a set of small navigation icons at the bottom left and the number '23' at the bottom right.

- This causes a particular group of statements to be chosen from several available groups.
  - Uses "switch" statement and "case" labels.
  - Syntax of the "switch" statement:

```
switch (expression) {  
    case expression-1: { ..... }  
    case expression-2: { ..... }  
  
    case expression-m: { ..... }  
    default: { ..... }  
}
```

Now, we come to a new construct you know that is the switch construct. Now in order to understand this. So, let us quickly go to the flow chart for a while see I draw flow chart like this.

(Refer Slide Time: 15:03)

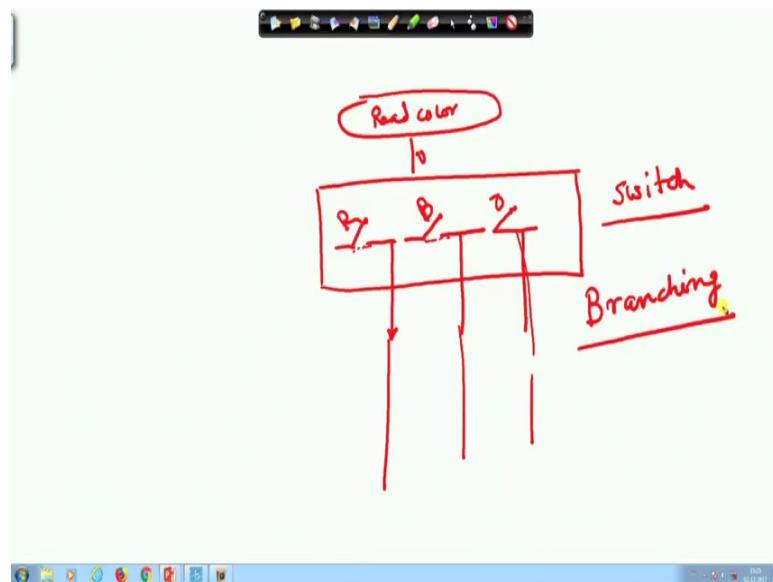


I am evaluating some values here all right or maybe I get some value from the user here, I read some value let me make it clearer.

I read some value from the user all right something read colour now here depending on the colour that the users applies earlier in this decision box whatever you doing? We were either going for true or for false. Now here what we are seeing if the colour is red then we do something, if the colour is blue we do something if the colour is magenta, we do something if the colour is orange then we do something if the colour is violet we do something ok.

So, the same diagram I can draw in a different way that is.

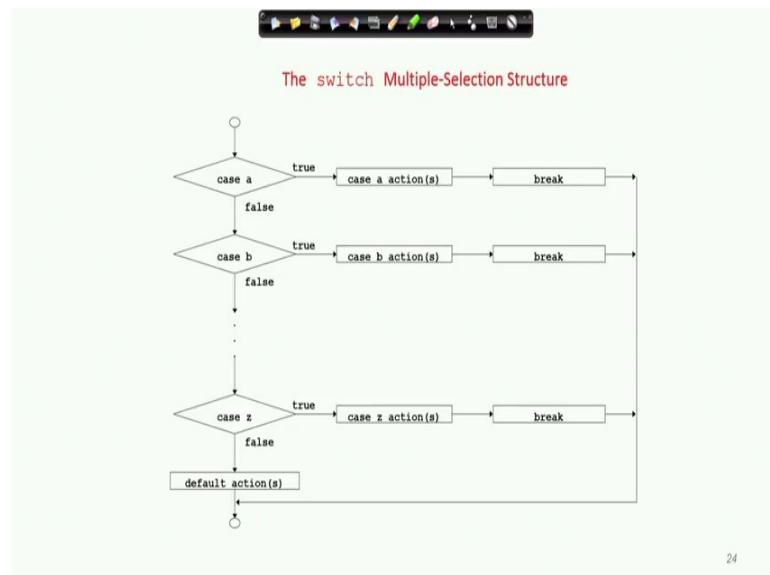
(Refer Slide Time: 16:36)



I am reading colour and then as if I am coming to a switch box, think of this to be a switch box where number of switches are there all right and number of switches are there now depending on what value of the colour is coming. So, this may be the red switch, this will be the blue switch, this one with orange switch, depending on which switch which value is coming. So, let the value be v. If v is r then this switch gets closed and we follow this path. If the v is blue then this switch gets closed and we follow this path. If v is orange then this switch gets closed and we follow this path all right that is why because of this analogy which switch this statement is also known as switch statement, this is also very much use for branching.

Branching we have seen with this else type of thing and here, also we will see another variety of a c construct called switch using which we will do that. So, let us see here now let us look at the construct of this. The syntax of the switch statement is this switch expression then case expression 1 expression 2 let me give an example first and it will be better I will come back to this.

(Refer Slide Time: 18:36)



24

So, here is the total switch structure, it is a multiple selection structure. So, I come to this point switch if the case is a, if it is the case that the colour is red like that if the case is a then I take the case a actions and then come out M otherwise I take the case reactions in the case be b then only I will take the case b action and then come out.

(Refer Slide Time: 19:18)

Example

```
switch ( letter ) {
    case 'A':
        printf("First letter\n");
        break;
    case 'Z':
        printf("Last letter\n");
        break;
    default :
        printf("Middle letter\n");
        break;
}
```

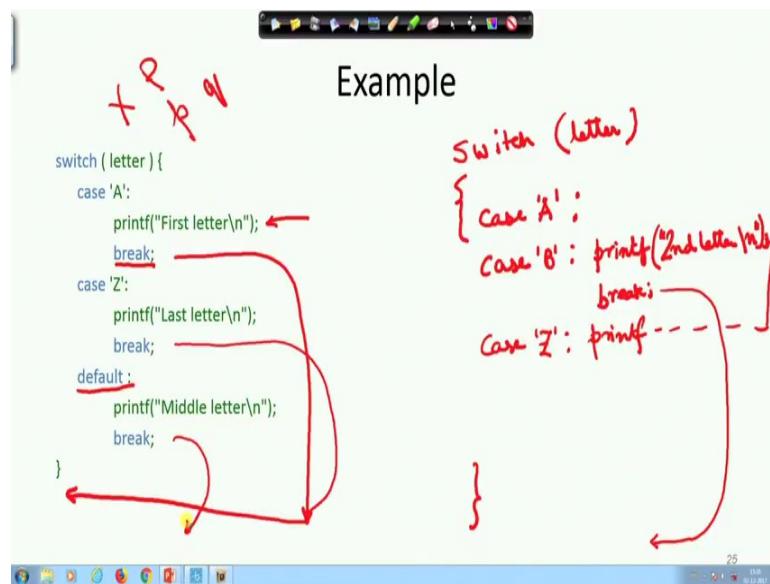
Switch (letter) A - Z  
Case 'A'

A video thumbnail of a man speaking is visible in the bottom right corner of the slide.

Similarly if there may be different cases. So, what we are getting here is an advantage is an example, that we are showing here what we are getting here is using this case construct multiple branching we are accommodating in one shot. Let us look at this I was giving

an example with colour here it is an example with a letter. So, I have read some letter from the user some letter has been now that letter can be anything A to Z say A to Z. So, if it be if the case is that the letter is A. Look at how it has been written case A before that switch what is my variable on which way I am switching? Switch on variable letter.

(Refer Slide Time: 20:19)



In the earlier example it was switch on the variable colour. So, here we start with switch on the variable letter.

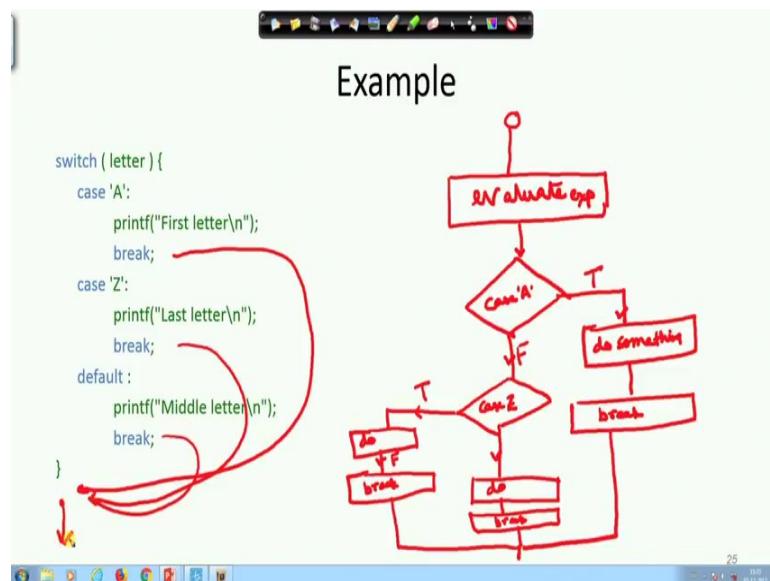
And what are the possible switches I encapsulate that in 2 braces just as we do. Now within this I am taking multiple decisions multiple possibilities are there for example, if the letter is a; that means, if it is the case that the letter is A then we put a semicolon then I write down print the first later. So, what will be printed first later and then there is a break statement what this break statement does? That you execute this and then you come out of this entire switch box come out here; that means join the next statement. If the letter was B then here it is not written I am writing case B printf second letter backslash n and then break; that means, if the case is B then.

I will print this and after that with the break I will come out of this now after that I have not written anything else, but I have written case z only this is case z all right printf last letter as is shown here and then break. Now if somebody this letter was p if this letter was q or some other alphabet sorry I mean anything small alphabet or capital alphabet x whatever in that case what do I do? This one only tells me what I can do if the letter is A

or if the letter is Z or in my case if the letter is B, but if it B something else otherwise then we use this statement called the default statement if anything is other than what has been specified here then we will say that it is some other letter and.

Then break it you see here if the letter is a then only I entered here and then went out if the letter is b then I only entered here if the letter is c then I will come to default if the letter is c it is not specified here I will come here and I will escape I will go right it is an intermediate letter and then come out. So, with this understanding if we can go back to the earlier slide, the flowchart then we can see that the flow chart will be something like this.

(Refer Slide Time: 24:00)



We start then we evaluate expression, we evaluate expression come here its equivalent to case a yes do something do something and then break otherwise it will come here is looking at I am just doing this case z, if true then do something oh and then break right otherwise.

So, this is true this is false, this is true this is false otherwise default I will come here do something and then break now all these breaks all these breaks brings me to the next statement after the case statement. So, anything that is here whenever I encounter a break this comes brings me here this also brings me here, this also brings me here and I continue following the sequential nature of program from here all right.

(Refer Slide Time: 25:43)

The screenshot shows a presentation slide with a title 'Example' and some C code. The code is as follows:

```
switch (choice = toupper(getchar())) {  
    case 'R': printf("RED \n"); break;  
    case 'G': printf("GREEN \n"); break;  
    case 'B': printf("BLUE \n"); break;  
    default: printf("Invalid choice \n");  
}
```

At the bottom of the slide, there are navigation icons and the number '26'.

So, we have seen. So, here is another nice example we can look at it say here switch, I am switching on an expression what is that? Choice to upper now if this has got some multiple parameters into this, let me take it up in the next lecture so that I can devote some time on this.

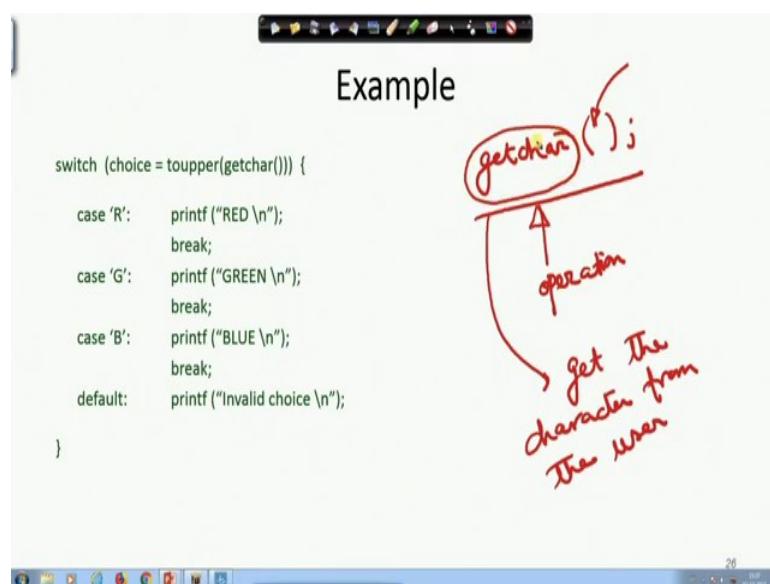
Thank you.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 18**  
**Switch Statement (Contd.) And Introduction to Loops**

So we are looking at a new construct that is switch and case.

(Refer Slide Time: 00:23)



So, here is another example, but this example has got many more things embedded in it. For example, in order to understand this you have to understand two things that are being introduced here, one is getchar. This is a construct we have seen scanf, we have seen printf as input output statements right, but getchar is also another statement another function actually, this is given by you can since we have not considered function in detail there is a built in library function or an operation you can consider that to be an operation also that getchar and then they there is a parenthesis as if we want to have some value here.

It means get the character from the user. So, where from does the user provide you the character whatever the user types, that will be captured by this function and this will get that particular character that is why it is called getchar, all right.

(Refer Slide Time: 01:57)

The screenshot shows a C code editor window with the following code:

```
switch (choice = toupper(getchar())) {  
    case 'R': printf ("RED \n"); break;  
    case 'G': printf ("GREEN \n"); break;  
    case 'B': printf ("BLUE \n"); break;  
    default: printf ("Invalid choice \n");  
}
```

Handwritten annotations in red ink explain the code:

- A large bracket covers the entire code block with the label "toupper ( )".
- "char mychar;" is written below the declaration of choice.
- "mychar='a'" is written below the assignment of choice.
- An arrow points from "mychar='a'" to the call "toupper (mychar);".

That other thing that we have here is another function to upper to upper and then you can see that there is another parenthesis to upper what it does is it converts any character variable to its upper for example, if I type a as a character say I have got a variable mychar. Now suppose I have declared mychar to be a character type of variable and somewhere here I assigned mychar to be a.

Then what does mychar get? Mychar gets a ascii value of small a, now if I say toupper mychar then what it will do? It will take the ascii value of small a and will return me the ascii value of capital A all right to the upper a. So, this one will be returned.

(Refer Slide Time: 03:23)

```
Example
```

```
switch (choice = toupper(getchar())) {  
    case 'R': printf ("RED \n"); break;  
    case 'G': printf ("GREEN \n"); break;  
    case 'B': printf ("BLUE \n"); break;  
    default: printf ("Invalid choice \n");  
}
```

So, here you see to upper what not mychar, but what? We have written to upper then getchar character. So, what it means is that, you will get the character from the keyboard suppose it is small a, and then that will be converted to capital A. So, it becomes capital A and then that is being fed assigned to choice ok.

Let us look at it because it gives us the opportunity to look at nesting of functions also here two built in functions which are being used here to get the character and assign it to choice all right. So, you can also have another example similarly we have got to lower getchar something, where if I had given capital A it will be converted to small a ok.

(Refer Slide Time: 04:47)

```
switch (choice = toupper(getchar())) {  
    case 'R': printf ("RED \n"); break;  
    case 'G': printf ("GREEN \n"); break;  
    case 'B': printf ("BLUE \n"); break;  
    default: printf ("Invalid choice \n");  
}
```

And then I assign it to something else other things also I could do by suppose I getchar, mychar and mychar is assigned a sorry mychar has been assigned a, then I can say again I can assign not mychar.

So, some others mychar and suppose the there was another variable mychar 1 to upper mychar that was also possible of course, mychar one has to be also declared that is not declared here ok.

So, this now that was the first part of the thing.

(Refer Slide Time: 05:53)

```
switch (choice = toupper(getchar())) {  
    case 'R': printf ("RED \n");  
    break;  
    case 'G':  
    break;  
    case 'B':  
    break;  
    default:  
    break;  
}
```

Now, let us look at this. So, we take a choice. So, what is the choice? We are taking a choice and what is the choice; choice is a character that has been given if the user gives a lowercase character then suppose I want a choice between a multiple choice answer scenario, type in the answer a b c. So, somebody can type caps cap a shift a or just a all right whatever you type internally I will convert it to capital.

So, through my; to upper functions. So, now, I take your choice now based on the choice I go on checking is it r if so, print red and then break its not R then let us go and check whether its green G. If it is G then print green and then print green and for break you will come and meet here it is not green also not R not green, I will come to this point is it blue yes then I will print blue, but suppose somebody has printed as typed in give the choice as y all right Y then neither it is R nor it is G nor it is B then we will come to default and we will print invalid choice.

Here since that is a last statement I am closing the bracket immediately after that I may not give the break statement here, I am free not to give the break statement here. Otherwise now suppose let us look at this break statement suppose I do not give the break statement suppose this is not there and my choice is r my choice is small r.

(Refer Slide Time: 07:54)

```
Example
```

```
switch (choice = toupper(getchar())) {  
    case 'R': printf ("RED \n");  
    break;  
    case 'G': printf ("GREEN \n");  
    break;  
    case 'B': printf ("BLUE \n");  
    break;  
    default: printf ("Invalid choice \n");  
}
```

A red annotation on the slide highlights the conversion of the lowercase 'r' in the 'case 'R'' block to a capital 'R'. It also shows the resulting printf call: printf("REI").

So, at this point the small r will be converted to capital R by this statement right by this. So, my choice will be capital R now I go here and I will my system will find that it is r. So, it will printf red I am sorry why should I the printf red will be worked on let me

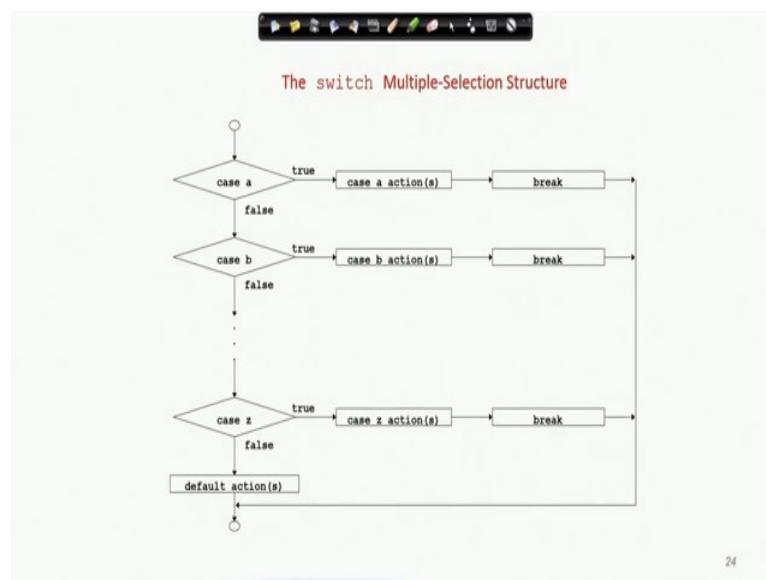
do it again, my choice was r that was converted to capital R and by this and then I come to this point and I find that it is r.

(Refer Slide Time: 08:28)

```
switch (choice = toupper(getchar())) {  
    case 'R': printf ("RED \n");  
    break;  
    case 'G': printf ("GREEN \n");  
    break;  
    case 'B': printf ("BLUE \n");  
    break;  
    default: printf ("Invalid choice \n");  
}
```

So, my system prints red, red has been painted now this break statement is not there. If the big break statement is not there, then it will not go to check this condition it will simply print all these things. Since it has follow this path unless I force it to break it will go on following this path. So, I will have green printed again which is not what I desired right. So, let us see. So, now, let us see here let us look at this flowchart.

(Refer Slide Time: 09:43)



The switch statement if the case is red let us take to the example that we are doing, red then I print red then break. If the sorry if the break was not there then I would be following this path again I mean this path would be followed that is why this break is essential will show that requirement of break in a moment.

(Refer Slide Time: 10:29)

The screenshot shows a presentation slide titled "The break Statement". The slide content includes a bulleted list and a video player interface at the bottom.

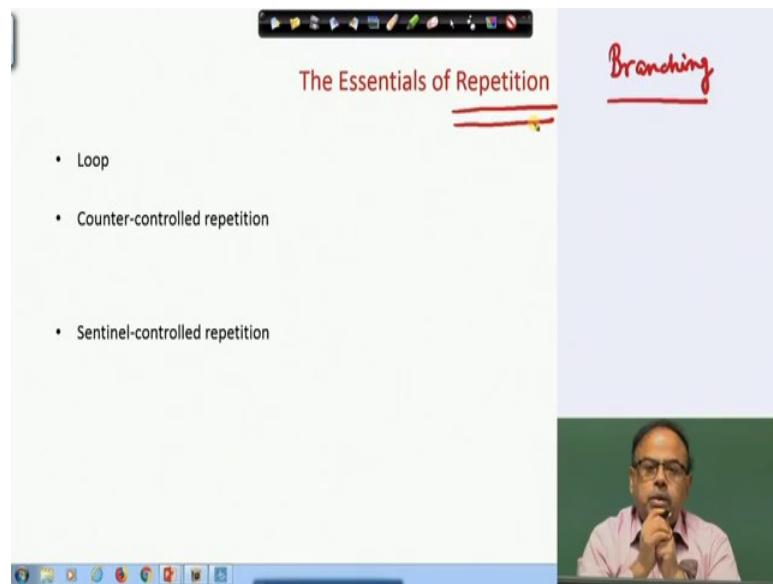
- Used to exit from a switch or terminate from a loop.
  - Already illustrated in the switch examples.
- With respect to "switch", the "break" statement causes a transfer of control out of the entire "switch" statement, to the first statement following the "switch" statement.

At the bottom of the slide, there is a video player interface showing a man speaking. The video player has a progress bar and a toolbar with various icons.

Now, let us look what the break statement is used for. The break statement is used to exit from a switch or terminate from a loop; we have just illustrated it for the switch statement it can be also used for some other purpose.

So, the break statement causes a break statement causes a transfer of control out of the entire switch statement to the first statement following the switch statement.

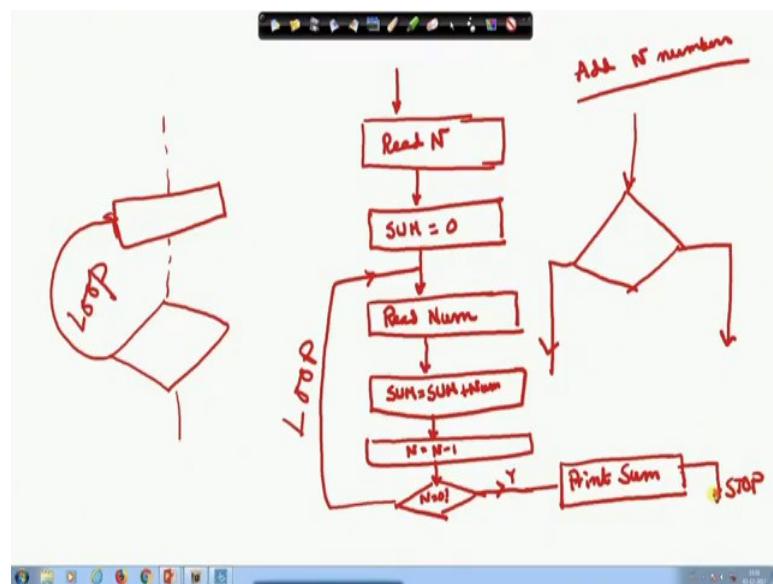
(Refer Slide Time: 11:06)



So, I hope you have understood what is meant by the switch statement how it can be used we will see that, through a number of examples later we will see with the number of examples. So, we will see that the break statement can be used for exiting from loop also we will see that, but what is a loop.

Next we come to see repetition, till now what we have seen is branching, but now we are going to look at another important construct which is repetition. Now just for a moment let us go back to our old friend example that is finding the sum of n numbers for example.

(Refer Slide Time: 12:32)

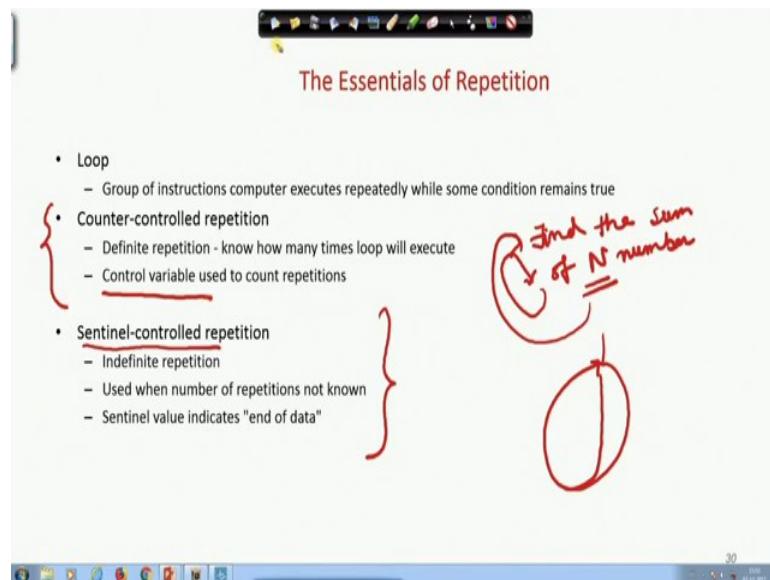


Now, how did you do that what was the flow chart? The flow chart was something like this read N there is a number of numbers all right then we make sum equal 0, I am adding 10 num adding n numbers then read number I am giving a different variable name num, then I am adding that with sum, sum is assigned sum plus num, then I am decrementing N.

Now, I look at is N 0. if yes then I go and print sum all right. So, if there be three then I first read one add it. So, n actually shows how many numbers are yet to be added. So, print sum then stop, but if N is not 0 then what I will do? I will go and continue this path this is another form of structure that we see that we have based on some decision we have got sum statements somewhere, and we are going back to that and repeatedly doing this this is we are repeatedly doing this this is known as loop this is known as repetition or loop all right.

Unlike the earlier case of if then where we look at the condition and then we follow either this path or this path both in the forward direction, here we are also taking up the backward direction. So, this is a very important concept we will need it at every step in our programming exercise. Now there can be different types of loops we will discuss that later, but first let us take.

(Refer Slide Time: 15:43)

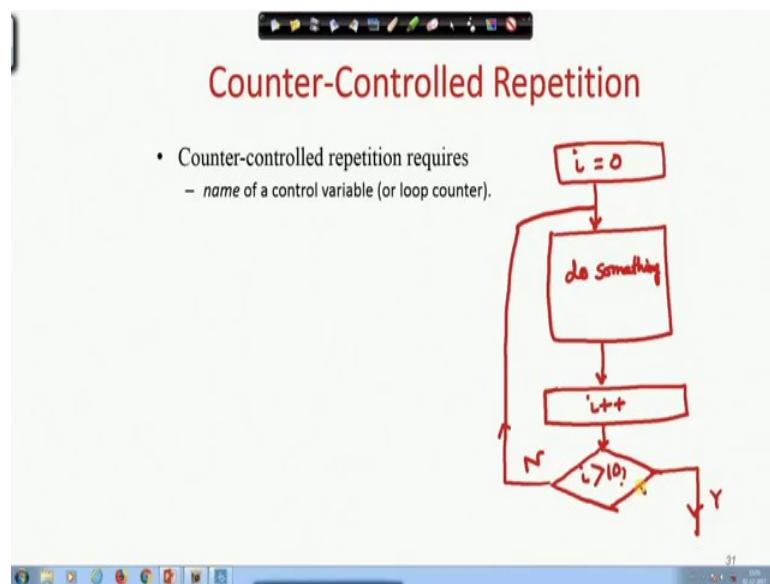


Let us look at a counter now there are three types of loops briefly let me tell you one is loop is something that is repeatedly executed. There can be counter control repetition where we know that that we know before and how many times I must repeat ok.

Suppose I know that find the sum of  $N$  numbers whatever that be  $n$  I know that this will have to be repeated  $N$  times all right  $N$  times, but there is another case where we actually go on repeating a particular set of sentences we go on repeating that until a particular condition occurs; that means, some value has become negative or something that say for example, I am taking the sum I am adding that and the sum goes beyond 9999 and if it goes beyond that I will stop, that sort of situation is known as sentinel control or even controlled repetition this will come later.

But most importantly we will look at this counter controlled repetition where we know how many times we should look and for that we need a control variable we will see what that is.

(Refer Slide Time: 17:38)



So, counter control repetition requires name of a control variable or a loop counter. If you recall in when we added n numbers what was my counter variable? What was my control variable or what was the loop counter, how many times will it do n the number of numbers right.

So, similarly I can have another some other variables say I say I can just draw arbitrary flowchart for you say is and integer initialized to 0, then I come here do something and then i increment i let me use what we learnt I increment i plus plus; then I check, i greater than 10 if it is yes I will again, if it is no sorry if it is no I will again do that if it is yes i will come out all right. So, I am doing it as long as i is not greater than 10.

So, this here what is my control variable? My control variable is i or this is a loop counter based on which how many times I will be carrying it out is determined. Now we also had an initial value of the control variable what was the initial value that I did in this case.

(Refer Slide Time: 19:32)

**Counter-Controlled Repetition**

- Counter-controlled repetition requires
  - *name* of a control variable (or loop counter).
  - *initial value* of the control variable.
  - condition that tests for the *final value* of the control variable (i.e., whether looping should continue).
  - *increment* (or *decrement*) by which the control variable is modified each time through the loop.

*while it is training  
stay home*

```
int counter = 1;           // initialization
while (counter <= 10) {    // repetition condition
    printf("%d\n", counter);
    ++counter;             // increment
}
```

*while  
counter++*

The initial value of i was 0 it was initialized to 0 if you have seen the condition that tests for the final value of the control variable whether the loop should continue or not, what was my condition test in the earlier case? It was i greater than 10 if i is greater than 10 then I will come out otherwise I will go on continuing with the looping and another increment or decrement operation here i was 0. So, I implemented it until it comes to 10 or crosses 10.

If you recall in the earlier example there was n. So, I have to read n numbers after reading one number I decremented n. So, depending on what I want to do I will have to increment or decrement the control variable and based on this condition I will come out. So, here is an example an initialization is done, you can see here a counter has been set to one it is an integer counter that has been set to one while counter is less than equal to 10, while it is a statement that you are getting. As long as counter is less than equal to 10. So, this is one new term you are learning.

While counter is less than equal to 10 do this printf the counter value and then increment the counter and then go on doing it; now this is this really does not make any difference if I had written counter plus plus that we equivalent because these are singleton here there is no assignment or nothing addition with this value. So, its standing alone it really does not mean me and does not matter whether it make it counter plus plus or counter

minus minus. So, this one is. So, here is the condition check counter is my control variable and increment and decrement operation.

Now, you are getting this while statement and now let us try to find the simple English meaning of while. We often write something like this while it is raining stay home; that means, as long as it is raining stay home. So, this is the conditions condition as long as the condition is true do this, here you see while the counter is less than 10 as long as this condition is true do this. If this condition is false then come here do not do this that is the meaning of the semantics of while.

So, while is used for one of the methods by which we can achieve counter control repetition now counter control this we have seen.

(Refer Slide Time: 23:11)

The screenshot shows a presentation slide with the title "while Statement". The slide contains a block of C code. Red arrows have been added to highlight specific parts of the code:

- A red arrow points from the word "while" in the first line of the code to the "while" keyword in the code sample.
- A red arrow points from the opening brace of the inner while loop to the opening brace of the outer while loop.
- A red arrow points from the closing brace of the inner while loop to the closing brace of the outer while loop.
- A red box highlights the entire code block, and a red arrow points from the word "/\*" at the start of the box to the "/\*" at the start of the highlighted code.

```
while (condition)
    statement_to_repeat;

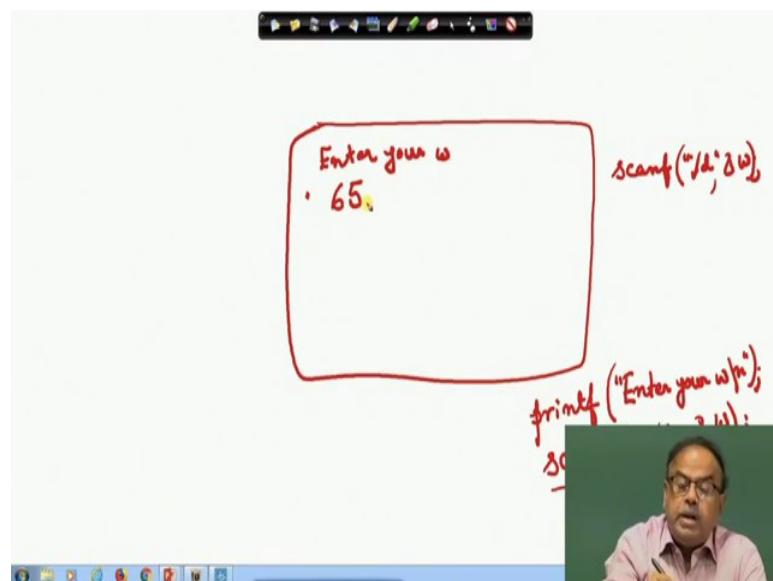
while (condition) {
    statement_1;
    ...
    statement_N;
}

/* Weight loss program */
while (weight > 65) {
    printf("Go, exercise, ");
    printf("then come back. \n");
    printf("Enter your weight: ");
    scanf("%d", &weight);
}
```

So, the while statement actually looks like this, while some condition the statements to repeat it can be one statement or it can be multiple statements as is shown here within this block all right, this entire statement should be this entire thing should be repeated; example here suppose here is a weight loss program I mean as if you have being told to lose weight by following this while weight is greater than equal to 65. While weight is greater than equal to 65; that means, as long as your weight is greater than 65, then printf go exercise ok.

Then come back again enter your weight and then read the weight, now here there is a nice thing often we had encountered the scanf and printf independently. Now look at this statement printf enter your weight scanf weight what does this two together mean.

(Refer Slide Time: 24:31)



I have got my screen here and a program is running and it just says scanf percentage d and w say weight, you do not see anything on the screen, but suppose somebody who is a little more helpful to the user writes printf enter your weight all right. Then here on the screen it is printed enter your weight and then since backslash n is there you are here.

Now, scanf percentage d and w. Now here then when you type your weight say 65 then that is shown here you know what data you are supposed to give. So, this sort of thing makes it more interactive all right. So, here you see these two I have made the program more interactive, but what is the program do? It checks at every point first it checks weight. If the weight is not greater than 65, it will simply come out here because this condition is false otherwise it will carry out all these statements and then it will read the weight again you see again, the weight has been read and after the weight has been read it is coming back here and checking it again whether the weight is greater than 65. If by one days exercise you have reduced your weight then; obviously, you need not exercise more you can come out otherwise you will have to again go and do this. So, while is a very important statement you should understand the meaning of this, and we will try to do a couple of more exercise on examples on this.

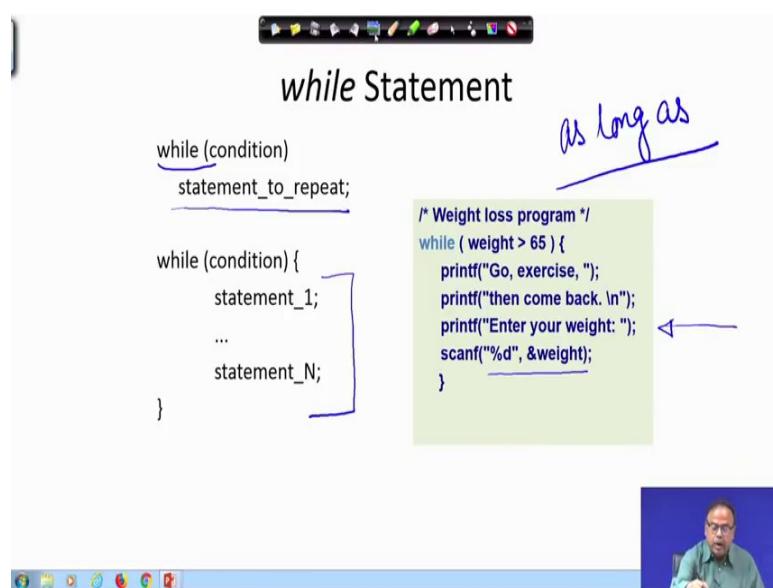
Thank you.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 19**  
**Implementing Repetitions (Loops)**

We were looking at C constructs by which we can achieve repetitions or loops; that means, a set of statements will be executed repeatedly for a fixed number of times ok.

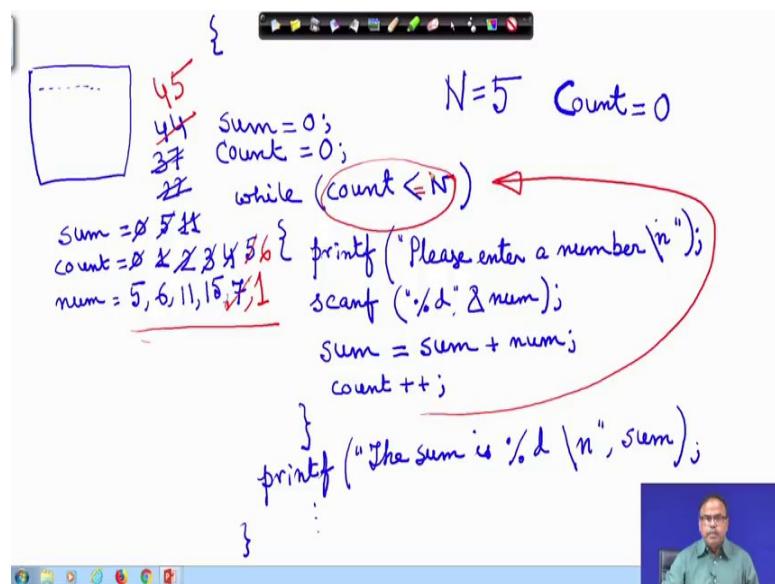
(Refer Slide Time: 00:38)



So, one of such constructs that we came across was the while statement as is being shown here while. And you can see the structure of this that while a particular condition is true, we will carry out a set of statements. It can be one statement or it can be a number of statements as is shown here a number of statements will be repeated as long as this condition is true. An example that we had seen in the last lecture was this that is as long as I can read this while as long as all right. As long as the weight is greater than 65 we will have to do exercise all right, and here you again check the weight and if it is greater than 65 you will carried out that is the structure of the while statement.

Now, how can you use it fruitfully? Let us take an example of using the while loop for some meaningful computation. For example, I want to add 10 numbers the example that we are doing earlier we had shown the 0 count for that. So, I have got a value N.

(Refer Slide Time: 02:05)



Let us see let us do it for a small n number N equals to 5. So, I will carry out the sum of 5 numbers. So, I will have a count another variable count which may be initialized to 0 ok.

Now, we can write something like. So, initially count is 0, count is an integer, count is 0 while count is less than N and also I do another thing I am going to add 5 numbers. So, I create another variable sum. So, sum is 0 and count is 0. So, while count is less than N, I can I write a complete program; now printf. Please enter a number and then. So, on the screen I will have please enter the number printed here, then I am doing scanf percentage d and num, num is the variable which I am reading, and then I am adding updating sum initially sum was 0. So, sum plus num all right I do this and then I how many numbers I have read I have read one numbers. So, I will do count plus plus; that means, count is now 1.

So, one number has been read. So, I will be completing here, now let us see what will happen. Suppose M was 5. So, sum is 0. So, the first number has been read. So, count was 0 please enter a number and some number has been entered and suppose that number was 5, then sum is sum plus number. So, this one becomes 5 then I increment count. So, count becomes 1. I come back here, I find count is still less than 5, because I am going to n is 5 right.

So, count is still less than 5 I again do that suppose I did the second number 6. So, then this becomes here I add this becomes 11 and count becomes 2. Please note that count in this case is therefore, keeping a count of how many numbers I have already read because I am reading the number here and then incrementing count. So, count now I have read 2 numbers and I have added the sum ok.

So, now after doing this count is becoming 2 average, 2 numbers I again go here now 2 is less than 5 I take another number say 11. So, I add sum. So, it becomes 22 and I increment count 3. I have read 3 numbers I go up again here and read the forth number 15 add that with sum. So, 22 plus 15 will be 37 and then count will be upgraded it will become. So, I have read 4 numbers 5 6 11 15.

Now, I again go up go up here and check count, count is 15 yeah sorry count is 4 which is less than 5. So, I read another number suppose that is 7 I come here add it, it becomes 44 and I increment count I first add that number. So, it becomes 44 and then I increment count. So, count becomes 5 meaning that I have already read 5 numbers.

Now, when I go back here, I check is count less than N? No therefore, I will come out of that and maybe here I will write something like printf the sum is assuming integers is percentage d back slash n and here I print sum and whatever I do here are the other components of the complete program declarations integers and all those things are there all right.

Now, point to be careful about, I must be very careful about expressing the condition. Look here if instead of this I had if instead of this I had made it less than equal to n then what would have happened? If instead of countless than n, if I had written count less than equal to n what would have happened? After count is five; that means, I have read 5 numbers have added 5 numbers and 44 is my result I would again go back here, and find that count is less than equal to 5 count is 5 therefore, this condition would be true, I would again come and would have read another number sorry I should cut it out. I would have read another number and then count would be incremented 6, I would have 45 which would be along result because here actually I have read 6 numbers. So, you must be very careful to specify this particular statement.

So, that the number of times you want the loop to work should be accurate should be correct. So, I think you have understand this example, a very simple program, but some theory is needed about this. So, that was an example of while statement.

(Refer Slide Time: 10:15)

```

while (condition)
    statement_to_repeat;

while (condition) {
    statement_1;
    ...
    statement_N;
}

int digit = 0;
while (digit <= 9)
    printf ("%d \n", digit++);

```

The slide illustrates the execution of a while loop. A green box highlights the initialization and condition of the inner loop: `int digit = 0;` and `while (digit <= 9)`. A red box highlights the `printf ("%d \n", digit++);` statement. Handwritten annotations show the value of `digit` being tracked: `digit = 0 1 2 ... 8`, with `9` crossed out, and the output of the loop as `0 1 2 ... 8`.

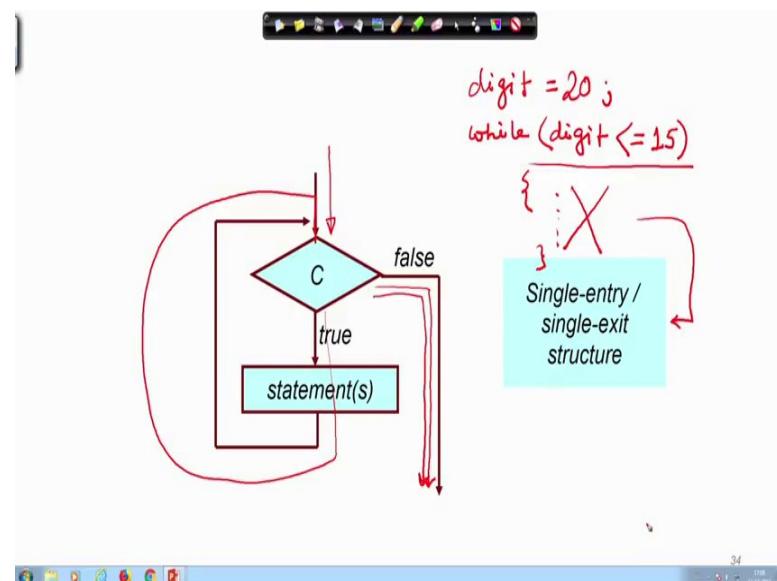
So, here is another example suppose the digit is 0 what would this one do can you find out what this will do? This is a while statement. All of you try to look at it and find out what this loop does this while loop does. If you look at it carefully you will see that I am starting the digit I have initialized digit to be 0.

Now, while digit is less than equal to 9, first time I come here it is less than equal to 9. So, what will it do careful it will come to this `printf` statement and we will print digit now it is a post increment or pre increment it is a post increment; that means, first digit will be printed. So, 0 will be printed then digit will be incremented. So, this digit will now become 1 I again go back here and find that digit is less than equal to 9 right. So, I will come here, print the digit 1 and then again increment digit, digit will become 2 I go back here check whether it is less than equal to 9, it is still less than equal to 9, I will come inside the loop will print the digit 2 in that way it will go on ultimately 8 will be printed and 8 has been printed, and I go back after printing 8; this has been incremented to 9, I go back here.

Now, you see here less than equal to 9 therefore, I will still executed; that means, I will print 9 and then increment it, it will be 10 and then when I go up here this condition is no

longer true therefore, I will come out of this loop. So, what will be printed 0 to 9 the 10 numbers will be printed. So, that is how the while loop works.

(Refer Slide Time: 13:05)



So, this flowchart is very important to remember what while does you have actually understood it by now, but the key point to note the most important point to note is this, that when I am executing the program in this direction, I first compute the condition. First I compute the condition and if the condition is true I execute the body of the loop then I again go up check the condition and this will be going on.

So, first the condition is checked if it is false I will go out of the loop. So, if initially say for example, I write something like this say digit, is equal to 20, and I start something like while digit is less than equal to 15, some things I will be doing. Now in this case when it comes to this condition at the very beginning it will fail, because this condition is not true. So, it will come out to this false path and this loop will not be executed, will be just coming to the next statement after the loop.

So, in while the condition is first evaluated and depending on the success of the test of the condition, will either enter the loop or will bypass the loop. So, this flowchart is very important.

(Refer Slide Time: 15:04)

The slide has a title 'do-while Statement' with a red underline. Below it is pseudocode:

```
do {  
    statement-1  
    statement-2  
    .  
    .  
    statement-n  
} while ( condition );
```

To the right of the pseudocode is a code snippet in C:

```
/* Weight loss program */  
do {  
    printf("Go, exercise, ");  
    printf("then come back. \n");  
    printf("Enter your weight: ");  
    scanf("%d", &weight);  
} while ( weight > 65 );
```

A callout box contains the text: 'At least one round of exercise ensured.'

At the bottom right is a small video frame showing a person speaking.

Now, we will be contrasting while statement with another statement which is the do while statement what is the difference looks like very similar. So, there is a scope of confusion while and do while. So, as the name implies here do comes first do something and then check the condition. What it means is say do some statements here while condition will do all these statements and then check the condition.

So, let us see what will happen here, the weight loss program again really with while is do go exercise. So, we start with a do you do, go exercise printf comeback whatever whatever, then read the weight and while weight is 65. So, at least the condition is being checked here the condition is being checked here. So, it will at least carry out the computation once.

(Refer Slide Time: 16:33)

The image shows a video call interface. On the left, there is a whiteboard with handwritten red text. On the right, a video feed of a man speaking is visible. The whiteboard contains the following C code:

```
digit = 20;
while (digit <= 15)
{
    digit++;
    printf ("%d\n", digit);
}
```

So, the digit thing if we do suppose I write something like this. So, digit is 20, and I write while digit is less than equal to 15 say I do digit plus plus, printf digit. Now in this case since the digit is 20 what will be printed here nothing all right or say let me make it even simpler.

(Refer Slide Time: 17:44)

The image shows a video call interface. On the left, there is a whiteboard with handwritten red text. On the right, a video feed of a man speaking is visible. The whiteboard contains two versions of the C code separated by a vertical line:

Left side (labeled 20):

```
digit = 20; ✓
while (digit <= 15)
    digit++;
printf ("%d\n", digit);
```

Right side (labeled 21):

```
digit = 20; ✓
do
    {
        digit++;
    } while (digit <= 15);
    printf ("%d\n", digit);
    }
```

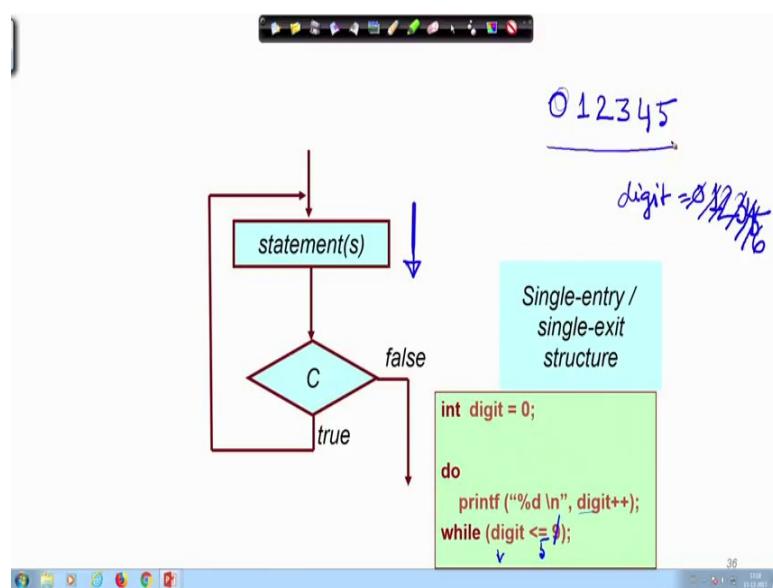
Let me make it a little different digit is 20, while 15 digit plus plus printf percentage d back slash n digit. If I do this since digit is 20 what will be printed here 20 will be printed because this plus plus will not be executed right because here it will feel.

But if I do it now if I do it as digit equals 20, do I need to give the bracket, I may or may not, but let me give it it really does not matter digit plus plus; while digit is less than 15 less than equal to 15 and here I write printf percentage d back slash n digit all right what will happen let contrast this here also digit is 20 here also digit was 20.

Now, here it was tested at this point. So, this digit plus plus will not be executed, it will straight way come here and will print digit it will be 20 whereas, in this case I first encountered this do, and as I do I check this digit I make this digit plus plus; that means, it becomes 21, and then I check while digit is less than equal to 15. Now I find digit is not less than equal to 15 of course, I will not do it again, but once I have already done it. So, when I take the printf digit will be printed to be 21.

So, here is the difference between while and do while. So, these are 2 constructs by which just 2 examples constructs by which we can carry out the loop. So, at least one round is carried out. So, if I take if I look at the flowchart, it will be looking like this it will have a set of statements which will be executed at least once whatever the condition b condition s. So, this will be executed first it will be first executed here.

(Refer Slide Time: 21:16)

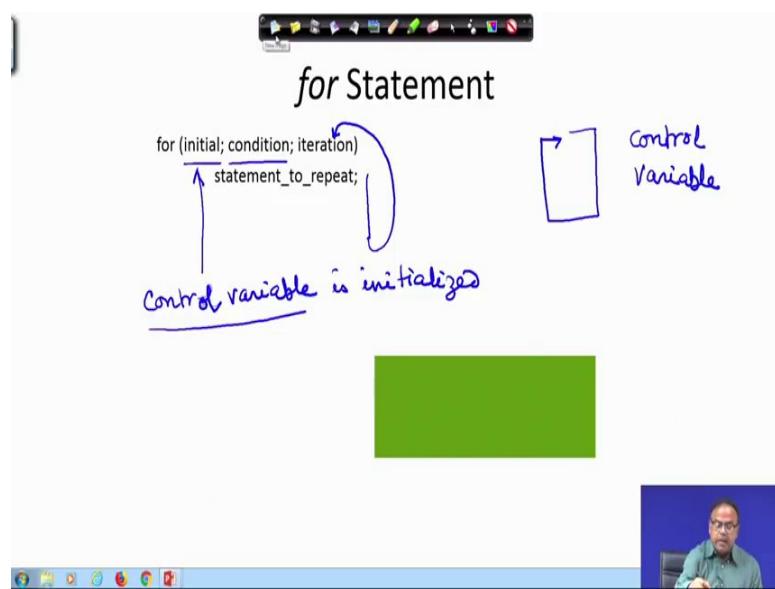


And then we check the condition if it is false I will not do it again, I will come out otherwise I will repeat. So, here again that old example digit was 0, do printf digit plus plus while digit is less than 9. So, you remember in the earlier case we had printed 0 1 2 3 4 5 6 7 8 9; now what will be printed in this case let us see digit is 0 initially.

Now, we come here first we print digits. So, 0 is printed, 0 is printed then digit plus plus is done. So, digit is 1 right. So, digit now becomes 1 digit was 0. So, it becomes 1 and then I check is digit less than 9 yes it is less than 9. So, I again print digit one is printed let us make this 9 let us say that is 5 all right let us say it 5. So, we have done made it one and after printing one digit has been made 2 plus plus has been done, I check again it is less than 5. So, I will do. Go to the loop again I will do it again. So, I will be printing 2 and then make digit to be 3 it is still less than 5, I will go again I will print 3 and then I make digit to be 4 still less than 5, I print 4 I make it 5 all right less than equal to 5 right. So, I print 5 print 5 I first print 5, then increment it to 6 as I do it I increment to 6 and then I check whether it is less than equal to 5.

So, if it is since it is less than not less than equal to 5, I will come out of the loop. So, what I will be printing is 0 1 2 3 4 5. So, this is an example of while do while next.

(Refer Slide Time: 23:55)



So, this is one type of statement that we have encountered here, that is while and do while. Here we are not specifying any number of times that it will be doing; as long as a condition is not met I will be doing it.

Now, how is it that the condition is being affected it is through the computation insides. So, for example, the computation here this is plus plus, that is being done by that this condition variable is changing all right there are cases where I know a priori beforehand

I know beforehand that I will have to carry it out 10 times I will have to carry it out 20 times so and so forth ok.

So, for that another construct is very important and is used in C language that is the; for construct. Welcome to the; for construct this a little complicated, you should be attention there 3 components of this for one is an initial a value, now this initial value can be assigned to a control variable. So, there is a loop program and how many times it will loop. How many times it will loop that is being determined by some control variable. In the case of while what was our control variable? Say for example, in the earlier is a case of earlier example the value of digit was the control variable. Here we put some control variable and initialize the control variable with some value; here some control variable is initialized and then we have a condition. If that condition is true, then I will enter the loop and do that and after doing this loop I have got some iteration parameter by which we update the control variable and check the condition in that way it will go on.

So, let us have a look at the structure in a little more detail.

(Refer Slide Time: 27:13)

The slide title is "for Statement". It shows the general structure of a for loop:

```
for (initial; condition; iteration)
    statement_to_repeat;
```

Below this, a detailed example is shown:

```
for (initial; condition; iteration) {
    statement_1;
    ...
    statement_N;
}
```

Handwritten annotations explain the execution of the loop:

- A vertical column of numbers 10, 10, 5, 1, 2, 3, 4, 5 is written to the right of the loop structure.
- A circled 1 is underlined with a blue arrow pointing to the first iteration of the loop.
- A green box highlights the loop body: `fact = 1; // Calculate 10!` and `for (i = 1; i <= 10; i++)`.
- Inside the green box, a blue arrow points from the handwritten 1 to the `i = 1` in the loop condition.
- Below the green box, a blue arrow points to the handwritten 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, which are listed in a sequence.
- A small video player window in the bottom right corner shows a person speaking.

See here we have got some initial value expression some condition and iteration and we have got some statements which will be looping. Here is an example say I am calculating the factorial of 10. So, I am trying to compute the factorial of 10, some people write it in this way some people write it in this way all right.

Now, here what is being done I will explain it in a little bit more detail, fact is a variable which is standing for factorial it is initialized to 1. Now you know factorial of 5 is what? 1 times 2 times 3 times 4 times 5 right. So, I am started with some variable and multiplying it with its successor, and then the product I take and multiply with the successor of the last integer and in that way it goes on. So, here is a look of how it will look like we have got a for statement here, and I am saying for I assigned one now this is what I was talking about I is a control variable here that is being initialized to 1.

So, that is less than 10 of course, because here I am trying to compute factorial 10 and then what I do fact was 1. So, fact will be fact multiplied by i what was i? I was 1. So, it is one times 1, then after this operation is done I go and do this iteration operation iteration operation what it is doing it is incrementing i and i is becoming 2, and then before I enter I just check whether it is still less than 10 yes it is less than 10, I will again do that. So, fact will be it was 1, 1 times what is i now 2, 1 times 2 after I compute 1 times 2 I will increment i. So, that will become now 3 it is still less than 10 please follow my pen it is still less than 10, I go in compute what was I three. So, I whatever was fact was the product of 1 and 2. So, I multiplied that with 3, and then I increment the iteration variable it becomes 4 again I come and check is it less than 10 yes it is I go on here and I multiply it in that way it goes on and on.

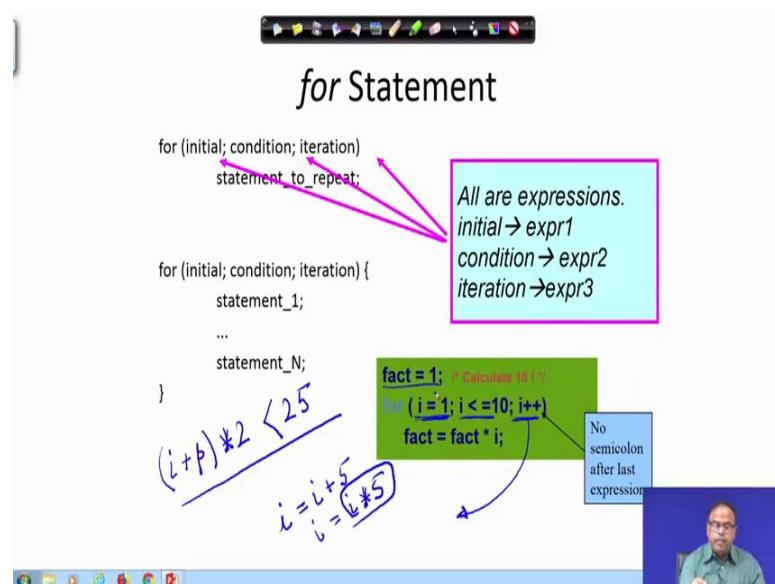
Now, ultimately it will be ultimately it will be like that 5 6 7 8 now suppose i is 9 sorry suppose i is 9. So, I come here I multiplying fact with 9 i plus plus. So, i becomes 10, i come here, i is still less than equal to 10 is it is equal to 10, I come here multiply 10 i plus plus it becomes 11, I come here this condition is not satisfied I come out all right, this is the for statement will look at it in the little more details in the next class.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 20**  
**Implementation of Loops with for Statement (Contd.)**

We were looking at new construct for building up loops in C language and that construct is for construct.

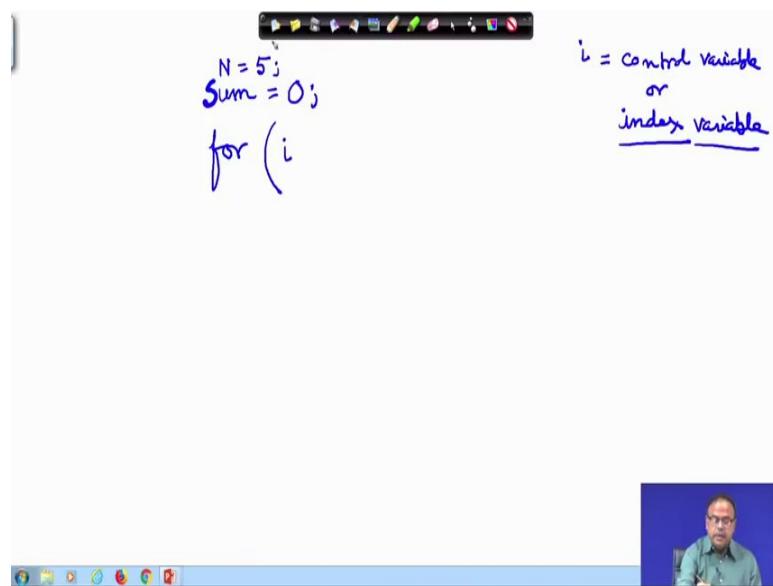
(Refer Slide Time: 00:26)



So, let me first write a simple program again the same program that we are writing for reading for finding the sum of 10 numbers. So, just I am writing and you try to follow what the meaning of this program is, then we will go and further explain it. My intention is to read n numbers and find their sum the simple thing, how can I write that.

So, I put a variable again sum is 0 and I have read some value n whatever that value is.

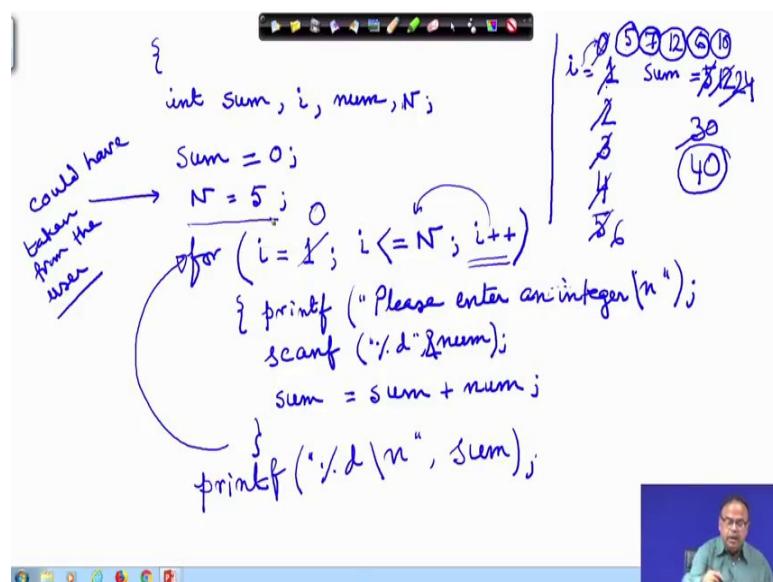
(Refer Slide Time: 01:04)



Suppose say n equals 5 user has provided me or I can initialize it to 5. So, 5 numbers I will read and add. Now, I write for i. What is this i? i is an control variable or we also call it as very common name of this is an index variable, index variable. It need not be i it can be anything, but an integer variable. So, that has to be declared at the beginning.

Si, I can declare let me start with normal declarations int, some program has start at main int sum i and the number that I will read, I will see if I need any other variables later and also n sorry and also n.

(Refer Slide Time: 02:32)



And here I do sum 0 n equals to 5. Now, this thing I could have read from the user, could have taken from the user the value. How could I take it from the user? By doing a scanf, scanf printf please enter the number of numbers you want to add then scanf percentage d ampersand n in that way I could read this But here for the sake of saving time I am just initializing it N is 5.

Now, what I do for i, this i equals 1 semicolon. So, this one complete statement i less than equal to N semicolon i plus plus. What does it mean? We will see what it means then what would I do I will printf. Please enter number let us say integer I call it please enter an integer scanf percentage d num ampersand num. Then sum assigned sum plus num that is all.

Here I do not have to increment this control variable why, after I add the number that has been read here after I add that with sum I am going back to this for loop and where am I going to, I am now doing this operation that is incrementing i. So, i will become it was i was 1, so i will become 2 and then again I add the number. After I make i equal i increment i, initially i was 1 here, so what is happening. Let us look at what is happening to the control variable of the index i was 1, I do it once. So, the sum becomes number suppose the number was 5 suppose the number was 5 that was entered. So, the sum becomes 5, I increment i so i becomes 2, I check it is less than 5 less than this n. I again read another number suppose I read 7 as the second number.

So, sum becomes 5 plus 7 12 and then I go back here increment it, it becomes 3 all right. I check again after doing this I come here check it is still less than 5, so less than n I again do this I need another number suppose it is 12. So, 12 plus 12 it will be 24. Here increment this, it becomes 4, i still go over there still it is less than 5. I read the next number which was 6 say, so sum becomes 30 here. Then i is incremented to be 5, i again come here and check that is less than equal to n true. So, I come here I read another number say 10 sum becomes 40, come back here i become 6, i come at this point this condition is not true therefore, i will come to this point where maybe I will be writing something like printf percentage d backslash n sum. So, the sum will be printed at this point 40.

Now, you see I have read how many numbers 5 7 12 6 10, 5 numbers and that is what I wanted to do. Now, I can look at this. Now, I could have done several this thing in a

different way also. For example, if I had just instead of this  $i$  equal to 1, I make it I initialize this to be 0 all right, I initialize  $i$  to 0 and keep everything the same. What will be changing?  $N$  is 5, so here  $i$  will start not from 1, but from 0. So, first will be 0 then it will be 1, then it will be 2, then it will be 3, then it will be 4, then it will be 5. So, ultimately how many numbers would I read? 6 numbers. But I was actually trying to read 5 numbers. I am sure you are confused. So, let me show it through another example.

(Refer Slide Time: 10:30)

Read 3 numbers

```
for (myIndex = 0; myIndex < 3; myIndex++)
    scanf ("%d", &num);
```

	myIndex	num
0	0 1	15 ✓ ✓
1	1 2	5 ✓
2	2 3	14 ✓ ✓
3	3 4	X X

Suppose I read, I want to read 3 numbers. If I write it in this way for  $i$ , now by the way it need not be  $i$ , I could have written declared it properly and I could have taken any variable to be my index, I could have saved that it is my index. But the only constraint is that this must be an integer variable. My index is 0, note the semicolon here because this is one statement then I write my index less than equal to 5 say less than equal to  $i$  am trying to read 3. So, my index is less than equal to 3 semicolon again and then my index plus plus and I just read the numbers. So, I am dropping off the printf just writing scanf percentage d and num.

Now, I since its only one statement I can simply remove this parenthesis. Now, let us see I wanted to read 3 numbers. Now, what will happen? Here I write my value the value of my index and the value of num. My index has been initialized to 0 and I check with my index my I check with this statement, it is less than equal to 3. So, I read the number

suppose the number is 15, I increment my index. So, my index now becomes 1, my index becomes one still less than equal to 3, I read another number 5.

Next I come here. So, I do this and come here and my index becomes 2, still less than equal to 3, I read the number say 14. I again come here my index becomes 3 less than equal to 3, I read another number 5. I come here my index becomes 4 I compare this condition is not satisfied. So, I come out of this loop.

But in the process how many numbers have I read? I have read 4 numbers 1 2 3 4. But what was my intention? My intention was to read 3 numbers. So, where did I go wrong where did my logic go wrong. This is what as a programmer you must be very careful and cautious about. Where did I go wrong? You can say that I have gone wrong in either of the 2 places, one is I could have simply initialized my index not with 0, but I could have initialize it to 1. In that case what would have happened, first this part we forget we will start with my index one, read one number less than 3. So, then become 2 less than 3 I read the other number increment it, it becomes 3 still less than equal to 3 I read this number, whenever I come to 4 then my index becomes 4 this condition is violated. So, I will not be read in this number it will be all right.

Otherwise, another thing I could have done what could I have done? You must have discovered it by. Now, that suppose I had kept my index to be 0, I decide no I like this circular figure 0 very much, so I keep it like that. Then what should I do in order that I can still be logically correct? I would have changed this condition from less than equal to to less than, then let us see what would have happened. Then also my index starts with 0 I read one number incremented. So, it becomes 1, I check for the condition my index is still less than 3. So, I read the other number 5 my index is incremented 2 still less than 3. I read the other number 14 fine as soon as I after I read I make it 3 incremented and I check the condition. Now, it is no longer true it is not less than, but equal to, but less than equal to is not my condition my condition is less than therefore, I will not read the forth number.

So, this is a point where often people make mistakes while writing for loops. So, I encourage all of you to very carefully study this we look into the for loop a little bit more. So, we have seen the initial. So, I have given you some example one point that is very important you must have noticed while I was writing this that there is no semicolon

at the end of this statement why, because the entire for statement has not ended here it is going on for this period to this, that is the end of the statement. So, no semicolon is given for this for conditional part all right. And here all these are expressions you can see that this I am sorry what happened. You can see that this is an expression sorry this is an expression, this is another expression, this is another expression.

Now, since these are expressions it can be very general for example, I could have written initialization is fine, but here I could have done something  $i + p \times 2 < 25$  that is very much valid and here also I could have done  $i \leq 5$  anything. This is a modified modulator I am changing the condition and changing the index variable and then testing that index variable with respect to a condition. Only point that you should remember is that that this must be an integer variable.

(Refer Slide Time: 19:44)

A screenshot of a presentation slide. At the top, there is a hand-drawn note in blue ink:  $(i=1; i \leq 5; i=i+2)$ . Below the note is a bulleted list of points about how a for loop works. To the right of the list is a code snippet in a light green box:

```

int digit;
for (digit=0; digit<=9; digit++)
    printf ("%d\n", digit);
  
```

**• How it works?**

- “expression1” is used to initialize some variable (called index) that controls the looping action.
- “expression2” represents a condition that must be true for the loop to continue.
- “expression3” is used to alter the value of the index initially assigned by “expression1”.

So, how it works? The expression one that is say typically  $i = 1$ , the expression 1 is used to initialize some variable called index that controls the looping action. Expression 2  $i \leq 5$  represents a condition that must be true so that the loop continues. An expression 3 say  $i = i + 2$  all right that is the bracket,  $i = i + 2$  or  $i += 2$  as you are seeing till now, they are used to alter the value of the index initially assigned by expression 1. We have seen this, so not much to worry about it.

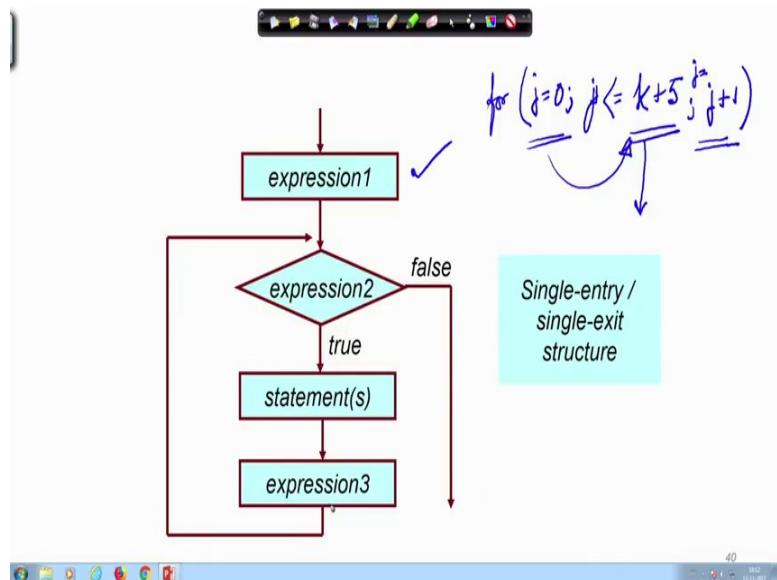
(Refer Slide Time: 20:52)

- How it works?
  - “expression1” is used to *initialize* some variable (called *index*) that controls the looping action.
  - “expression2” represents a *condition* that must be true for the loop to continue.
  - “expression3” is used to *alter* the value of the *index* initially assigned by “expression1”.



So, expression one is used to initialize, so here.

(Refer Slide Time: 20:58)



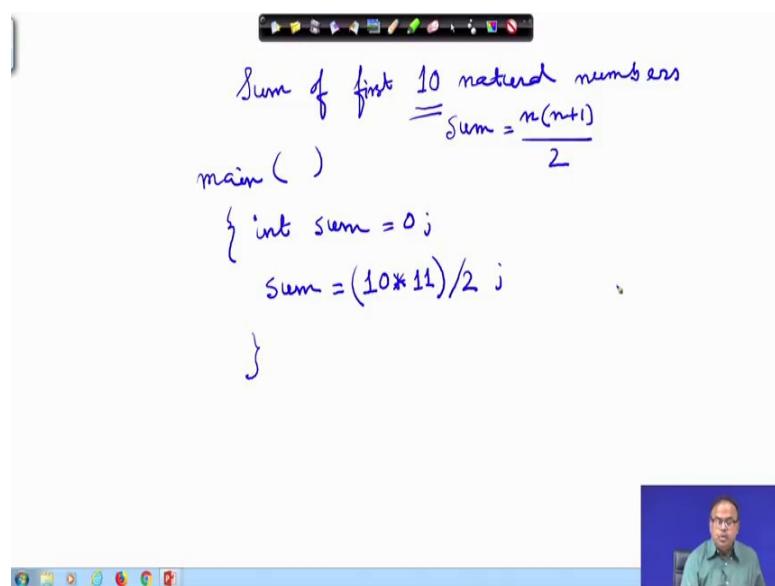
So, what is the way it is done is first expression 1 is executed. So, for j equals 0, j less than equal to k plus 5 can be anything it can be an expression j plus plus. So, first this expression is executed then after we execute this, I check by initialization have I violated the condition in that case of course, there is no point getting into the loop I will go out.

Otherwise if it is not done then I go inside the loop carry out the loop and then instead of going into the loop straight back I will first come to this alteration or modulator

statement. So, it is I am sorry what did I write here. I wanted to write j assign j plus 1 here or j plus plus whatever. I do that and after I do that what do I do, I immediately come back to this expression and test it again is it true, if so I will get into otherwise I will exit. So, this is the structure of the for statement.

Now, there are a couple of critical issues, but before that let us write a simple program with for. Suppose I want to find the sum of first 10 natural numbers.

(Refer Slide Time: 23:01)



What are the first 10 natural numbers? 1 2 3 4 5 up to n. So, in school mathematics you know that the sum for 10 numbers it is the formula is n times n plus 1 by 2 right. So, one simple program is if I want to find the sum of n numbers you can simply do in your main int sum assign 0 and then in one short you can write sum assign 10 because I want to write 10 natural numbers, 10 times 11 divided by 2 and then printf sum, that could be one way. But that is not what I am going to do. I want to illustrate the for loop.

(Refer Slide Time: 24:33)

The screenshot shows a video call interface. The main window displays handwritten C code for a main function. The code initializes a sum variable to 0, declares a myindex variable, and uses a for loop to iterate from 0 to 10. Inside the loop, it adds myindex + 1 to sum. Handwritten annotations explain the calculation:  $\frac{5 \times 11}{2} = 55$ ,  $myindex = 1, 2, 3, 4, 5, 6$ , and  $sum = 1, 3, 6, 10, 15, 21$ . A small video window in the bottom right corner shows a man speaking.

```
main ( )  
{ int sum = 0;  
int myindex ;  
for (myindex = 0; myindex < 10; myindex++)  
    sum = sum + myindex + 1 ;  
    1  
    3  
    6  
    10  
    15  
    21  
    sum = 1  
}  
1  
3  
6  
10  
15  
21
```

So, the way I can do it is main main function and here I put in int sum equals 0. Now, I know that it is not I am not going to find the sum of n natural numbers I am going to find the sum of 10 natural numbers. So, I can write it in this way for. So, in sum equal to 0 another variable I have to initialized that is my index I write it int my index not necessarily i or j all right, but it must be an integer. So, for, but I am not initializing it here, for my index assigned 0 my index less than 10, please verify whether I am right or wrong my index plus plus. Sum assigned sum plus my index plus 1 what will happen? Sum was 0, my index was 0, so my index plus 1, the sum will be 1. What am I expecting? 10 times 11 by 2 right. So, that will be 55 right, sum of first n natural numbers.

Now, my index becomes 1. So, my index is 1 now, less than 10 I again add that. So, sum was what was my sum? My sum was 1. So, here is first iteration sum was 1, then sum was 1, I have written it here. So, sum plus my index my index was 1. So, 1 plus 1 am I right. So, here it was right. Now, sum was 1 and my index is 1 plus, 2, 1 plus 1. So, it will be 1 plus 2, this will be added so the sum will be 3.

Next my index will be implemented to 2. So, now, sum is 3 my index is 2 plus 1, so 3 plus 2 5 and 1 so sum will be 6. In that way it will go on and ultimately I will come to this point where it will exceed 10 and then I can stop.

So, here is now, you can also do that, some of natural numbers you can do that all in a loop using the very the index itself being updated and creating the different natural numbers every time.

(Refer Slide Time: 28:58)

The slide contains handwritten notes in blue ink. At the top, there is a for loop definition:

```
for (i = 1; i <= 5; i++)
    sum = sum + i;
```

Below the loop, there is a table showing the values of  $i$  and  $sum$  at each iteration:

$i$	$sum$
1	1
2	3
3	6
4	10
5	15

To the left of the table, there is a calculation:  $\cancel{5} \cdot \cancel{4} \cdot \cancel{3} = 15$ .

At the bottom right of the slide, there is a video player window showing a person speaking.

So, if I had written it in this way for  $i$  equals 1, there is the first one less than 10 less than 10 would that be  $i$  plus plus sum equals sum plus  $i$ . What would have happened?  $i$  is 1, sum is 1 right.

Now, next  $i$  is incremented  $i$  is 2 and then sum was 1 so that will be added sum is 3. Again this will be added 3 sum is 4 sum is 3. So, then be 6 again 4 then be 10 in that way it will go on ultimately. So, suppose I was trying to do it for 5 numbers. So,  $i$  just  $i$  less than 5 first 5 natural numbers. So, sum is 5. So, now 4, I have already taken. Now,  $i$  plus plus it becomes 5, but I will get stuck here.

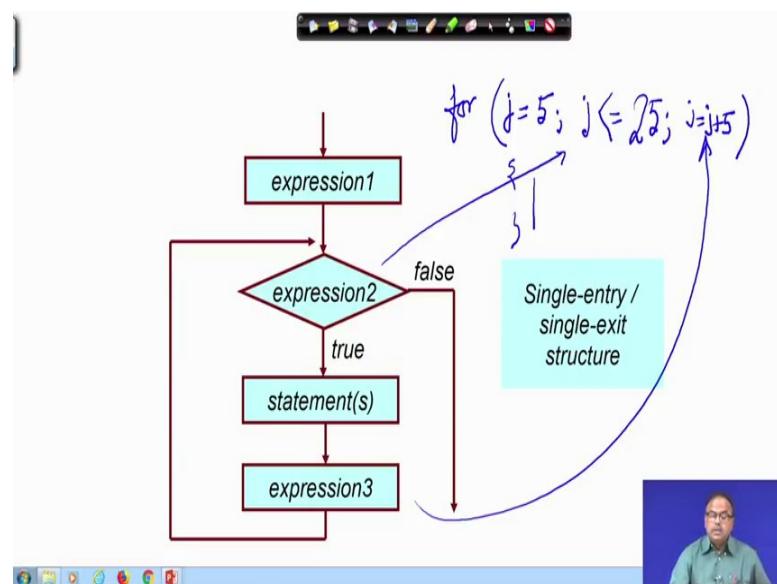
So, if I start it with 1 what should I do? I just did it in a couple of moments earlier in the earlier lecture. So, what should I do here? I should make it  $i$  less than equal to 5, in that case I will take this 10 plus 5 15. So, 5 natural numbers sum of 5 times 6 by 2 is 15 right. So, I can compute that using this loop all right. So, that is a very interesting application. So, for is a very powerful will see, in future application that for is a very powerful construct using which we can do many things.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 21**  
**For Statement (Contd.)**

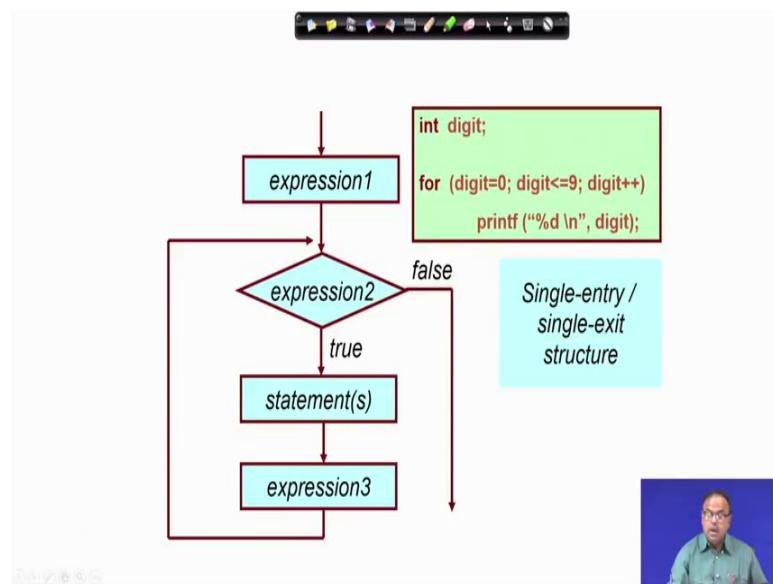
Now, we are discussing about formation of loops, creation of loops in a program using for statement. Before that we have seen the while statement and do while statement and then we have seen the for statement as well.

(Refer Slide Time: 00:35)



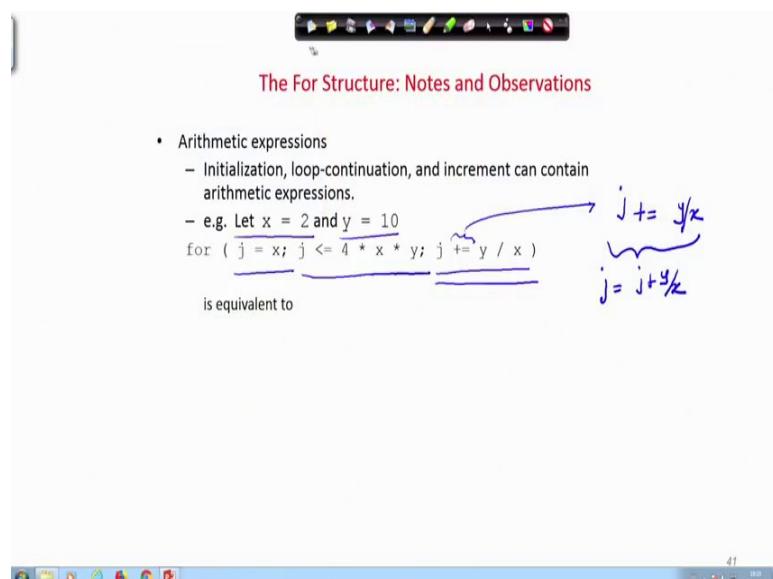
Now, the for statement as we had shown we had discussed in the last class it is basically starting to take an expression, arithmetic expression and then it starts with in sorry initialization expression and then we check a particular condition. So, something like this for some variable integer variable  $j$  assigned some value 5 maybe and then some expression here this one is  $j$  less than equal to 25 and then here there will be some statements which will be executed and after that there is a third expression which is altering it and that can be  $j$  assigned  $j$  plus 5 all right and the body of the x loop. These are structure that we had seen.

(Refer Slide Time: 01:52)



Now, there are some critical issues that are to be noted for the for structure.

(Refer Slide Time: 01:53)



So, if you look at this you can see that we are using arithmetic expressions. For example,  $x$  assigned 2 and  $y$  assigned 10 that is an arithmetic expression. So, similarly this is a valid arithmetic expression for  $j$  assigned  $x$ ,  $j$  less than equal to 4 times  $x$  times  $y$  that is the condition. That means, after some computation will have to find out whether  $j$  is satisfying this condition and here  $j$  here probably this is something this construct we have not shown you, this is I do not like it very much and initially you need not bother about

this j plus equal to y by x this is a C structure, say C syntax expressing j equals j plus y by x. So, the whole thing can be written in this way.

So, this, but that is not important for the purpose of understanding for expressions. So, here is just an expression.

(Refer Slide Time: 03:32)

The For Structure: Notes and Observations

- Arithmetic expressions
  - Initialization, loop-continuation, and increment can contain arithmetic expressions.
  - e.g. Let  $x = 2$  and  $y = 10$
- "Increment" may be negative (decrement)

`for ( j = x; j <= 4 * x * y; j += y / x )`

*is equivalent to*

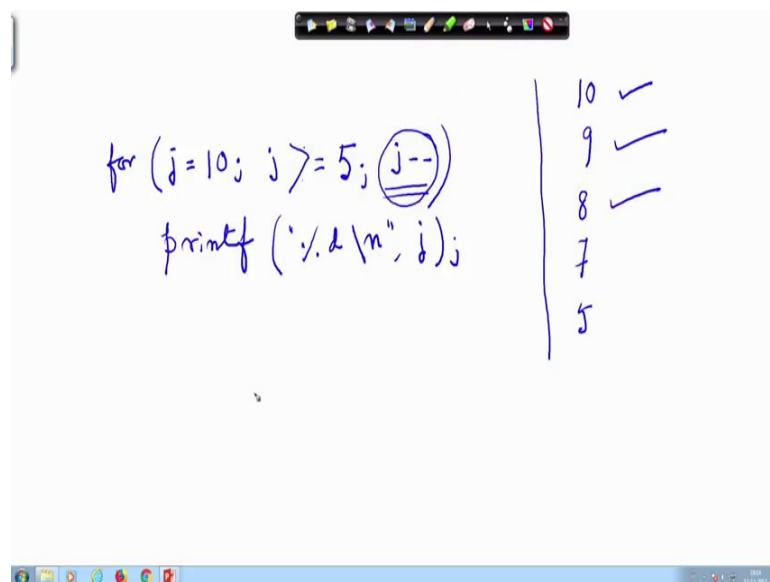
`for ( j = 2; j <= 80; j += 5 )`

*for (j=2; j<=80; j++)*

And this is equivalent to like this for say x was 2 and y was 10, then this is equivalent to j assigned 2 because j assigned x and j assigned 2 are essentially same; j as less than equal to 4 times x times y that means, 4 times 2 times 10 that means, 80 and j equals assigned j plus y by x is 5. So, this is equivalent all right. So, this is initialization, this is loop continuation condition and this is the increment or I as I was saying that this is a modulator or the alteration.

Now, increment can be negative. So, this increment all the way we are calling it increment that is why we should not call it increment let us call it a modulator because say the same thing I can write as for j assigned 2 maybe something j less than equal to 80 and might be j minus minus. It can be the case when can it be. Say for example, if I add if I want that I will be printing the numbers in the reverse order starting from 10. So, what should I do? So, can I do this what I want to do is, that I want to print something like 10 9 8 7 5 all right in this order.

(Refer Slide Time: 05:15)



So, simply I can do this repeatedly in a loop for example, for j assigned 10 and j sorry, j greater than equal to 5 j minus minus and here I just do printf percentage d backslash n j. So, what will happen? Initially j is 10 less greater than equal to 5, so 10 will be printed. Then I go back I decrement j. So, j will become 9 still greater than equal to 5 I come in here print j 9, I again decrement j decrement j becomes 8 I compare with this still it is greater than equal to 5 I get in and printed. In that way I can do it I can repeatedly do the same thing, but here as you can see I am being able to achieve this reverse order by instead of incrementing and decrementing this index. That is why I can increment to decrement, I can multiply it, I change it, I modulate it, that is why this to an alteration expression or modulation expression is a better term than increment or decrement.

(Refer Slide Time: 07:31)

The slide title is "The For Structure: Notes and Observations". It contains a list of bullet points and a diagram. The diagram shows a for loop structure with three main components: "Initialization", "Loop continuation", and "Increment". Arrows point from the text labels to the corresponding parts of the loop statement. The loop statement shown is:

```
for ( j = x; j <= 4 * x * y; j += y / x )
```

Below this, it says "is equivalent to" and shows another loop statement:

```
for ( j = 2; j <= 80; j += 5 )
```

Annotations with arrows point from the labels to the respective parts of the loop statements.

So, if the loop continuation, if the loop continuation condition is initially false is initially false then the body struck the for structure will not be executed. It will proceed with the statement for the next statement.

(Refer Slide Time: 07:57)

A handwritten example of a for loop is shown:

```
j = 20  
for ( i = 1; i > j; i++ )
```

Annotations include curly braces under "i = 1" and "i > j" with an arrow pointing down to an equals sign at the bottom, indicating that the loop does not execute because the initial value of i is less than or equal to j.

So, for example, if I write something like this as you have seen earlier j equal to 20 and for i equals i assigned 1, i greater than j i plus plus say and then I want to do some things here all right.

Now, j is 20 I will first initialize i to 1 and immediately I find the first thing that I do is its false I check this and its false i is not greater than j therefore, this stay these statements will not be executed even once and the statements following the for loop will be executed.

(Refer Slide Time: 08:55)

The screenshot shows a presentation slide titled "for :: Examples". On the left, there is a code snippet in a green box:

```
int fact = 1, i;
for (i=1; i<=10; i++)
    fact = fact * i;
```

Handwritten notes next to the code show the value of fact being multiplied by successive integers from 1 to 4:

fact = 1  
i=2      1.2  
3      1.2.3  
4      1.2.3.4

A video player interface at the bottom right shows a person speaking.

Now, here are some examples of for loops. This is again computing the factorial fact. Factorial is a the variable, fact is touring the factorial factorial all of you know as factorial n is n multiplied by n minus 1 multiplied by n minus 2 so on so forth up to 1.

So, it is 1 and I have got a variable i. Now, you can see simply I can do it like this i assigned 1, i less than 10 i plus plus fact assign fact times i. Now, what is happening here? So, I have got fact to be 1. So, i equals 1 then with that what I do is 1 times 1 is fine and I then make i to be 2 and then i, i becomes then 2. So, 1 times 2 then i becomes 3 then 1 times 2 times 3 then i becomes 4 then 1 times 2 times 4 times 3 times 4 and so on and so forth it will go on in this loop. So, that is a nice way of writing factorial.

(Refer Slide Time: 10:27)

The slide has a title 'for :: Examples' at the top. It contains two code snippets in green boxes:

```
int fact = 1, i;
for (i=1; i<=10; i++)
    fact = fact * i;
```

Handwritten notes next to it show a sequence of numbers from 1 to 10, with arrows pointing to each number labeled  $i = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$ .

```
int sum = 0, N, count;
scanf ("%d", &N);
for (i=1; i<=N; i++)
    sum = sum + i * i;
printf ("%d \n", sum);
```

Handwritten notes next to it show  $N = 5$  with an arrow, and the calculation of the sum of squares from 1 to 5:  $Sum = 0 + 1^2 + 2^2 + 3^2 + 4^2 + 5^2$ . Below this, the formula  $Sum = \underline{\underline{1^2 + 2^2 + 3^2 + \dots + N^2}}$  is written.

A small video player window in the bottom right corner shows a person speaking.

And here is another example you can see what it does quickly. You can explain it yourself what it does. Sum is 0 and N is a variable and count is another integer variable all right. Sum is 0, N is a 1 variable and count is another integer variable. Now, we are reading N and for i equals 1, i less than equal to N I am adding some. So, what is being done here what will happen if I press this program what is going to happen? Let us do that.

Suppose I have read N to be here, I have read N to be 5. Now, what am I doing here? I here there is one problem here i should have been declared there is a mistake here int N and here i also add i should have been declared here. Now, i assigned 1. So, i is 1, N was 5, i less than 5 less than equal to 5. So, sum is sum was 0. So, sum is 0 plus i times i, i was 1. So, 0 plus 1 square and then I check i increment i. So, i becomes 2, i becomes 2 and I check that i is less than it not less than a is still less than equal to 5 so I got this sum and with that I add now, i square i times i, so 2 square. Similarly, it will go on it will do 3 square. So, i will become then 4 and then still it is less than equal to, so 4 square then i will become 5 and still it is true. So, it will be 5 squared then i become 6 this condition will be violated and i will come to this printf.

So, what will sum be sum is 1 square plus 2 square plus 3 square and if I make it N then up to N square. So, this is the very well known series the sum of the square of natural

numbers we can compute by this small program using for loop right. So, it is a nice example.

(Refer Slide Time: 13:32)

- The comma operator
  - We can give several statements separated by commas in place of "expression1", "expression2", and "expression3".

for (fact=1, i=1; i<=10; i++)  
 fact = fact \* i;

for (i=1; i<=N; i++)  
 sum = sum + i \* i;

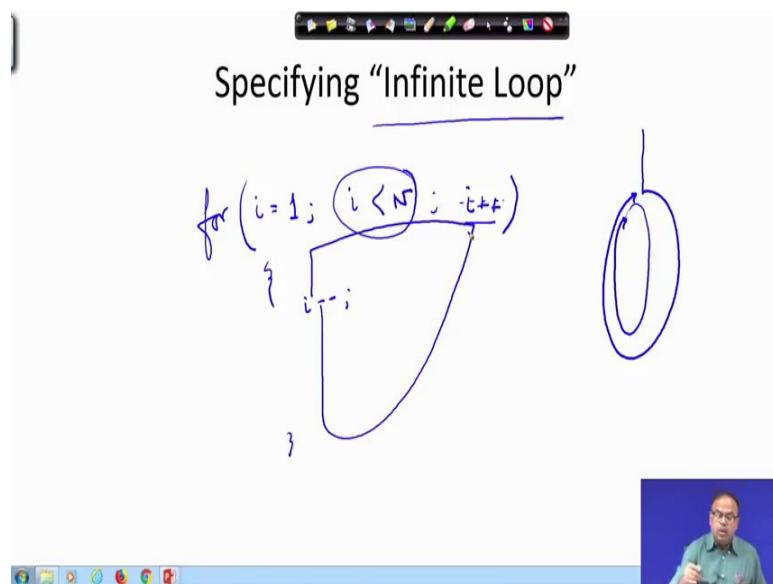


Now, we introduce the comma operator. As we say that here for when we write for then i assigned 1 instead of that I can put in more than one statements here using a comma operator. For example, for fact 1 i equals 1, i less than equal to 10. Now, this part is what? This part is the initialization. Now, remember that this will not be continuously initialized, this is just an just a statement that I put here an assignment statement, but my index variable for the control variable for the loop is i. So, I could have written this earlier example of factorial maybe, earlier example of factorial this could be initialized here also all right. So, that is just saving space saving the number of lines of code.

But personally I would suggest as I did earlier also that is for of those of you who are beginners in programming you should not try this tricks or should not try this thriftiness reducing the number of lines, that is not so important to how much you can reduce. The most important thing is to be logically can syntactically correct while you write a program ok.

So, we can give several things in comma like some here again. So, the program becomes even smaller look smarter, but sometimes at the beginning if you try to do that you try to be smart and in the process you may result in some long program long logic better avoid that.

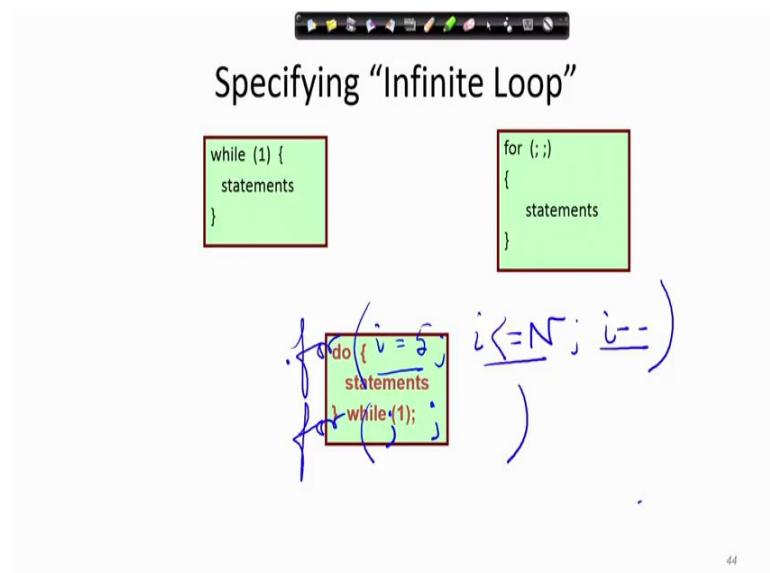
(Refer Slide Time: 15:27)



Now, infinite loop in general what is an infinite loop. When a program continues, program continues in a loop repeatedly it goes on and it is never completing its going on because the condition that is supposed to turn out to be false at a particular point of time. So, that it comes up the out of the loop never happens all right. That can always happen that say if I write something like for i equals 1, i less than n and whatever i plus plus. Now, every time inside the loop when you get in you read you do something and then increment i and when you come inside the loop you do i minus minus, then whatever has been done here will be canceled out here. So, this loop will never reach this condition I less than i therefore, that this loop will continue forever such situations are known as the condition of infinite loop.

Now, sometimes usually we do not like that, but sometimes it may be necessary to specify that something will happen forever all right. Say some particular work has to be done continuously.

(Refer Slide Time: 07:23)



44

Now, in order to specify that for loop provides us some facility like say for and while everything say this one we had discussed earlier while 1, that means, always it is true it is a nonzero this while loop this expression part condition expression part should return nonzero and so it will go on. Now, here for and I put null; that means, the for has had 3 parts, for the initialization part, some condition part and some incrementation decrementation part whatever.

Now, I just keep everything blank. So, I turn it to be for nothing and nothing all these things are blank. In that case what will happen? In that case what will happen? If I keep some statements here that will go on forever. Similarly if I in the case of do while if I just put while 1 that is again in finite loop. So, by this I can express my desire that some things will have should continue forever all right. So, this is another trick that you can utilize in some cases for the for and the while constraints.

(Refer Slide Time: 19:05)

The slide has a title 'break Statement' at the top. Below it is a bulleted list:

- Break out of the loop {}
  - can use with
    - while
    - do while
    - for
    - switch
  - does not work with
    - if {}
    - else {}

A handwritten note 'Break out of the loop.' is written next to the bullet point about loops.

Below the list is the text: 'Causes immediate exit from a while, for, do/while or switch structure'.

Further down, it says: 'Program execution continues with the first statement after the structure'.

Handwritten notes include:

- 'Common uses of the break statement'
- 'Escape early from a loop'
- 'Skip the remainder of a switch structure'

On the right side, there is a code snippet for a switch statement:

```
switch color
{
    case 'R': break;
    case 'G': break;
}
```

The slide has a footer with the number 45 and a date 11/11/2017.

Now, we come to another statement which we have encountered a little few lectures earlier in the context of switch statements. While we are considering the switch statements if you recall you have seen that after every case statement switch on a particular variable then their case red case, green case, blue if you recall then we did something and then give a break, did something give a break like that we proceeded right.

So, the break statement of course, we know can help us we can use it with several things one is while, do while, for and switch. Now, this switch, switch part we have seen, but we can also use it for while do while and for, for breaking out of the loop, we want to break out of the loop. Sometimes that is required it works with while, do while, for, but does not work with if and else statement. It causes immediately exit from a while for a do while or case we have seen that switch statements earlier.

So, the program continues after with the next statement let us see. Why do we want to use it? It helps us to escape early from a loop. Sometimes we want to escape early I do not want to go till the end of the loop, when a particular condition is met I want to come out. I am doing it in a loop, but waiting for some condition to take place as soon as that condition takes place I come out of the loop. Maybe, as we have seen in the case statement switch some expression all right color, then case R we do something and then

give break right, case G we do something and then give break right we do like that. So, we skip the remaining part of the switch block right. We come out of that.

(Refer Slide Time: 21:54)

```
#include <stdio.h>
main()
{
    int fact, i;
    fact = 1; i = 1;
    while (i<10) {           /* run loop -break when fact >100*/
        fact = fact * i;
        if (fact > 100) {
            printf ("Factorial of %d above 100", i);
            break;             /* break out of the while loop */
        }
        i++;
    }
}
```

5.4.3.2

1 2 6 24 120

So, similarly we will see a couple of examples of this. Say here there is a complete example let us look at that. Here is a complete example complete program, once again include stdio dot h you are running some programs main, then fact and i are integers fact we are computing factorial. So, fact is 1, i is 1, while i is less than 10; that means, I want to break out while i is less than 10 fact times i, if fact is greater than 100, then factorial is above 100 factorial of a number is above 100. So, for example, I am going on up to 10 numbers. So, what will happen with the factorials? Let us see. So, factorial 1 will be 1, factorial 2 will be 2, factorial 3 will be 3 times 2 that is 6, factorial 4 will be 4 times 3 for 4 times 6, it will be 4 times 3 times 2, so 4 times 6; that means 24, then 24 times 5, so that will be how much it will be, more than how much will it be if I have got this 4 3 2 and multiply that with 5, so 26.

So, here just with 5 numbers I am exceeding the value 120, but I do not know when I am going to reach the value 100. So, I have written the program in this way, fact while i is less than 10 I will test it up to 10 numbers every time, I compute fact if fact is greater than 100 print. So, here as soon as fact becomes 100 will say factorial of 5 is above 100, factorial of 5 is above 100 and then I break out. Otherwise if I had not put this break then this would have continued till this loop all for all the 10 numbers up to factorial 10, but I

just want to stop whenever my result becomes more than 100. So, this is one way we can utilize a break right.

Similarly there is another statement although we do not use it often it is better to know that sometimes it can come handy, that is a continued statement; that means, the remaining part of the body of a loop we will skip if I put a continue all right.

(Refer Slide Time: 25:05)

The slide has a title 'continue Statement' at the top. Below the title is a bulleted list of points:

- continue
  - Skips the remaining statements in the body of a while, for or do/while structure
    - Proceeds with the next iteration of the loop
  - while and do/while
    - Loop-continuation test is evaluated immediately after the continue statement is executed
  - for structure
    - Increment expression is executed, then the loop-continuation test is evaluated.
    - expression3 is evaluated, then expression2 is evaluated.

At the bottom of the slide, there are navigation icons and the number 47.

So, let us see and it proceeds to the next iteration of the loop. So, what happens is let us look at one example here.

(Refer Slide Time: 25:11)

The slide has a title 'An Example with “break” & “continue”'. Below the title is a code block with annotations:

```
fact = 1; i = 1;
while (1) {
    fact = fact * i;
    i++;
    if (i<10)
        continue;
    break;
}
```

Annotations include:

- A blue arrow points from the word 'continue' to the line 'if (i<10)'.
- A blue arrow points from the word 'break' to the line 'break;'.
- Handwritten notes on the right side:
  - 'fact = 1.1'
  - '= 1.1.2'
  - 'i/13'
- A note at the bottom right of the code block says: 'It just skips the rest of the loop and continues with the next iteration.'

At the bottom of the slide, there is a video player showing a person speaking and the number 47.

So, while one that means what; that means, it is always true. It will continually go on doing it. So, fact is fact times i, i plus plus. So, I start with i 1, then 2, then 3. If i is less than 10 I will go into the loop otherwise I will break. So, this continue is basically remain forcing me to go back to this point, by forsaking this part. I am not coming to the remaining of the loop, but if this condition is not holding then I will not execute continue I straight away come here and do break. I think it will need a couple of moment for you to just realize what is happening here all right.

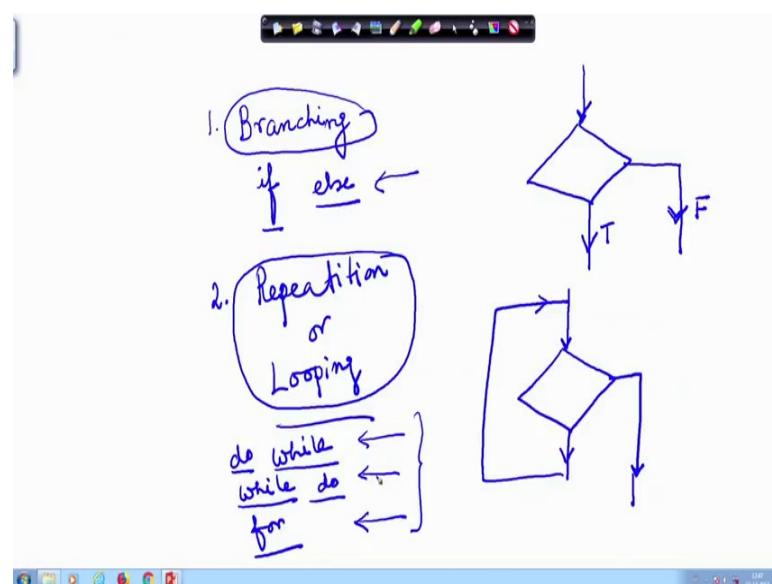
So, just look at this. So, in this way it will I think you can understand it. So, I have got i to be 1. So, fact will be 1 times 1, then i becomes 2 and then I go back. So, it will be fact will be 1 times 1 times 2, i will be 3 and in that will go on, but as it will go on as long as i is 10, less than 10. Then i will not come to break I will go back here. But if i is less than 10 i will come to break and i will come out of the loop. So, it is a combination of continue and break this is. But what I want is that you should understand how the for loop is used for repetition that is very important to understand and will carry out with some examples in the next lecture.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 22**  
**Example of IF-ELSE**

Till now, we have learnt about some constructs of programming. To implement some very generalized structures, programming structures, right. What are they? First, we have seen, how we can do Branching, right.

(Refer Slide Time: 00:36)



And Branching means that whenever there is a sequential flow, we take a decision pop or we come to a decision box and depending on whether the result is true or false; we take different paths right. That is what we have seen and that is implemented by if else statement right.

We have seen that and the other thing that we have learnt is repetition or looping all right. That means, we have come to a particular point, we take a decision and based on the decision, we decide whether we will go back to an earlier point or we will continue forward. So this is, this repetition we have we could achieve by While statement, do While statement or While do statement or for statement. So, if else and do while, while do, for all these are constructs in the C language.

Whereas, the concept of Branching and Repetition or Looping is general for most of the programming languages. So, these are general concept, while this is a specific construct corresponding to the C language. Now to start our journey into the world of real examples, let us first start with the school level equation solving.

(Refer Slide Time: 03:11)

A screenshot of a video call interface. The main area shows a handwritten equation:  $ax^2 + bx + c = 0$  with a horizontal line underneath it, followed by  $3x^2 + 5x + 1 = 0$ . To the right of the first equation, the text "Roots of the eq^n" is written in blue ink. In the bottom right corner of the video frame, there is a small video window showing a man with glasses and a mustache, wearing a blue jacket, speaking. Below the video frame is a taskbar with several icons.

When we try to solve a quadratic equation of the form  $ax^2 + bx + c = 0$ . We tried to solve it, means we wanted to find the roots of the equation. Roots of the equation of this form and what are the roots? The roots are the values of  $x$  that will satisfy this equation.

For example I can have  $3x^2 + 5x + 1 = 0$  all right. What are the values of  $x$  that will satisfy this equation? So, we knew.

(Refer Slide Time: 04:11)

$ax^2 + bx + c = 0$

$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$

$b^2 - 4ac = 0$

$x = \frac{-b}{2a}$

$(x-a)^2 = [x^2 - 2ax + a^2 = 0]$

Sreedhar,  
Acharya's  
Method

We know that, we can solve this general equation by the well-known Sreedhar Acharya's Method; that is  $x$  is minus  $b$  plus minus root over  $b$  square minus  $4ac$  divided by  $2a$ . By this, we can find the roots of this equation and since it is a quadratic equation how many roots will be there? There will be 2 roots. We also know that if  $b$  square minus  $4ac$  is equal to 0 then, this part will be 0. What will be the roots? Roots will be minus  $b$  by  $2a$  only.

So, there will be 2 roots which are equal. The same roots. For example, if I have  $x$  minus a whole square that is equal to  $x$  square minus  $2ax$  plus, what will that be? Plus a square right. Now if I saw and if I say that this is equal to 0, from here I say that this is equal to 0, if I solve it I will make this as  $x$  minus a whole square and; that means,  $x$  assigned a will be the root right. That means,  $x$  minus  $a$  is 0. So,  $x$  both the roots will be equal. So, if I have root over  $b$  square minus  $4ac$  equal to 0, in that case I will have both the roots equal.

(Refer Slide Time: 06:32)

The image shows handwritten mathematical notes on a whiteboard. At the top, there is a toolbar with various icons. Below it, the discriminant is defined as  $\text{Disc} = \sqrt{b^2 - 4ac} = 0$ . This leads to two cases: if  $\sqrt{b^2 - 4ac} = -ve$ , the roots are imaginary. If  $\sqrt{b^2 - 4ac} > 0$ , the roots are real. The formula for the roots is given as  $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ .

If root over b square minus 4ac is negative, we know then, our roots which are minus b plus minus root over b square minus 4ac by 2a; if this part is negative, then, the roots will be imaginary right. If b square minus 4ac is greater than 0, then, we will have Real roots right and the Real roots will be minus b plus this and minus b minus this by 2a. So, that will be that is a school level knowledge that we have.

Now, if I want to apply this knowledge, to write a program, that can take as input any equation of that form ax square plus bx plus c and find its root. How will the program look like? Suppose, I want to write such a program and now, henceforth in my discussion about this problem I will called, I will call this b square minus 4ac to be the Discriminate, I write it for short, Disc, as the Discriminate. Because that is, what is telling me discriminating between the roots Disc. So, whenever I refer to Discriminate; that means, I am talking of b square minus 4ac.

(Refer Slide Time: 08:52)

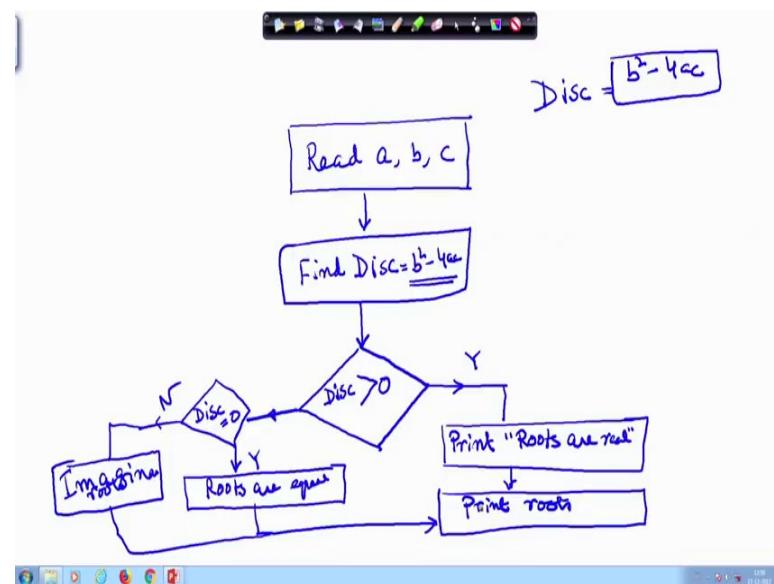
$$ax^2 + bx + c = 0$$
$$3x^2 + 4x + 5 = 0 \quad [a, b, c]$$
$$25x^2 + 17 = 0$$
$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{ or } \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Now, I want to write a generalized program, for finding the roots of  $ax^2 + bx + c = 0$ . So, I want to write a program that will be equally applicable to solve this equation, solve this equation or  $25x^2 + 17 = 0$ . Sorry here I should not write  $c$ , let me put it something like  $5 \cdot 25x^2 + 17 = 0$  or any such equation. So, you see here what is the coefficient of  $x$ ? Coefficient of  $x^2$  is  $25$ . What is the coefficient of  $x$ . In this equation it is  $0$ , that is why this term has vanished and what is the value of  $c$  here? It is  $17$ .

Whereas, in this equation the coefficient of  $x$  is  $3$ ,  $x^2$  is  $3$ , the coefficient of  $x$  is  $4$  and the constant term is  $5$ . So, since I am going to solve any general equation of this form, how can I define the specific equation? An equation of this form can be specified by specifying the coefficients. So, I have to accept the values of  $a$ ,  $b$  and  $c$  from the user and you can see that the root is dependent on the value of these coefficients, because we know the root  $x$  is  $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ .

So, the roots are determined only by the value of these coefficients. Therefore, the 1st step that I will have to do here we can start with a flow chart.

(Refer Slide Time: 11:19)



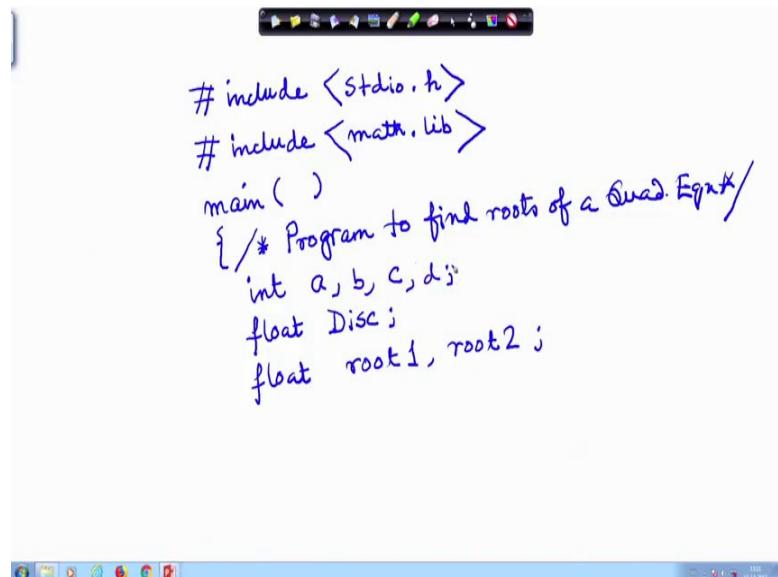
So, I will 1st, I write Read a, b, c. Read them from the user. Then, I compute, Find Discriminate equal to b square minus 4ac. I just compute this part, b square minus 4ac. Because depending on this, we will have the different values of the root or I would say all right. So now, so earlier probably I said that the Discriminate is root over b square minus 4ac. I would like to correct myself. I would say Discriminate is this part b square minus 4ac.

Because depending on the value of this, if this is negative, then, the roots will be imaginary. Because the square root of this will be imaginary. So, this is the Discriminate part. If this is 0 then, the roots are equal. If this is greater than 0 then, the roots are positive and real and if it is negative then, the roots are imaginary. So, at this point, I will make a decision. Discriminate greater than 0, yes. Then, what should I do? I will Print Roots are real and then, I will print the roots. If the Discriminate is not greater than 0 then, I come here and I check whether Discriminate is 0, if Yes then, my decision is Roots are equal and then I go and Print the roots.

If this is No, then, obviously, I will print Imaginary roots. I can see this part and then, I will go and Print the roots. That is the flow chart of the whole thing. So, I am computing this value and based on that, I am finding the roots. Now if the flow chart is clear to you, let us proceed to write the program for this. Now here, obviously, I should have said that

the Discriminate is actually this part b square minus 4ac and then, I take the square root of that.

(Refer Slide Time: 15:35)



```
#include <stdio.h>
#include <math.lib>
main()
{ /* Program to find roots of a Quad. Eqn */
    int a, b, c, d;
    float Disc;
    float root1, root2;
```

So, now if we start writing the program, if I am a nice programmer, I will always start with so, first of course, hash include stdio dot h. Then, I am including another thing which is a library called math dot lib. Dot lib stands for library. Why I am including that? I will come to that later. Now we start our program, main and we start. I can give a comment, program to find roots of a Quadratic Equation all right. So, that is a comment. So, anybody can understand, what I am writing here.

Now, I will have to declare a number of variables. I will declare the, lets say the coefficients are all integers. So, int a, b, c. Now b square minus 4ac can be anything. So, I put float Discriminate; also float root 1, root 2; 2 roots. Next, what should I do? Let me proceed, to the next page, with this declaration, I hope you have been able to take down this declarations.

(Refer Slide Time: 18:27)

The screenshot shows a Windows desktop with a Notepad window open. The code in the Notepad is handwritten in blue ink and follows the standard C syntax for printing prompts, reading coefficients, calculating the discriminant, and printing results based on its value.

```
printf ("Enter integer coeff a then b then c");
scanf ("%d %d %d", &a, &b, &c);
Disc = b*b - 4*a*c;
if (Disc > 0)
    printf ("Roots are real\n");
else if (Disc == 0)
    printf ("Roots are equal\n");
else
    printf ("Roots are imaginary\n");
```

Now, let's proceed and go to the next page. I am continuing with that. I start with printf Enter coefficients a then, b then, c all right.

So, the user is told that, he is supposed to enter the coefficients now. Next thing will be scanf, since I know, I can also say that please Enter integer coefficients; I am just restricting it here like that, though it's not necessary. So, scanf percentage d percentage d percentage d, and a, and b, and c. So, what I will do? My system will read the 3 integers a, b and c. Now starts the main program and as if you have studied the flowchart that I had shown, here you can see that, I have got only 1 decision box and I can proceed through that decision box. There are 2 decision boxes actually, one in this direction; another one in this direction.

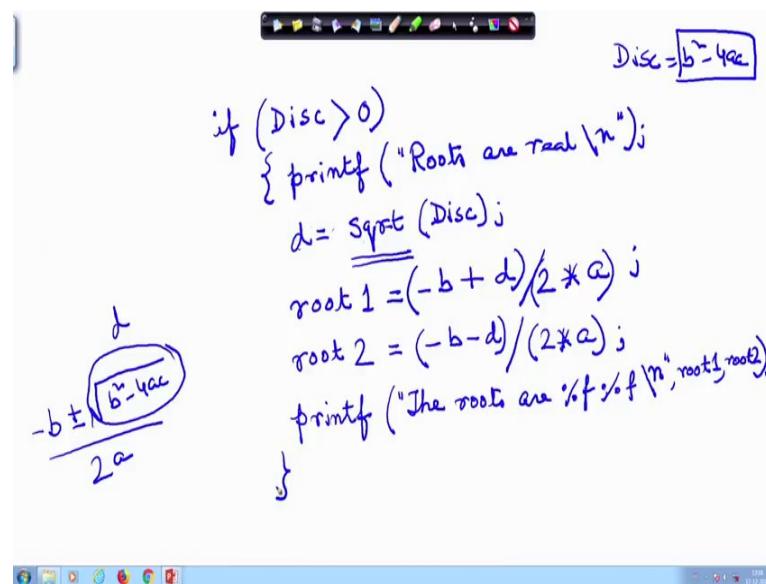
So, accordingly I do not have any loop here. So, what is the construct that I will be needing here. The construct that I will be needing here is if. So, what should I do? First, let me compute the Discriminate all right. I have got this b, a and c. So, I do Discriminate assigned b square minus 4ac. So, b times b minus 4 times a times c. So, this is an arithmetic expression. I get Discriminate. Now I will have to take the path depending on the Discriminate. So, what I will write here is if then, a parenthesis for the condition if Discriminate is greater than 0, printf Roots are real and I can put a backslash in here else I have not computed the roots as yet ok.

So, let me write the else as close to the if as possible, as we have learnt about indentation. So, else if, I should have written this if also and small just for the sake of uniformity and at not only for the sake of uniformity, C is very much case sensitive. So, capital if is not if. So, if Discriminate is equal to 0, printf Roots are equal backslash n; else now this, else comes under this if, you remember that else and ifs are tied up, based on the closest the else is connected to the closest if. So, these 2 else, else the only option is Discriminate is neither greater than 0 nor equal to 0.

So, Discriminate is less than 0. Then printf Roots are imaginary and backslash n again. Now so, I have just said what type of roots they are. But I have not printed the roots as yet, we have not I have not found out the roots. So, what I can do? I can modify this a little bit or how do I find out the roots? Let me go to the next page. I do this and then, I come back to a particular stage or I can do it here itself. How would I find out the roots? Say, let me go back and do it here. Here, I have I have not written down, what the roots are?

How could I? Suppose I modify this part a little bit; I modify this part a little bit. So, let me modify this part, I will rewrite this part in the next page.

(Refer Slide Time: 25:06)



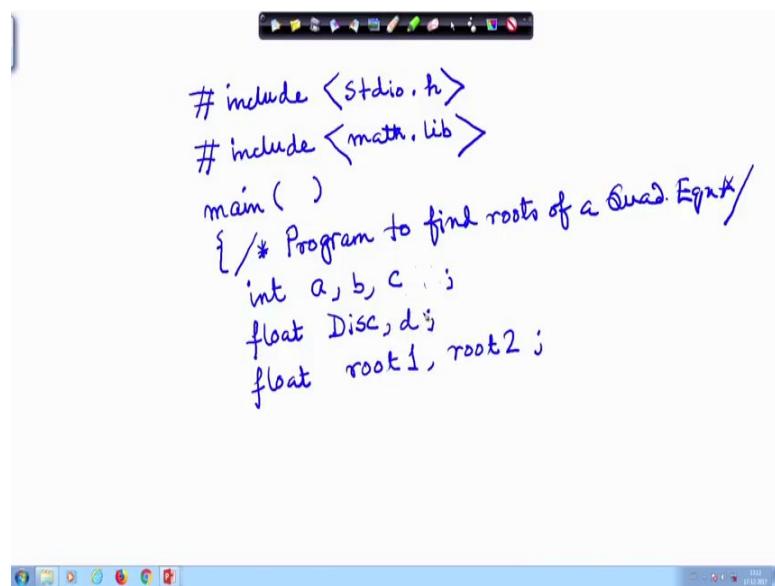
So, I rewrite it as, if Disc is greater than 0, printf Roots are real I find. Now I compute say I compute, lets call it, I should have declared this also Discriminate 1, another part that is, I had the Discriminate part. So, I can certainly make another variable. So, I add 1

more variable here, int a, b, c and here I entered another variable say d. Why, I will show later.

So, I have entered another variable d there. So, printf Roots are equal and d is square root of Disc. So, I called Disc to be b square minus 4ac. Now I am finding the root of that. Now here I have committed 1 mistake. Can you tell me what the mistake is? So, this Discriminate can be integer, but when I take the square root of that, it can be a float right and also here I have already declared, if you look at I Disc, I somehow declared Disk as float, but since a, b, c are integers, then, this will also be an integer.

So, I should correct it a little bit. What should I do? I or it really does not matter if I keep it as float. So, only thing is that I cannot keep the d here. I will have to remove this d all right and I will also remove this.

(Refer Slide Time: 28:08)



```
# include <stdio.h>
# include <math.lib>
main( )
{ /* Program to find roots of a Quad. Eqn */
    int a, b, c ;
    float Disc, d ;
    float root1, root2 ;
```

And will make this small d as a float. So, then I come here; d is square root of the discriminant. Now what is this square root? Who will compute this square root? The square root computation is done by an inbuilt function in C library. C compiler provides us with a library of some inbuilt function, mathematical functions which we can call and invoke whenever we need them.

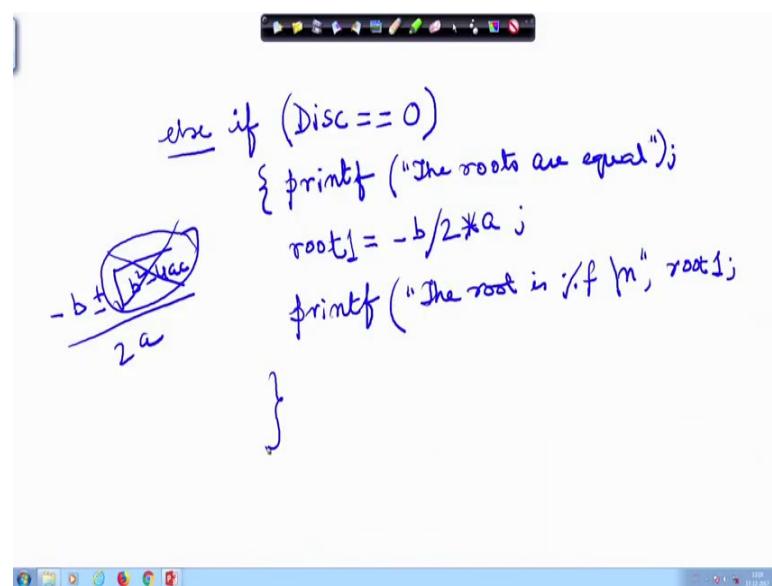
So, you can see here, that is why I have included this math dot lib; that means, I will be using some library function here, dot lib means library. So, you will find in the course of

this core lectures and more examples, where you will see more number of library functions. But just for this, we need the square root function. So, I compute the square root of Disc d assign the square root of Disc and then, root 1 is what root one is minus b plus d by 2a. Is it clear? The 2 roots, one is I have my formula is b plus minus root over of b square minus 4ac by 2 a.

So, this b square minus 4ac is d here all right. So, b plus d whole divided by 2a, I should also put a parentheses here, is root 1 and root 2 is minus b sorry minus b minus d divided by 2a. Then, I have got the 2 roots and I can print here. So, I have computed the roots. So, I can either print it here. I can printf that The Roots can be float. I can say the roots are percentage f, roots are floats percentage f, percentage f backslash n and root 1 comma root 2. So, this is the whole part compound statement if Disc is greater than 0.

So, if we go up. So, I will modify this part in this way. We will also have to modify this part. What should you do? Simply, what can I do to write this? If Disc is 0 then, I will print roots are equal right, as I have written roots are real and then, I will find out; now I do not need to find out this part. So, let me just show you this part and I leave the other part for you.

(Refer Slide Time: 32:01)



That second part, if Disc, oh my god what did I do here; this all right. It is greater than equal to, if disc is 0. Then, printf that the more than once sentences. So, this is coming else that else part, else if disc is 0; printf the roots are equal and here I donot need to

bother much I can put a backslash n, that you know by now printf The Roots are equal.  
Root is what?

Now, if in this, b plus minus root over b square minus 4ac by 2a, if this part is 0. Then, my root is only minus b divided by 2a 2 multiplied by a and printf The Root is percentage f backslash n root or I can say root 1, because I did not declare anything as root. So, root 1 anyone root that is same. This is this part if the Discriminate is 0. See yeah. So, I have just mentioned root 1. Why? Sorry, you see here I declared only root 1 and root 2, I did not declare any root. So, I can make root 1 root 2. Now, so this is replacing this part.

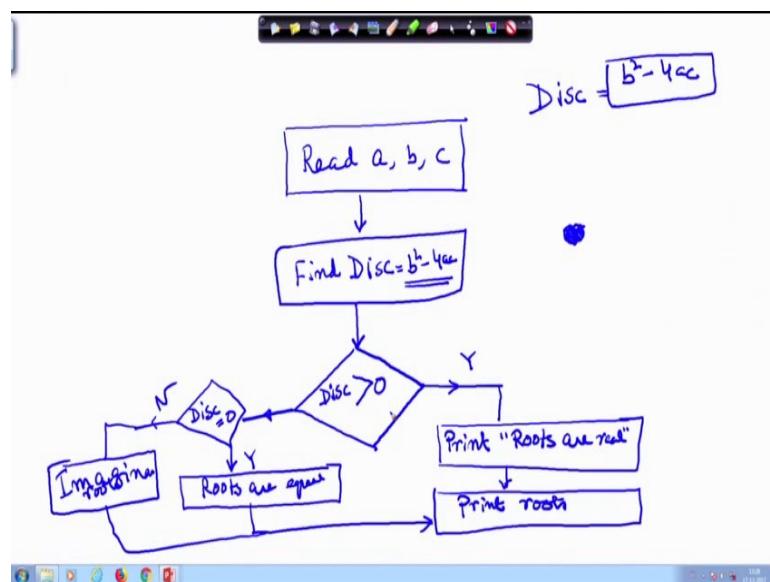
Now what will you do here? The roots are imaginary, but the same thing the values will be the same; only thing is roots are imaginary. I leave it to you as an assignment; you think off, what will you do for that part? For the imaginary part. You can understand that only thing that you need to do is to print some roots as say the real part. Real part is this and imaginary part is this. So, you think about it; we can discuss about it in the next lecture.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 23**  
**Example of Loops**

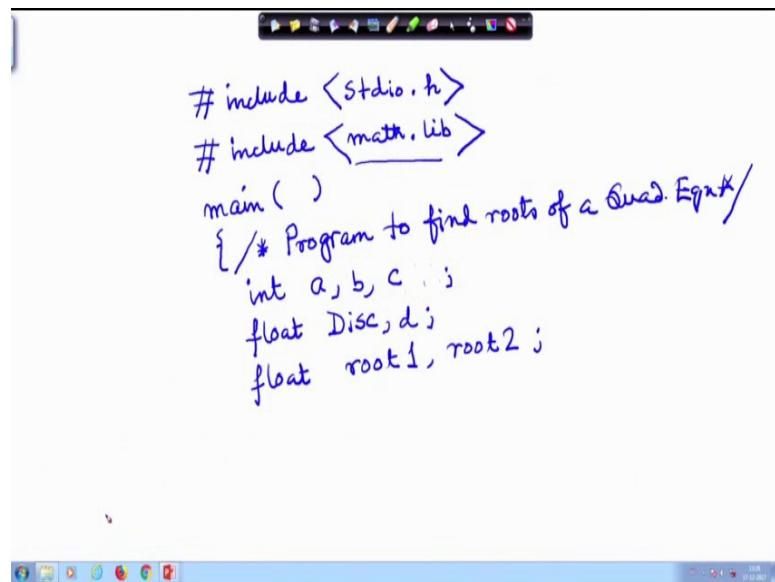
So, we were looking at solving a quadratic equation and we have seen that there are three cases. One is if the discriminant is 0 we print the roots that their roots are real, sorry if the discriminant is greater than 0 then the roots are real, otherwise if it is equal to 0 then the roots are equal and if the discriminant is neither greater than 0 nor equal to 0; that means, it is less than 0 then the roots are imaginary.

(Refer Slide Time: 00:19)



So, next what we did is we started with a program like this where we declared the coefficients here and then the discriminant and the intermediate variable d and 2 roots right root 1 and root 2.

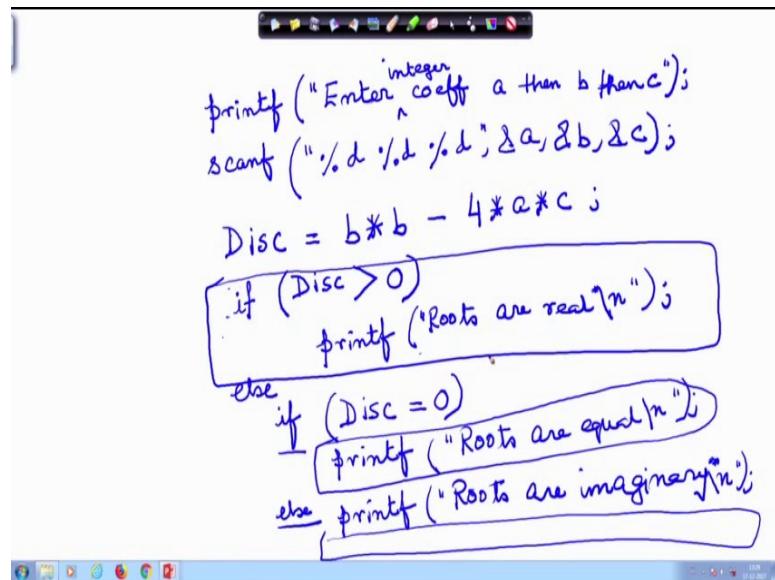
(Refer Slide Time: 00:45)



```
#include <stdio.h>
#include <math.lib>
main()
{ /* Program to find roots of a Quad. Eqn */
    int a, b, c;
    float Disc, d;
    float root1, root2;
```

Next we proceeded with this and we modified this part.

(Refer Slide Time: 01:04)



```
printf("Enter integer coeff a then b then c");
scanf("%d %d %d", &a, &b, &c);
Disc = b*b - 4*a*c;
if (Disc > 0)
    printf("Roots are real\n");
else if (Disc == 0)
    printf("Roots are equal\n");
else
    printf("Roots are imaginary\n");
```

If disc is greater than 0 printf roots are real and then I modified this, this part and replaced this part with this, if discriminant is 0 prints the roots are real and also compute the root.

(Refer Slide Time: 01:16)

```
Disc = b^2 - 4ac
if (Disc > 0)
{
    printf ("Roots are real\n");
    d = sqrt (Disc);
    root1 = (-b + d)/2*a;
    root2 = (-b - d)/(2*a);
    printf ("The roots are %f,%f\n", root1, root2);
}
```

Now, if the discriminant is equal to 0 then prints the roots are equal compute the root and print it. Now, I asked you to see how you will deal with the third case if the discriminant is not equal to 0.

(Refer Slide Time: 01:44)

```
else if (Disc == 0)
{
    printf ("The roots are equal");
    root1 = -b/2*a;
    printf ("The root is %f\n", root1);
}
```

So, this if, when this if comes then we was already failed the condition the discriminant is greater than 0 and if the discriminant is not equal to 0 then we will come to just an else here right, and let me go to the next page to write the else part of this.

(Refer Slide Time: 02:07)

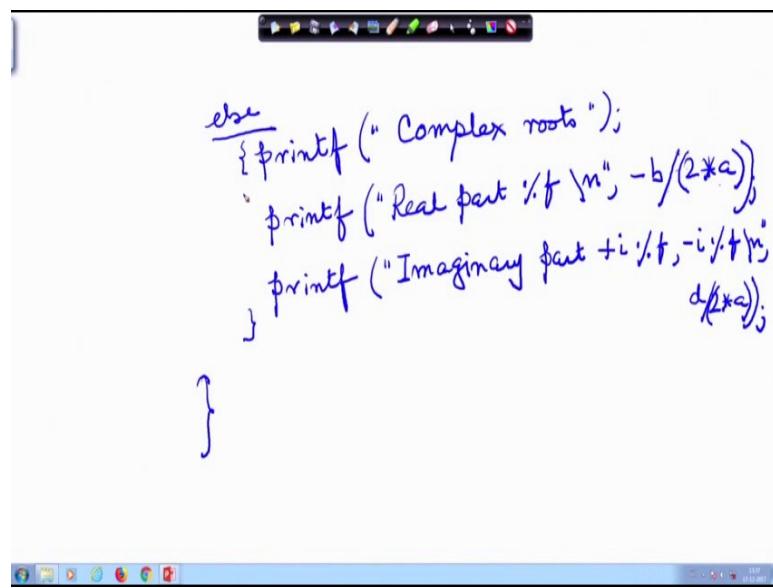
The image shows handwritten mathematical notes on a whiteboard. On the left, there is a diagram of a rectangle divided into four quadrants by a diagonal line. The top-left quadrant contains the expression  $\frac{-b + i\sqrt{d}}{2a}$ , and the bottom-right quadrant contains  $\frac{-b - i\sqrt{d}}{2a}$ . Above the rectangle, the word "else" is written above a brace, followed by the text "{ printf("The roots are imaginary\n");". To the right of the rectangle, the quadratic formula is derived:  
$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$
$$= \frac{-b \pm \sqrt{\text{Disc}}}{2a}$$
$$= \left( \frac{-b}{2a} \right) \pm \frac{i\sqrt{\text{Disc}}}{2a}$$
$$= \frac{-b}{2a} \pm \frac{i\sqrt{d}}{2a}$$

So, it will be else simply I am sorry here did I give the parenthesis, yes I did. So, I am ok. This is please note these are the points where we often make the mistakes. Else printf “the roots are imaginary backslash n” fine, and what are the roots? So, if we if the roots are imaginary then the real part say one, one root is minus b by 2a is the real part and plus i that is what we, i d by 2a is it clear another this is one root another root is b by 2a minus i d by 2 a.

What does, so this d, basically you see what we are doing is x is minus b plus minus root over b squared minus 4 ac by 2a. So, this is my discriminant and I take the square root of that. So, plus minus root over d by 2a right. In my program what did I write d or discriminant square root of disc sorry disc. So, I should not use different name. So, it should be disc by 2a and that square root of disc can be either imaginary, so negative if this is negative then I will find it is its absolute value and, this can be written as minus b by 2a plus square root of disc by 2a is one root another root is b by 2a minus root over disc by 2a and since its imaginary it should be i, the imaginary part.

So, I can print it as a real part to be b minus 2a and imaginary part is square root of disc by 2a and the square root of disc is nothing but d. So, minus b by 2a plus i d by 2a that is what I am writing here all right. So, this is the real part this is the imaginary part. So, I can print it in different ways.

(Refer Slide Time: 05:28)



```
else
{ printf ("Complex roots");
  printf ("Real part %f \n", -b/(2*a));
  printf ("Imaginary part +i %f, -i %f \n",
         d/(2*a));
}
```

I can just say else printf roots are imaginary or rather I should have said this is wrong to say. What should I have said, you should have I mean I am doing a mistake here all through, starting from this point, you must have observed it.

Here the roots are complex roots then in that case, not the imaginary roots because it has got a real part and the imaginary part. So, the roots are complex. I should have written roots are I should have written complex roots and then I could print real part percentage f backslash n, I can put an expression here minus b divided by 2a. We can make it a little cleaner, looks very nasty here, b divided by 2a that is the imaginary part that is a real part all right.

And printf imaginary part is plus i percentage f comma minus i percentage f backslash n followed by. What is my imaginary part? Imaginary part is d by 2a. So, you see I have put this i part plus i and i here comma b divided by again d divided by twice a all right. So, this and then of course, everything is done so I come to the end of my program. Before that here there should be a parenthesis. So, this is the complex part.

Now, as we will become little more conversant with programming you can see that I have repeated many things right. So, instead of doing it there are some common you can think of how can write it in a much more elegant and shorter way. But this is an example which gives you an exposure to the use of if then else statements, if else statements in C.

Next, well another example we can take up. Say for example, we are going to compute the income tax of a person.

(Refer Slide Time: 09:56)

The image shows a hand-drawn diagram on a computer screen. On the left, there is a C program code. On the right, there is a table showing the tax calculation based on income levels.

**Hand-drawn C program:**

```
# include <stdio.h>
main()
{
    /* Tax computation */
    float income, Tax;
    printf("Please enter your income\n");
    scanf("%f", &income);
```

**Hand-drawn Tax Table:**

Income	Tax
100,000	NIL
100,000 to 200,000	10% of the amount above 100,000
> 200,000	20% of the amount

Suppose the if the income tax is less than 100,000 rupees then income tax is nil, if it is between 100,000 to 200,000 rupees then you pay 10 percent of the amount above 1 lakh, 100,000. For income above this 200,000 if the income is above 200,000 then you pay 20 percent of the amount whole amount not how much is exceeding 10,000 suppose this is our income tax policy. How can we do this?

So, what is the input that I need from the user? I need this income from the user right; and what is the other variable that I want to compute? The tax. So, here earlier I needed math dot leave in this case probably I will not need that, but, I will not need that if needed I will can always add it later main and then you can say put a comment computation of income tax, tax computation all right.

Now, I start my program all right I can put the parenthesis above, does not matter . Then printf its always better to do this, "Please enter your income" because if you do that then you make your program interactive; that means, the user can see what you are doing right, please enter your income. Well I have not yet declared the variables at all. So, here I should write float income and tax. Then here scanf, percentage f and income, so I read the income here fine. Then what should I do? Next let me come to the next page then or

because there will be number of if conditions. So, this part is ok, I have read then come here.

(Refer Slide Time: 14:11)

```
if (income <= 100000)
    printf("No tax\n");
else if (income <= 200000)
    { Tax = 0.1 * (income - 100000);
      printf("Tax = %.f\n", Tax);
    }
else
    { Tax = 0.2 * income;
      printf("Tax = %.f\n", Tax);
    }
```

Now, if income is less than or equal to 100,000 printf no tax, else what is my principle; 1 lakh to 2 lakhs, 10 percent of the income, 10 percent of the income above 100,000, so else if income is less than equal to 200,000. So, it is not when it is coming here it is already greater than 100,000, if it is less than 200,000 then tax will be 10 percent; that means, 0.1 times the amount that the amount exceeding 100,000, so income minus 100,000. Printf tax equals percentage f backslash n tax, is this part clear.

This part is so when am I coming here this if condition as failed. So, I am coming here; that means, it is not less than 100,000 so it is more than 100,000, but if it is less than 200 less than equal to 200,000 then I am the second bracket and I will be paying 10 percent of the amount that is exceeding 100,000 that is why I computed this and then I am printing this. So, here it is more than one statement I put a bracket. Here you see under if I had only one statement. So, putting a bracket is not mandatory, but here it is mandatory otherwise it will mean something else.

So, else this, now if this is also not true, that is if this if does not satisfy then I am in the third bracket that is here what should I do else tax is 20 percent of the income and printf tax assigned percentage f backslash n tax. Now, here then I will come to the end. So, I come to this beginning this, this beginning point and I end the program here.

Now, you can see that I can reduce this program a little bit. How? I have written this printing the text this thing the same thing twice, I am sorry here they should be another bracket for completing this. This should be completed and then the next bracket. Now, this I could have done later after these 2 use all right, but under this else under this else. So, you can also try to reduce it and as an assignment you should run this program and get yourself satisfied all right. So, this is another classical example of if then else usage all right.

Given this we will. Now, move to some more examples, some more examples of the other construct that is while and do while type of constructs. So, here is an example to show if a number is prime or not.

(Refer Slide Time: 19:23)

```
#include <stdio.h>
main()
{
    int n, i=2;
    scanf ("%d", &n);
    while (i < n) {
        if (n % i == 0) {
            printf ("%d is not a prime \n", n);
            exit();
        }
        i++;
    }
    printf ("%d is a prime \n", n);
}
```

The slide includes handwritten annotations: 'n' above the variable declaration, 'i' above the loop counter, and a vertical list of numbers 2, 3, 4, 5 next to the loop condition. A circled '12' is visible in the bottom right corner.

Now, how do I go about it let us first think of the algorithm for finding out whether a number is prime or not.

(Refer Slide Time: 19:45)

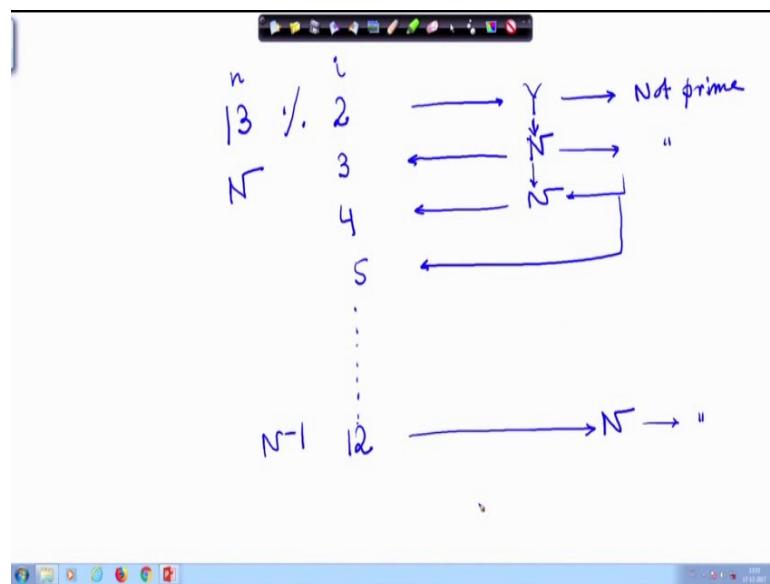
The whiteboard contains the following handwritten notes:

- A circled number 13 with a curved arrow pointing from it to the number 2.
- Handwritten modulus operations:
  - $14 \% 2 = 0$
  - $26 \% 2 = 0$
  - $13 \% 2 = 1$  (This line is enclosed in a rectangular box.)
  - $15 \% 3 = 0$

Say for example, I take a number 13. I want to find out whether 13 is prime or not. So, when is a number called prime? When the number is not divisible by any other number other than the number itself and 1. So, in order to find out whether 13 is prime or not, what should I do? I will start with 2 and I will try to see whether the 2 is dividing 13. How do I know where the 2 is dividing 13? By the modulus operator, if there will be 2 integers say 14 modulus 2 the modulus leads gives me the remainder, sorry remainder. So, if 14 is divisible by 2 the remainder will be 0. 26 divided by 2 the remainder will be 0, but 13 divided by 2 the remainder will be 1.

So, similarly, suppose I want to see where the number 15 is divisible by 3 I will do the modulus operation of 15 with 3 and the remainder will be 0. So, 15 is divisible by three right.

(Refer Slide Time: 21:32)



So, now if I want to find out whether 13 is a prime or not; I will first try to divide see where that is divisible by 2. If it is divisible yes, will tell me not prime right it will tell me not prime, but no then I will try to divide it by 3, if this is divisible by 3 then again not prime. No, I will divide by 4 then; obviously, not divisible then I will divide it by 5 so on so forth I will go till I divide it by 12. Since of course, by 13 it will be divisible, if the answer continuously up to 12 is no then we say it is not a prime right, not a prime.

So, here you see there are 2 things happening. One is I am taking a check branching here I am dividing it by 2, if it is divisible is it is not a prime, if no then I am taking another path. What is that path? That path is again trying with the next successor of this 2, yes; if the result is yes I take this path otherwise I will follow this path with a successor of this. So, the same thing trying with the successor I am doing continuously repeatedly if the divisibility result is false; that means, if it is not divisible I will continuously go on till I reach if this number is N, till I reach N minus 1 right, up to that I will try up to N minus 1 I will try. So, this is an example where there is a branching as well as a looping, another real life mathematically important application. So, let us look at the program now.

Let us try to understand the program here. This include stdio dot h is known to you mean. I am declaring 2 variables one is n another is i; n is an integer, i is also an integer, initialize to 2. I am reading the number here say 13 while i is less than n; that means, i

was 2 if we look at this page we started with 2 we started with 2 and then went on right, this is the value of i and this is the value of n.

Now, while i is less than n, I had 13 and i is 2 this is n, I will go on up to 13 up to while i is less than n that means up to 12, I will go on trying this thing what; if n is divisible by i. Is n divisible by i, yes then print that the number n is not a prime and then exit we come out of this loop. Just like break we come out of this loop, but break is not applicable in if statement exit we can write. So, I come out of the loop.

Otherwise what I am doing still in the while loop, the while loop is starting from this point and going up to this point. I am incrementing i, so i becomes 3 and again I go and try to see if it is divisible with 3 know then I come make it 4, make it 4, I will again try this one fails, it is not divisible so I do not do this part, I do not do this part. I come here make it 5 and go on in this way while i is 12, up to 12 I check and then it becomes 13. So, I come here and ultimately I come out of the loop and till 13 I could see that no, I could find no number no integer that is dividing it. So, we say that this is a prime number.

Say for example, what would have happened if I had given the instead of n 13 if I had given it 14. What would be the change in this flow? I would start with the 2 and again I divide it by 2, try dividing it by 2 it is divisible I print that it is not a prime and I exit and come out of this entire while loop. So, in the while loop here this I am exiting from the while loop all together, not from the if loop I am exiting from the while loop if is not a loop by the way. I am exiting from the while loop completely. So, this example also illustrates the use of if here and the while loop here all right.

We will see a few more examples in the future lectures.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 24**  
**Example of Loops (Contd.)**

So in the last lecture we have seen how we can write a program to find out whether it is prime or not.

(Refer Slide Time: 00:19)

The slide title is "Example 1: Test if a number is prime or not". The C code is:

```
#include <stdio.h>
main()
{
    int n, i=2;
    scanf ("%d", &n);
    while (i < n) {
        if (n % i == 0) {
            printf ("%d is not a prime \n", n);
            exit();
        }
        i++;
    }
    printf ("%d is a prime \n", n);
}
```

A hand-drawn diagram to the right shows a vertical list of numbers from 2 to 12. Above the list, "n" is written above 13, and "i" is written above 2. A bracket labeled "n-2" covers the numbers 3 through 11. A bracket labeled "n" covers the number 12. To the right of the list, "11 times" is written, indicating that the loop iterates 11 times from i=2 to i=12. A blue arrow points from the word "times" to the number 11.

Now, this program that you see that we have we have discussed in the last class, we will certainly give us correct result. Now our job is not only to write a correct program it is also very stimulating intellectually to think of how we can make it a more efficient, program often the program can be correct, but it is not very efficient. What do I mean by this efficiency? Say if we look at this program and again I take the earlier example that suppose I have my n is 13.

Now, and I start with the value of i to be 2 and n is 13 and since it is prime I am checking this condition time and again and since this is third I mean prime number how many times it will never come here, it will never succeed. So, I will have to carry on this loop and ultimately when I exhaust all the possibilities then only I will come to this point.

So, how many times do I have to do this let us see, I am starting with 2 checking ones then start incrementing and making it 3, then 4 then testing with 5, then testing with 6, testing with 7, testing with 8, testing with 9 10 11 12. So, how many times did I check it 1 2 3 4 5 6 7 8 9 10 11 times, I had checked it with 11 times

Now, do I really need to check it with 11 times? If you think a little bit if this number is  $n$  then, I did not check it and I am starting with 2. So,  $n$  minus 2 times I am checking right do I really need to check  $n$  minus 2 times. If I do not find it divisible by squared within square root of  $n$  times. So, square root of 13 as an integer what would it be? 13 would be 3 point something right. So, 4 times within 4 if it is not divisible, it will not be divisible late by the future numbers also right.

So, if I just apply this knowledge that if it is not divisible within square root of  $n$ , then it will not be divisible later I can make this program much more efficient how let us see, let us look at the and a more efficient version of the program.

(Refer Slide Time: 03:48)

```
#include <stdio.h>
main()
{
    int n, i=3;
    scanf ("%d", &n);
    while (i < sqrt(n)) {
        if (n % i == 0) {
            printf ("%d is not a prime \n", n);
            exit();
        }
        i = i + 2;
    }
    printf ("%d is a prime \n", n);
}
```

So, here you see I am starting with 3 and I am reading the value of  $n$  fine. So, far no problem I am starting with  $n$ .

Now, here I am checking not up to  $n$  I am checking I mean in the earlier case what we did is sorry earlier case what we did is we tested with up to  $n$ , but here what we are doing is I am testing with up to square root of  $n$  and if  $n$  is divisible by  $i$ ; that means, say 13, I

first try with 3 it is not divisible is try with 4 and. So, square i is for that is less than the greater than the square root of n right. So, with that I will come out and every time what I am doing is I am implementing i by 2. So, I am testing with 3, I am testing with 5, I am testing with 7 like that I am going on. If I apply this then I can always make this program more efficient clear.

(Refer Slide Time: 05:48)

The slide has a title 'More efficient??' with a question mark. Below it is a C code snippet. Handwritten annotations include: 'i' above the variable 'i' in the code; '21 ← 3 ✓' where '21' is crossed out and '3' is written with a checkmark; '23' below '21'; and '5' below '23'. There are also small checkmarks next to the printf statements in the code. The code is:

```
#include <stdio.h>
main()
{
    int n, i=3; /
    scanf ("%d", &n);
    while (i < sqrt(n)) {
        if (n % i == 0) {
            printf ("%d is not a prime \n", n); ✓
            exit;
        }
        i = i + 2; /
    }
    printf ("%d is a prime \n", n);
}
```

So, you see by the. So, let us take another example here say I am trying to find out whether 21 is a prime number or not. So, I start with i to be 3. So, 3 is of course, 21 will not work sorry let me change it 21 is not a prime. So, first time in first short it will give me that it is not the prime fine, but if instead I had a 23 then what will happen? I start with 3, it does not divide then I come and make it 5. If something is not divisible by 3 it will not be divisible by I mean next square root of that I am just going by 1 more I come to 5 it is not divisible by that, what is the square root of 23? It should be 4 point something. So, again quickly I am coming to the point where I can see that it is not divisible. So, by doing this using this small technique, I can make it a little faster the number of checks will be much more reduced.

So, now, let us go to another example.

(Refer Slide Time: 07:04)

The slide contains the following text and code:

Example 2: Find the sum of digits of a number

```
#include <stdio.h>
main()
{
    int n, sum=0;
    scanf ("%d", &n);
    while (n != 0) {
        sum = sum + (n % 10);
        n = n / 10;
    }
    printf ("The sum of digits of the number is %d \n", sum);
}
```

Handwritten calculations for the sum of digits:

$$\begin{array}{r} 123 \\ +2 \\ +3 \\ \hline 6 \end{array}$$
$$\begin{array}{r} 243 \\ +4 \\ +3 \\ \hline 9 \end{array}$$

This is again finding the sum of and sum of the digits of a number. Now this 1 we did not do sum of the digits of a number what is meant by this? Suppose I have got a number 123, sum of the digits of a number means 1 plus 2 plus 3 6. If my number is 243 then the sum of the digits will be 2 plus 4 plus 3 6 and 3, 9 this is what I want to find out.

So, how do you go about doing that, how do I find out the digits? Say 1 2 3 from there how do I extract out 1 2 and 3.

(Refer Slide Time: 08:05)

The slide contains the following text and code:

Example 2: Find the sum of digits of a number

```
#include <stdio.h>
main()
{
    int n, sum=0;
    scanf ("%d", &n);
    while (n != 0) {
        sum = sum + (n % 10);
        n = n / 10;
    }
    printf ("The sum of digits of the number is %d \n", sum);
}
```

Diagram illustrating the extraction of digits from 123:

123 is divided by 10, resulting in 12 and a remainder of 3. The remainder 3 is then divided by 10, resulting in 3 and a remainder of 0. This process is shown for each digit of the number 123.

Just think over if I have got 123 and I divide it by 10, I will get a quotient 12 and a remainder. So, the remainder is 1 digit, now I take the quotient divided by 10, I will get a remainder sorry a quotient and the remainder, the remainder is another digit and then I go on till it is the number is not equal to 0. So, next I take one and divide it by 10, the quotient is 0 and the remainder is 1.

So, you see I have got all the digits from the right hand side one after another right. So, I can add them that is my basic way. So, let us keep it on the side and see what we have done in this program. I have got a number n, now this number that has been given initially 123, I do not need to preserve that number. So, I can play with that and I have to make a sum of this right 1 2 and 3. So, I keep a sum to be initially 0. I keep a variable sum which is initialized to 0 as we saw in the earlier examples and sum is also an integer then I read the number that is say 123 has come.

Now, while the number n that is 123, 123 is not zero then I take add n modulus 10 n modulus 10 will give me what? The remainder 3 and add it to the sum. So, my sum now becomes 3, and then I want to find out this thing also. I make my n to be n divided by 10 now this operation gives you the quotient right this operation gives you the remainder.

So, I now I get n by 10 means now 12, and I go back to this loop I check that 12 is not zero. So, I will again do that, sum will be sum what was that 3 plus n modulus 10. So, I again divided by 10 and take the remainder and that remainder is added sum plus this. So, 3 plus 2 will be 5 and then I divide n by 10. So, now, I find out 1 still I go back 1 is not I go back actually here, 1 is not zero sure. So, I again divide it by I take the remainder divided by 10 and take the remainder. So, the remainder is 1, I added with the sum. So, it will be 6 and then I divide I find the quotient and that is 0, I go back again here and find that n is not equal to 0 condition is not true. So, I come out of the loop and I apply this printf statement what is the printf statement doing? The sum of digits of the given number I cannot say given number n because n has already changed I will come to that in a moment.

Now, it is sum. So, I have got the sum. So, this is another example of while loop now my question is if you have understood it, I want the output to come as printf; the sum of digits of the number 123 is 6 how can I do what modification should I do in this program.

(Refer Slide Time: 13:02)

The screenshot shows a Microsoft Word document containing a C program and its execution output. The program calculates the sum of the digits of a number entered by the user. Handwritten annotations explain the logic: 'num = n' points to the assignment in the main loop; '123 -> 12' shows the division by 10; '12 -> 1' shows the next division; and '1 -> 0' shows the final digit. A blue box highlights the printf statement.

```
#include <stdio.h>
main()
{
    int n, sum=0;
    scanf ("%d", &n);
    while (n != 0) {
        sum = sum + (n % 10);
        n = n / 10;
    }
    printf ("The sum of digits of the number is %d \n", sum);
}
```

The output window shows the result: "The sum of digits of 123 is 6".

What I want to be printed is the sum of digits of 123 is 6 what should I modify? Of course, I should modify this statement, this statement should be the sum of the digits of the number percentage d is percentage d, and here before sum what should I do? My n that was provided by the user has already been destroyed in this loop.

So, what because every time I am dividing it by 10 and making a new number, 123 is becoming 12 and then it is becoming 1 then is becoming 0. So, I have to save it somewhere this 123. So, what I can do here after this reading this, I can save it in another variable num has n and here I will put num comma now because I am destroying n here, but I am not touching num. So, num will remain intact only another thing that I have to do I have to add this num here I must declare it num as an integer otherwise it will give you syntax error all right.

So, this is another nice example of while using which we can find the number of digits the sum of the number of digits of a number.

(Refer Slide Time: 15:04)

The screenshot shows a Windows desktop environment. A Notepad window titled "Example 3: Decimal to binary conversion" is open. It contains the following C code:

```
#include <stdio.h>
main()
{
    int dec;
    scanf ("%d", &dec);
    do
    {
        printf ("%2d", {dec % 2});
        dec = dec / 2;
    } while (dec != 0);
    printf ("\n");
}
```

Handwritten notes next to the code show the conversion of the decimal number 100 to binary. A vertical line is drawn through the numbers 4, 5, 1, and 2. To the right of the line, the binary digits 100, 101, 001, and 010 are written vertically.

A small video window in the bottom right corner shows a man in a green shirt speaking.

With that we move to another example; decimal to binary conversion some of you may know it and some of you may not be very conversant with that. We know that we have got the decimal binary number system where the base is 2. So, everything is expressed using zeros and one.

So, sorry if I have a binary digit sorry decimal digit 4, it is binary equivalent is 1 0 0, if we have 5 that is 1 0 1, if it is 1 then it is 0 0 1 or just 1, if it is 2 it is 0 1 0 like that the question is that given our decimal digit how can I convert it to binary what is the algorithm? The algorithm is something like this say I take 4 and divide it by 2.

(Refer Slide Time: 16:23)

The slide contains a C program and a binary division diagram.

```
#include <stdio.h>
main()
{
    int dec;
    scanf ("%d", &dec);
    do
    {
        printf ("%2d", (dec % 2));
        dec = dec / 2;
    } while (dec != 0);
    printf ("\n");
}
```

Hand-drawn binary division diagram:

$$\begin{array}{r} 2 \mid 100 \\ 2 \mid 0 \\ 2 \mid 0 \end{array}$$

The quotient is 100 and the remainder is 0.

So, the remainder is 2 a sorry the quotient is 2 and the remainder is 0 right and then I divide it by 2 again, the quotient is 0 and this is 0 now when I divide it by 2 further what will happen [FL] [FL] [FL][FL]

[FL]

[FL]

(Refer Slide Time: 17:50)

The slide contains a C program and a binary division diagram.

```
#include <stdio.h>
main()
{
    int dec;
    scanf ("%d", &dec);
    do
    {
        printf ("%2d", (dec % 2));
        dec = dec / 2;
    } while (dec != 0);
    printf ("\n");
}
```

Hand-drawn binary division diagram:

$$\begin{array}{r} 2 \mid 100 \\ 2 \mid 0 \\ 2 \mid 0 \end{array}$$

The quotient is 100 and the remainder is 0.

[FL] [FL] yeah [FL].

(Refer Slide Time: 18:36)

The slide title is "Example 3: Decimal to binary conversion". Below it is a C code snippet:

```
#include <stdio.h>
main()
{
    int dec;
    scanf ("%d", &dec);
    do
    {
        printf ("%2d", (dec % 2));
        dec = dec / 2;
    } while (dec != 0);
    printf ("\n");
}
```

To the right of the code is a hand-drawn binary division diagram for the number 4. The divisor is 2. The quotient is 2, and the remainder is 0. This step is repeated once more, resulting in a quotient of 1 and a remainder of 0. The final remainder is 0. An arrow points from the number 100 at the top right down to the final remainder 0.

Let us take the number 4 and I want to find out what is the binary equivalent of this, the algorithm goes like this I first divide it by 2, because 2 is the base of any binary system I divide it by 2. So, I get the quotient as 2 and remainder as 0.

Next I divide it again by 2. So, my quotient is 1 and remainder is 0. Next I divide it again by 2 my quotient is 0 and remainder is one. So, I go on dividing till I get a quotient to be 0 and I have remembered all the remainders that I got now I can get the binary if I did it in this direction. So, it is 1 0 0 all right.

(Refer Slide Time: 19:53)

The slide title is "Example 3: Decimal to binary conversion". Below it is a C code snippet:

```
#include <stdio.h>
main()
{
    int dec;
    scanf ("%d", &dec);
    do
    {
        printf ("%2d", (dec % 2));
        dec = dec / 2;
    } while (dec != 0);
    printf ("\n");
}
```

To the right of the code is a hand-drawn binary division diagram for the number 15. The divisor is 2. The quotient is 7, and the remainder is 1. This step is repeated twice more, resulting in quotients of 3 and 1, and remainders of 1 and 1 respectively. The final remainder is 1. To the right of the division steps is a binary powers of 2 diagram showing  $2^3, 2^2, 2^1, 2^0$  with values 8, 4, 2, 1 below them, and a sum of 15. A vertical arrow points from the remainder 1 in the third division step down to the 1 in the binary powers of 2 diagram. A small video window in the bottom right corner shows a teacher speaking.

Let us take another example suppose I want to have the number 15, now what is the binary of 15 to understand that you let us look at the weights of the different positions in binary, this one is with 2 to the power 0, 2 to the power 1, 2 to the power 2, 2 to the power 3; that means, this is 8 4 2 1 now 15 if I have to have I must have a 1 here and 7 more. So, 1 here 8 and 4 12 and 1 here 12, 13, 14 and 15 all this should be 1.

So, can we find out find this out let us see 15 I divide by 2 my quotient is 7 and remainder is 1. I again divide it by 2 my quotient is 3 and remainder is 1. I divide it by 2 again my quotient is 1, remainder is 1, I divide it by 2 again my quotient is 0 and the remainder is 1, I did it in this direction and we get 1 1 1 1 all right.

(Refer Slide Time: 21:22)

Let us take another example again the weights of the system are 1 2 4 8. 2 to the power 0 2 to the power 1, 2 to the power 3, 2 to the power sorry 2 to the power 2, 2 to the power 3 now let us take a number and next one will be 16 is 2 to the power 4.

Now, let us take a number 23, what will be the binary representation of that let us see here 16 will be there of course, I have greater than 16. So, 23 16 and 6 and 7 more. So, this should be 0 I cannot put a 1 here, because there to be 16 and 8 will be 24. So, 1 here 20 this this. So, my binary is 1 0 1 1 1 let us try in our algorithm divide it by 2, my quotient is 11, I have got a remainder 1 I divide it by 2 my quotient is 5 remainder is 1, I divide it by 2 quotient is 2 remainder is 1 divided by 2 is 1 and 0 and divided by 2 is 0 and 1. So, I reach a 0 I stop and this is the pattern 1 0 1 1 1, 1 0 1 1 1 all right.

Now, let us see how this algorithm that we are talking of can be encoded in C program. You see here I have declared 1 variable as dec; that means, the decimal number that I will be reading. So, I am reading that number here. Now in the earlier example we did it while do now here we are doing do while we are doing something and then I am putting the while condition later. Do what am I doing printf this means percentage 2 d means I am printing in binary, I mean this I am taking this dec and I mean 1 beta I am printing I am defining out the remainder I find out the remainder and then divide it.

And I go on doing this I divide it and I go on doing this while this is the number the remainder is not equal to 0. I am dividing it by 2 and finding out the new number, that this is this point what am I doing? I am finding out the quotient, here I am finding out the quotient and I am based on the quotient I am going on till I get this 0. Do not bother about this I can simply take the this thing, first this thing and I go on printing it.

So, here what I am printing I am printing in this in this format I am printing in this format let us forget about this for the time being all right let us put percent d then I am dividing it like this way all right. I divide by 2 continually and I am printing in this way 1 then I change it to 11 and then divide it then dec becomes 5, I print it and go on doing it. So, at least once I am dividing till it is 0. This is the algorithm for decimal to binary conversion you can also try to write it not with do while, but just by while you can I leave it as an assignment for you to write it using while some condition and not do while not do while.

(Refer Slide Time: 25:57)

```
#include <stdio.h>
main()
{
    int A, B, temp;
    scanf ("%d %d", &A, &B);
    if (A > B) { temp = A; A = B; B = temp; }
    while ((B % A) != 0) {
        temp = B % A;
        B = A;
        A = temp;
    }
    printf ("The GCD is %d", A);
}
```

Initial: A=12, B=45  
Iteration 1: temp=9, B=12, A=9  
Iteration 2: temp=3, B=9, A=3  
B % A = 0 → GCD is 3

So, next let us move to another example that is also very interesting and known to us that is finding the greatest common divisor of 2 numbers what is the algorithm? The algorithm is shown here, suppose I have got a number 45 and 12, I am to find out the greatest common divisor or the hcf of these 2 numbers. What I do is I divide the bigger number by the smaller number and get a remainder. You see remainders are becoming. So, important they come to the remainder

And then if the remainder is not zero, I divide now the smaller number this 1 with a remainder and then I get again a remainder which is nonzero, then I take the divisor current divisor and divide it by this until we get 0 that is what we used to do in school right. So, ultimately when I get 0 whatever is the divisor currently current divisor that is our hcf or gcb. Here you see how the code is read you just look at the code a little bit and try to understand it.

A and B are 2 numbers, now I do not know as it which one is smaller and which one is bigger. Here in our example we can see one is 12, one is 45, but one is 12 one is 45, but I am going to try this here the computer does not know which one is 12 which 1 is 45 and I am taking another variable temp I am reading the numbers A and B. Now if a is greater than b; that means, I am taking the greater number and moving that in temp out of this 12 and 45 which 1 is the greater number 45.

So, temps becomes 45 and b becomes a B is the smaller number. So, 12 becomes c and b becomes temp, now you see I have written these 3 in 1 shot. So, let us see here what is happening after this first temp is 45, A is 12 and B is 45 now what I am doing here I am keeping this temp what I am doing here I am dividing B by A. So, 45 is being divided by A 12 and I am checking the remainder the remainder is not zero you can see that. Since it is not zero I am taking temp to be the modulus of 12 sorry b is what? B was 45 right and A was 12 by this point. So, then and temp was 45 I try this it is not zero.

So, then I take this remainder 9 and again make that to be b. So, now, B becomes 9 and A becomes the temp which was this number and I go on dividing this and I go on carrying out this loop or till this becomes 0. As soon as this becomes 0 then I have got my GCD stored in A, because every time I am taking the A in A I am storing the temp and what is temp? Temp is after I divide I get the remainder and that I am storing in temp and that temp is coming in A all right it is being stored in a and that is also copied in B, but in the next loop that is being divided by. So, here you are having 3 9 and here 3. So, look at this iteration 1, here looking at this picture first temp will become 9, B is 12 and A is 9.

Next time temp is becoming 3, B is 9 this 9 and A is becoming 10 and whenever I find that B divided by A this part is 0 remainder is 0 then that is my GCD A is my GCD, I get the 3 there is another very interesting example of computing using this while loop and repeatedly I am doing this till I find the remainder to be 0. So, there are many such nice examples and we will give you some more during for assignments, which we will have to solve and get more you will get more confidence about it. So, next we look at we will come to some of the pitfalls of loop, and then move to something more useful something very useful that is the concept of arrays ok.

Thank you.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 25**  
**Example of Loops (Contd.), Example of For Loops**

So, we have seen the application of the constructs of if then and if else as well as while, do while and for constructs. Till now you have seen examples in of while and do while, will see more examples for loops for example.

(Refer Slide Time: 00:47)

Some Loop Pitfalls

```
while (sum <= NUM) {  
    sum = sum+2;  
}  
  
for (i=0; i<=NUM; ++i) {  
    sum = sum+i;  
}
```

Sum = 0  
NUM = 10

```
for (i=1; i!=10; i=i+2)  
    sum = sum+i;
```

```
double x;  
for (x=0.0; x<2.0; x=x+0.2)  
    printf("%.18f\n", x);
```

2  
4  
6  
8  
10  
12

Let us also see another application for the application of for loop. Say for example, I want to print I want to add 20 numbers I think we have seen such examples will see more interesting examples a little later. But before that let us look at some common errors that take place often unintentionally in writing loops and that gives rise to a number of logical problems in a program.

For example, let us look at this. Here you can see the first line what will happen, the intention was that while sum is less than equal to NUM; that means, it is less than equal to sum a particular value may be 20 we are adding sum plus 2. What does this mean? Let us try to understand this example forget about this, forget about this part. What would what is the intention of doing this? That means, suppose NUM is 10 and sum is 0. So, while sum is less than NUM I will be adding sum and 2. So, sum will be, sum will be 2

and then sum is still less than num so again it will be 4, sum is still less than NUM it will be 6, still less than NUM it will be 8, still less than equal to NUM it will be 10, then still less than equal to NUM it will be 12 and then when it goes there it will stop. So, what will be the sum? Sum will be 12. But that was my intention of the program.

But unfortunately I have put a semicolon here. What does that imply? That implies that the entire while statement ends here; that means, while sum equal to NUM do nothing all right. So, that is the end of the statement. So, nothing has been specified there and whatever some was there suppose sum was 0 sum will perpetually remain less than 10 or NUM and will go on forever. So, this semicolon should not to be given because the while statement is actually extending up to this point up to this point that is the whole statement.

So, next example we take is this one, for i assign 0, i less than equal to NUM plus plus i, forget about, again about this.

(Refer Slide Time: 04:08)

**Some Loop Pitfalls**

NUM=20

i>0      SUM

0
1
2
3

↓

double x;  
for (x=0.0; x<2.0; x=x+0.2)  
printf("%f\n", x);

What is the intention of this program? What will it do? i is 0 all right and NUM was say something like 20 then sum will be added to i. So, sum will be 0, then i will be incremented, so i will become 1 less than NUM it will be added. So, sum will now be, sum was 0 so sum will now be 1, here was sum, sum was 0, sum becomes 1, then sum becomes 2, sum becomes 3 like that it will go on all right.

However, since I have put a semicolon here this part is not a part of this for statement. Consequently this loop is a non loop nothing is being done here and for i equals 0, less than equal to NUM i plus plus do nothing all right. So, nothing will happen here this statement will not be executed.

Here is another type of pitfall where let us study this for i equal to 1, i not equal to 10, i assigned i plus 2. What will happen to this loop? Can anyone guess?

(Refer Slide Time: 05:42)

The slide has a title 'Some Loop Pitfalls' and contains the following content:

- Top Left:** A code snippet with a circled 'i' in the condition:
 

```
while (sum <= NUM);  
    sum = sum+2;
```
- Top Right:** A code snippet with a circled 'i' in the condition:
 

```
for (i=0; i<=NUM; ++i);  
    sum = sum+i;
```
- Middle Left:** A code snippet with a circled 'i' in the condition:
 

```
for (i=1; i!=10; i=i+2)  
    sum = sum+i;
```

Handwritten notes next to it: 'infinite loop' and 'i = 1, 3, 5, 7, 9, 11'.
- Bottom Left:** A code snippet:
 

```
double x;  
for (x=0.0; x<2.0; x=x+0.2)  
    printf("%.18f\n", x);
```
- Right Side:** Handwritten calculations showing the value of 'sum' increasing from 1 to 25:
 

sum = 0 + 1 = 1  
4  
9  
16  
25

i is 1, so sum has been computed sum is sum plus i sum was 0. So, 0 plus 1 sum is 1, then i is incremented to 3, then sum equals sum plus i. So, sum becomes 4 then this is incremented to 5. After each incrementation I am checking with this condition, so i is 5 not equal to 10 fine. So, I will add 5 with this, it will become 9, then becomes 7 i is change to 7, I check with this still not equal to 10. So, 7 is added to this 16, this becomes 9 still not equal to 10. So, then 9 plus; 16 plus 9 maybe 25 and then this is incremented to 11 because i plus 2, still it is not equal to 10. So, what will happen? It will go on it will never be equal to 10. This means as long as i is not equal to 10 you will go on. So, this will be a case of any another infinite loop all right.

Just as in this case this is a null statement this will be done and then this statement will be done only once. Here it will be an infinite loop because this condition will never be made. So, these are some points where we should be very careful about.

(Refer Slide Time: 07:40)

The slide title is "Nested Loops: Printing a 2-D Figure". Below it is a hand-drawn diagram of a 2D star pattern. The pattern consists of three rows of five asterisks each. A bracket on the right side groups the top two rows, and another bracket on the left side groups all three rows. To the right of the diagram is a C code snippet with handwritten annotations:

```
for(i=0; i<5; i++)  
    printf(" * ");  
    printf("\n");
```

Annotations include:

- A bracket on the left labeled "repeat 3 times print a row of 5 \*'s" points to the outer loop.
- A bracket on the right labeled "repeat 5 times print \*" points to the inner loop.
- A callout bubble points to the "printf(" \* ");" line with the text "repeat 5 times print \*".
- A callout bubble points to the "printf("\n");" line with the text "repeat 3 times".
- A callout bubble points to the "for(i=0; i<5; i++)" line with the text "for(i=0; i<5; i++)".

Now, let us have a very interesting program. We want to print we want to print, we want to print a pattern like this. We want to print the pattern like this say 5 stars in a row and 3 such rows, this sort of pattern. How can I do that? My algorithm will be, I want to print, I want to print a row of stars. How many? 5 stars. So, how can I print 5 stars in a row? If I just write one statement `printf`, star I do not give a backslash n; then what will be done? A star will be suppose this is my screen all right a star will be printed and if I put it in a loop say for i assign 0, i 5 I want to do, less than equal to 5 i plus plus and I do this then what will happen, i 0, so once it is printed then i 1 once it is printed, again in the same line then again another one.

So, since I am giving a gap what I can do here I can keep a space here star and then a space I show space by blank. So, exactly a star in the blank will be printed. So, i 0, i is 1, i is 2, i is 3, i is 4 and then it is incremented and checked is 5, so less than 5 it will not happen then. So, this will be a loop after doing that. So, in a loop I will be printing one row then I will give `printf` I have to come to the next line. So, I will simply give a backslash n; that means, I will come to the next line. And this again loop I will carry out how many times? 3 times because I need 3 rows. So, what should I do? I should do this again loop this 3 times so how should I write it, how would that look like now.

(Refer Slide Time: 11:11)

Nested Loops: Printing a 2-D Figure

- How would you print the following diagram?

```
*****  
*****  
*****
```

for (j=0; j<3; j++)  
{  
 for (i=0; i<5; i++)  
 printf("\*\b");  
 printf("\n");  
}

repeat 3 times  
print a row of 5 \*'s

repeat 5 times  
print \*

\* \* \* \* \*

The slide shows a C code snippet for printing a 2D figure. The code uses nested loops: an outer loop for 'j' (0 to 2) and an inner loop for 'i' (0 to 4). It prints a star followed by a backspace to align the stars in a single row, then prints a new line after each row. Handwritten annotations explain the logic: 'repeat 3 times' points to the outer loop, and 'repeat 5 times print \*' points to the inner loop. The resulting output is a 3x5 grid of stars.

It should be something like this for  $j$ ,  $j$  is another variable assign 0,  $j$  less than 3,  $j$  plus plus for  $i$  assign 0,  $i$  less than 5,  $i$  plus plus and then here `printf` star followed by a blank and then the quote no backslash at the end of this. So, after this loop is done then I will do `printf` backslash n backslash n. So, this part will be repeated 3 times and in this part it will be this one will be done 5 times. So, star star star star star will be printed. Then we will come to the next line and here this is my next for loop. So, I come to `printf` and again do the same thing this part start start start start star 5 times, this `printf` by this loop. This is printed row, printing a row by 5 times. And then again I will come and do a backslash n, I come here and do the same thing 3 times and then come here backslash n and then stop.

So, print a row of 5 stars and repeat therefore, I am repeating this print star 5 times in a loop. So, that is a very nice interesting application of for loop. I hope you have understood this.

(Refer Slide Time: 13:38)

```
const int ROWS = 3;
const int COLS = 5;
...
row = 1;
for (row = 1; row <= ROWS; row++) {
    /* print a row of 5 *'s */
    ...
    ++row;
}
```

```
row = 1;
while (row <= ROWS) {
    /* print a row of 5 *'s */
    col = 1;
    while (col <= COLS) {
        printf("* ");
        col++;
    }
    printf("\n");
    ++row;
}
```

So, here that is exactly what I have been showing you. Look at this, here the number of rows and columns have been made little flexible rows 3 columns 5. Now, row equal to 1, while row is less than rows print a row of 5 stars. This I have done it is the for, here it is shown using a while.

So, let us see whether you can understand this also with the while. Row is 1, now row is less than rows; how many rows we will do, row is less than rows that means, as long as it is sorry as long as it is 3 rows will print a row of 5. And how do I do a row of? I will that we have already shown that how we do it and then we increment the row all right. So, here while row is less than rows print a row of 5, printing a row of 5 is done through in this manner. So, this is the outer loop, this is outer loop. Column is 1, while column is less than columns, so 5 columns 1 2 3 4 5 while column is less than column printf star and blank and then column is incremented.

Now, since I am doing it in a while it is done in this way. I have already shown you in the earlier this thing how we can do it with for. I can do the same thing with for right, the same thing with for and here it is being shown how it can be done with a while. And then I print of n and do this. You can try to understand this again yourself.

(Refer Slide Time: 15:54)

2-D Figure: with for loop

```
Print
*****
*****
for (row=1; row<=ROWS; ++row) {
    for (col=1; col<=COLS; ++col) {
        printf("*");
    }
    printf("\n");
}
```

Next, say here this being done again in the way that I had written using for. Here only 3 and 5 these things are variable for row equals 1 to row less than equal to 3, plus plus row. Here it is plus plus row; that means, first is incremented. Then column is less than equal to columns. Why it is less than equal to, while if you remember when I was doing it here when I was doing it I had less than 3 less than 5, less than 3 less than 5, but here it is being less than equal to 3 less than equal to 5 why because I started my index with 0 and here I am starting my index with 1, all right. So, this you should be very careful and you should always hand trace your program and see whether you have done it correctly if there is a little bit of confusion because this is very, this very important and you should be very careful about it.

(Refer Slide Time: 17:19)

The slide title is "Another 2-D Figure". To the left, there is a hand-drawn diagram of a right-angled triangle pointing downwards. The first row has one asterisk (\*). The second row has two asterisks (\*\*). The third row has three asterisks (\*\*\*) and so on, up to five rows. A bracket on the left is labeled "Print". To the right is a code box containing:

```
const int ROWS = 5;  
....  
int row, col;  
for (row=1; row<=ROWS; ++row) {  
    for (col=1; col<=row; ++col) {  
        printf("* ");  
    }  
    printf("\n");  
}
```

Below the code box is the file name "2d-figure.c".

So, same thing that I have shown is written again here. Another 2D figure this a little more interesting. First row we print 1 star, second row we print 2 stars, third row we print 3 stars, 4th row we print 4 stars and then 5 stars. So, how many stars I will print that is also variable. So, if we think about that how many times I will print in a row that is also a variable. How many times I am doing this?

(Refer Slide Time: 17:59)

The slide shows a hand-drawn diagram of a right-angled triangle. The first row has one asterisk (\*). The second row has two asterisks (\*\*). The third row has three asterisks (\*\*\*) and so on, up to five rows. To the right, the variable "row" is defined as 1, 2, and 3. Below this, the text "I am printing 'row' stars" is written. A large bracket groups the first two rows as "2 times", the next two rows as "5 times", and all five rows as "row times".

So, for the first when a row is equal to 1 then I am printing 1 star, when row is equal to 2 I am printing 2 stars, when row is equal to 3 I am printing 3 stars. So, every time I can

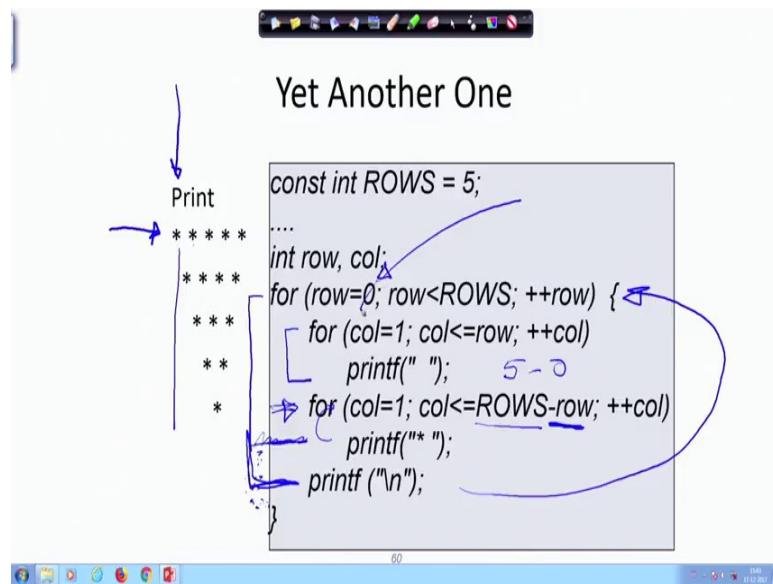
also say that I am printing row stars, I am printing row stars, row number of stars therefore, how many times I will do in a loop in the inner loop. You could see that when I we had drawn this thrice or twice whatever there was here, there is an inner loop 5 times and an outer loop that was doing 2 times right. Now, here what will change it to is inner loop row times outer loop maybe 3 times. So, this is variable.

Now, let us see how we can program it. Constant integer rows is 5 that I have not made variable there are 2 integers row and column. For row equal to 1, I start with rows row as an index and row less than rows less than 5 plus plus row. What do I do? Column equals one I will do up 2 column less than equal to row. So, first row column 1, column is less than equal to the only ones I will print then plus plus column, so column becomes 2, but what is my row? Row is 1 still I am pointing at this row, row is 1. So, I will do printf. Now, the value of row becomes 2 as I increment, but then the column this is a column this is row the column will be less than row because row has become 2. Now, it is pointing to this row, this row. So, column being less than that I will come out of the first row.

Second row, what will happen? Column will start from one and row is 2. So, column less than row I will print once, I will go back here column is becoming 2 and column is still equal less equal to row all right not less than, but equal to row therefore, I will again print here then I will come back here and now column is 3 column is 3, but row is 2 therefore, I will not print any longer will come out of this loop and go here. Then low will be incremented here all right, row has been incremented here. And, now again column is initialized to 1 for the third row first it is printed incremented column comes here all right then columns column comes here, so column is 3 still less than equal to row. So, I print the third one and then it is incremented, so the row becomes 4, but my column is sorry the column becomes 4, but row is 3 therefore, the column is I go out of the loop and again column is initialized to 1 and my row is incremented to this.

So, that is how it is done. You please look at it more carefully and you will have to understand it and this will give you a very clear idea how a nested loop is working. So, this is one example.

(Refer Slide Time: 22:52)



Here, what we are trying to do just think of this figure. First row is 5. So, row is this, column is here how long shall I print in the columns keeping the row fixed. What will be my logic? The logic will be well here I started row with 0 and less number of, less than number of rows, rows is 5 that is not that important. Here let us look at this, first I will do 5, then I will do 4, but there is another one. For the second row I am shifting one space I am shifting this should have been aligned and then I am shifting and giving a space and then doing it. So, gradually it is being shifted.

So, let us see what is being done. Let us look at the first for loop here this is the outer for loop, let us outer for loop is up to this, outer for loop is up to this and let us see what is happening. Row is 0 to number of rows less than. So, I have started with 0. So, I did not make it less than equal to it is less than less than 5 I will do this number of times. Internally what am I doing this doing here column is 1 and column is less than row because less than equal to row because row is 0, row is 0 and column is 1. Look at the trick here, the trick that has been applied is column has been in is starting with a value 1 and as long as column is less than row I am printing blank.

So, how many blanks should I print for the first column? 1 blank then column less than. Now, here from the first column look at this point. Column equals 1, 1 column less than equal to rows, rows is 5 as long as column is less than equal to rows minus row. What is my row? Initially my row is 0, so rows minus row is 5, it is 5. So, column is one column

is less than equal to 5 plus plus, so I do printf, I do a printf and I go on doing this as long as the column is, so column is now incremented to 2 3 4 how long will it come for the first row it will come 5 times because rows minus rows 5 minus 0. So, I will print this and then I will come back this loop is over, this loop is over, I will a print a backslash n up to this sorry I am sorry this for loop is actually extending up to this. So, from here I go back. So, I come to the second row.

(Refer Slide Time: 26:37)

```

const int ROWS = 5;
.....
int row, col; AS 2
for (row=0; row<ROWS; ++row) {
    for (col=1; col<=row; ++col)
        printf(" ");
    for (col=1; col<=ROWS-row; ++col)
        printf("* ");
    printf ("\n");
}

```

Now, row becomes 1 let us see, now row becomes 1. Less than 5 for column equal to 1, column less than equal to row 1 still valid I give one blank here, one blank here for the second row. My row pointer as come here, row is 1. Then I will do column 1, 2 column minus rows ones blank I have already given. So, I am. Now, my starting is here how many? Now, row is 1, 5 minus 1 so that means 4, 4 times this will loop and print star and then printf n.

Next time I go back here and this becomes 2. So, row 2 to less than 5 column is again now 2 blanks, column is 1 2 less than equal to row and row is 2, so 1 to 2. So, there will be 2 blanks here one blank here, one blank here. So, I am coming here and then I am printing this is 2, so 5 minus 2 3 stars. So, in this way I can go on and print this figure by an intelligent way of applying the for loops or the nested loops and putting in the spaces together spaces properly. I think this gives a very interesting example for you to look at.

(Refer Slide Time: 28:23)

The slide has a title 'break and continue with nested loops'. Below the title is a bulleted list:

- For nested loops, break and continue are matched with the nearest loops (for, while, do-while)
- Example:

A code snippet is shown:

```
while (i < n) {  
    for (k=1; k < m; ++k) {  
        if (k % i == 0) break;  
    }  
    i = i + 1;  
}
```

A red arrow points from the word 'Breaks' to the 'break' keyword in the inner loop, with the text 'Breaks here' written below it.

So, these are some of the examples that we have seen. We will come to this for thing again later, but let us just remind you a little bit about some things that we had mentioned in passing.

(Refer Slide Time: 28:31)

The slide has a title 'Shortcuts in Assignments'. A bullet point says:

- Additional assignment operators:

Below this, there is a table showing the equivalence of standard assignment operators and their shortcut counterparts:

<u><math>+ =</math></u>	<u><math>- =</math></u>	<u><math>* =</math></u>	<u><math>/ =</math></u>	<u><math>\% =</math></u>	$+ =$
$a += b$	$a -= b$	$a *= (b+10)$	$a /= b$	$a \% = (b+10)$	$a = a + b$
a += b is equivalent to a = a + b			$a = a - b$		
a *= (b+10) is equivalent to a = a * (b + 10)			and so on.		
$a = a * (b+10)$					

For example, this operator plus equal to as for example, here a plus equal to b this means a assigned a plus b these are some shortcuts all right, a minus equal to b that means, a is assigned a minus b. Here a star b plus 10, a as star assigned b plus 10; that means, a will be assigned a times b plus 10. So, in that way we have got this one also, say a assigned b

that means, a is assigned a divided by b these are some of the shortcuts. So, this is just to wrap up some of the assignment operations that we are talked about. But these are as I said that you can keep this for later use. Right now more fundamental thing that you need to know is a use of if else for, while, do while etcetera.

In the next lecture we will start with a new concept called arrays and there you will find that these loops are becoming so important and will have many interesting applications using arrays that will be done from next lecture onwards.

Thank you.

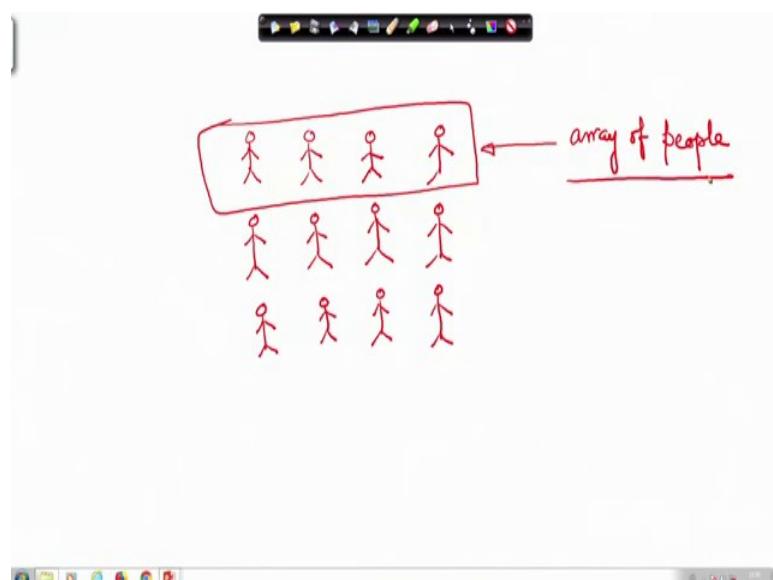
**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 26**  
**Introduction to Arrays**

Till now we have looked into, we have seen different programming constructs using which we can implement repetition; that means, a set of statements will be repeated in a loop. And in C such contracts where while do while and for loop, right? Now also we have seen how we can branch out. Coming through a sequential one after another execution of the instructions depending on the condition we can go out to one path or another path through if else structures. And we have also seen some examples by in which the combination of the, if else structure as well as the while structure are combined together to give us more powerful and useful programs.

Today we will start discussing about a new form of representation of data that is very useful and very fundamental that is called arrays ok. Arrays the word array simply means arrangement of data. So, we can think of for example, an array of soldiers or array of people standing in line.

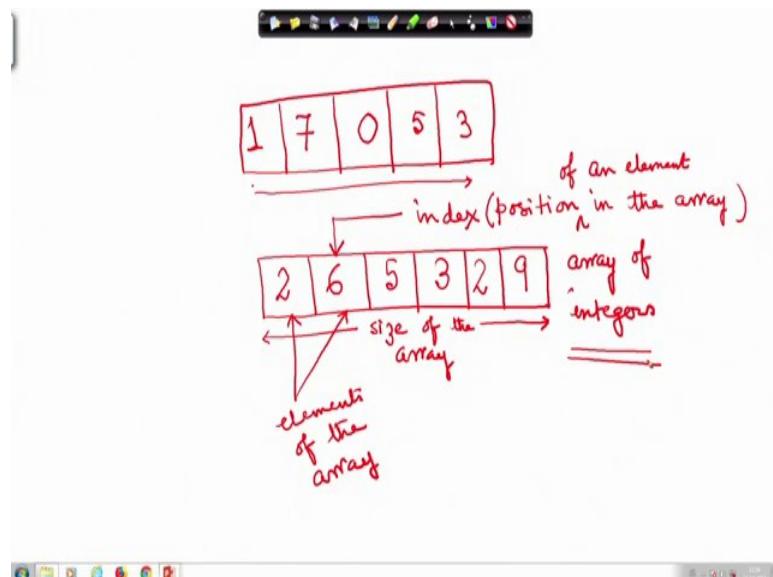
(Refer Slide Time: 01:42)



So, people are standing in a line, this is an array ok. We can also have 2 rows of people standing alright. This is a very regular structure we can have 3 rows also. So, these are arrays of people.

Now, if I just have only one row; in that case say for example, I consider one row then it is a one-dimensional array. When I consider rows as well as columns then it is a 2-dimensional array; however, we will now initially concentrate on one dimensional array like this ok. What is this? This is an array of people similarly, we can have array of numbers.

(Refer Slide Time: 02:54)



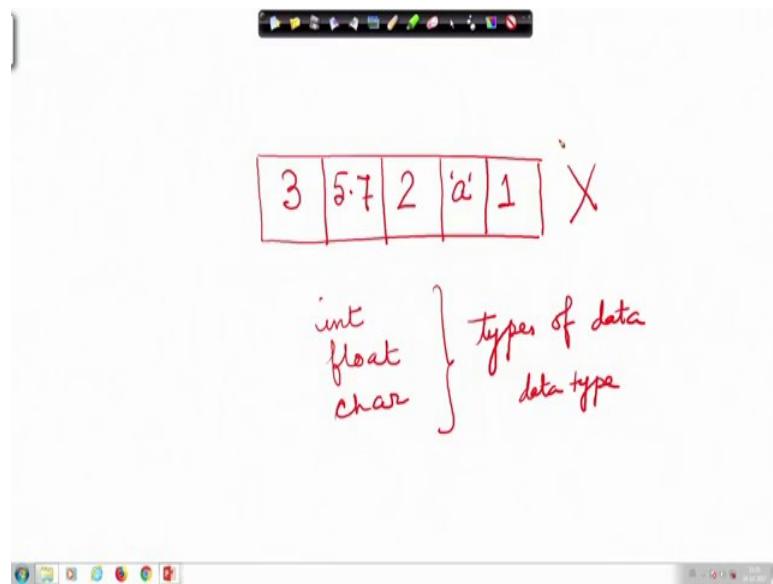
For example, 1 7 0 5 3 that is again an arranging of data arrangement of data in a linear fashion ok, this is the first element and this is the last element of this arrangement. So, this is also an array, array of some numbers. So, in an abstract way we can say that an array can be drawn in this form it can be an array of integer or an array of real numbers depending on what I want to store here, suppose this b and array of integers. Then each of this places can have one integers stored in this 2 6 5 3 may be again 2 and 9 so this is an array.

Now, how many elements are there in the array? Each of these are elements of the array how many elements are there in the array 1 2 3 4 5 6. So, this is the size of the array and these are the elements of the array this all these are elements of the array. Also another thing we need to know is how do I identify one element of the array, say this element 6

of the array this array how do I identify that what is it is identification, the identification is the position. So, this is a second position alright this is the first position this is the third position 4th position so and so forth.

Now, this positions in the array is known as index. Index is the position in the array; index determines position in the array position of what? Position of an element in the array so, for the time being we assume that arrays are linear structures linear arrangement of data. Now one point to remember is one array can store data of the same type. For example, it is not possible to have an array.

(Refer Slide Time: 06:16)

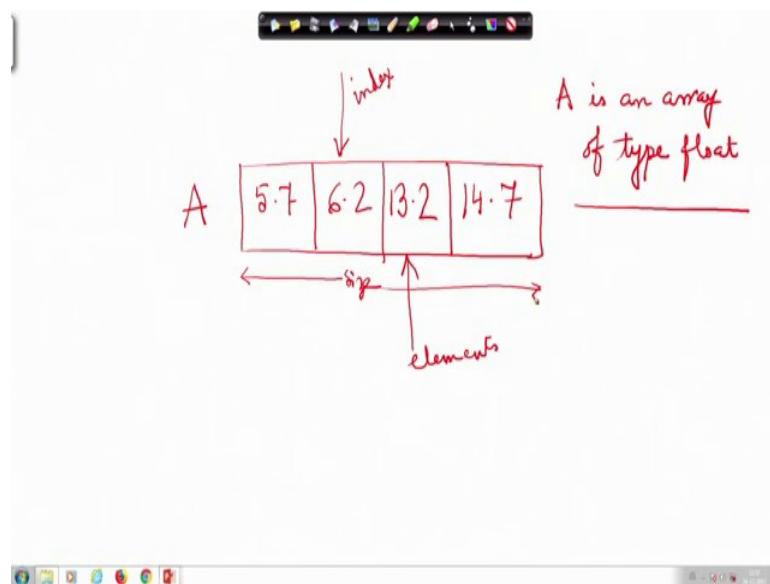


In which there will be some integers some real numbers and some again characters ok. So, that is not possible that is not allowed.

So, an array will be allowed to store data of only one type. We know that int float, char all these are defining different types of data or data type, right or data type. So, an array can store data of only one data type whatever that is it can be all floating point numbers, it can be all characters, it can be all integers whatever, alright?

So, these are 2 words of cautions. So, what have we learnt? We have learnt that array is an arrangement of data of the same type and the array has a name identified as a whole by a name and say for example, this array this is not a valid array so let us have a valid array.

(Refer Slide Time: 07:51)



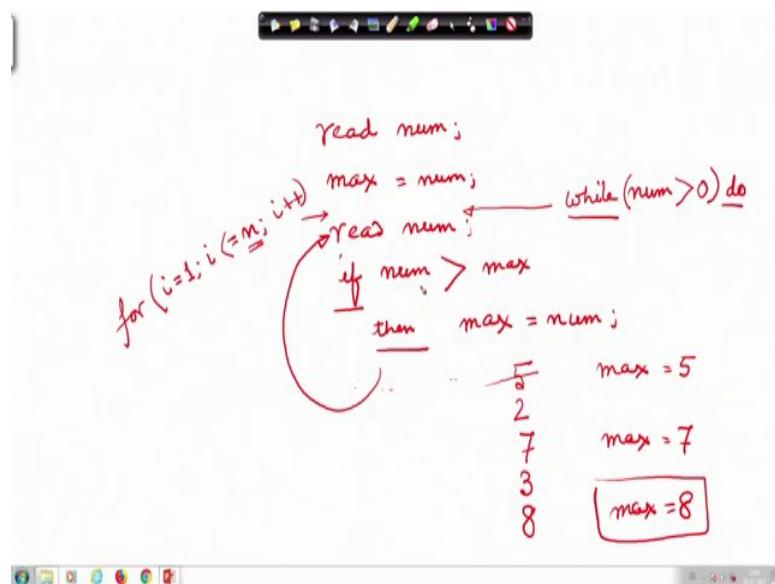
Let us have a valid array of 4 elements say of real numbers may be, right?

So, this array will have name let us say arbitrarily put a name A ; that means, A is the name to this array and this array has got a type, what is the type of this array? The type is float so array A is a float. So, A is a is an array of type float; that means, it can store only floating point data. Also, an array consists of different elements each of these are elements and each element each element can be a floating-point number. And the particular element is identified by the index of an array, alright?

Now, and also, we have we know that an array has got a size, given this let us think of why are we talking about all these things, why is this needed after all? We have encountered till now quite a few example problems for example; let us again come back to the old problem of finding the maximum of a set of integers.

Now, in that case what did you do? We read one integer we initialized a variable.

(Refer Slide Time: 10:01)



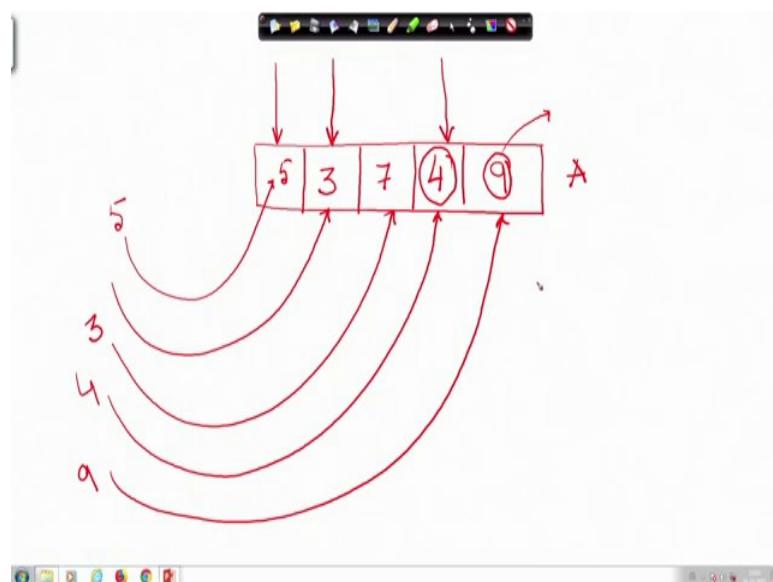
So, first integer we read let me once again do it. So, if you recall we I am writing the flow chart I am not writing the c program. Read a number you know how to read a number by scanf then put that number to be max read again read number. And if number is greater than max then I am writing pseudo code. So, I can write then max is getting the number, again I read number and in that way I go on or you can say that ok, why should I write in this way? I shall simply implement a loop while loop or for loop. So, that this thing is carried on till I stop, say for n numbers or for ah. So, here I can use for I assigned 1 to I less than equal to n I plus plus to implement the loop here, but that is possible when I know the total numbered of numbers.

If I didn't know the number of numbers there are some other ways of doing it say for example, I am reading integers and I say when I want to stop I will enter a 0. So, here I could have put in something like this while num is greater than 0 do so we do this as long as num is greater than 0 I am going on doing this. And if num is 0 or num that we decide or some negative number I know that my job is done. So, but in this case if I want to remember what were the numbers already given to me suppose the numbers that were given were 5 initially here then 2. So, nothing was done next was 5 then 7 then here there has been a interchange. So now, here max becomes 7 then again it was 3. So, nothing was done then it was 8 and max became 8.

So, that will work for this sort of program, but if I want to know what where the numbers he was saying max is 8, but what where the numbers that where input by the user we have not remembered them anywhere is not it, you have not remembered them anywhere. So, here in my program, I will not be able to say what was the second number entered, what was the first numbered entered, was there any 0 entered or was there any anything greater than 3 entered, we cannot answer these questions because we read this and we operated on this and we forgot them.

Now, if you can think of that if I had used an array.

(Refer Slide Time: 14:02)

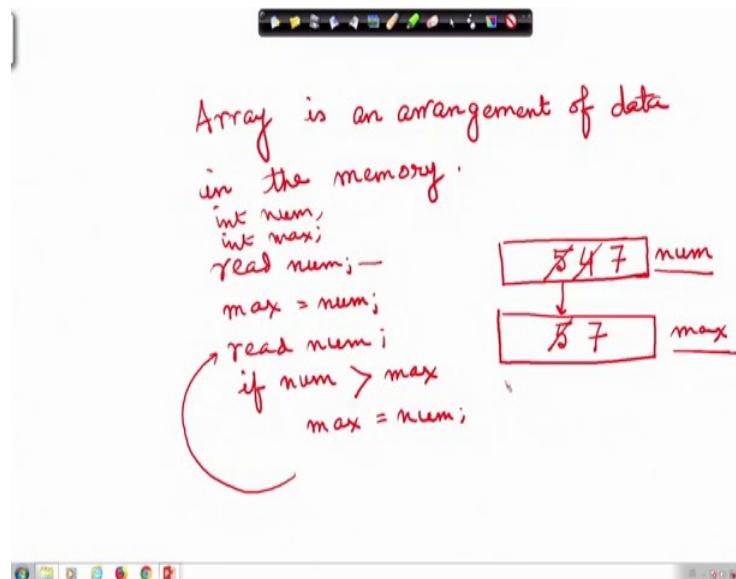


This sort of thing and I had as the user give the numbers I stored them here 5 then 3 user gave 3. I stored 3 here then user gave 7 I stored 7 here, then the user gave something may be again 4 I stored 4 here, and then the user gave 9 I stored 9 here. Then even after or 8 here I even after reporting the max that this is the max I have got these in my array. So, my name of the array is still A which A is an array of integers. Then this is stored here I have remembered them in the array.

So, I can now say what was my first element I go to the first element of the array using the index value ok, and second, I go to the this value ok. What was the 4th value? First second third 4th I come to this point and I can say the 4th value was this. So, all these things I can remember because I have been able to store them in the form of an array.

So, that is. So, array is again an arrangement in the memory alright.

(Refer Slide Time: 15:49)

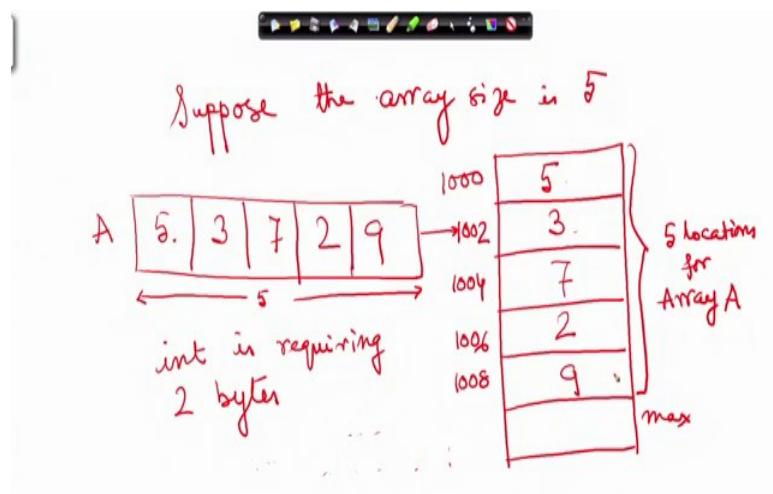


So, an array when I. You know that every variable is stored in a memory location, right? So, when we had num and the program that I had written that num initially I sorry I initially I read num. And then I do this and then again read num if num is greater than max, max assign num when I do this thing in a loop or whatever, what is happening? Whenever I have got variable num is a variable one location in the memory. This is num and max is another variable I have declared all of them here int num, int max I have declared them.

So, here there is another location which is known as max. So, when I did this read num and suppose some data 5 was read that 5 is stored here alright, 5 is stored here when I come here and I put max num assigned to max so it comes here. Now I read num again so suppose num is 4 and no update in max next time I read it 7. So, this is updated here and this becomes 7. So, this is the process that goes on alright.

Now, may so every time I am losing the old value of num and storing the current value of num and similarly whenever there is an update required I take the current value the max up to now ok. So, I am having 2 variables of integers, but if I had done it in the form of an array then I could have thought of like this that I have got suppose the array size is 5.

(Refer Slide Time: 18:58)



Suppose the array size is 5. So, that means, conceptually I am talking of something like this that there are 5 positions in this array. And as I am getting the data it is coming over here 5 3 7 2 9 whatever. Now actually internally what it is happening internally this is being stored in the memory the array size is 5 here. So, I have got 5 locations in my memory 4 5.

So, these 5 locations of the memory are being given to the array a suppose the name of the array is array A. So, for the array A so, the first element is coming here when I am actually writing it here it is coming here, second element is coming here, next element is coming here, next here next here so these are the memory locations. So, an array is actually a representation in the memory which are contiguous and you know each memory location has got an address.

So, suppose hypothetically the address of this is 1000 and each of them are integers and integer if I say integer is requiring 2 bytes. Then if this is thousand location the next one will be 1002, next one will be 1004, then 1006, then 1008, 1008 to 1009, right?

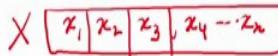
So, 1000 to 1001 is for the first element this one coming here 1002 to 1003 for this element, 1004 to 1005 is this element. 6 to 7 is this element 1008 to 1009 will be this element. So, 1000 to 1009; that means 10 locations we are requiring for these 5 elements of the array. Now since also in my program I have got 2 other variables one is the num or one is the max, alright? I can manage it with one more variable max is separate variable.

Suppose. So, that is here this variable is max let me delete this for you this is not required, alright? It is because this diagram has redundant because max is another variable which I have here. Will write a program will show later how we can write a program to deal with this.

So, everything. So, since I have computed I have read the numbers I have stored them in the memory locations and since I am going through an array I am not written overwritten one number by another therefore, I can remember all the numbers. So, this is one basic reason of usage of array where I want to remember the numbers in the memory given this let us quickly recapitulate what we have said till now.

(Refer Slide Time: 23:08)

The slide has a title 'Basic Concept' in bold black font. Below the title is a bulleted list of points. The first point has a red handwritten note 'int/float/char' next to it. The second point has a red handwritten note 'x<sub>1</sub>, x<sub>2</sub>, x<sub>3</sub>, ..., x<sub>n</sub>' next to it. The third point has a red handwritten note 'Take an example.' next to it. The fourth point has a red handwritten note 'Finding the minimum of a set of numbers.' next to it. There is also a red box around the list items.

- Many applications require multiple data items that have common characteristics. *int/float/char*
- In mathematics, we often express such groups of data items in indexed form:  
•  $x_1, x_2, x_3, \dots, x_n$  
- Why are arrays essential for some applications?
- Take an example.
- Finding the minimum of a set of numbers.

So, many applications require multiple data items, but have a common characteristics, what is that common characteristics? Common characteristics like whether they are integer or float or character whatever that is it can even more complex. So, we call say; so that numbers that we are showing say we can in mathematics we often represent them as  $x_1, x_2, x_3, x_4$  say up to  $x_n$ . And that hold thing I can represent as an array  $x$  where  $x_1$  is the first element,  $x_2$  is the second element,  $x_3$  is the second third element like that

Now, here is the example finding the minimum set of numbers minimum of a set of numbers. How we can find a minimum of a set of numbers. So,. So, this is what I have already discussed.

(Refer Slide Time: 24:20)

The slide shows two code snippets side-by-side:

**3 numbers**

```
if ((a <= b) && (a <= c))
    min = a;
else
    if (b <= c)
        min = b;
    else
        min = c;
```

**4 numbers**

```
if ((a <= b) && (a <= c) && (a <= d))
    min = a;
else
    if ((b <= c) && (b <= d))
        min = b;
    else
        if (c <= d)
            min = c;
        else
            min = d;
```

But just for the sake of revision let us look at it once again say for 3 numbers you know we can this program must be familiar to you now, you should be able to understand it quickly if a is less than b and a is less than equal to c then a is the minimum I was showing the maximum here it is an example of the minimum else if b is less than c then b is the minimum else c is the minimum.

So, first so fine as soon as I extend from 3 numbers to 4 numbers you see the program becomes a little bigger. If a is less than equal to b and a is less than equal to c and a is less than equal to d then minimum is a, otherwise if for you see if everywhere I have brought in more number of comparisons and the program code has also been bigger alright? And I am not storing the numbers as yet I am not storing the numbers that is one issue.

(Refer Slide Time: 25:30)

## The Problem

- Suppose we have 10 numbers to handle.
- Or 20.
- Or 100.
- How to tackle this problem?
- Solution:
  - Use arrays.

The problem is so from 3 to 4 we need needed so much extension.

Now, suppose we have got 10 numbers to handle that will be even bigger 20 numbers even bigger 100 numbers. So, how do you do that? The solution is the solution to the problem is what is a solution? The solution to the problem is use of arrays right using arrays, right?

(Refer Slide Time: 26:00)

## Using Arrays

- All the data items constituting the group share the same name.
- Individual elements are accessed by specifying the index.

Diagram illustrating an array:

NUM	5	7	6	3	2
-----	---	---	---	---	---

Arrows point from labels NUM<sub>1</sub>, NUM<sub>2</sub>, ..., NUM<sub>n</sub> to the first, second, ..., nth elements of the array respectively.

Now as I said that all the data items constituting the group share the same name all these data items that I had shown say integers like 5 7 6 3 2 all these can share the same name x or may be let me give a meaning full name NUM.

Now, this one is NUM 1, this one is NUM 2, this one is NUM 5 ok. So, I have got just as we had shown  $x_1$   $x_2$  up to  $x_n$  so here  $n$  is 5 so I have got this 5. Individual elements are accessed by specifying the index; this is the index which is telling me in which position which number in this array of numbers I am looking at ok. We will look at this further the use of arrays and anymore things are to come.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 27**  
**Arrays (Contd.)**

So, we were looking at arrays and we have seen that all the data items constituting the array shared the same name.

(Refer Slide Time: 00:22)

Using Arrays

- All the data items constituting the group share the same name.
- Individual elements are accessed by specifying the index.

int x[10];

A  $a_1 \ a_2 \ \dots \ a_n$

1 2 3 4 5

So, we can call an array to be A, an array A or a array num. If the array be A then the elements are a 1, a 2 like that up to a n, where n is the size of the array and each individual elements are being accessed by this indices, these are the different indices.

(Refer Slide Time: 01:11)

The slide has a title 'Using Arrays' and two bullet points:

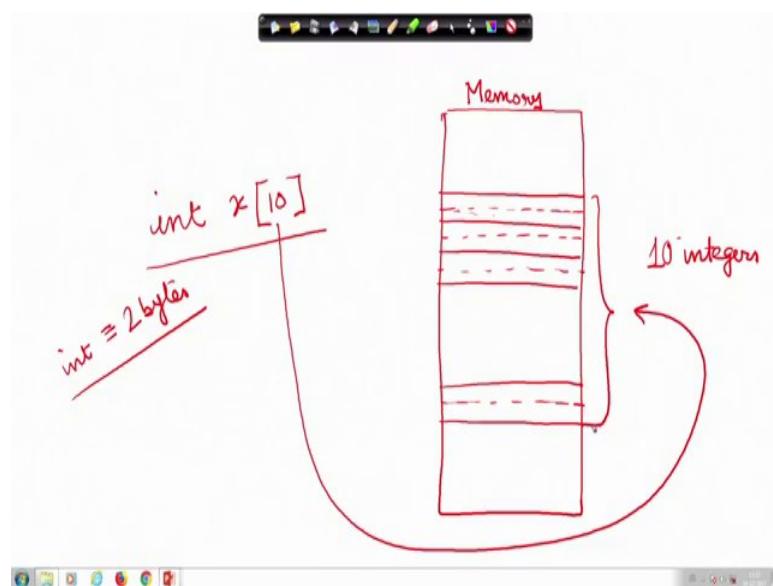
- All the data items constituting the group share the same name.
- Individual elements are accessed by specifying the index.

A hand-drawn diagram illustrates an array `x[10]`. It shows a horizontal row of ten empty boxes. Above the first box is the letter 'x'. To the left of the first box is an equals sign (=). A red bracket underneath the first five boxes is labeled '10' with a double-headed arrow, indicating the size of the array. A red curved arrow points from the word 'dimension' to the number '10'.

So, for example, here when we declare here comes a new thing. Suppose we have an array where the size is 10, so there will be 10 elements of the array, 10 elements of the array. Just as and just like variables array as a whole is also a variable. So, suppose I name this array to be `x` these are `x 1, x 2, x 3, x 4, x 5` like that up to `x 10`. Now, just like we had to declare the variables like `int float` here, here also we need to declare the array and the way we declare the array is shown here `int x 10` what does it mean; that means, `x` is an array of now I am introducing a term of dimension 10. Earlier I was talking calling it size and here I am calling it dimension. For the time being assume that both these both of these are the same thing, but actually there is a difference, we will explain that in the course of discussion.

So, we declared it first of all the type of the array as I had said that an element all the elements of an array has to be of the same type. So, either all of them will be integers or all of them will be floats therefore, this entire array has is of a type `int` which says that all the elements of this array are integers and `x 10`. So, let us quickly have a look at the memory scenario. Say in the memory, this is my memory and this is the memory and in the memory some locations are kept for the array `x`; that means, start from here and maybe up to this.

(Refer Slide Time: 03:31)



Now, how much space will be given to the array must be known to the compiler because the compiler just like the compiler allocates variables to different memory locations the compiler will also have to allocate locations memory locations for the array elements. Now, when I said int x 10; that means, the compiler will store space equivalent to 10 integers, 10 integers in the memory, 10 consecutive places for 10 consecutive one after another no gap elements, 10 consecutive integer places in the memory. Therefore, how many can you tell me, how many bytes are required for that? Integers and I have assumed that any every integer is taking two bytes, integer is taking two bytes. If I assume that then I will require 20 bytes here. So, there will be 20 locations each location consisting of two bytes in this way we will go on, right.

So, you must understand when I declare this, what is the meaning of this. Dimension of the array means that I am storing, I am reserving, I am reserving the compiler reserves the space for 10 integers and that has to be done before the array is used, before the array is used therefore, it must be declared before so that the compiler while it is compiling the program can allocate enough space.

(Refer Slide Time: 06:06)

Using Arrays

- All the data items constituting the group share the same name.  
`int x[10];`
- Individual elements are accessed by specifying the index.

X is a 10-element one dimensional array

$x_1 \equiv x[0]$

$x_2 \equiv x[1]$

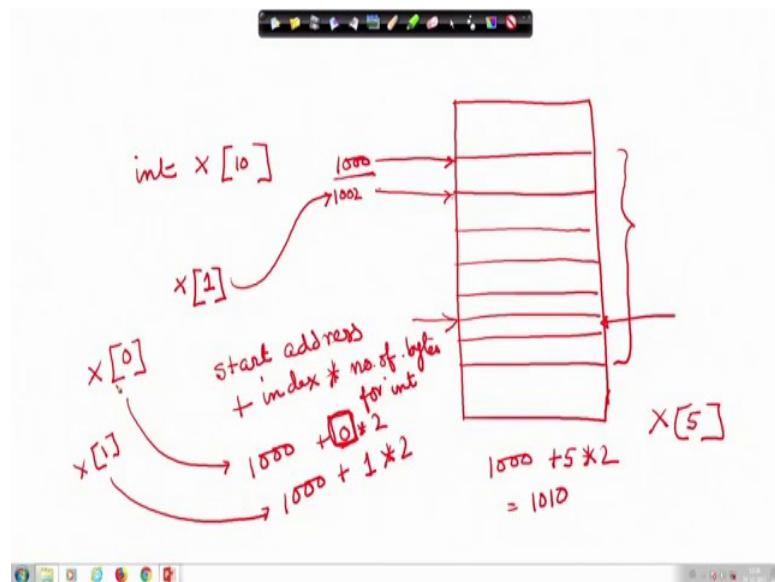
$x_5 \equiv x[4]$

index

So, and now individual elements are accessed by specifying the index. So, here x, this x is a 10 element one dimensional array linear. Now, comes a peculiar thing if you look at this, we are saying that each of these elements are stored in these locations and which are identifiable and accessible by the indices. Now, in C language each index, the index value starts from 0. So, the first element although while discussing we are saying x 1 say I was drawing an array like this where I was saying that these elements are the first element x 1, x 2, that is true, x 3, x 4, x 5, but in C these elements are counted the counting is started from 0, there is a reason for that which will also be evident soon.

Just remember for the time being that the first element therefore, x 1 is actually represented in C as x then the square bracket followed by the index and the first one will be x 0, second one will be x 1, therefore, can you tell me what will be the last element of this array? How do I represent that? Yes, you are right x 5 will be nothing, but x 4 yeah. Since I am starting with 0 I can go up to 4. Here in this example as you can see that I have got 10 elements in the array therefore, I start from 0 and go up to 9 that is what is followed in C language. There is a reason for that let me just briefly mention the reason the reason would be this.

(Refer Slide Time: 08:54)



As I said that in the memory the array is stored in a particular region. When I declared it as int x 10 then 20 bytes have been given to me and the array starts from this point which has got some address maybe 1000 and this address is starting is 1002.

Now, and I know that each one is of size 2. So, when I say what is the address of x 1. What is x 1? x 1 means here and I know the starting of this array. So, start address plus index times number of bytes for integer will give me the actual address. So, for the first address what will be the index x 0? So, my start address is thousand for the first one thousand plus 0 times 2. So, that will be 1000 for x 1 1000 plus 1 times 2. So, that will be 1002, for x 5, x 5 in this way is which element; 6th element.

So, that one will be here 1 2 3 4 5 6, 1 2 3 4 5 6 this element. And what would be its address? The address will be 1000 plus 5 times 2; that means, 1010 it starts from 1010. So, 1002, 1004, 1006, 1008, 1010 here it will start. So, this is basically the offset. How much I shift from the top, that is the reason the ease of computation of the address of any of the index any index, any element with any element with a particular index can be done using this method this formula and therefore, we start with 0 that is a specific reason for starting with 0, all right. So, this is a 10 element one dimensional array that we have seen. Let us move ahead.

(Refer Slide Time: 12:22)

The slide has a title 'Declaring Arrays' and two bullet points:

- Like variables, the arrays that are used in a program must be declared before they are used.
- General syntax:  
`type array-name [size];`  
– `type` specifies the type of element that will be contained in the array (int, float, char, etc.)

Handwritten notes in red ink:

- `float myarray[20];`
- `myarray` (with five empty boxes below it)
- `ma[0]` (under the first box)
- `ma[19]` (under the fifth box)

So, like variables the arrays that are used in a program must be declared before they are used the general syntax. Just like in integer or float we had declared them as in float before they are used. Why do I need to declare them before they are used? Because the compiler needs to allocate space for that otherwise the compiler does not know what type of variable it is therefore, we also have to do the same for the array and the general syntax will be type, array name, size. Now, again here the size I mean the dimension. Type specifies the type of the element that will be contained in the array it can be int, float, char or whatever.

So, I have got fields like say here I can have an array I can dictate as float the array name can be my array and size may be 20. Now, this one specifies that this array can hold only floating point numbers, and suppose floating point numbers are 4 bytes each then for how much memory will be reserved for this 80 bytes will be reserved for this because 20 is a size; that means, the maximum size that the array can take. And what is my array? Everything a just like every variable must have a name this array my this area also has got a name called my array. And so what will be the indices? I am repeating the thing my array is 0. So, I am writing ma for short 0 and what would be the last element here ma for size 20, it will be 19, is it all right.

So, this is the general syntax which a must put and also there should be a semicolon at the end, just like other declaration there is no other major difference here.

(Refer Slide Time: 15:04)

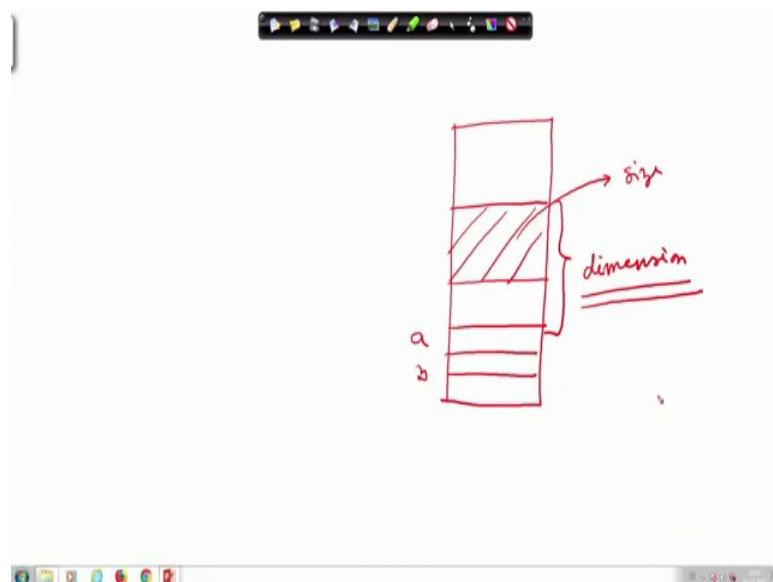
The slide has a title 'Declaring Arrays' at the top center. Below the title, there are two bullet points:

- Like variables, the arrays that are used in a program must be declared before they are used.
- General syntax:  
`type array-name [size];`
  - **type** specifies the type of element that will be contained in the array (int, float, char, etc.)
  - **size** is an integer constant which indicates the maximum number of elements that can be stored inside the array.

Below the text, there is a hand-drawn diagram of an array. It consists of a rectangle divided into six smaller rectangles. The first five rectangles contain checkmarks (✓). The sixth rectangle is empty and has three arrows pointing to it from above, indicating that it represents the maximum capacity of the array.

Size is an integer constant here I am calling it dimension that is it is a maximum number of elements that can be stored inside the array. Suppose I have got so much space kept for storing an array all right, so much space many many elements can be stored I have reserved so much space, but in my actual working I am using only some spaces and these are not touched that is allowed that means, but the reverse is not allowed. Unless I can, I have got the space unless I have got space I cannot store them. So, I must reserve the space beforehand but. So, that is why I want to I was mentioning this maximum space as dimension and the actual number of spaces which you are using to be size of the array all right.

(Refer Slide Time: 16:21)



The reverse is not true because suppose I have an; I have reserved. So, much space for an array and I go I go beyond that then these marks are spaces for other variables like a b c, their data will be destroyed by my data which I am taking as an array. Therefore, we must restrict to this, but suppose out of this I am using only this much in my parlance I am calling it size and this to be the dimension you can call it in this example in the slide we are calling this to be the size of the array there is a maximum size of them. So, what is dimension? Dimension is the maximum size that the array can be off, but in practice in actual running it can be less than that, but not more than that.

(Refer Slide Time: 17:22)

## Declaring Arrays

- Like variables, the arrays that are used in a program must be declared before they are used.
- General syntax:

```
type array-name [size];
```

  - **type** specifies the type of element that will be contained in the array (int, float, char, etc.)
  - **size** is an integer constant which indicates the maximum number of elements that can be stored inside the array.

~~int marks[5];~~

~~float marks [20];~~

So, I hope that part is clear to you. So, here what did I write? int marks 5 what does it mean? It means I am storing marks and the marks are all integers right 50, 55, 60, 100 I am storing integer marks; and how many marks can I store? 5. So, what will be the indices? 0 marks 0 to marks 4 that is the index limit because I am starting from 0 I can have only 5 positions. So, marks is an array containing maximum of 5 integers.

If a teacher decides that he will give fractional marks that somebody can get 62.5 somebody can get 70.2, somebody can get 59.7 then what change should we do the change that we should do is this will be replaced with float. And if I say in my class there can be maximum 20 students then what else should I change, I should change this and make it 20 that is how I should declare all right.

(Refer Slide Time: 18:58)

• Examples:  
int x[10];  
char line[80];

line

a	b	c	d	- - -	- - -	+
---	---	---	---	-------	-------	---

80

Here are few examples int x 10; that means, x is an integer. How do we read it? x is an integer x is an array of integers of size or dimension 10. char line 80; that means what? that means, line is a variable array of type character. So, what will it be? There will be 80 such positions usually when we take a printout usually when we take a printout of characters the lines were conventionally 80 characters in a line. So, there will be there will be positions like this all through maximum 80 all right. So, the size will be 80 and this variable is known as a line and what can it hold it can hold a characters like a, sorry a there can be a space, space is also a character I denote space as a blank like this then

maybe x is a character, then there is a blank a blank is also varied character then c, then d, all those things can be there.

So, it is an array of say ultimately say p is the last character something like this. Altogether 80 spaces for character I keep in my variable line and what is the type of the variable that type of the variable is an array, array of character all right.

(Refer Slide Time: 21:03)

The slide shows a list of examples and a diagram illustrating character arrays.

- Examples:
  - int x[10];
  - char line[80];
  - float points[150];
  - char name[35];
- If we are not sure of the exact size of the array, we can define an array of a large size.

Diagram illustrating character storage:

A box labeled "S. ravikumar" is shown above a row of boxes representing memory storage. The storage row contains the characters 'S', '.', 'r', 'a', 'v', 'i', 'k', 'u', 'm', 'a', and 'r' in sequence, with each character in its own box. A red bracket underneath the row is labeled "name".

Below the storage row, a large rectangular box represents memory space, divided into several smaller sections by vertical and horizontal lines, illustrating the allocation of memory for the array elements.

Similarly, I am saying that there are 150 points, there can be 150 points and each of those points are floating point numbers. Say name, I just stored a name, I want to store the name of a person say name S Ravikumar is the name of a student and I want to store it in a computer. How do I store it? I can store it in the form of an array where each of the elements is a character first one is S then dot then r then a v i k u m a and I need one more space r.

So, you see I needed one more space here. Why? Because I had taken I did not have enough space here. Now, each of them is a character by the way you know that a character is denoted like this so and so forth. Now, what this line says is that name is a variable which has got that the capability of holding 35 such characters, it has got the ability of storing 35 such characters at the most.

Now, how much space should we keep? How much should I keep here? If we are not sure of the exact size of the array we can define an area of large size. Say if I think if I

have got no idea of how the what are the typical Indian names for example, or American names for example, I can keep name 100, but is it advisable not always. When I really do not have any idea I have got no other way, but to do it, but if I have an idea of how much the name length can be at the most I should keep so much size because if I keep an arbitrarily large size then what am I wasting, I am wasting my memory space because each of these are a memory location.

Suppose beyond this never a name can extent to then I should not keep this part in the name. But when I have got no idea of course, I have to keep a bigger large size. So, that is the difference between the actual dimension of the array how much reserve we do and for a S Kumar, S Ravikumar for example, my actual size is 1 2 3 4 5 6 7 8 9 10 11 all right out of 35, 35 was the available number of spaces.

(Refer Slide Time: 24:43)

The slide shows a presentation interface with a toolbar at the top and bottom. The main content area contains the following text:

- Examples:  
int x[10];  
char line[80];  
float points[150];  
char name[35];
- If we are not sure of the exact size of the array, we can define an array of a large size.  
int marks[50];  
though in a particular run we may only be using, say, 10 elements.

Handwritten notes in red ink are present on the right side of the slide:

- int marks [10]
- int marks [20]

A red arrow points from the handwritten note "int marks [10]" down towards the underlined text "10 elements".

So, when we say it is int marks 50. Suppose in my class I have got say 10 students and I am going to store the marks of only one subject in that case there is no point storing so much space I mean it is like reserving so much space for the variable marks. If I know since I know only 10 students are there, if I know beforehand then I can I could have written int marks 10 or if I had known that sometimes the number of students in the class are 10 sometimes 15, but never more than 20 then I could have kept marks 20, but I should not keep marks 50. I hope the point is clear now.

(Refer Slide Time: 25:40)

## How an array is stored in memory?

- Starting from a given memory location, the successive array elements are allocated space in consecutive memory locations.

Let

x: starting address of the array in memory  
k: number of bytes allocated per array element

Element  $a[i]$  :: allocated memory location at address  $x + i * k$

Now, I have already described this how an array is stored in a memory. Starting from a given memory just let us have a revision of this, starting from a given memory location the successive array locations this is very important, the successive array elements are allocated spaces in consecutive memory locations one after another.

So, array a will have memory locations one after another like this let now the same computation let us do it again. Let x be the starting address of the array sorry let x be the starting address of the area address, this is the address and k is the number of bytes allocated per array element I had shown it for an integer now I am showing it in a generalized form, x was the first starting location say 1000 or whatever and k is the number of bytes allocated.

Now, then the next one will be x plus k because it started with x and needed k locations the next one will start from x plus k, the next one will start from x plus 2 k like that the ith element therefore, will be as we had computed earlier will be x plus i k and since we started with 0 that will be perfectly all right. Element  $a[i]$  will be located some  $a[i]$  will be located the address say  $a[i]$  here will be located starting at x plus i times k. Just a little bit of puzzle here I have written it here. Like this a common mistake could be that I what would have happened if I had written it like this. What is the problem of writing in this way? the problem is; what is  $i \cdot k$ ?  $i \cdot k$  is not  $i$  times  $k$  not  $i$  multiplied by  $k$ , but  $i \cdot k$  is another variable name all right.

So, these two are not the same thing. Whenever we go for programming we should be very careful. So, first array index is assumed to start at 0, in C we are doing that.

(Refer Slide Time: 28:24)

The screenshot shows a presentation slide with a dark blue header bar containing various icons. The main title 'Accessing Array Elements' is centered in a large, bold, white font. Below the title, there is a bulleted list in white text. The slide has a light gray background and a dark blue footer bar with small icons.

- A particular element of the array can be accessed by specifying two things:
  - Name of the array.
  - Index (relative position) of the element in the array.

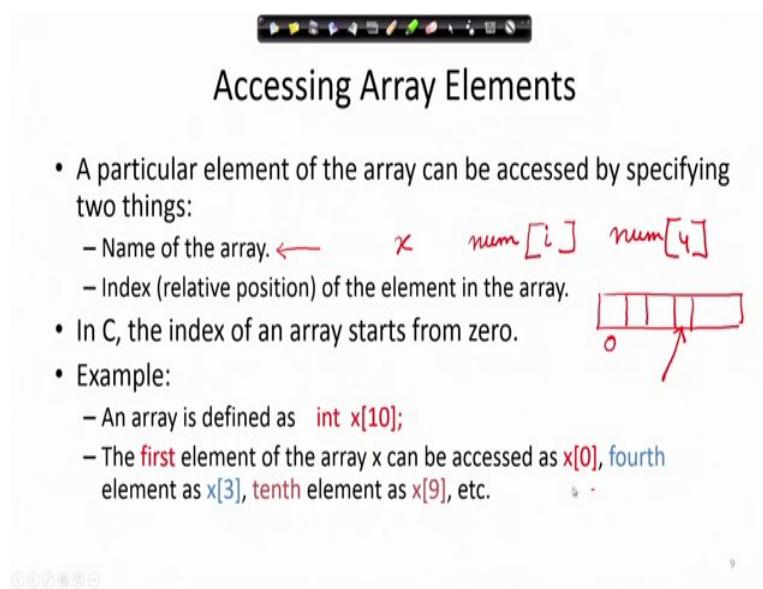
So, a particular element of an array can be accessed by specifying two things. What are the two things? Name of the array? Index, index is nothing but the relative position of the array. We will continue with this discussion further.

**Problem Solving through Programming in C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 28**  
**Arrays (Contd.)**

So, we now know that an array is identified by a name of the array. Some name like x, num, whatever and there is an index num i or say num 4, num 5 etcetera. So, by this index, by this index we identify which particular element of the array I am referring to.

(Refer Slide Time: 00:24)



**Accessing Array Elements**

- A particular element of the array can be accessed by specifying two things:
  - Name of the array.  $\leftarrow x$
  - Index (relative position) of the element in the array.  $num[i]$   $num[4]$
- In C, the index of an array starts from zero.
- Example:
  - An array is defined as `int x[10];`
  - The **first** element of the array x can be accessed as  $x[0]$ , fourth element as  $x[3]$ , tenth element as  $x[9]$ , etc.

Diagram: A small diagram shows a horizontal array of five boxes. The first box is labeled '0' below it, and an arrow points to it. The last box is labeled '4' below it, and an arrow points to it.

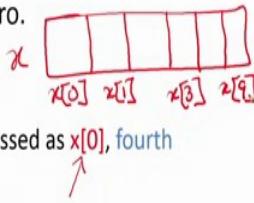
Now, in C also you have said that the array starts from 0, the index of, the value of the index starts from 0 ok.

Next. So, when we have an area defined as `int x[10]`, the index are from 0 to x 0 to x 9 all right. The first element can be accessed as  $x[0]$  first element can be accessed as  $x[0]$  and the fourth element as  $x[3]$ , the tenth element as  $x[9]$ .

(Refer Slide Time: 01:40)

## Accessing Array Elements

- A particular element of the array can be accessed by specifying two things:
  - Name of the array.
  - Index (relative position) of the element in the array.
- In C, the index of an array starts from zero.
- Example:
  - An array is defined as `int x[10];`
  - The first element of the array  $x$  can be accessed as  $x[0]$ , fourth element as  $x[3]$ , tenth element as  $x[9]$ , etc.



I hope this is clear to you by now that this one is arrays  $x$ . So, this is  $x[0]$ , this is  $x[1]$ , the fourth element will be 3 and the last element will be  $x[9]$ .

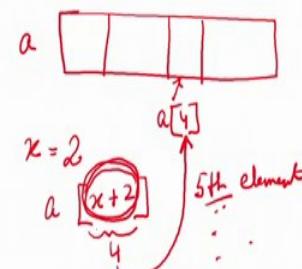
So, that is how the array elements are addressed, how they are referred to by the indices.

(Refer Slide Time: 02:14)

### Contd.

• The array index must evaluate to an integer between 0 and  $n-1$  where  $n$  is the number of elements in the array.

$a[x+2] = 25;$        $b[3*x-y] = a[10-x] + 5;$



Now, these indices must be integers. Now, the (Refer Time: 02:16) indices must be integers that is very important. So, if I have got an array defined as `int x[n]`, where  $n$  is the size of the array then the value of this  $n$  this index value of this index must lie between 0

to  $n$  minus 1. So, if it were twenty then the value must lie between 0 to 19 where  $n$  is the number of elements in the array.

Now, you can see here in this example that it is not the case that always I have to put a fixed integer as the index of course, I can write for an array, say an array is  $a$  I can always refer to some element of the array as say  $a[4]$  for the fifth element or I can have some other integer value  $x$  and suppose  $x$  is 2 now and if I refer I can refer to the same thing as  $a[x+2]$ ; that means, this will also evaluate to 4 and this will also point to the fifth element of the array. Now, one thing that we have to be careful about that this value that is computed lies between in this range between 0 to  $n$  minus 1.

So, here is another example where the array index value has been computed through an expression 3 times  $x$  minus  $y$  is assigned. So, suppose here you can see that we are talking of two variable, two array variables.

(Refer Slide Time: 04:25)

The diagram illustrates the state of two arrays,  $a$  and  $b$ , after the following assignments:

- $a[x+2] = 25;$
- $b[3*x-y] = a[10-x]+5;$

Given  $x=2$  and  $y=3$ , the assignments simplify to:

- $a[2+2] = 25; \rightarrow a[4] = 25$
- $b[3*2-3] = a[10-2]+5; \rightarrow b[3] = a[8]+5$

The arrays are shown as follows:

- Array  $a$  contains elements:  $\underline{a[4]}$  (labeled  $\underline{\underline{25}}$ ) and  $a[5]$  (labeled  $\underline{\underline{30}}$ ).
- Array  $b$  contains elements:  $b[2]$  (labeled  $\underline{\underline{35}}$ ) and  $b[3]$  (labeled  $\underline{\underline{30+5}}$ ).

So, let us draw a picture. A picture always clarifies the things much better. So, let us have an array  $a$  and array  $b$ ,  $a$  and  $b$  are two arrays right.

So, suppose  $x$  is 2 and  $y$  is 3 then this statement what does it do? A assigned  $x$  plus a  $x$  plus 2,  $x$  is 2 plus 2; that means, a 4 right. So, a 4 the fifth element is being assigned the value 25. And what is being done here?  $x$  is 2 a 10 minus  $x$ . So, that is a 8; that means, the ninth element, let us have the fifth let me make it a little bigger suppose this is the

suppose this is the ninth element a 9, a 8 sorry a 8 this is 8. So, that was suppose 30 that is being taken plus 5 is being added to that. So, 30 plus 5 that is 35.

Now, note this, this part is being evaluated. What is being done in evaluating this part? I have got  $x$  to be 2. So, I am computing  $10 - x$  I get that to be 8; that means, I want a particular variable which is stored in stored as a 8; that means, in the nineth location of the array  $a$  and that is 30. I take that value 30 and add 5 to that, so I get 35. And then what do I do? I store it in the array  $b$  and wearing in the array  $b$ .

Now, I will compute this part 3 times  $x$ ,  $x$  was 2. So, 3 times 2 is 6 minus  $y$ ,  $y$  was 3. So, it will be 3. So, in  $b[3]$ ,  $b[0]$ ,  $b[1]$ ,  $b[2]$ ,  $b[3]$ ; that means, again in the 4th element of this array  $b[3]$  I will be storing the value 35. So, now, the array  $b$  will have the value 35 here. I hope this is clear. So, you have to be very clear about the distinction between the array index and the value of the array.

So, this 30 that I got after computing the index  $a[8]$  that was telling me where in the array I should look at to get that value. And then that was just a value and with that value I had another value 5 I added them together, I added them together and got the value 35. Now, I see where I am going to store that for that again I compute the index here and I compute the index the index must be within the range 0 to  $n - 1$ , I compute that and I find that the index is 3; that means, in  $b[3]$  I have to store the value and go to  $b[3]$  and store the value. I hope this is clear right.

(Refer Slide Time: 08:14)

A Warning

- In C, while accessing array elements, array bounds are not checked.
- Example:  
`int marks[5];  
:  
:  
marks[8] = 75;`

The above assignment would not necessarily cause an error.

Now, in C, there is a word of caution here that in the language C the array bounds are not checked for example, int I have got declared int marks 5; that means, I have got an array marks whose size is 5, a dimension is 5; that means, at least I can have 5 elements in that all right. Now, here what I am trying to do, here what I am trying to do is I am trying to write 75 in marks 8, but there is no space for marks 8, it was 5 6 7 8 this is the place where I am going to trying to write 75.

Now, it would be good if the compiler could give me any error that hey you had declared it that size to be 5 and you are going beyond the border all right, you are crossing the border. So, be careful warning syntax error or whatever syntax error and you are not allowed to do that, but unfortunately array bounds are often not checked. So, what can happen if you are not careful, then it will not necessarily cause an error, but will be writing something here all right, it will be writing something at this point which might be when I allocated the compiler allocated the memory this part was for the marks and this part were for other variables x y p and whatever value of p was there that is overwritten with 75. So, one must be very careful about this when writing the programs and running it.

Suppose it is a very common thing that often you will find that there are some funny errors occurring and the reason for that maybe you have crossed the array boundary.

(Refer Slide Time: 10:43)

The slide has a title 'Initialization of Arrays' and a subtitle 'int x=25;' with three underlines. It contains three bullet points:

- General form:  
type array\_name[size] = { list of values };
- Examples:  
int marks[5] = {72, 83, 65, 80, 76};  
The code is annotated with red arrows: one arrow points to the first brace {}, another points to the first element 72, and a third points to the label 'marks' above the array name. The array itself is shown in a red-bordered box with elements 72, 83, 65, 80, and 76.
- Some special cases:

At the bottom left is a video frame showing a person speaking. At the bottom right is a small navigation bar with icons for back, forward, and search.

So, it can result in unpredictable results. So, general form of initialization of arrays whereas, the array name size with some list of values how do we. So, recall that we had done we can do initialization slight int x equal to 25 semi code, where I declare x to be an integer and also initialize it to a particular value.

The same thing can be done for arrays for example, I can write int marks 5 72, 83, 65, 80 16. Note that this initialization is within this two curly braces. That means, here I am creating an array whose name is marks and the values are 72 for mark 0, 83 for marks 1, 65 for marks 2, 80 for marks 3 and 76 for marks 4, 0 1 2 3 4 5 elements. I can declare in this way this is one form of initialization. Remember that here we are putting square brackets for declaring the dimension, but here we are putting curly braces.

(Refer Slide Time: 12:36)

- General form:

```
type array_name[size] = { list of values };
```

- Examples:

```
int marks[5] = {72, 83, 65, 80, 76};
char name[4] = {'A', 'm', 'i', 't'};
```

- Some special cases:
  - If the number of values in the list is less than the number of elements, the remaining elements are automatically set to zero.

float total[5] = {24.2, -12.5, 35.1}; total [24.2 | -12.5 | 35.1 | 0.0 | 0.0 | ]  
 → total[0]=24.2, total[1]=-12.5, total[2]=35.1,  
 total[3]=0,  
 total[4]=0

Similarly, this one char name 4 I have just typed in an array, so the array has got 4 elements and what is stored here A m i t, Amit. So, basically the name is Amit. So, that is coming as a string of 4 characters. Although I use the word string be careful about that because string has got a little more thing to it which I will describe later, but right now it is an array of characters, the characters array of characters of size 4.

Now, there are some special cases the number of the, if the number of values in the list is less than the number of elements the remaining elements are automatically set to 0 are automatically set to 0. For example, in this case I have got a variable what is the name of the array here the name of that is total and what type of array is it, it is an array of

floating point numbers and so there are 5 elements, 1, 2, I have got 5 spaces, 2 3 4 and 5. There are 5 spaces yeah and I have loaded initialize it to this value.

So, here it is 24.2, the next one is minus 12.5, the next one is 35.1. The remaining elements which are not filled up are filled up with 0s automatically by the compiler if it is less. Now, be careful that sometimes some compilers behave in a are implemented without doing this bit and therefore, you must check whether your compiler usually the standard compilers do that, but you should be careful about it.

(Refer Slide Time: 15:05)

The slide shows a presentation interface with a toolbar at the top and a title 'Contd.' in the center. Below the title is a code snippet:

```
int flag[] = {1, 1, 1, 0};
```

Handwritten annotations explain the code: a red bracket under 'flag' indicates it's an array, and a red bracket under '[]' indicates the size. To the right, a red box labeled 'flag' contains the values 1, 1, 1, 0. Below the code, two underlined 'int flag' labels are shown, with a red bracket under the second one indicating it's a variable.

So, total will be this as I have said here. The size may be omitted in such cases the compiler automatically allocates enough space for the initialize variables. So, when I am initializing it then it is possible that I can write I can leave out this size because I am already writing it here in this form therefore, in this case what will happen flag and array will be created just for the initialized values. So, a 4 element space will be given to you which will be 1 1 and 0 because I have not mentioned anything here, but I must give this symbol because just to distinguish it between int flag and int flag this because this is just a variable integer variable this is an integer array. So, you must be careful about this too.

(Refer Slide Time: 16:24)

## Contd.

- The size may be omitted. In such cases the compiler automatically allocates enough space for all initialized elements.

```
int flag[] = {1, 1, 1, 0};  
char name[] = {'A', 'm', 'i', 't'};
```

So, similarly I could have done char name, I did not put anything here and just write A m i t.

(Refer Slide Time: 16:33)

Example 1: Find the minimum of a set of 10 numbers

```
#include <stdio.h>  
main()  
{  
    int a[10], i, min; ✓  
    printf("Give 10 values\n");  
    for (i=0; i<10; i++)  
        scanf ("%d", &a[i]);  
  
    min = 99999;  
    for (i=0; i<10; i++)  
    {  
        if (a[i] < min)  
            min = a[i];  
    }  
    printf ("\n Minimum is %d", min);  
}
```

Now, let us come to this example finding the minimum of 10 numbers. Now, we are doing our first programming with arrays. We are doing the first programming using the arrays.

Now, here what we are doing is first we are reading the numbers I am reading the numbers all right. I am reading the numbers so and then I am storing them in an array.

So, let us look at this line by line. Let us look at the declarations; int a 10 what does it mean that I have got an array of size 10 and the name of the array is a all right, there is space for 10 elements a 0 to a 9.

Next, I have got another variable I one variable another variable mean. So, that is my declaration part now I am printing here give 10 values. So, on the screen I will find that give 10 values and then backslash n.

So, now the user is entering 10 values and what am I doing? I am reading those 10 values. Now, compare when you till now when we read the different integers what did you do we wrote it like this right, scanf, percentage d meaning that is an integer ampersand, num something like that we did to read an integer called num. Compare that with what we have done here. Here what we have done is scanf percentage d remains the same because whatever I am reading is nothing, but an integer and ampersand a i; that means, this array is ith element I am reading, this means a I means I am reading the ith element of this array a. What is the value of i? i is 0 initially.

So, first time whenever I am asked 10 values. So, i is 0, first i is pointing here. So, i is 0. So, this; that means, this is pointing here. I read a particular value 25, I store it here all right. Then I go in the loop what happens? i plus plus, i is incremented to 1; please observe this carefully this variable is incremented to 1; that means, now the index shifts from here to here a 1 and I read and that is less than 10 typical for loop. So, I come here read the second value say 3. Go back increment I it becomes 2 less than 10 I come here let this brings 2, means again this is changed to the next element all right and I read the next value which may be 37. In that way I go on and read 10 values. Every time I am reading one particular integer and that is being, where am I storing it that that is the importance of this ampersand; that means, that I am reading this at the address of a i, so at this location.

So, this is the first part, first part of the program that has read the array. Now I want to find out the minimum of the array. Now, let us see.

(Refer Slide Time: 21:46)

```
#include <stdio.h>
main()
{
    int a[10], i, min;
    printf("Give 10 values \n");
    for (i=0; i<10; i++)
        scanf ("%d", &a[i]);
    min = 99999;
    for (i=0; i<10; i++)
    {
        if (a[i] < min)
            min = a[i];
    }
    printf ("\n Minimum is %d", min);
}
```

Let me now. So, is this part clear how we read the array now? So, in this way suppose I have read the array some array like say 5, 3, 200, 75, 1. Now, in this part of the program I am trying to find the minimum and that is typically what we did last time, done here in a little bit different way I am starting with a min mean which is very large. So, I am assuming that there will be no data given at this point, at this point no data will be given which is as large as 9 9 9 9 9, I am assuming that.

So, again in a for loop here  $i$  is starting from 0, that is  $i$  is pointing here. I am comparing  $a[i]$ ; that means, a 0 a 0 we with  $min$  a 0 is less than  $min$  therefore, I can assign a 0 to  $min$ . So, my  $min$  becomes 5. I go back to the loop  $i$  is incremented. So,  $i$  is incremented and  $i$  comes here; less than 10, so I come here and again compare  $a[i]$ . What is  $a[i]$ ?  $a[i]$  is this value 3, is 3 less than  $min$  yes. So, the  $min$  becomes 3. I again go back increment the index.

So, you see in this for loop I am incrementing the index. We will see how we can write it in different ways. Now, this is a very important exercise and you will have to program it yourself meet with errors at several times, but then ultimately we will find it that is not that difficult. So,  $i$  is now 3, I again compare 200 with  $min$  now 200 is not less than the  $min$ . I go back to the loop  $i$  is incremented, 75 is not less than the  $min$  I come here, 1 is less than the mean. So, this will be like this in this way it goes on.

So, here we could find two distinct applications of this for loop, one for reading the array and storing it in a set of locations and the other one is looking through that array for every element I am looking through this area and I am finding the minimum. Now, let us do a little bit of exercise before we conclude this lecture today. Say I want to read an array of integers and I want to print the array. So, what should I do? I am leaving out the standard include stdio dot h declaration, declarations let me do; int let me call it, an array m underscore a meaning my array and let the size be 10 all right.

(Refer Slide Time: 25:24)

```

main()
{
    int m-a[10], i;
    /* Reading the array */
    for (i=0; i<10; i++)
        scanf ("%d", &m-a[i]);
    /* Printing the array */
    for (i=0; i<10; i++)
        printf ("The value of m-a[%d] is %d\n", i, m-a[i]);
}

```

Also i, and then index i index need not be i, but it must be an integer variable and then I am starting to read the array, I am trying to read the array. So, my program started here. For reading the array, I am doing for i assign 0, i less than less than 10, i plus plus scanf percentage d. And what am I reading? m a i. So, in a loop I will be reading m a 0, m a 1, m a 2 like that and then I want to print the array.

So, suppose I have read an array which is something like 3 2 4 5 1. Now, I want to print the array. So, what shall I do? I will just simply again I will take one elements one by one and will be printing them. So, I can write for i assign 0, i less than 10 i plus plus; that means, I will be doing this. So, i is starting with 0 and will go up to this here I am assuming that there are 10 elements here although I have shown only 5, then printf I can say that, so I am printing the value of m a this percentage d. So, these things will be as it

is printed except for this is percentage d backslash n and sorry backslash n and then it is continuing m underscore a i. Let us see what will happen and then I conclude this.

So, I have stored this now I am printing this. What will be printed? Some printing will be something like this let me use another color for this. What will be printed is the value first iteration. First iteration what will be printed. The value of m a; what is the value of i? I am sorry here, there should be another one I have written wrong they are two percentage d is, so here should be i comma this all right. So, I will repeat again. The value of m a. Then this, then this part will come and this will correspond to the first variable that is i. What is the value of i? Value of i is 0 is, then the second with, second place holder that is m a 0s value; that means, it is 3 backslash n. Again it will go in the loop and what will it again print, the value of m underscore a will be printed as it is percentage d the value of the index of m a 1 is then the value m a 1 is 2, is 2 like that it will be printed. So, these are two fundamental operations of reading an array and printing an array. We will continue further with the arrays.

**Problem Solving through Programming in C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 29**  
**Program Using Arrays**

So, we saw a program where we are finding the minimum of a set of 10 numbers.

(Refer Slide Time: 00:22)

The slide title is "Example 1: Find the minimum of a set of 10 numbers". Below the title is a C program:

```
#include <stdio.h>
main()
{
    int a[10], i, min;
    printf("Give 10 values \n");
    for (i=0; i<10; i++)
        scanf ("%d", &a[i]);
    min = 99999;
    for (i=0; i<10; i++)
    {
        if (a[i] < min)
            min = a[i];
    }
    printf ("\n Minimum is %d", min);
}
```

The code is annotated with three boxes:

- A red box labeled "Array declaration" covers the line `int a[10], i, min;`.
- A red box labeled "Reading Array Element" covers the line `scanf ("%d", &a[i]);`.
- A red box labeled "Accessing Array Element" covers the line `min = a[i];`.

To the right of the code is a diagram of an array a[10] with 10 slots. Arrows point from the annotations to specific parts of the array diagram.

So, here the program is restricted as you can see to 10 numbers and accordingly I have declared the array a to be of size 10 and also this iteration values are also iteration limit has been kept to 10. Now, when we did that just a quick recapitulation of what we did in the last lecture that here is the array declaration and this is where we are accessing the array elements. Here you can see I am accessing the array element a[i], a particular array element here and assigning that to another variable mean.

Now, here it is very important how am I reading the element. Now, since an array consists of a number of elements sorry, the since the array consists of a number of elements I have to read them in a loop that is a very fundamental thing that you one should understand that at a time I cannot read the entire array in one shot, except for the case where I initialize it and when I declared the array I give `int a,` so and so values within curly bracket that is one way. But if you have to do it at runtime; that means, whenever the you asking the program is running and you asking from the user that you

entered the values then you have to take one bit of value at a time just like this and store it in the array.

So, that has to be done in a loop for by varying this indexes one after another as the index moves I load the different locations with the values. Now, given this we can have an alternate version of the same program.

(Refer Slide Time: 02:26)

Alternate Version 1

```
#include <stdio.h>
#define size 10
main()
{
    int a[size], i, min;
    printf("Give 10 values \n");
    for (i=0; i<size; i++)
        scanf ("%d", &a[i]);

    min = 99999;
    for (i=0; i<size; i++)
    {
        if (a[i] < min)
            min = a[i];
    }
    printf ("\n Minimum is %d", min);
}
```

#define PI 3.14  
#define size 10  
int size[10]

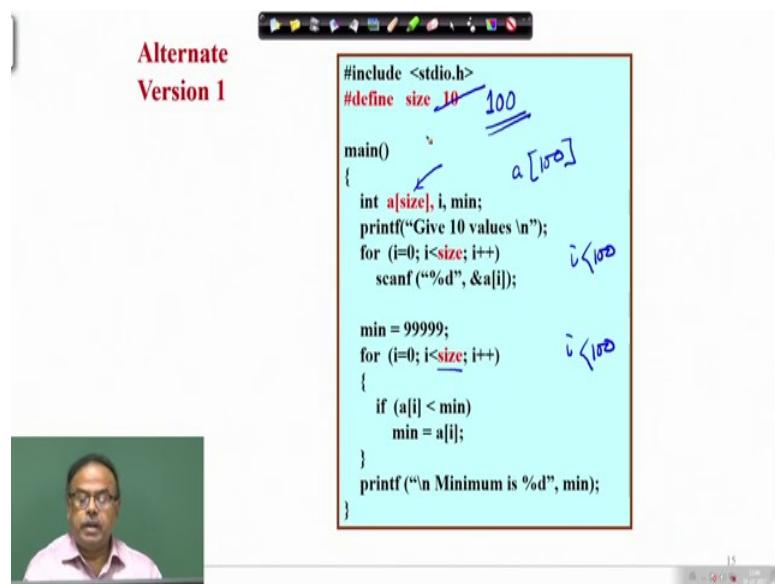
You can see that here we have added one line that is this line, define size 10. This is again the use of hash define that we had learnt when we defined pi to be 3 point we had encountered this defined pi 3.14 etcetera etcetera. So, similarly here I am defining size a variable size to have the value 10, therefore, I now, can write this dimension as size. Now, please note that the compiler look at it from the angle of the compiler, the compiler will try to allocate some memory location some amount of memory location ok. Now, if there be a variable size it really does not know how much memory location to allocate, but since I have defined it earlier it now, knows size means 10. So, it will replace int size with int size sorry, int a size to be int a size 10.

Now, next thing you do is wherever I had i less than 10 I make it size i less than size here also for the for loop I make i is less than size. Why did I do that? What is the advantage of doing it? Now, suppose you have got this program and how many places is size being used it is being used in 1 2 3 places in another program it could have been used in more

number of places. Here we have not printed the array, I have only printed the minimum. So, there could be more places where sites could be used.

Now, suppose I want to modify this program, or reuse this program for an array of size 100 only thing I need to do is I change this 10 to 100. And please note that after defined any hash define does not have a semicolon because these are not part of the program, these are commands to the compiler or pre processing commands.

(Refer Slide Time: 04:55)



Alternate Version 1

```
#include <stdio.h>
#define size 10
100
main()
{
    int a[size], i, min;
    printf("Give 10 values \n");
    for (i=0; i<size; i++)
        scanf ("%d", &a[i]);
    min = 99999;
    for (i=0; i<size; i++)
    {
        if (a[i] < min)
            min = a[i];
    }
    printf ("\n Minimum is %d", min);
}
```

So, if I could make it 100 then this would be replaced by 100 it would be a 100 and here it should be i less than 100 automatically, i less than 100. I need not change it at all the places that is the advantage of the hash define.

(Refer Slide Time: 05:42)

**Alternate Version 1**

Change only one line to change the problem size

```
#include <stdio.h>
#define size 10

main()
{
    int a[size], i, min;
    printf("Give 10 values \n");
    for (i=0; i<size; i++)
        scanf ("%d", &a[i]);

    min = 99999;
    for (i=0; i<size; i++)
    {
        if (a[i] < min)
            min = a[i];
    }
    printf ("\n Minimum is %d", min);
}
```

15

So, we can change only one line to change the problem size. The problem size from 10 to 100 to 1000 we can change by just changing one line.

(Refer Slide Time: 05:56)

**Alternate Version 2**

```
#include <stdio.h>

main()
{
    int a[100], i, min, n;
    printf("Give number of elements (n) \n");
    scanf ("%d", &n); /* Number of elements */

    printf("Input all n integers \n");
    for (i=0; i<n; i++)
        scanf ("%d", &a[i]);

    min = 99999;
    for (i=0; i<n; i++)
    {
        if (a[i] < min)
            min = a[i];
    }
    printf ("\n Minimum is %d", min);
}
```

Here is another version of the program. Here we have not defined hash defined, but I have kept a large size 100. Now, so, I have got an array space of 100 elements all right 100 elements. So, much space is kept for me by this line. Now, what I am doing is I am asking from the user you please tell me how many elements you are going to input. Suppose here and that I am taking as a variable n, n is a variable where I am just asking

the user give the number of elements n and when the user is typing say 25 then that goes as n. So, out of now, this 25 is of course, less than the size of this which is 100 it is less than that.

So, now, I will just work not for 100 elements, but for 25 elements why because the user has already told me that I want to input 25 elements. Therefore, what modification should I do? Again here I want to make it interactive. So, I again tell the user on the screen you will see this message input all in integers and the user is entering them one after another. He can type in 25 enter 35 enter 42 enter like that or he can give space and go on typing that.

So, for i less than 0 to n, now what is n? n is 25. So, now, it is for i to i, 0 to 25 I read the number. So, I will be reading 25 numbers here 5, 20, 16, 12, 9 etcetera etcetera I go on and I will go up to 25 not 100 all these. So, this space will be wasted, there is a wasted space. Why wasted space? Because I had deserved 100 locations, but the user has just agreed to give only 25 numbers and then the remaining part remains the same here also I will be dealing with n. Why n? I will be searching. So, when I did this the array that I got is actually although they were 100 elements the array was actually of 25 elements right, the array was of 25 elements. So, I will have to find the minimum within this zone and I need not go here.

(Refer Slide Time: 08:57)

Alternate Version 2

```
#include <stdio.h>

main()
{
    int a[100], i, min, n;

    printf("Give number of elements (n) \n");
    scanf("%d", &n); /* Number of elements */

    printf("Input all n integers \n");
    for (i=0; i<n; i++)
        scanf("%d", &a[i]);

    min = 99999;
    for (i=0; i<n; i++)
    {
        if (a[i] < min)
            min = a[i];
    }
    printf ("\n Minimum is %d", min);
}
```

So, that is another version of the same program. So, here what is done is we define an array of larger size and only use the required amount out of that all right. Now, let us do a programming ourselves. Suppose I want to read the some integers and I want to find out the sum of all the integers that old example that we had done earlier. How did we do it earlier let us recall.

(Refer Slide Time: 10:03)



```
for (i=0; i<10; i++) {  
    sum = 0;  
    printf("Enter the number\n");  
    scanf("%d", &num);  
    sum = sum + num;  
}
```

We did let me write it in this way sum there was something like sum assign 0, then printf, enter the number, then scanf percentage d and num sum equals sum plus num and then this has to be done in a loop ok.

So, watch how can I make a loop out of this. So, I also read another variable the number of numbers. So, sum n was read here. So, I had done some scanf. I would say let us say of 10 numbers. So, I need not scanf. So, this thing I have to loop right this part I have to loop how long. So, what should I put here? At the beginning for doing the loop for i, since I know beforehand that there are 10 numbers, i assign 0, i less than 10, i plus plus then I put a parenthesis here and a parenthesis here. So, and the first statement comes here. Confusing? Should I write it again or is it clear.

(Refer Slide Time: 12:07)



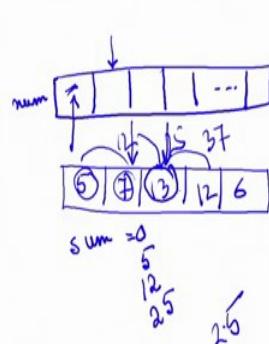
```
sum = 0;
for (i = 0; i < 10; i++)
{
    printf ("Please enter the number\n");
    scanf ("%d", &num);
    sum = sum + num;
}
```



So, I can write for i assign 0, i less than 10, i plus plus, printf please enter the number backslash n, then here I had done sum assign 0 scanf percentage d and num sum assigned sum plus num that is how the loop will go on. But here I am not remembering the number.

So, what I can do using an array what I can do is, let me now do it using an array int, num say 10, comma i is another integer variable. Now, and also I need sum sorry sum is also another integer, sum.

(Refer Slide Time: 13:26)



```
float avg;
int num [10], i, sum;
/* Read the numbers */
for (i = 0; i < 10; i++)
{
    printf ("Enter number\n");
    scanf ("%d", &num[i]);
}
/* Add the elements */
sum = 0;
for (i = 0; i < 10; i++)
{
    sum = sum + num[i];
}
/* Find average */
avg = sum / 10;
```

Now, I will first read the array, the first is reading the array. So, let me put a comment read the numbers all the numbers earlier you see the scanf every time I was doing scanf I was reading the number. Here I am reading the numbers and storing them.

So, what should I do? For  $i = 0$  to  $i < 10$ ,  $i$  plus plus let us increment the array. What am I doing? Printf enter number backslash n scanf. Now, what should I do? I am reading an integer, so percentage d and ampersand. What should it be? What am I reading first? Here is an array whose name is num. So, what is the first thing that I will be reading? First thing that I will be reading is num 0 this element, so scanf percentage d and num i. Why  $i$ ? Because  $i$  is the index and I can see that initially  $i$  is 0. So, whatever is being scanned will be stored here then I will go back in this loop then the next one will be stored then the next one will be stored in this way the entire array will be stored after that I am trying to find the sum. So, now add the next part that I do is add the elements.

Now, in this here I will have to again add the elements. So, suppose the, suppose the array that has been formed has got some elements like say 5 7 13 12 6 so and so forth. So, this I have to add this with this then I will get a sum and with that 12, I will be adding 13. So, I will be getting 25, then 25 and 12, 37 in that way I will go on.

So, here I initialize sum to be 0 and then for again I assign the same index I can use, I could have used other index also  $i < 10$  sorry  $i$  plus plus. I hope you remember that after each of these we put a semicolon right in for loop, some assigned some plus num  $i$ . So, what will happen? Sum was 0, sum is 0. First, in the first iteration of the loop what will happen? I take my value of  $i$  is 0. So, num  $i$  will be taken and what is num  $i$ , the first element of the num array. So, that is 5 sum plus 5. So, sum will be 5.

Now, next I go back here  $i$  becomes 1. So,  $i$  come to the second element and  $i$  add that with sum. So, sum becomes 12 automatically go there I take the index is updated,  $i$  is being updated here. So, I take the third element and add with sum, so it becomes 25 and in this way I go on adding them all the elements in an area. So, the same thing I could do using, but in this case what is the advantage that I am getting? I have got, I have not forgotten the numbers that I have read. So, I can do that and again I can print all these numbers I can say now, the sum of these numbers is so much.

So, that is how we can use an array. Now, suppose I had, I had to find the average of the elements what should I do? I have got the sum here. And then find average, what should

I do here? Here of course, I will need another variable average and so I can say and average is sum divided by 10. There is a mistake that is there. What is the mistake can you find out? Here I found sum that is ok, and then I am finding the average and why I am why while I am finding the average I am dividing some whatever sum I got here by the number of elements which is 10, I knew beforehand that these are 10 elements. Where is the error? What is the mistake? The mistake is that the average cannot be need not be an integer right. Suppose my sum was 25 and I divide by 10, so it to be 2.5. Therefore this average should not be kept here I should have declared it as a float average and not with the int right. So, you must be very careful about it. So, this is how what we have shown here is how we can read an array and how you can use an array.

(Refer Slide Time: 21:25)

Example 2:  
Computing gpa

Subject	Grade Point	Credit
0	3	5
1	5	5
2	4	4
3	5	3
4	3	4
5	5	4

```
#include <stdio.h>
#define nsub 6 /* number of subjects */
main()
{
    int grade_pt[nsub], cred[nsub], i,
        gp_sum=0, cred_sum=0, gpa;
    printf("Input gr. points and credits for six subjects \n");
    for (i=0; i<nsub; i++)
        scanf ("%d %d", &grade_pt[i], &cred[i]);
    for (i=0; i<nsub; i++)
    {
        gp_sum += grade_pt[i] * cred[i];
        cred_sum += cred[i];
    }
    gpa = gp_sum / cred_sum;
    printf ("\n Grade point average: is %d", gpa);
}
```

15  
+25  
+16  
+15  
+12  
+20  
103  
25  
= 4.15

So, let us look at this example of computing the grade point average of the students. Let us try to understand what this program does.

We have defined a variable n sub to be 6, this n sub means number of subjects all right. Now, there are 6 subjects. So, 0, 1, subject 0, subject 1, subject 2, subject 3, subject 4, subject 5, now for each of them we are taking a great point which is nothing, but a mark say, an integer 3 or 5 or 6 whatever, for every how many 6. Here you see I have used define so this line essentially means great point, 6, for 6 subjects and credit for each subject. So, suppose I have got the subject and the great point and each subject has got a credit and I have got 6 subjects.

So, subject 0 has got a great point, he has got a great point of 3, maybe he has got a b grade or whatever and the credit; that means, the importance of that subject is 5, subject 1 he has got a great point of 5, he did really well here and the credit of that was also 5. There was another subject 2 in which his great point was 4 and the credit of that subject was 4 that is the importance weightage. The subject 3 he got 5 and the credit was 3 like that and since we have got 6 subjects let us do that. Subject 4 his credit he got 3 and the importance was 4 and this one he got 5 out of 5 and the credit was again 4.

Now, the credit point is the great point is computed here as for each of them we will multiply the great point with the credit. So, it will be 3 times 5; that means, 15 plus 5 times 5 that is 25 plus 4 times 4 that is 16, then 15, then 12, then 20 and all these are added up. So, 40, 56, 66, 71 and 12, 71 and 12 is a 83 and 20. So, he got 103 points. And how many credits were there? 5 5, 10, 10 and 7 17 and 8, 25.

So, his grade point average gpa will be when I divide this by 25 his thing will be 4 point something, 4.15 etcetera. So, 4.15 or something like that. So, this is what I want to compute.

(Refer Slide Time: 25:42)

Example 2:  
Computing gpa

```
#include <stdio.h>
#define nsub 6

main()
{
    int grade_pt[nsub], cred[nsub], i,
        gp_sum=0, cred_sum=0, gpa;
    printf("Input gr. points and credits for six subjects \n");
    for (i=0; i<nsub; i++)
        scanf ("%d %d", &grade_pt[i], &cred[i]);
    for (i=0; i<nsub; i++)
    {
        gp_sum += grade_pt[i] * cred[i];
        cred_sum += cred[i];
    }
    gpa = gp_sum / cred_sum;
    printf ("\n Grade point average: is %d", gpa);
}
```

$\sum_{i=0}^{nsub-1} \text{credit}_i * \text{gpt}_i$

$\sum_{i=0}^{nsub-1} \text{credit}_i$

Therefore, what I need is to find out that, I have to find the sigma of the sum of the credit for a course  $i$  times the great point of that course  $i$ , I will do it for all the courses and then the whole thing should be divided by the total number of credits, total number of credits.

Now, let us see how we have done, we can do this program. Here we can find that the great point for 6 and credit for 6 and there is I the great points sum. So, I have to do this sum is 0 credit sum is 0 and I have to find out a gpa. So, what I am asking from the user here input great points and credits for the 6 subjects.

(Refer Slide Time: 26:43)

Example 2:  
Computing gpa

```

#include <stdio.h>
#define nsub 6

main()
{
    int grade_pt[nsub], cred[nsub], i,
        gp_sum=0, cred_sum=0, gpa;
    printf("Input gr. points and credits for six subjects \n");
    for (i=0; i<nsub; i++)
        scanf ("%d %d", &grade_pt[i], &cred[i]);
    for (i=0; i<nsub; i++)
    {
        gp_sum += grade_pt[i] * cred[i];
        cred_sum += cred[i];
    }
    gpa = gp_sum / cred_sum;
    printf ("\n Grade point average: is %d", gpa);
}

```

So, where are those being stored? Those are being stored in two different arrays. Here is the great point array, this thing and there is another credit array 6 subjects.

So, I read for each of the subjects i less than, i 0 to n sub, I read the great point say 3 and the credit for that subject say 4. And again I go and store the great point of that subject great point that is been obtained is 5 and say credit is 3. In that way I go on filling up reading 2 arrays. So, here I am reading two arrays you see, here I am reading two arrays in one scanf statement and through one for loop, through one for loop I am filling up these two arrays. So, the reading part is complete here.

Next I come to the computation. Grade point sum will be you, now know what is meant by this. This means grade point sum is equal to grade point sum plus grade point i times credit i. So, it is gp sum is whatever is the gp sum initially gp sum was made 0, gp sum plus grade point i times the credit of i and that has to be done in a loop just as we had added the elements of an array. So, and credit sum also I am also finding out the credit sum the credit sum was 0. So, whatever the credits are I am also adding them. So, by adding this array I am getting the total number of credits here and here I am getting the

grade points sum. Now, the grade point I have sum, grade point average is grade point sum by credits sum and then I print out the gpa.

So, please try to understand this part. What I am trying to do is sigma grade point i, times credit i divided by sigma of credit i. So, the sigma of credit i is being computed here this is computing sigma of credit i. Why? Because this is being done in a loop and here I am doing this part sigma of grade point 5 times credit i and adding that with grade point sum. So, these are very useful program which illustrates a number of things and in the assignments you will be given more number of programs to do which will give you a very good practice.

**Problem Solving through Programming in C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 30**  
**Array Problem**

We were discussing about arrays and we have seen how we can read an array, we have seen how we can print an array, we have also seen how we can access the different elements of an array using for loops and while loops.

(Refer Slide Time: 00:44)

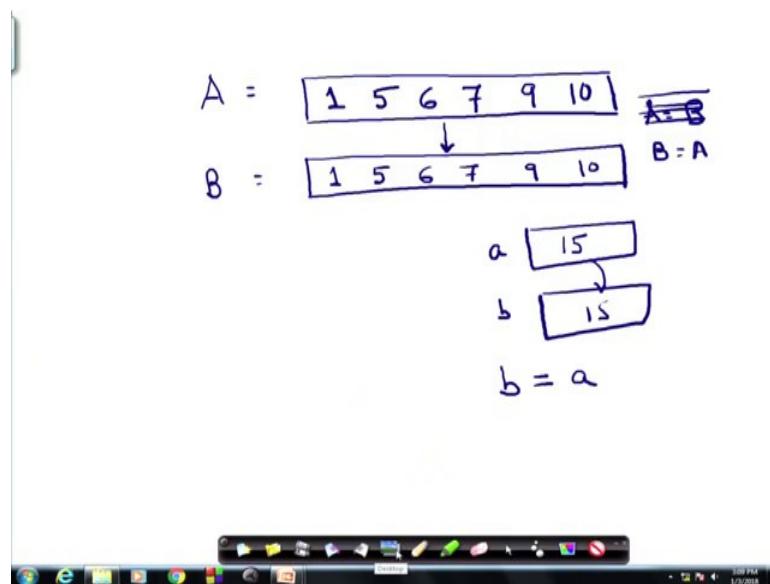
Things you cannot do

- You cannot
  - use = to assign one array variable to another  
a = b; /\* a and b are arrays \*/
  - use == to directly compare array variables  
if (a == b) .....
  - directly scanf or printf arrays ✓  
printf(".....", a);

18

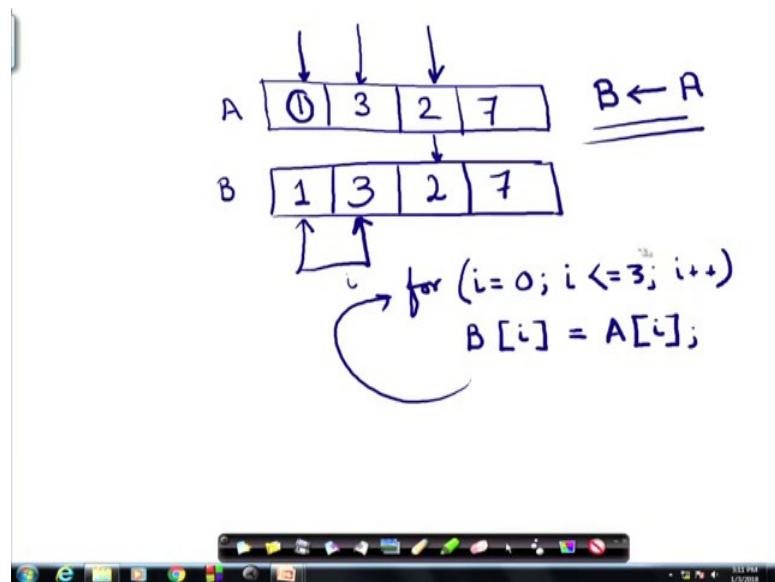
Now, couple of things that we cannot do using an array are as you can see here we cannot use. These equalities, this assignment for assigning one array variable to another, for example, if I have, suppose I have one array A suppose I have one array A which is this array having some elements suppose it is an integer array and the elements here are 1 5 6 7 9 10.

(Refer Slide Time: 01:05)



Now, and there is another array B like this, I want that this data be transferred to 1 to this array B as 1 5 6 7 9 10. Now, that I cannot do using A assigned B, I cannot do this or B assigned A, I am sorry this is wrong using B assigned a i cannot do that. However, it is true that A is also a variable of type array and B is also a variable of type array. But for other variable types for example, if A let us say A is a integer variable and B is another integer variable and a was 15 and then I could have assigned B assigned A. So, B will become 15, but this sort of straightforward assignment cannot be done in the case of an array. What we have to do in order to, this such in order to do such assignments in an array is that we will have to do it component by component element by element.

(Refer Slide Time: 03:07)



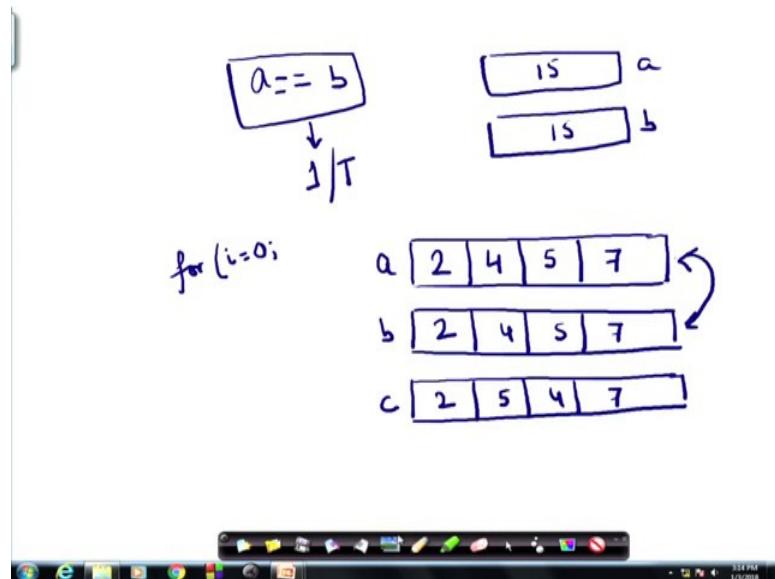
For example, I can do this let me again take another array A. Now, this time I take an array A to be a little smaller of say 4 elements 1 3 2 7 and there is another element B and I actually intend my intention is that the data in A goes to the array B. Now, as I said for an array I have to do it element by element. So, suppose this there are 4 elements in the array and I write a for loop for  $i = 0$ ,  $i \leq 3$ ,  $i++$ .  $B[i]$  is assigned  $A[i]$  what will happen through this?  $A[0]$ ,  $i = 0$ . So,  $A[0]$ ; that means, 0 this element will be taken and that will be copied to the 0th element of B here, both of them are 0, so that 0. So, this will become 1.

Next  $i$  is incremented, so  $i$  becomes 1. So, this element is being pointed at and I transfer this element to the corresponding element this will also be incremented  $i$ , this is also  $i$ , so  $i$  is 1. So, this will be copied here similarly for this element next in this loop I will be incremented and this will be loaded here. Similarly, for the last element it will go on. Therefore, ultimately I will get the same thing 1 3 2 7 as was my intention, but I cannot do it directly I have to do it in the form of a loop, element by element I have to transfer them all right. So, we cannot use this assignment directly. What else am I prohibited to do. So, A and B are two arrays I cannot do this.

Also I cannot compare two arrays using this sort of equality checker. Earlier we have seen that I can check whether two elements are equal logical operation I could do it using normal  $a == b$ , I am sorry  $a$  is  $b$ . If  $a$  was a variable 15 and  $b$  was another variable 15

in that case a equality checker b will lead to a value 1 or 2. But if a is an array and b is another array I cannot do it directly in that way.

(Refer Slide Time: 06:12)



There also I can write a piece of program in which I can check them element by element. So, please note suppose I have got an array  $a$  which is 2 4 5 7. Another array  $b$  is 2 4 5 7 then these two arrays are equal. However, if I have an array with the same elements, but in a different arrangement say for example, 2 5 4 7 these two I this  $a$  and  $c$  are not the same therefore, how can I compare how can I compare these two arrays how can I compare them. In order to compare these two arrays  $a$  and  $b$  what I need to do is I have to again write a small loop like for let me do it clearly.

(Refer Slide Time: 08:19)

a [ 2 5 7 4 ]

b [ 2 5 7 4 ]

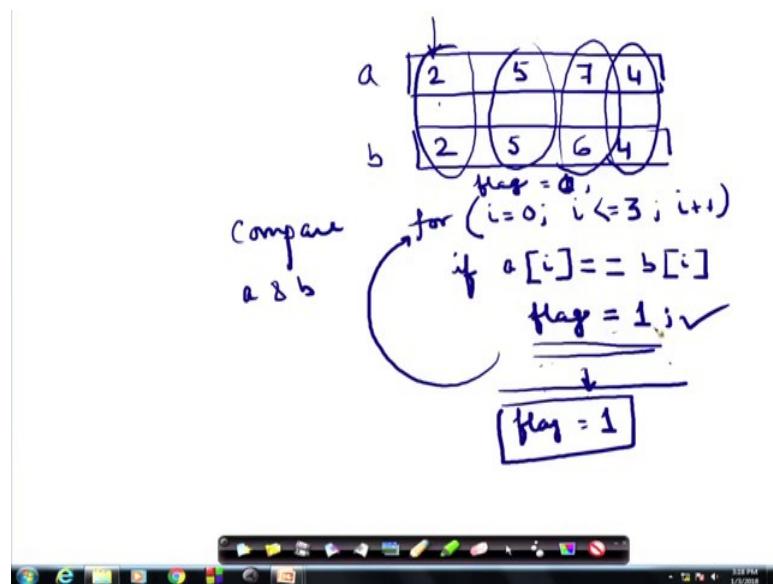
Compare a & b

```
for (i=0; i<4; i+1)
    if a[i] == b[i]
        flag = 1;
    else
        flag = 0;
```

Say I have got an array 2 5 7 4 another array 2 5 7 4, a and b are the equal I can compare them I want to compare them compare a and b. Now, what can I do? I can do it in this way for i 0, i less than equal to 3 because there are 4 elements in the array, i plus plus. Suppose I put a flag here a variable flag equals 1, suppose I keep a flag variable I have initialize it to 1 all right. Now, for i equal to 1 to 0 or let me make it simpler I keep it 0 initially this is 0.

So, now I do if a[i] is equal to b[i] flag is assigned 1 and flag is assigned 1, else flag assigned 0 and I come out of this. So, what I am trying to do is here let me let me the avoid this part for the time being let us see we will develop it gradually. What will happen here if this is done let us see, for if I do this piece if I run this piece of code then what will happen. For i equal to 0; that means, I will be comparing with this.

(Refer Slide Time: 10:47)



And if *a* [*i*] is equal to *b* [*i*]; that means, I am comparing these two if these two are equal then *flag* is 1; that means, till. Now, it is one, but this will go on here, so here 1. Now, 5 and 5 is 1 fine, 7 and 7 is compared again it is written 1, 4 and 4 again it is written 1. So, it has been done and ultimately come out with the value of *flag* ultimately I come out when I come out of this the value of *flag* is 1 and so I can say if *flag* is one then they are equal. But there is some problem with this program what is that? First of all every time I was assigning *flag* to be 1, that is one problem.

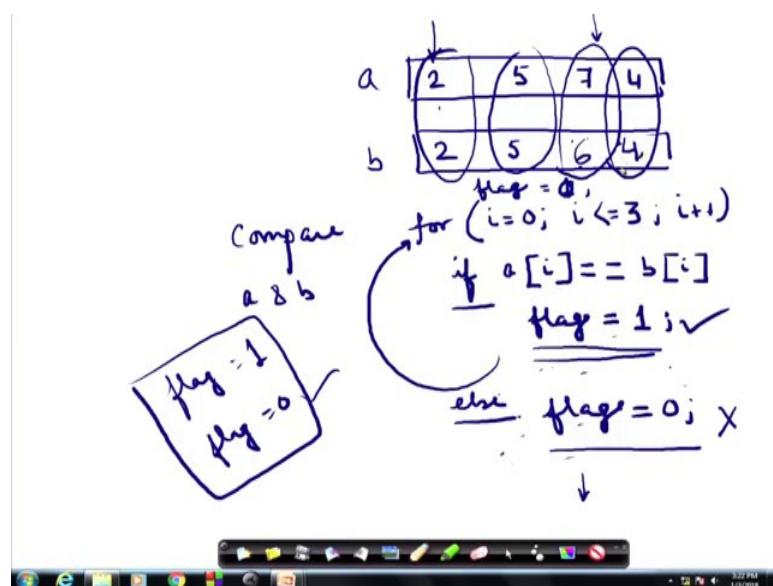
Now, suppose another problem is suppose this value is 6 then what will happen let us trace this once again 2 and 2 will match. So, the value of *flag* will be 1 fine. 5 and 5 will be matched, again I am not bothering about how many times unnecessarily I am writing into *flag* that apart *flag* will be 1. Now, 7 and 6 they are not equal to 1. So, I will come out of this if statement, but will go again in the loop. So, what should I do here? If these are not equal, not equal then I will have to set *flag* to 0 if there is a mismatch. So, if I do it like this then what will happen? It is this part, if *a* [*i*] is equal to *b* [*i*] then *flag* will be one and else; that means, with this else *flag* will be 0.

So, ultimately now, suppose, but here again there is a problem what will happen. So, 7 and 6 are not matching. So, this is not true. So, *flag* will be 0 and then I will again go into the loop because the loop is loop has not yet ended and here I will find that these two elements are equal 4 and 4. So, it will come at this point and *flag* will be 1 and then I

come out of the loop I will come out with flag equal to 1, but these two arrays are not equal because I can see that the elements are different.

So, this problem this program will not solve my purpose. I leave it to you for a while to see how to think how you can solve this problem you need to apply your mind a little bit to write this piece of program which will simply check whether two arrays are equal or not if they are equal. That means, if the elements are the same at every position corresponding position the elements are same in that case it will come out with a flag value is 1, otherwise it will come out of the flag value of 0, that is the task. I hope you understand where the problem is in this problem, in this program.

(Refer Slide Time: 14:31)

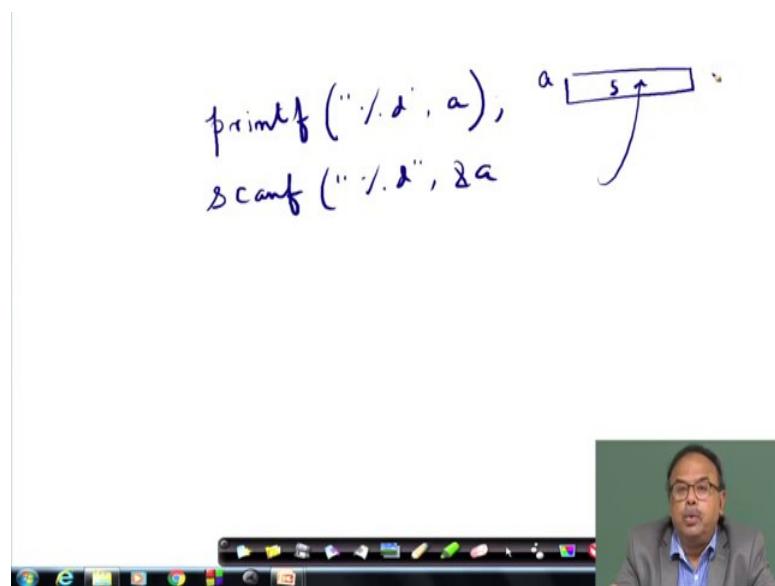


You see here had the mismatch been at the last point last moment, at the last element 7 6 then it would have worked fine. So, that is why at the beginning we said this is not just C programming, it is learning to write programs using logic. Suppose this. this was 7 and this was 3 then the errors are not equal, array are not equal, here flag would be 1, here flag would be 1, here flag would be 1, here flag would be 0 and I had come out and my answer would be correct by chance. The answer is correct by chance because as we have seen that here if I have an error if I have a mismatch here and suppose here the things are all right then I would once gate the flag is 0, but next time that flag will be made to 1 and ultimately when I come out I will have the flag value to be 1. So, the correct answer has

eluded me. So, you just think of how I can modify this program such that I get the correct result as I intended.

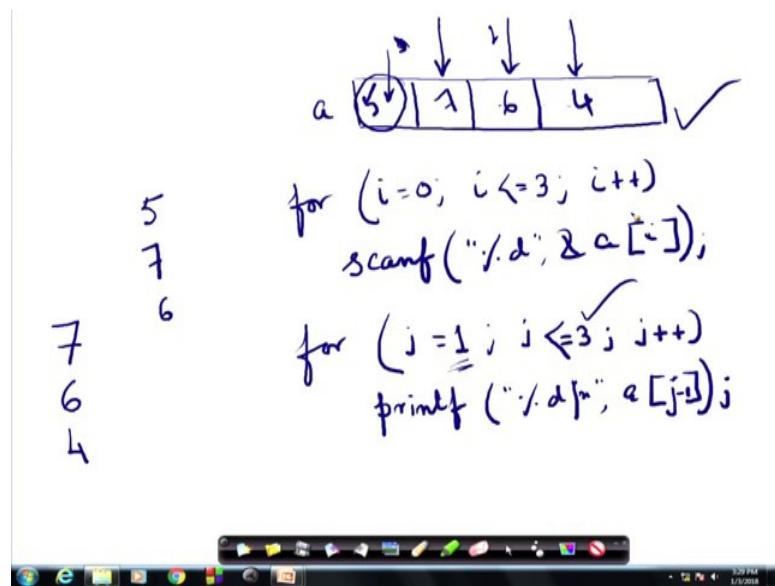
So, we cannot use these directly, directly it compare the array variables I cannot compare. What else? So, if a assigned b this sort of thing I cannot do. Also the other thing that I cannot do is scanf and printf statements I cannot read the array in one shot that is not allowed; scanf and printf I could normally, normally what could I do.

(Refer Slide Time: 17:13)



Normally if I have a variable a sum variable value 5 I could have simply done printf percentage d a. So, this could be printed. Similarly I could have done scanf, percentage d and they I could have read a value here. But if it is an array I cannot do it in this way we already know how we can do that. We know that in order to read an array what we need to do is read it again in a loop in a for loop element by element so, it can be for i assign 0, i less than equal to whatever 3, i plus plus, scanf percentage d and a i.

(Refer Slide Time: 17:51)



So, this array  $a$  will be read element by element I equal to 0. So, first this element will be read same might be 5, whatever element is 5 there might be 7 etcetera in that way I can read them element by element. Similarly for printing an array 6 4, for printing an array I can simply again do that for say  $j$  assign 0,  $j$  let me change a little bit, is any trouble here  $j$  less than 3,  $j$  plus plus,  $\text{printf}$  percentage  $d$  a i. Here of course, I do not give the  $\text{scanf}$  again.

Now, is any problem with this? Here I made  $i$  less than equal to 3, here I am making  $j$  less than 3 for the same size of array you can think of, you can look at it this is also correct because here I am starting with 1. So, 1 I am looking at. Now, there is a mistake here, I have done a couple of mistakes. What will happen here?  $j$  first of all this was a unintentional mistake. So, this is  $j$ .

Now, how will the array be printed here if I do it in this way? What will be printed? Here first thing that we printed is  $j$ ,  $a[j]$ . What is the value of  $j$ ? 1. So, what is  $a[1]$ ; that is this element. So, 7 will be printed. Suppose I have got a backslash n here. So, 7 will be, 7 will be printed, then 6 will be printed because  $i$  is now 2,  $i$  is now 3, then 4 will be printed, but till less than 3. So, therefore, 1 2 3 this will not be printed. So, and also again I am missing this one. So, what should I do? What should I do? I can make it  $j$  minus 1 what will happen in that case? I am printing  $j$  is 1 and I am printing  $a[j-1]$ ; that means, I am printing a 0 5. Next  $a[j]$  becomes 2. So, I will be printing  $j$  minus 1; that

means, 2 minus 1 a 1, 7 then 6. So, 7 then j becomes 2 then j becomes 3. So, 3 minus 1 2, I have printed a 2 have printed.

Now, j will be increment it will be 3 the last one will not be printed. So, the here also I have to make it less than equal to 3. Was this one alright? Just quickly check. This is a common source of error I am reading this array. Let me do it again I am reading this array, a 0 then I have read a 0, then i is incremented, so the 4 elements, so I will come up to a 3. So, the second one I am reading the second one a i is 1. So, a 1, i is next 2 a 2 and then a 3. So, this is ok. So, you have to be a little careful about all this.

So, reading and printing, reading and printing we cannot directly do we have also to do it through a loop. So, I cannot do like this.

(Refer Slide Time: 23:13)

The image shows a presentation slide with a black header and footer bar. The main content area has a white background. At the top, the title 'How to copy the elements of one array to another?' is centered. Below the title, there is a bullet point followed by a code snippet. The code is as follows:

- By copying individual elements

```
int a[25],b[25];
for (j=0; j<25; j++)
    a[j] = b[j];
```

The footer bar contains several small icons, likely for navigating the presentation, and the number '10' indicating the slide number.

How to copy elements of one array to another? By copying individual elements I have already shown that. So, that is what we have to say about arrays right. Now, will come to some examples say I want to this is an example of copying an array there are, there are two arrays a 25 and B 25. So, I am trying to copy them. So, you know that I can do it by copying it in this way.

(Refer Slide Time: 23:47)

## Some Problems to Try Out

- Find the mean and standard deviation of a set of n numbers.
- A shop stores n different types of items. Given the number of items of each type sold during a given month, and the corresponding unit prices, compute the total monthly sales.

Now, some problems we will like to try out. Now, one is finding the mean and standard deviation of a set of n numbers. I leave this standard deviation part we can try to do that mean you know mean is the arithmetic average. So, I will first try to find write a program where all of you will be able to write find the mean and standard deviation of a set of n numbers.

(Refer Slide Time: 24:30)

$$\text{Mean} = \frac{\sum_{i=1}^n a_i}{n}$$

a [ ]  
ai is an element

```
int A[5], sum=0;
float avg;
/* read array */
for(i=0; i<5; i++)
    { scanf("%d", &a[i]);
        sum = sum + a[i];
    }
avg = sum / 5;
```

So, what are you going to do? Mean means the average right. So, I add the numbers all the numbers suppose my array is a. So, each element I designate as a i is an element. So,

I will have sigma a<sub>i</sub>, i varying from 1 to n whatever this size is divided by n that is the average I add all the numbers. So, it is simple to write the program. Suppose I have read the array I have first of all I have to read the elements in an array.

So, summarizing all what we have learnt what should be my first statement first statement is declaring the array. So, let me declare an array, int a suppose the size of the array is 5, maximum size possible. And I also have sum and I have got another variable avg all our integers, again avg may not be integer, mean may not be integer. So, and then I have got another float avg right.

Now, first I have to read the array. So, read the numbers, I am making a shortcut I am leaving out the printf statement please enter the array that part I am leaving out. Then what I am doing here? Here of course, for i assigned 0, i less than suppose there are 5 elements, i less than 5, i plus plus. If it was some other value then I would have put it n here and I had to read the n before I read the array, all right.

So, I do this and then initially I can make it the sum here to be sum is initialized to 0 all right. Now, for i equal to 0, i less than m, i plus plus I read the array I am just now reading the array. So, what I do? Scanf, percentage d, ampersand, a i, I am reading that and then here in this for loop I can repeatedly compute the sum, sum equals sum plus a i. So, I will get the sum here ultimately at the end of the loop, at the end of the loop I will get the sum. So, now, here I can write average is sum divided by whatever the value was I have taken 5 to be a fixed value, so I divide with 5 and that is the average. So, in that way I can find out the mean this value. The next task is computing the standard deviation. So, that I leave to you for the time being. We will take it up later.

**Problem Solving through Programming in C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 31**  
**Linear Search**

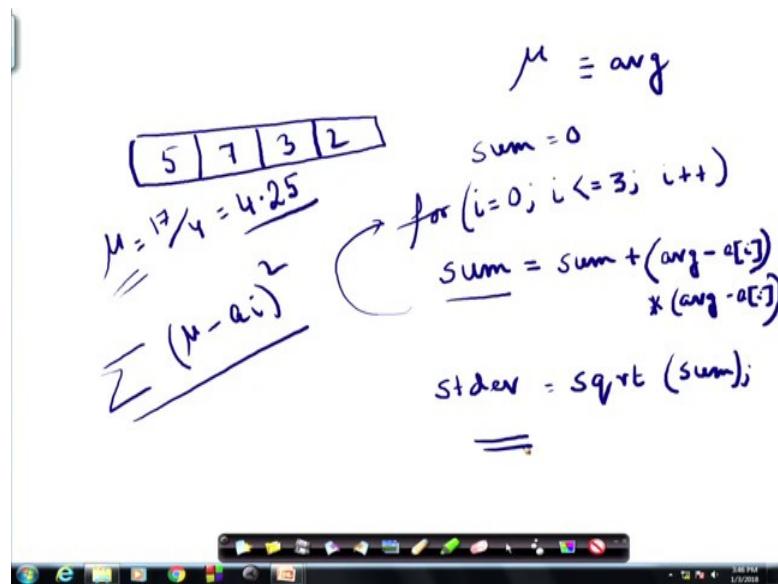
So, we were looking at computing standard deviation. So, standard deviation is mean; we have seen.

(Refer Slide Time: 00:28)

The image shows a whiteboard with handwritten mathematical notes. At the top, it says "Suppose mean =  $\mu$ ". Below this, there is a diagram of an array with elements labeled  $a_1$  through  $a_i$ . To the left of the array, the formula for standard deviation is written:  $\sqrt{\sum_{i=1}^n (\mu - a_i)^2}$ . A small coordinate system is drawn next to the formula. At the bottom of the whiteboard, a computer taskbar is visible, showing various icons and the date/time as 3:43 PM 5/1/2010.

Now, standard deviation is if the mean is suppose the mean is represented by mu then I take the sum of the deviation from the mean for every element of the array. So, the array was a with every element being called a i.

(Refer Slide Time: 01:56)



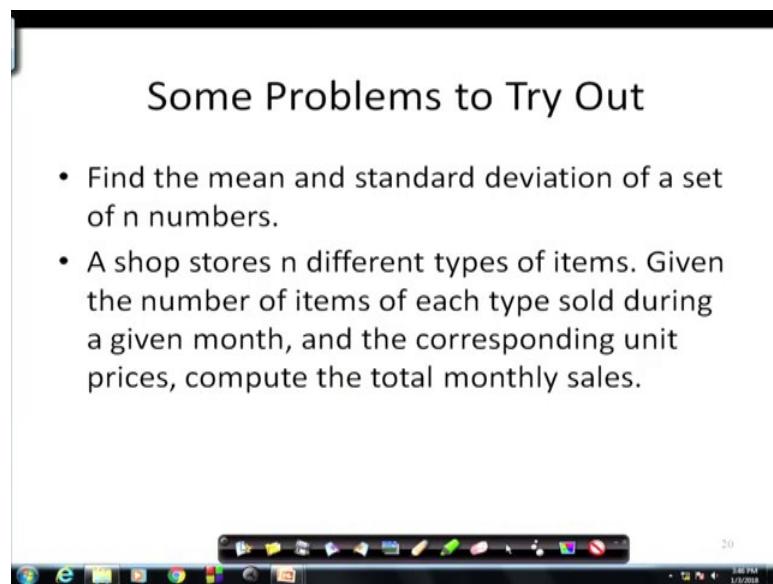
So, what I am trying to do is, I am taking the difference of a  $i$  from the mean and since its difference can be suppose there are some values and this is the mean and the value can be a little away from the mean on this side more than the mean or less than the mean. So, we take the square of the mean square of the difference and I do that for all the elements  $i$  equals 1 to  $n$  right and then I take the square root of the whole thing. That is my standard deviation other my variance. So, I can compute either this or this, whatever I like.

So, you will simply understand that in the code that we have given little earlier then there we had computed the mean; that means,  $\mu$  has been computed which was the average that we computed last time. So, if I have read the elements in an array whatever the elements are 5 7 3 2 and I have computed the mean  $\mu$  who has been computed mean is of 12 3 15; 17 divided by 4. So, it is 4 point something 4.1, 4.25 right that is my mean. So, now, for every element again in this array I find out for  $i$  assign 0,  $i$  less than equal to 3,  $i$  plus plus what do I do? For every element I have got the  $\mu$  and let us call it average right.

So, sum was 0 what is it look like, sum was 0, sum will be now sum plus I am getting sigma of  $\mu$  minus  $a_i$  whole square. So, sum plus average minus  $a_i$  times average minus  $a_i$ , all right. So, that is the square average is  $\mu$  minus  $a_i$  that is. So, I am doing this square and I am repeatedly doing this and getting the new sum and at the end of this

loop therefore, I have got this. So, I can see std, it is a dev let me call it standard deviation is square root of the sum all right, in that way I can find out the standard deviation also.

(Refer Slide Time: 04:24)



## Some Problems to Try Out

- Find the mean and standard deviation of a set of n numbers.
- A shop stores n different types of items. Given the number of items of each type sold during a given month, and the corresponding unit prices, compute the total monthly sales.

Now, let us come to the now what is the application what is the meaning of this. So, with these say in a class you are you are supposed to write a program where you want to find out say in a class of physics. What is the average of the numbers of all the students? So, I will find out the mean and mean of the class marks right. Similarly I can find out that what is the standard deviation, how much did it vary, that also I can find out.

Now, let us look at this new another problem. Say a shop stores in different types of items, n different types of items. Now, given the number of items of each type sold during a given month and the corresponding unit price is compute the total monthly sell. So, what is the scenario? The scenario is this.

(Refer Slide Time: 05:24)

The image shows a computer screen with a slide from a presentation. The slide contains handwritten notes and two tables.

Handwritten notes:

- item-cost
- item-sold

Tables:

0	1	2	3	4
7.50	25.0	12.5	10.0	50.0

5	6	2	4	2
---	---	---	---	---

$(7.5 * 5) + (25.0 * 6) +$



I have got say 5 items, item 1, item 2, item 3, item 4 and item 5. And let me call it the item, let me call it on this side let me call it this array is item price, item cost, item cost. Suppose the cost here is 7 and half rupees per item of type 0, 25 for item of type 1, this is 0, this is 1, this is 2, this is 3, this is 4. So, the item cost for this for item of type 2 is 12.5, item of type 3 is 10, item of this is item of type 4 is 50 rupees.

Now, I want to store how many items of each type has been sold. So, I take another array and call it items sold. Suppose 5 items have been sold of type 0, 6 items here of type 1, 2 items of type 2, 4 items of type 3 and 2 items of type 1. Now, my question is what is the total sell? So, what should I do? You can easily understand that the here is the item cost. So, I have to multiply this with this, this with this and add all these costs. So, it will be 7.5 times 5 it will be 7.5 times 5 plus 25 times 6 plus so and so forth in that way I can find the total cost. So, what will the program look like? The program will look like I will be needing to arrays item cost and items sold and they should be of the same size, assuming that I know beforehand that there are 5 items.

(Refer Slide Time: 08:46)

The image shows a screenshot of a computer desktop. On the desktop, there is a handwritten note in blue ink. The code starts with variable declarations: `float total-sale = 0.0;`, `float item-cost[5];`, and `int item-sold[5];`. To the right of these declarations are two boxes, each containing three checkmarks (✓✓✓). Below the declarations, a bracket groups them together with the handwritten note "Reading the array". The next part of the code is a `for` loop: `for (i=0; i<=4; i++)`. Inside the loop, there is a brace grouping a series of assignments: `{ total-sale = total-Sale + item-cost[i] * item-sold[i]; }`. The desktop background shows a standard Windows-style taskbar with icons for various applications like Internet Explorer, Google Chrome, and Microsoft Word.

```
float total-sale = 0.0;
float item-cost[5];
int item-sold[5];
    ] Reading
    the
    array
for (i=0; i<=4; i++)
{
    total-sale = total-Sale
    + item-cost[i] *
    item-sold[i];
}
```

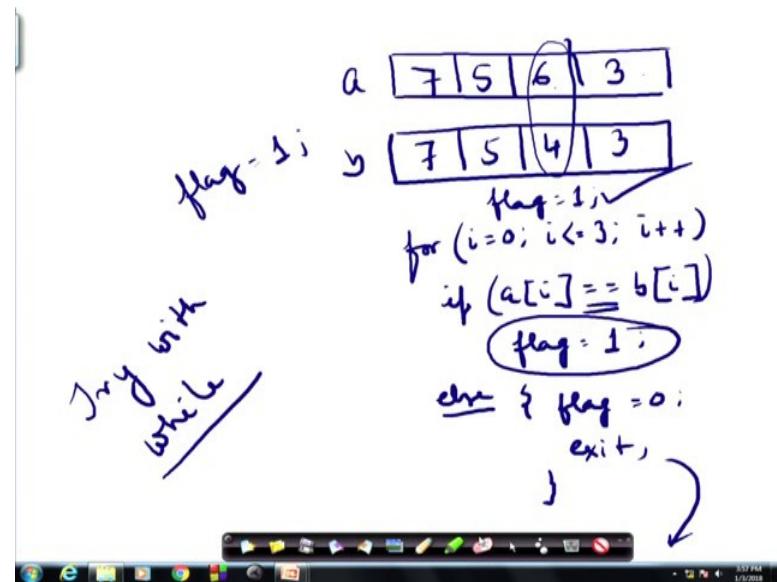
So, I can start with item cost of type 5, but this will be cost will be of type float and there will be another the number of items is, item sold the number of items sold is integer. So, I will have this, I am not showing the part that here I am reading the two arrays. So, after I read the arrays I will have two arrays like this, one is an integer array another is a floating point array this is the float and this is an integer.

Now, my actual body of the program will be in a loop for say I am doing it using for say `i assign 0, i less than equal to 4 since the size is 5, i plus plus`. And what do I do here? What do I do in the body of the for loop? I take ok, I write actually only one statement will do there is no harm in putting this bracket. Total sale which was a variable of type float total sale is, total sale was initialized to 0, total sale plus item cost  $i$  times multiplied by item sold  $i$ . So, this will be done in a loop.

And so I will take the first item, item cost 0 multiply with that with item cost item sold 0, add that with the total sale which was initialized to 0. So, here I can have float total sale initialized to 0, 0 point 0 all right, I can do that. So, now, I am doing it in a loop. So, first I multiply these two add it to total sales, next again in the next iteration  $i$  is implemented I take these two and multiply them and add it to the total sale then I do this two and multiply them add it to the total sale and I go on in this way. This is, this is in this way by using this array I will be able to add all these values and I will get the total sale at the end.

So, here we could see two very nice examples of application of arrays. Now, one another problem that I gave you I mean as I was while comparing the arrays you can do it in multiple ways that they were as two arrays you must have solved it by now, that they were two arrays like this 7, 7, 5, 5, here 6, here 4, but again 3, 3.

(Refer Slide Time: 12:24)

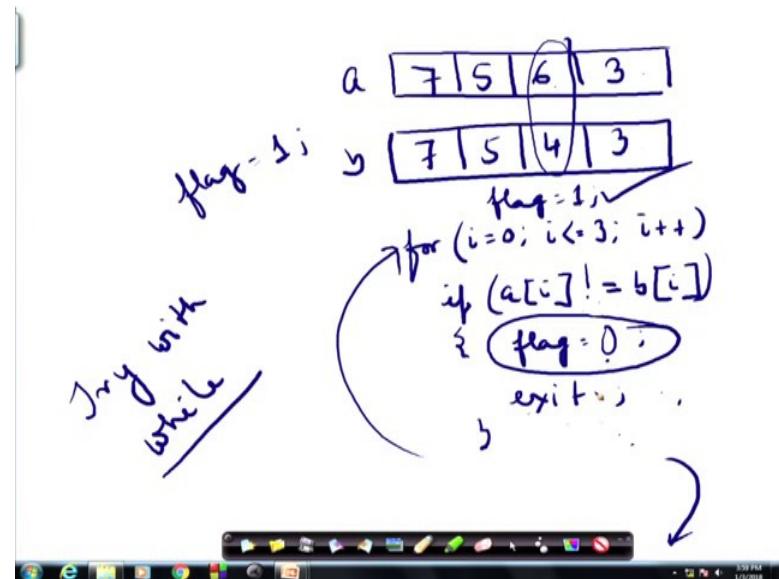


So, everywhere they are matching, but intermediate they were affected and I had a flag value so flag was initialized to 1. And then in a for loop then in the for loop I was checking this was a and this was b, flag was initialized to 1, if a[i] is equal to b[i] this I can do because here I am doing it element wise then flag equal to 1 else flag equal to 0 and I can do exit and I come out of the loop straightway, I come out of the loop. Because it does not really matter at which position the mismatch occurs, as soon as the mismatch occurs I can say that flag is 0 and so I come out. So, when I was comparing this whenever I find a mismatch the flag will become 0 and it will not be reset to 1 again because of this match, because this part is not being computed I need not compute it. I need not compute it because my objective was to see if the two arrays are equal and here the violation has already occurred. So, they are not equal.

However so, that is one way you can try it with while loop also. You can try with while to solve the same problem. Another point is here the time and again here, here, everywhere I am setting the flag to 1, I could have changed that also here if what did I

need to do I go on I have in take flag to 1 and the condition I simply change I just change the condition if a[i] is not equal to b[i], then make flag 0 exit.

(Refer Slide Time: 15:20)



I could have done this. As long as this condition is not holding I am going on doing the loop is it clear, I will go on doing the loop as long as there is no mismatch, this condition means mismatch. As soon as there is a mismatch I will set the flag to 0 and exit. There is another way of solving the problem. So, you have to think logically what exactly you need to do and what exactly you are writing what is the flow and what is happening with the variables and I always suggest that you have small pictures of the different variables and see how they are changing in the course of running the program.

Now so, we have seen a useful commercial. So, called toy commercial problem that how I can find out the cost of a total sales or monthly sales. So, and here the number of items sold per month are given then you can do it.

Well, next let us look at a very important thing called searching. Searching is a fundamental task in any and in fact, in many computations. In many computations we need to search. What do we search? There are different types of searches, but we will be now talking about the simplest possible search that is we are trying to find out whether a particular element is there in an array right.

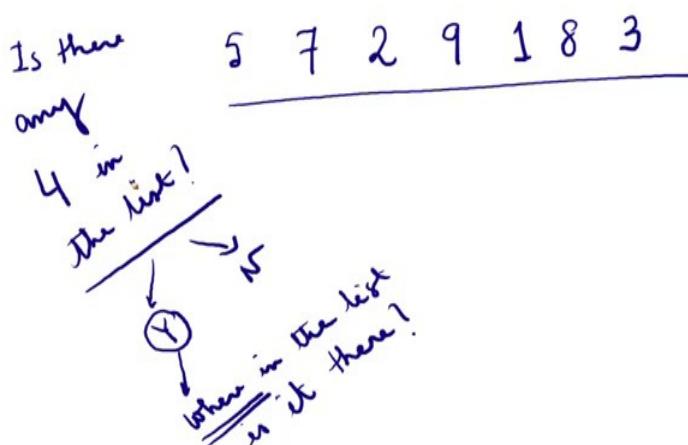
(Refer Slide Time: 17:24)

## Searching

- Check if a given element (**key**) occurs in the array.
- Two methods to be discussed:
  - If the array elements are unsorted.
  - If the array elements are sorted.

Say, so the purpose is to check if a given element which is known as the key is there in the array or not. We will first talk about the array is not arranged in any order and we will do that.

(Refer Slide Time: 17:56)

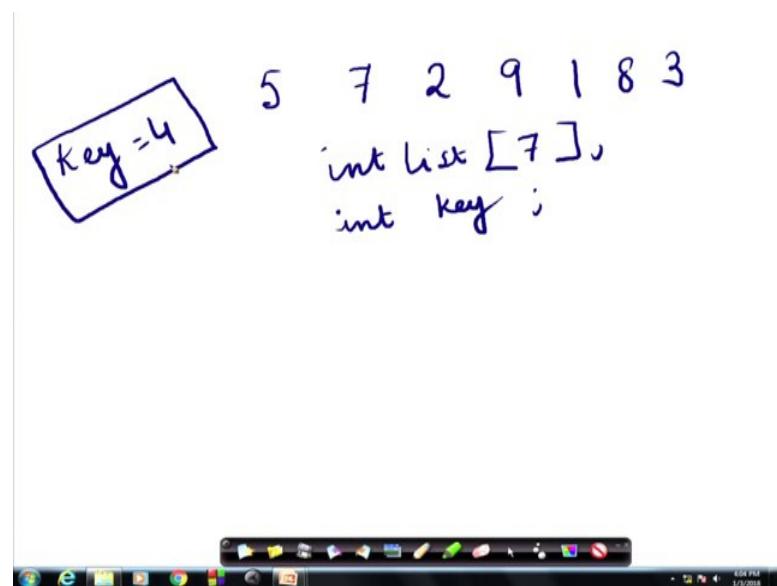


So, suppose I ask the question that is there any even number in the set of numbers given, suppose there are some numbers given 5 7 2 9 1 8 3 like that. I want to see and suppose this is a huge this is a list of hundred numbers. I want to find out whether there is any particular number forget about even number, for the time being for the timing let us

assume that I want to see whether in this list any 4 is there, is there any 4 in the list that is the question that we are asking. The answer can be either yes or no.

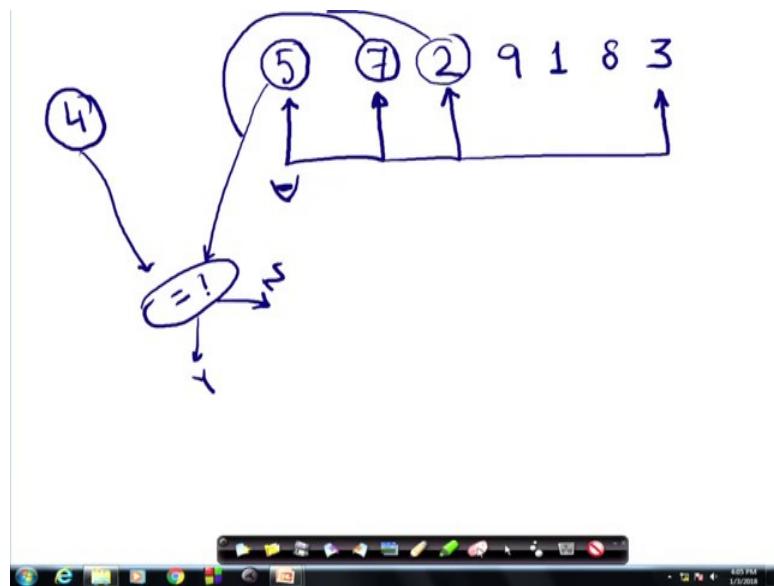
If it is yes then the next question comes where in the list is it there, is it where means in which position it is there. I may ask like to know the position or I may not like to know the position I would be satisfied to know whether this list contains any 4 or not all right; so 5 7 2 9 1 8 3.

(Refer Slide Time: 19:28)



So, again 5 7 2 9 1 8 3 and my key is 4 because I am interested in the existence of 4. So, instead of writing the c which you will be writing I will be discussing how to approach this problem what would the pseudo code be and I am sure in the assignments in your practice you can write the program. So, I know I need to know beforehand this list. So, so I need to know a list which may be an integer list of might be here 7 numbers, I need to know that also I need to know which key I am searching for.

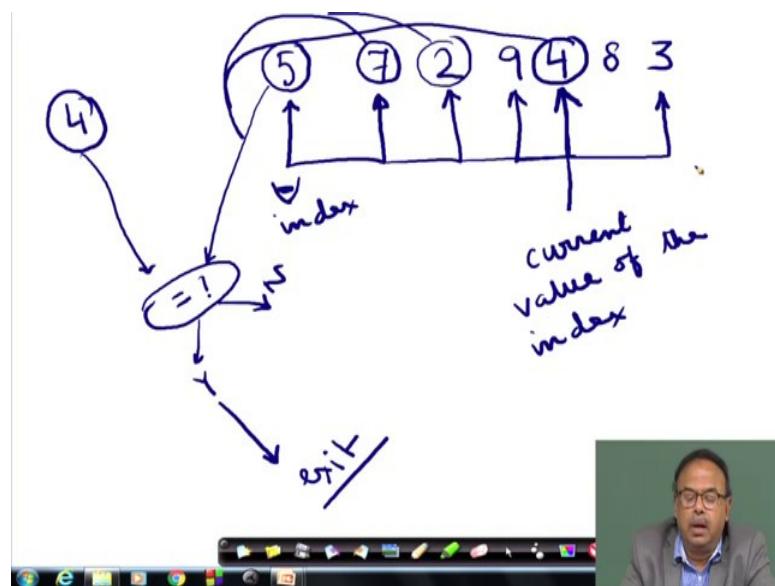
(Refer Slide Time: 20:46)



Once I know these two, I know 5 7 2 9 1 8 3 is my list and I know that 4 is my key then how should I go about it.

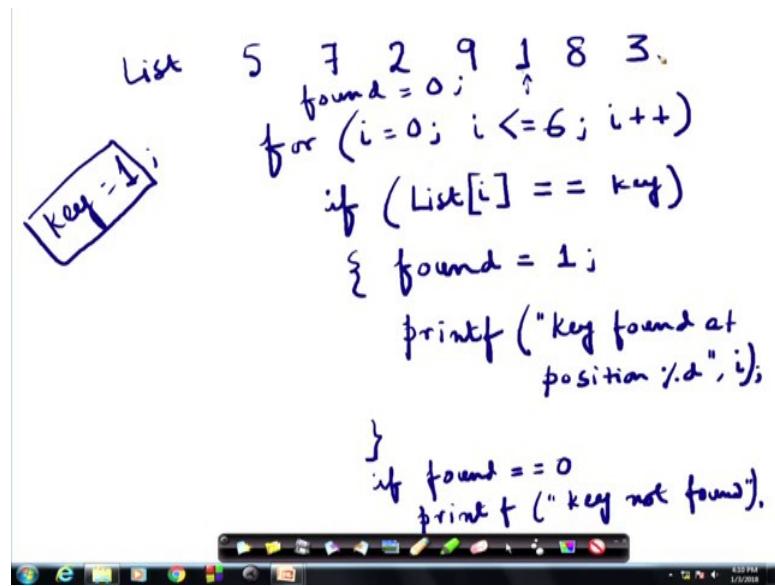
I have got 4 in mind. I start to look at as if I am looking at through some means at different positions I look at this position and check this element and compare this element with the key and I ask are they equal. If the answer is yes, then obviously I can say that 4 is in the list. But as you can see it is not true, so in case of no I will shift my focus from here to here and I will now compare with 4 this element 7, are they equal, no then I will again shift the focus and I will go on shift the focus and I will compare with this element with the key all right. In that I will go on. How long shall I go on till the end of the list.

(Refer Slide Time: 22:27)



If suppose here there was a 4, instead of 1 there was a 4 here, suppose instead of 1 there was a 4 here. Then when my focus changes to this point and then again comes to this point and then I find that this element is matching the key value, then I can exit and say yes, 4 is there in the list. Then if there is other question has to be answered that 4 is there, but where is 4, where is 4, in that case what would be an answer be. Your answer would be this position. And what is this position? This position is nothing, but the current index current value of the index. Here the index started this is my index which was shifting.

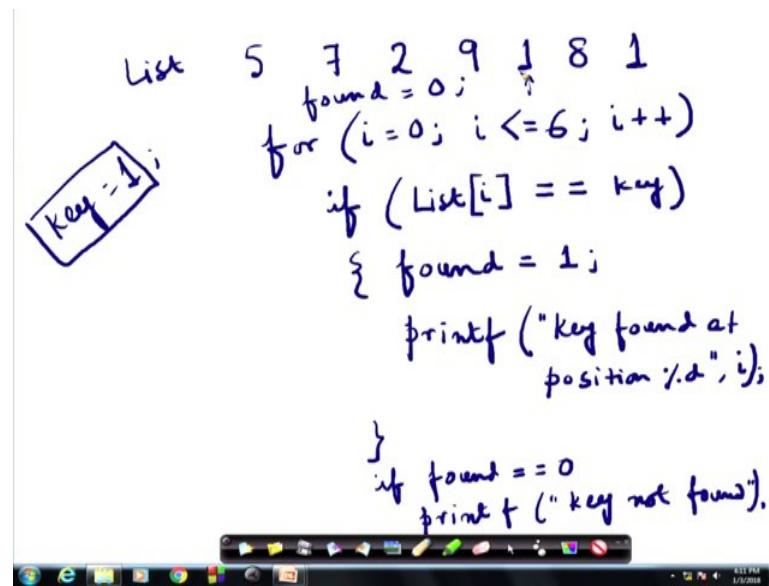
(Refer Slide Time: 23:48)



So, now we can think of the algorithm I have got 5 7 2 9 1 8 3 and my key is 4. So, I will be doing. So, this is a loop which is a list for i equals 0; that means, the focus or the index i less than equal to 1 2 3 4 5 6 7 6, i plus plus, i plus plus. If least i, that means, the ith element of the list is equal to the key then I can say found assigned 1. And what is found? Found is some variable which I have initialized at this point initially nothing is found it is not found. So, found is 0 initially I have not found the key here I am comparing as soon as I compare I put found equal to 1 and then I can exit or this automatically this loop will go on.

Now, if I do it in this way what is the problem? Suppose my key was, suppose my key was 1 then it goes on i 0, this is never happening found is still 0, it goes on it comes to here and I count come to found equal to 1 and then I can say if found equal to 1, I can printf, printf key found at position percentage d i. So, at that point I can also print that it has been found here right. In the worst case what can happen? Found will remain 0, found will remain 0 and I will come to the end of this point when I come here I can check if found is 0, printf key not found; if I do not find it. There can be different flavors of the same problem.

(Refer Slide Time: 27:46)

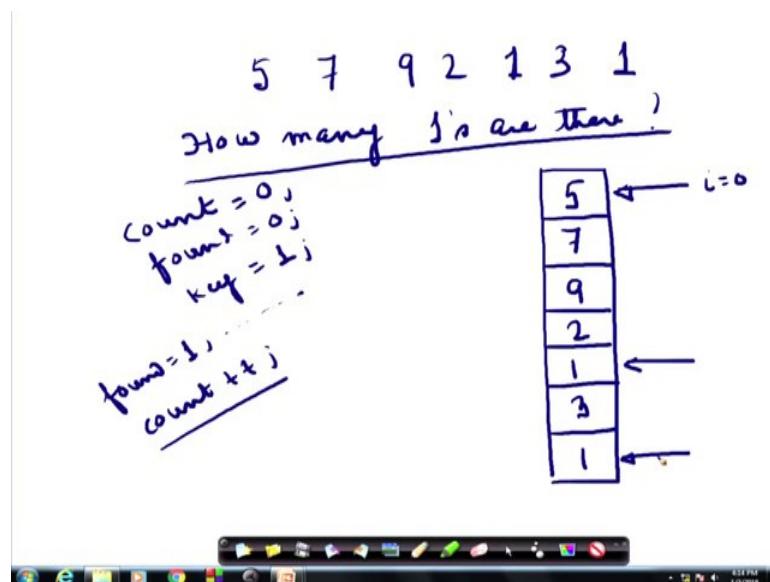


The other flavor could be that suppose this element 1 is there at multiple places suppose it is also here. Then what will this program result in? What would be its output? I will go on checking here I will check the for loop is extending up to which part; the for loop is

extending up to this, this position and this is separate I should not have I should have written it on the on this side.

So, I go on checking this. So, key is found at position number 0 1 2 3 4, key is found at position 4. The list is not exhausted ultimately it will come to this point and when it and again it will say the key is found at position 6, twice it will it is found it will be told like that keys it will print twice. If at the end it comes and still the key the value of the variable found is 0 then it will say print is not found.

(Refer Slide Time: 29:05)

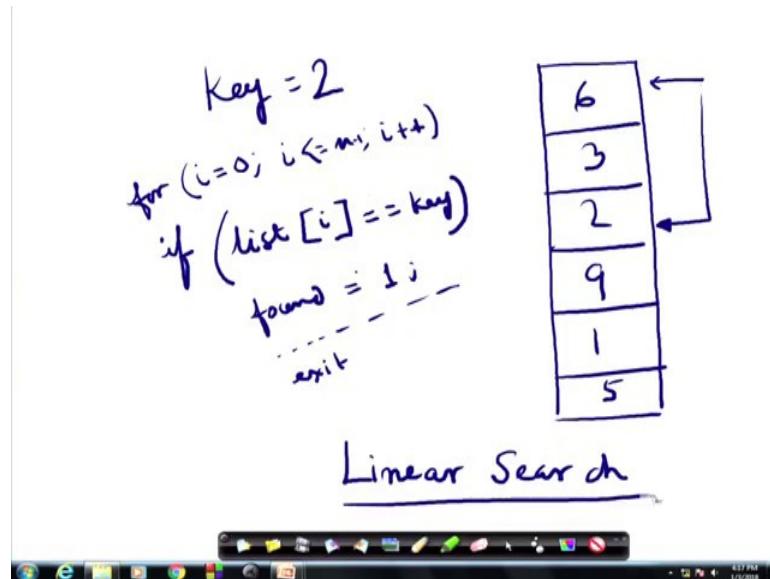


I could have also say the same thing same problem I do, say suppose I have got a list 5 7 9 2 1 3 1 the question is how many ones are there. The same algorithm will do, the same algorithm will work right. What is the same the algorithm? The algorithm is I start from one point from this beginning I let me draw it in this way if this be an array where all my elements are there, 5 7 9 2 1 3 1 I start from the beginning i equals 0 and for every element I compare with the key and go on till the end.

Now, if I want to do this what is the additional variable that I will require? I will require another variable count which is initially 0 other than found which is also false. So, whenever I find a 1 and my key is 1. So, whenever I find a 1 I will say found is equal to 1 and also I will do count plus plus and I will continue. Here I come and I will find I will have the value of count to be 2. So, I can also, so I can say that here I could print where

found is becoming 1. So, I can also, I can say at which position it is found and how many times it is found.

(Refer Slide Time: 31:27)



Now, it can another flavor could be that I have got this say this array; I have got this array and some array 6 3 2 9 1 5 and whenever I have been given a key say the key is 2, as soon as I find 2 that is enough for me. I just want to know whether 2 is there in the list, I am not interested to know how many times it is there or in which position it is there or at best I may like to know at which position I you found it first. So, what I can do, I will go on searching like this and whenever you find 2 then you print that I have found 2 here and exit.

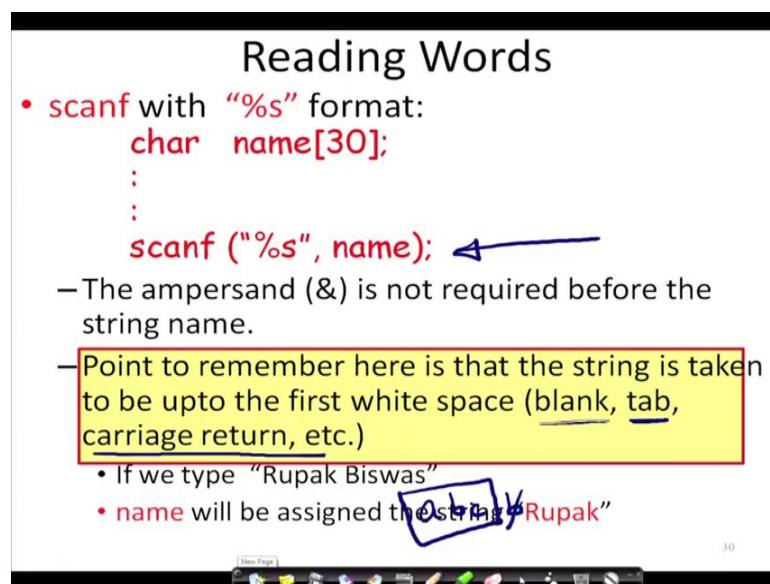
So, what will you do? The loop will be, the loop inside the loop you will have you can do it like this that if a i or list i, let us I was writing list i is not equal to key all right for you could have done it by while also n minus 1 i plus plus you go on doing this. If list i is not is sorry is if it is equal to the key what I have done is equal to key say found equal to 1 and printf the position and exit and you need not go through the entire loop. Now, there are, so that is this sort of search which I am doing in a linear way from one side to the other is known as linear search. It is the simple, very simple search for a particular element. We will see a little bit more on this in the next lecture.

**Problem Solving through Programming in C**  
**Prof. Anupam Basu,**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 32**  
**Character Arrays and Strings**

We were discussing about strings and in today's discussion, let me recapitulate what we or the last part of what we discussed in the earlier lecture.

(Refer Slide Time: 00:29)



**Reading Words**

- `scanf` with "%s" format:  
`char name[30];`  
:  
:  
`scanf ("%s", name);` ←

– The ampersand (&) is not required before the string name.

– Point to remember here is that the string is taken to be upto the first white space (blank, tab, carriage return, etc.)

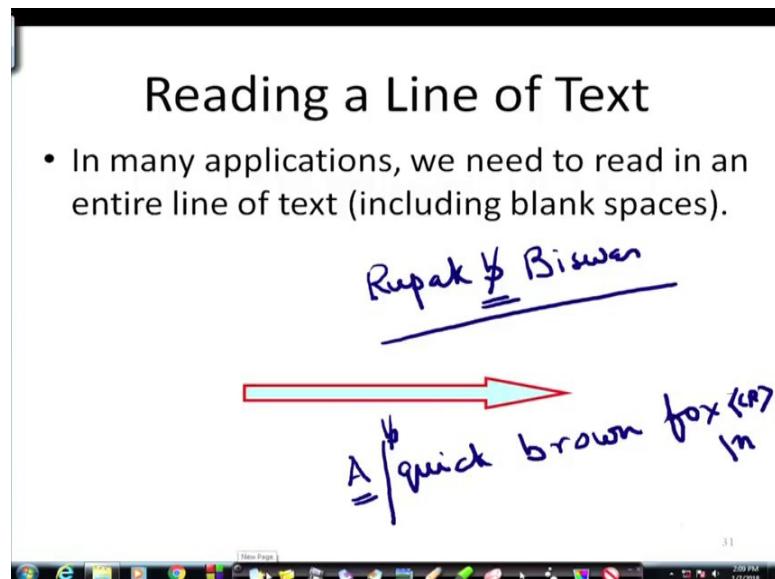
- If we type "Rupak Biswas"
- `name` will be assigned the string "Rupak"

How can we read words from this from an input say? So, there we had introduced, this percentage is format with that, with percentage is if we read a name, then it has to be noted that the ampersand is not needed in this case right the ampersand is not needed when I am reading a string when I am reading a string I do not need that ampersand.

Now, the point to remember is that whenever I am reading this in this way the string will be read until there is some blank white space this is also called as white space or a tab sign or a carriage return or enter is pressed. So, if I type on a b c and then I type blank, then this a b c will be taken as the word. So, when I perform scan if in this form in name then a b c will go in the name alright. So, this is what we had seen last time.

So, that is why this example was given if we type Rupak Biswas since after Rupak, there is a blank we will stop at that point of time. The name will be assigned to Rupak ok name will be assigned to the string Rupak name will be assign the string Rupak.

(Refer Slide Time: 02:25)



On the other hand when we read a line of text if I suppose I want to read say this Rupak Biswas, I want to read this blank also this entire thing the entire sentence, I want to read a quick brown fox I want to read.

So, I want to read this characters as well as this blank space is everything then what will be my delimiter? My delimiter will be the carriage return the return carriage return or that is often designated as backslash n. So, till I get a backslash n, I will go on reading this that is how I want I can read a line.

(Refer Slide Time: 03:26)

## Reading a Line of Text

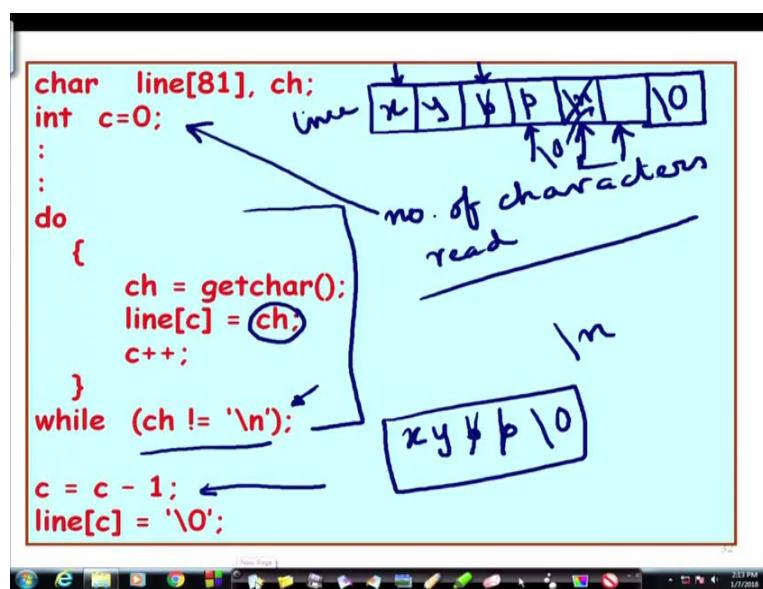
- In many applications, we need to read in an entire line of text (including blank spaces).
- We can use the `getchar()` function for the purpose.



31

So, in order to do that we can use the gate care function for the purpose by gate care function we will be getting the characters one after another, and I will go on getting the functions till the I mean I will be going on getting the characters, till I come to a character that is carriage return designating that there is an that is an end of the line.

(Refer Slide Time: 04:56)



So, for example, here a line has been defined to be of maximum length 81, typically as I said a line consist of 80 characters and you must be understanding now realizing now that why I put 81, the reason is here I need a space for backslash 0. And I put c to be 0 c is the number of characters c is here designating the number of characters read, all right.

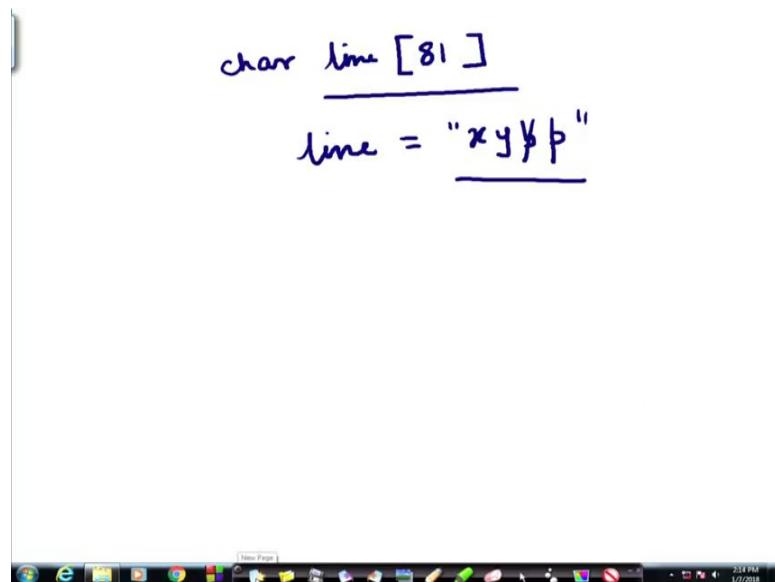
Now, here we are doing it through a do while statement, you can see that we can do a we are doing it through a do while. So, I am reading one character here say x and that I am reading it though to get char, and that character that has been read is coming to the variable array line all right line c c was initially 0.

So, here I put x, then I go up then character I go on I read it and then found that their character is not ampersand. I go back again and read another character say x y increase c. So, c is now pointing here, c is now pointing here and I again go back now suppose a blank has been typed in. So, I come here get char is blank. So, that comes here and then c is pointing to the next position here, and then I go again here and suppose I type in p. So, the c is now pointing here, and then after that I read the character and the character was ampersand I mean the character was backslash n; that means, the end of the character.

So, I put backslash n here, whatever it was there and c is pointing here. Now I have after insertion of backslash n, I come and find here that well what I have entered is backslash n. Now backslash n should not be there instead of that we should have backslash 0 therefore, I will come back I will decrement c by ones position. So, it will come here and I will replace this with backslash 0 right here I will bring in backslash 0 will come here.

So, the pattern that will be stored do will be x y blank p backslash 0. So, that is what this thing will look like in that way I am reading a line that is how we are reading a line. So, we have seen that we can read characters as a character array.

(Refer Slide Time: 07:22)



Char some line in this way line 81. Now this line I can read character by character or I can type in line to be x y blank p that is also will do the same thing, otherwise, I could have copied it character by character both of them are equivalent, all right.

(Refer Slide Time: 08:02)

```
char line[81], ch;
int c=0;
:
:
do
{
    ch = getchar();
    line[c] = ch;
    c++;
}
while (ch != '\n');

c = c - 1;
line[c] = '\0';
```

Read characters until CR ('\n') is encountered

Make it a valid string

So, here we have read the character until the char return or backslash n is encountered and then we make it a valid string by replacing backslash n by backslash 0.

(Refer Slide Time: 08:15)

### Reading a Line :: Alternate Approach

```
char line[81];
:
:
scanf ("%[ ABCDEFGHIJKLMNOPQRSTUVWXYZ]", line);
```

→ Reads a string containing uppercase characters and blank spaces

```
char line[81];
:
scanf ("%[^\\n]", line);
```

→ Reads a string containing any characters  
wild card

An alternative approach of reading a line can be this that I just specify the format, here till now what we have seen is that our formats could be percentage s percentage d

percentage f like that. I can also specify my format in this way as you can see here, what it means is anything I am sorry what happened; anything that is a b c d or whatever that this entire thing and here there is a blank, I do not know what is happening here, here you can see there is a blank here.

So; that means, what is allowed? Anything including bank blank capital A to Z everything is allowed as members of the variable line. Similarly here the specification is that it can be char at this is a wild card it is called; that means, we can put in anything preceding backslash n. So, this means it is a wild card; that means, anything can come here as a character, that can be a member of the variable line. So, that is also another way of specifying it.

Now, these are specifics to the language c of course.

(Refer Slide Time: 10:13)

## Displaying Strings on the Screen

- We can use `printf` with the “%s” format specification.

```
char name[50];
:
:
printf ("\n %s", name);
```

34

Now it is easier relatively simpler to display strings on the screen we can simply do a string, we can display the string with percentage x s and followed by the string name that is simpler now we come to a very important aspect how do we process character strings.

(Refer Slide Time: 10:30)

## Processing Character Strings

- C library functions for character string manipulation.
- It is required to include the following

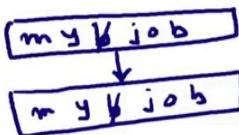
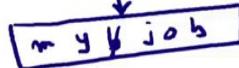
35



Now, for that we have got several c library functions we will soon come to functions and we have already seen different types of functions like square root, maybe did you see and we have seen we have seen the standard functions like scan f, print f all those things. Similarly we have got c libraries built in library functions, for character string manipulation how are they looking like? They are we have to in order to do that.

(Refer Slide Time: 11:21)

## Processing Character Strings

- C library functions for character string manipulation.
  - strcpy :: string copy A 
  - strlen :: string length B 
- It is required to include the following

35



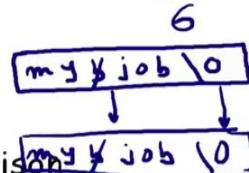
Say one is `s t r c p y`; that means, string copy; that means, if I have a string here say string here is my sorry let me put a blank here, my space job. Suppose these are string A and I want to copy it to another string B. So, B will also therefore, have my blank job.

So, that is that can be done by string copy function `s t r c p y` how can that be done? We also have a similar thing like string length `s t r l e n` which means that I have got a string say my job again, blank job and automatically there is a backslash 0 at the end.

(Refer Slide Time: 12:24)

## Processing Character Strings

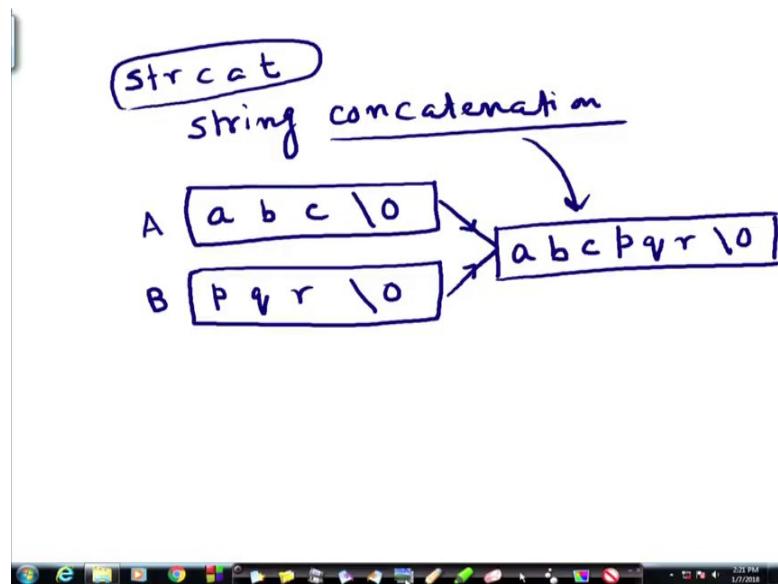
- C library functions for character string manipulation.
  - `strcpy` :: string copy
  - `strlen` :: string length
  - `strcmp` :: string comparison
  - `strcat` :: string concatenation
- It is required to include the following



So, when I copy it, then this will also be with `s t r c p y` we will have my blank job backslash n will come here sorry backslash 0 will come here, but `s t r l e n` means string length I will count how many elements are there in the string; can you tell me how many elements are there in the string? 1 2 3 blank is a valid character 1, 2, 3 4, 5, 6. So, the string length will be returned that will be returned is 6, similarly we have another function very popular function string comparison `s t r c m p`; that means, I have got two strings my job and here my job, and I compared them if sorry if they are the same then I will have a one ok.

So, that is another function the fourth function is `s t r c a t` what is meant by concatenation? Concatenation `s t r c a t` is actually `s t r c a t` means string concatenation.

(Refer Slide Time: 14:08)



That means, if I have a string here say a b c ended with this, and there is another string say p q r ended with this, when I concatenate it that these two strings what I get is a joining of these two strings one after another. When I say that string b is concatenated with a then my pattern will be a b c note that this backslash 0 will not come p q r and backslash 0. This is known as concatenation all right of two strings. So, that is being given by a function `strcat`.

Now. So, these are the very common functions for string operations. For our job if we need some other functions we can always write functions which will learn in a couple of lectures from now, but these are already available in the c library all right `stdio.h` and also when we use that square root function .

(Refer Slide Time: 15:51)

## Processing Character Strings

- C library functions for character string manipulation.
  - `strcpy` :: string copy
  - `strlen` :: string length
  - `strcmp` :: string comparison
  - `strcat` :: string concatenation
- It is required to include the following

```
#include <string.h>
```

35

And therefore, in order to do that we have to include st string dot h, if you recall we had said that if you recall we can include always you do that hash include.

(Refer Slide Time: 16:12)

```
#include <stdio.h>
#include <math.h>
    sqrt
#include <strings.h>
```

22/10/2018

Earlier we had included math dot h; that means, all the mathematic library of all the mathematical functions math dot h when we use for example, s q r square root.

So, this is this is a function that is already inbuilt, inbuilt in this math library similarly for strings if I use this `strcpy`, `strcat`, `strcmp`, `cmp` compare, then I have to include `strings.h` in my function alright. So, here are some examples.

(Refer Slide Time: 17:26)

## strcpy()

- Works very much like a string assignment operator.

`strcpy (string1, string2);`

- Assigns the contents of `string2` to `string1`.

- Examples:

`strcpy (city, "Calcutta");`

`strcpy (city, mycity);`

- Warning:

- Assignment operator does not work for strings.

`city = "Calcutta"; → INVALID`

36

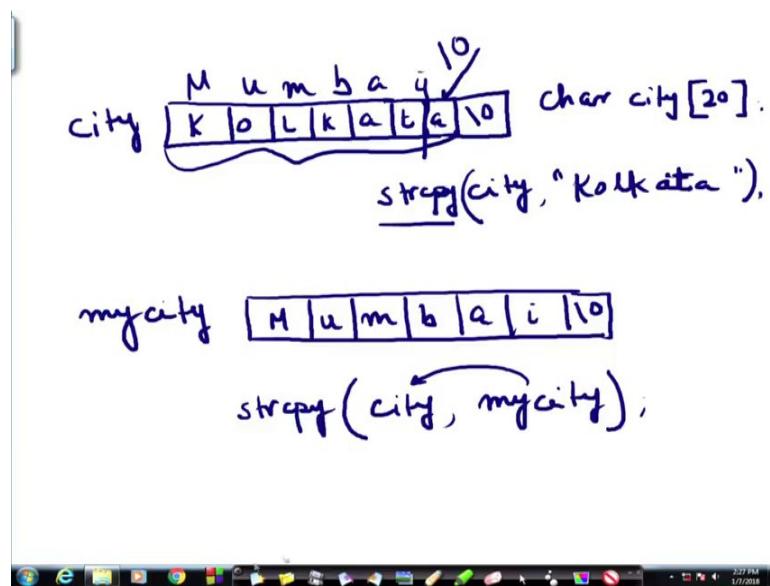


Very much like a string assignment operator string copy just like an assignment, for example, we have we are saying string copy string 1 string 2.

So, both string 2 will be copied in string 1. So, assigns the content of string 2 to string 1 assigns the. So, this is the source this is the destination. So, it is copied from here to here say string copy city Calcutta what will happen here; that means, I am trying to copy city is a string that is already defined alright I have defined city and I am copying this string now this is a string constant, this part is a string constant this is a string constant.

So, what I am doing is I am I have got a variable city, variable of type string.

(Refer Slide Time: 18:42)



Now this how did you define it to be a type of type string it was char city say something like this char city 20. So, 20 characters can come there.

Now, when I write city strc copy city to Calcutta or right now we say Kolkata say we do Kolkata. Then in this variable Kolkata will be loaded all right a followed by a backslash 0 right. On the other hand suppose city is here and suppose another city another string variable is there, which is maybe Mumbai and I copy string copy city my city; that means, what will happen this string, this string my city will come into this variable city.

So, this Kolkata will be overwritten and will be replaced by Mumbai. Now Kolkata is of length 1 2 3 4 5 6 7 and Mumbai is 1 2 3 4 5 6 in that case what will happen? As I copy this whole string over here then here Mumbai m will come u will come m b a i and this backslash 0 will come here and so, the entire string will be kept here the last sorry up to this and with a backslash 0 here, and the remaining part of unused part of Kolkata will be lost.

Now, warning there is an warning here that is assignment operator does not work for string. So, I could not have done string 1 assign string 2 or as we are doing here city assigned, my city that would not do that assignment operator will not work in the case of strings. So, city assigned Calcutta is invalid.

(Refer Slide Time: 22:08)

## strlen()

- Counts and returns the number of characters in a string.

```
len = strlen (string); /* Returns an integer
*/  
           strlen (city)  
  
M u m b a i \0  
len = 6
```

37

String length s t r l e n this is you I will ask you when we teach you function, to write a function for finding the string length although it is available in the standard c library. Counts and returns the number of characters in a string. So, len suppose len is a variable of type integer, len is of type integer len is of type integer and we say s t r l e n string.

So, string is some string all right some variable name s t r l e n city. So, city is a variable and whatever is the n suppose city is Mumbai, it will find out the length of the characters in this string and that will come in to len. So, len here we will get the value 6 when I do this function on this on the string.

(Refer Slide Time: 23:25)

## strlen()

- Counts and returns the number of characters in a string.

```
len = strlen (string); /* Returns an integer */
```

- The null character ('\0') at the end is not counted.
- Counting ends at the first null character.

37

The null character at the end as I said is not counted because that tells you that you need not count any further; counting ends with the first null character.

(Refer Slide Time: 23:41)

```
char city[15];
int n;
:
:
strcpy (city, "Calcutta");
n = strlen (city);
```

n is assigned 8

38

So, if I do Calcutta if I copy Calcutta the string, constant calculative city and I find out the length of city it will be 1 2 3 4 5 6 7 8 and backslash 0 will be left out. So, n is assigned 8 writing the string length.

(Refer Slide Time: 24:03)

## Writing String length....

```
int strlen(char str[])
{
    int len = 0;

    while (str[len] != '\0')
        len++;

    return (len);
}
```

can pass an array or pointer

Check for terminator

- Provided by standard C library: <iostream>

So, one problem that can be given to you that I am not talking about this function part right now.

Suppose I am trying to find out the string length of a character string I am trying to find how do you find out string length I mean its writing a program that is finding the string length you need not bother about the structure of this as yet, but let us look at the algorithm purely what is being done? Len some variable has been put to 0 and then while I am not encountering now s t r is an array of course, array of characters. As you can see its an idea characters as long as that is not equal to backslash 0; that means, not the end of the string I am going to going in going on incrementing this len, len was 0 len becomes one like that and then we return len after I completed then I return len .

Now, this is provided in the of course, I have to include the string dot h, but this thing this is what I am writing if, but actually this is a already available. So, I need not write it, I need to simply include hash include I call it hash include strings dot h that library. There similar program is already written and when I write strlen that is the program that is activated.

String compare is comparing two strings.

(Refer Slide Time: 25:53)

## strcmp()

- Compares two character strings.  
`int strcmp (string1, string2);`
  - Compares the two strings and **returns 0 if they are identical**; non-zero otherwise.
- Examples:  

```
if (strcmp (city, "Delhi") == 0)
{ ..... }
```



40

So, we are comparing two strings and returns 0 if they are identical what I just now said was just the opposite. If they are matching then it returns 0 and if they do not match a little non intuitive. So, keep it in mind that if the two strings match, then we will return a 0 otherwise will return a non-zero.

So, example is here if I compare city with the string Delhi is 0 then I do something. What does it mean city is a string already, suppose that is again Chennai suppose that is Chennai and I am comparing. So, city is a variable which has got this value, and I am comparing with Delhi of course, now they are not matching. So, it will not it will return non zero. But if the city was Delhi then these two have matched and I would have got a one here right.

Now; obviously, you can also just think and decide how this algorithm can be written that is 00730, simple now you have learnt all the tidbits of writing such a program.

(Refer Slide Time: 27:22)

## strcmp()

- Compares two character strings.

`int strcmp (string1, string2);`

— Compares the two strings and **returns 0 if they are identical**; non-zero otherwise.

- Examples:

```
if (strcmp (city, "Delhi") == 0)  
{ ..... }  
if (strcmp (city1, city2) != 0)  
{ ..... }
```

40



So, if this is not equal to 0 then then we do this. So, similarly I can do city 1 city 2 now.

(Refer Slide Time: 27:31)

## strcat()

- Joins or concatenates two strings together.

`strcat (string1, string2);`

— **string2** is appended to the end of **string1**.

— The null character at the end of **string1** is removed, and **string2** is joined at that point.

- Example:

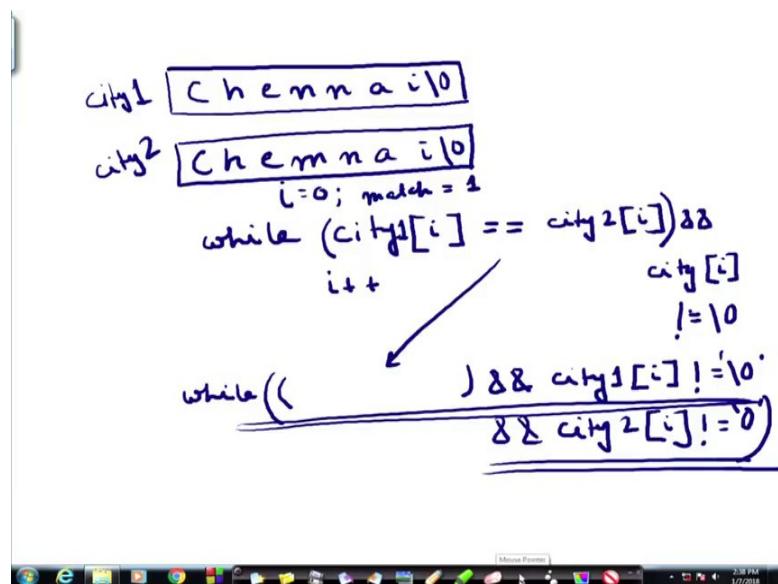


41



So, before that you can simply think of how the algorithm will look like in case of this suppose I have got two strings.

(Refer Slide Time: 27:40)



One is say Chennai and the other string is of course, backslash 0 here instead of n there is a name here.

So, how will you do that? So, you will compare these two character by character. So, this is suppose city 1 and this is city 2. So, you can very easily compare while city 1 i of course, i has been assigned to 0 here is same as city 2 i what shall we do? We will go on incrementing I plus plus, but and if it is not equal we will come out.

So, this here I compare here I compare I come here and here I find that city 3 here, city 1 3 and city 2 3 are not the same. So, I will come out, but what happens if they are same. So, I need to put in if they are same I how long shall I go on no mismatch is there suppose this is also n while city is city 1 is not equal to city 2, while city 1 is equal to city 2 do I need another condition here yes and city i is not equal to backslash 0.

So, are you understanding this, while this condition which I have already written there. That should be true in order to proceed in order to proceed further for further checking, I should they are matching. So, I should proceed and, but and also the fact should be that none of these two cities names. So, two strings have reached the end is not equal to (Refer Time: 30:33) this and city 2 i if any one of them has matched has reached the end then my while condition will be violated. So, I will not continue any further.

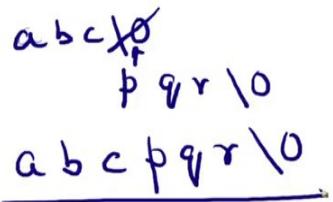
So, I can keep a flag here that what should I say match is one. So, if I go on doing this match will be remaining 1, whenever I come out of this loop I will make match 0 that means, it has there has been a match because the convention is that if they are matching then it should be 0. So, you please also try to write this function right this program right we will ask you to make it a function a couple of lectures later. I hope this is clear this part please try to understand this condition try to understand this condition all right.

Next we have got the last one that is a s t r c a t. that is rather simple you will also be able to write the program for that it is two strings are just being kind of concatenated, but one thing that you must remember appending concatenating means joining, while appending means at adding one at the end of the other. So, when I write this a b c, p q r then p q r has been appended to a b c .

So, string 2 is appended to string one; that means, it is joined at the end of string 1. So, the null character at the end of string 1 is removed and string 2 is joined from that point.

(Refer Slide Time: 32:48)

## strcat()

- Joins or concatenates two strings together.  
**strcat (string1, string2);**
  - **string2** is appended to the end of **string1**.
  - The null character at the end of **string1** is removed, and **string2** is joined at that point.
- Example:  


as we said that there can be a b c backslash 0 and then when I append p q r to that, then p q r will replace this and backslash 0 will come here. So, ultimately we will have a b c, p q r backslash 0 all right.

(Refer Slide Time: 33:20)

## strcat()

- Joins or concatenates two strings together.

**strcat (string1, string2);**

- **string2** is appended to the end of **string1**.
- The null character at the end of **string1** is removed, and **string2** is joined at that point.

- Example:

strcpy (name1, "Amit "); —→ A m i t \0  
strcpy (name2, " Roy"); —→ R o y \0  
strcat (name1, name2); —→ A m i t R o y \0

So, string copy suppose I have name one a string assigned by string copy Amit; A m i t. So, it looks like this a m i t backslash here there is a backslash 0 and there is a blank here you see the string is not a m i t a m i t blank and another string is name two which is R o y, then when I append them concatenate them there it will be a m i t blank this blank and then R o y and this this blank has been replaced by r. So, r has gone here and we will get this.

So, this is string concatenation this will often come handy when you type in some character strings or type and lines or compute using some text strings text English sentence has been given and you are trying to find out where the verb is and all those then you need a lot of string operations like this.

(Refer Slide Time: 34:33)

### Example

```
/* Read a line of text and count the number of uppercase letters */
#include <stdio.h> ←
#include <string.h> ←
main()
{
    char line[81]; ←
    int i, n, count=0;
    printf("Input the line \n");
    scanf ("%[^\\n]", line);
    n = strlen (line); ←
    for (i=0; i<n; i++)
    {
        if (isupper (line[i]))
            count++;
    }
    printf ("\n The number of uppercase letters in string %s is %d",
           line, count);
}
```

42



So, here is an example here we are reading a line of text and counting the number of uppercase letters how many uppercase letters are there. So, what are the things we are including here? s t d i o dot h and old friend is here, string dot h is also there then look at this function I have defined a line to be of length 81 big one 81.

Now, I have got the variables I and current count is equal to 0, I am asking the user to input the line I am reading the line using this format which we just discussed; that means, anything can come here I mean anything can come here like a b c slash, dash alright percentage p whatever is coming except backslash n its coming here and then I am finding the length of the line.

Suppose the line that was actually typed in in this way through this scan f is say apple blank is red.

(Refer Slide Time: 35:52)

### Example

```
/* Read a line of text and count the number of uppercase letters */
#include <stdio.h>
#include <string.h>
main()
{
    char line[81];
    int i, n, count=0;
    printf("Input the line \n");
    scanf ("%[^\\n]", line);
    n = strlen (line);
    for (i=0; i<n; i++)
    {
        if (isupper (line[i]))
            count++;
    }
    printf ("\n The number of uppercase letters in string %s is %d",
           line, count);
}
```

All right and then there is backslash here now this string length s t r l e n will find out the length of the string 1 2 3 4 5 6 7 8 9 10 11 12. So, 12 is n is becoming 12.

So, for i equal to 1 to n, now here is another new function that we are finding is upper if the character is upper case letter, is upper if the character that is being red that that is being passed here I put in some character and if the character is an upper case letter then I will count that. So, suppose here I make a little change I say this is I say that this is capital and I say that this is capital.

So, what is happening is it is reading character by character from here I 0 onwards, and checking whether this character is an upper character if that is so, count is becoming 1. So, count becomes one as a is upper case letter, then we go on in this loop i is being incremented until it comes to 12 less than 12. So, here is another one. So, I will get the count to be 2, it goes on and here I will get another one. So, its free print f the number of uppercase letters is percentage d is now look at this. The number of uppercase letters in string percentages s is percentage d. So, line will be printed as a line apple is red the number of uppercase letters in apple is red is 3.

So, here this program demonstrates a couple of things.

(Refer Slide Time: 38:14)

### Example

```
/* Read a line of text and count the number of uppercase letters */
#include <stdio.h>
#include <string.h>
main()
{
    char line[81];
    int i, n, count=0;
    printf("Input the line \n");
    scanf ("%[^\\n]", line);
    n = strlen (line);
    for (i=0; i<n; i++)
    {
        if (isupper (line[i]))
            count++;
    }
    printf ("\n The number of uppercase letters in string %s is %d",
           line, count);
}
```

is upper  
upper

One is first of all this is a new thing that you have learned is upper, I am just writing it separately. So, that it is clear, but in actual c library the function is written in the form without this gap or without any special character in between, this is the first thing that we have learned. And how we have already seen it is an application of what we learnt how we can read a line using this wildcard format. And then we found out what is the length of the line by our newly learned function `strlen`; and using that we have this is this looping we already know we have practiced it so often.

So, we now using this value using this for loop for so, many iterations. So, many repetitions that is determined by the value of `n` will check the entire string. So, this is an example of applying the string function in string operations, will go further with some more examples later.

**Problem Solving through Programming in C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 33**  
**String Operations**

We have looked at arrays and their utility, we have in particular looked at one dimensional array and today we will introduce another variety of array, which is 2-dimensional array. In fact, we can have multi-dimensional arrays in general arrays can be; we can have in n dimensional array.

(Refer Slide Time: 00:35).

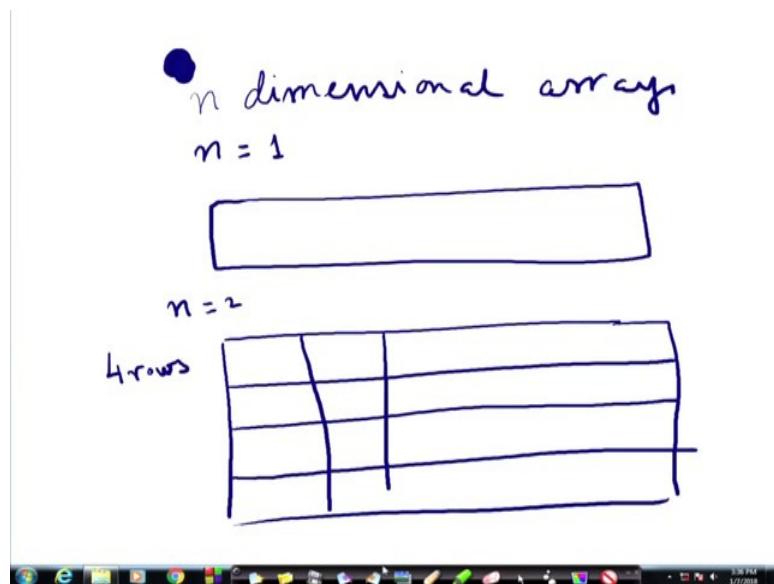
**Two Dimensional Arrays**

- We have seen that an array variable can store a list of values.
- Many applications require us to store a **table** of values.

	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5
Student 1	75	82	90	65	76
Student 2	68	75	80	70	72
Student 3	88	74	85	76	80
Student 4	50	65	68	40	70

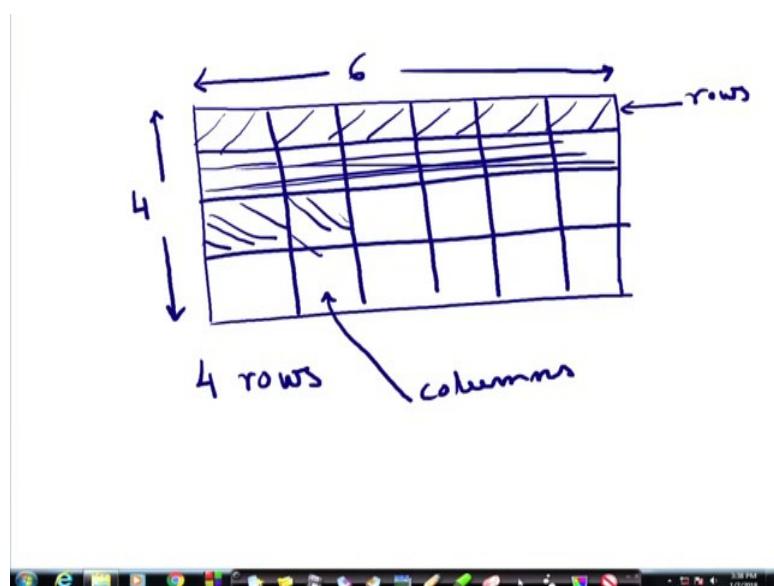
75

(Refer Slide Time: 00:51)



For a particular value of n equal to 1, we are having this one-dimensional array, right? Where, I have got some say 1 dimensional array of integers. So, for n equal to 2 we can have 2 dimensional arrays; that means, where there will length and there will be a breadth of the array. So, that array can be considered of consisting of a number of 1 dimensional arrays for example, as I am showing in this diagram I have got 2 dimensions on this dimension, I have got 4 rows on and on this dimension, I am sorry, this let me draw it a fresh.

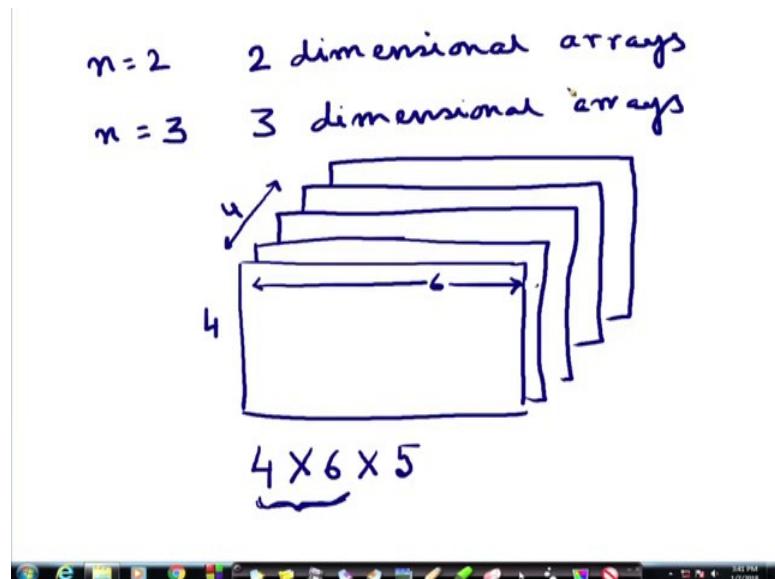
(Refer Slide Time: 01:54)



So, on this side I have got 4 rows and, on this side, I have got 1 2 3 4 5 6 positions. So, these are called columns and these are called rows, right? So, here I have got 2 dimensions one is the length, that is 6 or 6 columns and the breadth which is 4 rows since, these 2 dimensions. So, it is a 2-dimensional array.

So, a 2-dimensional array like this can be considered to be consisting of see in this case 4 1 dimensional arrays arrange one after another. So, this is one array, which we learnt till now this is another array this part is another array this part is another array. So, we have got 4 different arrays 1 2 3 4, 4 different arrays arranged one after another.

(Refer Slide Time: 03:54)



So, I can have for  $n$  equal to 2, I have got rows and columns. So, we get 2 dimensional arrays. Similarly, for  $n$  equal to 3 we can have 3 dimensional arrays, like so I have got one 2 dimensional arrays like this, and we can think of another 2 dimensional array line here, another 2 dimensional array line here, in that way I can have an arrangement of say may be 3 or 4, 2 dimensional arrays one after another, that is also an arrangement each of these may be 4 rows and 6 columns, sorry may be like this 4 rows and 6 columns.

So, this row this 3-dimensional array will be 4 rows 6 columns and 4 such, 2 dimensional arrays this is one 2-dimensional array this another 2-dimensional array, I am and this one 2-dimensional array and we have got 4 such 2-dimensional arrays arranged one after another, right? If I had a one more here, if I had one more here, same array then this would be changed to 5, in that way I can extend my concept of arranging data as we had

done for 1 dimensional array, we are extending it to 2 dimensions 3 dimensions in that way, it be difficult to concept I mean show it on the this screen we can go up to n.

So, n dimensional arrays, which will have different dimensions all together, but today we will be just discussing about 2 dimensional arrays and the thing is quite general and can be extended to other dimensional arrays also

So, we are now, talking about 2 dimensional arrays you can see the most common example of 2-dimensional array is a table means an arrangement of data just like this. This is list arrangement of information is known as a table, often we see table in other forms also for example, I can have a table of something like this.

(Refer Slide Time: 07:02)

A hand-drawn table is shown on a computer desktop. The table has 4 columns labeled 'name', 'age', 'school', and 'address'. It has 4 rows numbered 1, 2, 3, and 4. Below the table, handwritten text reads 'Table organized in a tabular form'. The computer taskbar at the bottom shows various icons and the date 1/7/2018.

	name	age	school	address
1				
2				
3				
4				

Table  
organized in a tabular  
form

So, the first here I put name of the boy, name of a student then I store age of a student, school to which the student goes and address of the student may be, and I have got such scope of keeping it for say 4 students one student number 1, student number 2, student number 3, student number of 4 and I go on. So, this is this is known as a table because, I am organising them in a organised in a tabular form.

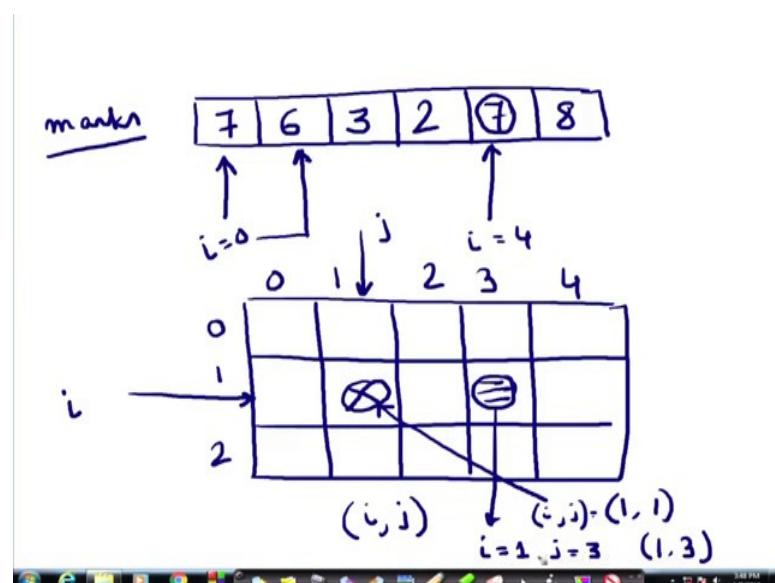
So, this is a table. So, what we were looking at now, is another table where here I am showing that, there are 4 students student 1, student 2, student 3, student 4 for each of the students I have got some marks in the different subjects, subject 1, subject 2, subject 3, subject 4 and subject 5. So, I can consider that all the marks of student 1 in the different

subjects are stored in a row, this row for the first student similarly, this row for the 2nd student, right? Similarly, this row for the 3rd student and this row for the 4th student.

So, each row wise I get the information of all students and column wise if I look in this direction, what do I get in subject 1 what are the different marks? And who have got what marks? And most interesting part is the intersection say for example, this element of the array what will this element tell me, this element has is identified by its row and column, which row it is in and which column it is in, this element is 75 when what is the meaning of this 75 is the student 2's sub marks of subject 2, it is in whatever we tell you can tell, that is the subject to marks of student 2, this is the subject 3 marks of student 3.

So, in that way, I can represent any of these as identified by some row, name row number and column number recall that, in the case of 1 dimensional array.

(Refer Slide Time: 10:25)



When we are talking of one dimensional array there we had indices index, right? So, that index was telling us say for example, there are suppose these are marks again out of 10 alright, and I wanted to know what is the marks in subject 1. So, the  $i$  value  $i$  equal to 0 was pointing me to this, suppose this is marks  $i$  value was showing this, if I take the  $i$  value 1, then I get the marks of this subject like that now, in that case in the case of 1 if I wanted to have this element, that was identified by  $i$  value equal to 0 1 2 3 4, but in the case of a 2 dimensional array just as we are seeing here, we will be getting to a particular

element again in the same way with index, but here say for example, this element this element here.

How do I identify this element? This element will be identified by this row and this column. So, my index while this was for one dimensional array the index was i, in this case the index will be i and j. So, any pair i j pair will identify which row and which column. So, this particular element is determined by the i value again just because, it is see I am saying that, it is starting from 0 1 2 and here 0 1 2 3 4. So, say this element can be identified by i is equal to 1 1, right? For example, this element this element is identified by sorry, index value i j I am sorry here, the index is 1 1.

So, the i j pair a pair is 1 1 for this i should be 1 and j should be 3. So, this pair is identified by 1, 3 whenever I get. So, suppose here this array was marks, similarly I will have an a name for this 2-dimensional array and I can access any element by this indices, but in this case since it is a 2-dimensional array, I have got 2 pointers 2 parts of this row part and the column part. So, here you can see that, the subjects are organised in this way subject marks are organised in this way and we can access any of this elements using the proper indexes student number and the subject number.

(Refer Slide Time: 14:07)

### Contd.

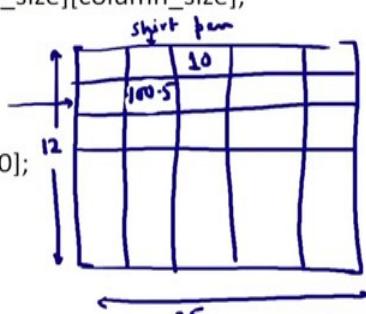
- The table contains a total of 20 values, five in each line.
  - The table can be regarded as a **matrix** consisting of **four rows** and **five columns**.
- C allows us to define such tables of items by using **two-dimensional** arrays.

In the table that was shown here, a showing how many elements 1 2 3 4 5 into 420 elements, right? So, the table contains 20 values 5 in each line and it can be arranged organises as a matrix 2-dimensional matrix consisting of 4 rows and 5 columns.

(Refer Slide Time: 14:40)

## Declaring 2-D Arrays

- General form:  
`type array_name [row_size][column_size];`
- Examples:  
`int marks[4][5];`  
`float sales[12][25];`  
`double matrix[100][100];`



Now, C allows us to define such tables by using 2-dimensional array what does it mean, the general form is like this type, array name, row size, column size. Always you compare with what we learned for 1 dimensional array is just an extension of that, for example, marks 4 5. So, in this case which we saw just now, it is an array with and let us call this say, the name of this array is it also marks.

(Refer Slide Time: 15:12)

## Two Dimensional Arrays

- We have seen that an array variable can store a list of values.
- Many applications require us to store a **table** of values.

	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5	marks
Student 1	75	82	90	65	76	
Student 2	68	75	80	70	72	
Student 3	88	74	85	76	80	
Student 4	50	65	68	40	70	

`marks [4] [5];`

And marks has got. So, let me write here, because it is marks and it has got 4 rows and it has got 5 columns. So, that is how I can describe this array, and then we can go on filling

up the values of this array. So, array name, row size and column size. So, for example, int marks, similarly I can have cells of 25 items over 12 marks may be say, I can have something like, so let us try to see what this possibly can mean is that suppose I have got a table where, there are 12 rows alright, I will have 12 rows here and there will be 25 columns here alright, there will be 300 items so I am not showing that.

But say each of these rows may mean say this is January row, this is a February row, this is a March row etc, and suppose here there are different items or shop has got different items and in that way it has got 25 different items and here. We write how many units of a particular this row is shirts alright, this one says that, in the month of February how many shirts have been sold and may be this says in the month of say this is pen in the month of January or in the month of January how many pens have been sold?

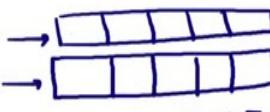
So, in that way this entire matrix or this array can be named as the arrays cells, which will have 12 rows and 25 columns, similarly and another very important thing to note is this type, this type that we declared also in the case of 1 dimensional array is also applicable here, because each of these elements in this array will be of the same type. So, here cells I am saying and I have said cells is float.

So, the way I wrote it is not correct I should write it as say, how many elements are there? What it can be that in the month of January? What is the worth of not how many shirts were sold? But, shirts worth how many rupees were sold. So, it can be here 100.5 say because, each of these elements have floating point numbers, alright? Similarly, matrix can be a 100 by 100 matrix where, each of the elements will there double precision number. So, we have got 2 places, 2 dimensions that we are specifying here.

(Refer Slide Time: 19:03).

## Declaring 2-D Arrays

- General form:  
`type array_name [row_size][column_size];`
- Examples:  
`int marks[4][5];`  
`float sales[12][25];`  
`double matrix[100][100];`

*int marks [5]*  
→   
→   
*int marks [2][5],*

Unlike what we did in the case of a 1-dimensional array where, we did something like int marks say 5. So, int marks 5 is a 1-dimensional array with 5 elements, I can say that these are with 5 columns of this 1-dimensional array, which is 1 row had there been more rows like this, then immediately I have to identify which row and which column and if there be 2 such rows, then this array should be described as int marks say here, 2 5 because, they are rows.

(Refer Slide Time: 20:01)

## Accessing Elements of a 2-D Array

- Similar to that for 1-D array, but use **two indices**.
  - First indicates **row**, second indicates **column**.
  - Both the indices should be expressions which evaluate to **integer values**.
- Examples:

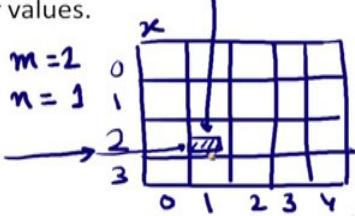
*marks [2][5]*  
↑ ↑

Similar to that of 1 dimensional array, but we use 2 indices, indices is the plural of index indices, right? So, the first one indicates the row and the second one indicates the column both the indices should be expressions which evaluate to integer values and that was the case for 1 dimensional arrays also. So, if I have something like marks 2 5 these integer, these values must be integers, I can write expressions we will show there later, but those expressions must evaluate to integer values, alright?

(Refer Slide Time: 21:14)

## Accessing Elements of a 2-D Array

- Similar to that for 1-D array, but use **two indices**.
  - First indicates **row**, second indicates **column**.
  - Both the indices should be expressions which evaluate to integer values.
- Examples:  
 $x[m][n] = 0;$



The diagram shows a 4x5 grid with the letter 'x' in the top-left corner. The rows are labeled on the left with 0, 1, 2, 3 (from bottom to top). The columns are labeled at the bottom with 0, 1, 2, 3, 4 (from left to right). A blue arrow points to the element at row 2, column 1. This element is highlighted with a yellow box. To the left of the grid, there are two equations:  $m = 2$  and  $n = 1$ .

Examples are  $x[m][n] = 0$ . So, whatever  $m$  value is  $m$  has to be an integer, suppose  $m$  here is 2 and  $n$  is 1, then in this array whose name is  $x$  we have got a number of rows and a number of columns, which element am I referring to here, I am referring to row to 0 1 2 3. So, I am referring to this row and which column I am referring to 1. So, this was 0 1 2 3 4. So, I am referring to this column therefore, I am actually pointing at this particular element, alright? This particular element.

(Refer Slide Time: 22:25).

## Accessing Elements of a 2-D Array

- Similar to that for 1-D array, but use **two indices**.
  - First indicates **row**, second indicates **column**.
  - Both the indices should be expressions which evaluate to integer values.
- Examples:
  - $x[m][n] = 0;$
  - $c[i][k] += a[i][j] * b[j][k];$

3:59 PM  
1/7/2018

Similarly, I can do operations like this what does this mean let us try to understand this means, I have I am referring to 3 different 2-dimensional arrays one is a, which has got some rows and columns and these another array this is a this is b and I have got another array c..

Now, I am taking what is being done here, this means  $c[i][k]$ . So, this  $c$  is being designated by an index  $i$ , another index  $k$  this one is being done by referred by  $i$  and some  $j$  this 1 by  $j$  and  $k$  what is being done here, that for different values of  $i$   $j$  and  $k$ , I am selecting this has got a special application, but I am not going to that I am looking at taking at the taking a particular value from here, that is  $c[i][k]$  is pointing here.

So, this one I am taking and adding that, with the product of  $a[i][j]$ ,  $j$  might be pointing somewhere here, I am taking this element and I am taking may be  $b[j]$  for  $b$  array  $j$  row I am sorry,  $j$  row  $j$  should be here,  $j$  row and  $k$  column. So, I am taking another element from here. So, I am taking this element multiplied by this element here, and adding that with this element that is taken from  $c$ , that is what I am doing. So, this simple example, but this has got for mode implication you will realise that soon.

(Refer Slide Time: 25:02)

## Accessing Elements of a 2-D Array

- Similar to that for 1-D array, but use **two indices**.
  - First indicates **row**, second indicates **column**.
  - Both the indices should be expressions which evaluate to integer values.
- Examples:
  - $x[m][n] = 0;$
  - $c[i][k] += a[i][j] * b[j][k];$
  - $b = \sqrt{a[j^3][k]};$

$j = 2 \quad k = 2$

$a[6][2]$

Here you see, what we have done here why is this example shown here, you see we are specifying the index in the form of an expression, suppose  $j$  was 2 and here, which one I am referring to a  $j$  into 2 so; that means, a 6 followed by whatever value of  $k$  was might be 6 2; that means, I am referring to the element of the array  $a$ ,  $a$  is an array whose 7th row and 3rd column element is being taken.

So, if this be like this I am taking this 7th row element and 2nd or 3rd column element, I am taking that value and finding this square root of that, and that is again being put to the corresponding value of that, is coming to some other variable name actually this should be since, this an array this should have been kept as some other variable in  $b$ , alright?

(Refer Slide Time: 26:19).

The slide has a black header bar. The main title is 'How is a 2-D array is stored in memory?' in bold black font. Below the title is a bulleted list:

- Starting from a given memory location, the elements are stored **row-wise** in consecutive memory locations.
  - x: starting address of the array in memory
  - c: number of columns
  - k: number of bytes allocated per array element
- $a[i][j] \rightarrow$  is allocated memory location at  
$$\text{address } x + (i * c + j) * k$$
  
a[0][0] a[0][1] a[0][2] a[0][3] a[1][0] a[1][1] a[1][2] a[1][3] a[2][0] a[2][1] a[2][2] a[2][3]

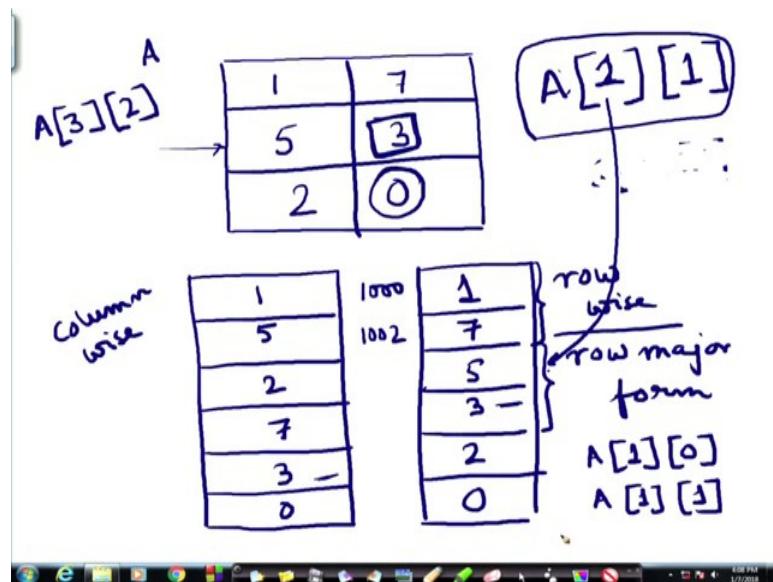
So, how is 2 dimensional arrays stored in memory we know that, a 1-dimensional array is stored in a memory row I mean in the contiguous memory locations.

(Refer Slide Time: 26:46)

The slide shows a horizontal array of four boxes containing the numbers 3, 5, 4, and 7. Below it is a vertical stack of four boxes, each containing one of the same numbers (3, 5, 4, 7), representing their memory locations. The windows taskbar is visible at the bottom.

So, just for the sake of recalling if we had, if I have a 1 dimensional array say 3 5 4 7, they were being mapped to the memory actual memory, here is the memory location the different memory locations and I am storing them here, 3 5 4 7 in contiguous locations, but what happens to the case of 2 dimensional array for example, if I have the array that I am drawing here, with 3 rows and 2 columns, alright?

(Refer Slide Time: 27:24)



So, what is the dimension of this array  $a$ ,  $a$  is 3 2 3 rows and 2 columns now, suppose it is 1 7 5 3 2 0 say those are the values in that, case if I stored it in the memory I can stored it now, I am drawing how it can be stored in the memory I need 1 2 3 4 5 6 locations in the memory. So, I will have 6 locations. So, I can store them 1 5 2 7 3 0 or I can store them as 1 7 5 3 2 0. Now, what is a difference between these 2 storages this one is I am storing them column wise and, in this case, I am storing them row wise row after row, right?

In C the compiler stores the arrays row wise, which is often known as in the row major form and this is the column major form, but in C we are doing it in row major form there is a specific significance of this. Because, whenever I want to access an element say a array, I want to find the particular element say 2 3, which element shall I find out here for example, here let us take this case and let me try to find out the element say  $A$  this is  $A$  capital  $A$  I want to find out the element 2 1; that means, I am actually trying to find out the row number 2 and element number 1. So, I am trying to find this one out. Now, where will it be if I do not know. So, say this is accidentally I came coming to the same let us say, let us make this 1 1 1; that means, which one row 1 column 1. So, I am actually trying to find this element 3.

Now, what is the address of this element, where is it stored you can see that 3 has been stored here if I store in the column major form, but if I store in the row major form it is

here. So, whenever I am trying to find this element out of this memory representation I will actually have to fetch it from the memory, right? So, therefore, I have to find out the address of this and for that, I need to know where it is stored row wise or column wise since, I know it is row wise.

So, how should I go about it since, there are 2 columns, so for the first row I will say that, if this been my starting address 1000 in the first row we will take 1000 and 1000 this is starting from 1002, alright? This is the first row, this is the second row, and this means the second-row part and second rows this is one what is this element 5, this is second row and 0th element and this one is second row and then the next element.

So, it is 1 1. So, using knowing that, whether it is stored in the row major form or column major form is very essential in order that, I can find out the address of the element in a 2-dimensional array, we will continue this further more explanations are required about this.

**Problem Solving through Programming in C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 34**  
**2-D Array Operation**

So, we were discussing that how a 2 dimensional array is stored in the memory.

(Refer Slide Time: 00:23)

**How is a 2-D array is stored in memory?**

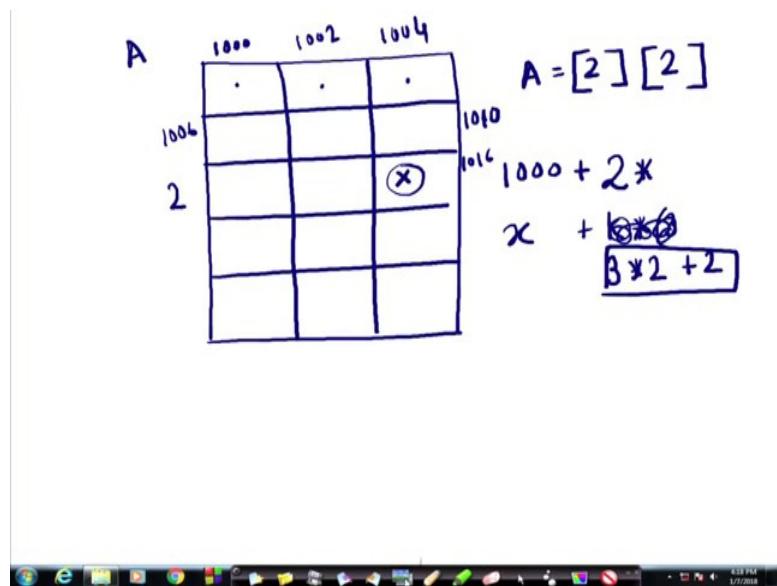
- Starting from a given memory location, the elements are stored **row-wise** in consecutive memory locations.
  - x: starting address of the array in memory
  - c: number of columns
  - k: number of bytes allocated per array element
  - $a[i][j]$  → is allocated memory location at  
address  $x + (i * c + j) * k$

Row 0	Row 1	Row 2
a[0][0] a[0][1] a[0][2] a[0][3]	a[1][0] a[1][1] a[1][2] a[1][3]	a[2][0] a[2][1] a[2][2] a[2][3]



Now, we have already said that that is and 2 dimensional array is stored in a row wise form. So, we have to find out the address of a particular element in the memory, right.

(Refer Slide Time: 00:47)



If we have a 2 dimensional array and the row and the column is specified all right. So, say here there are 1 2 3 4 5 rows and 3 columns say a particular element here has got the address say the name of this array is A this particular element has got the address A 0 1 2, A 2 3, A 2 0 1 2, A 2 2 right this element. So, it is stored how it is stored this stored in this consecutive way first this one is stored then this one is stored, then this one is stored row wise right.

So, if in my each of them have got a memory location, so if the location of this is 1000 and if it be an integer array then this location will be 1002, this location will be 1004, this location will be 1006, 1008, 1010, right, in that way it will go on. So, a little thought will tell you that if the starting address. So, what will be the starting address? What will be the address of this? It will be 1000 plus how many columns have to come for each column shift for each column. What will be the address of this? 1010, 1012, 1014, 1016, right, actually this should be in the location 1016 of the memory.

So, we start with 1000 and then since each of the elements each of the elements are taking 2 bytes. So, I will have to go for each shift 2 bytes and how many such shifts. What is the distance from here to here element wise? We have got 8 such positions right and so with that I can find out the address of this x. So, if this was x and this was element was k then what would be my address; k into the offset is so many columns 2 3, 2 columns sorry 2 means 3 columns per row and 3 such rows. So, it will be 6 rows. So, in

general we can say that if  $x$  be the starting address of the array in the memory and  $c$  be the number of columns and  $k$  be the number of bytes allocated then the address will be  $x$  plus  $i$  into  $c$  plus  $j$   $c$  is the column number plus  $j$ . How many columns are there? So, if I go back to this if I go back to this I can say the how many columns  $c$  is each it has got 3 columns. So,  $k$  into 3 plus sorry 3 into  $k$  plus  $j$  right 3 into  $k$  means 2 plus  $j$ ,  $j$  was 2.

So, that will be my result 1000 plus 6 plus two 8. So, that will be the location that I will be finding out. So, here that is the formulation;  $x$  plus  $I$  into  $c$  plus  $j$  times  $k$ ,  $k$  is number of bytes allocated per element plus please look at this. So, the rows are the matrix is actually stored as a 0 0, a 0 1, a 0 2. So, this entire thing is a 0 3, so 1 2 3 4 so there 4 columns in one row. Then, the second row a 1, a 1 column 0, a 1 column 1, a 1 column 2, a 1 column 3, then, I proceed in this way. So, this is the arrangement of 3 rows and each row having 4 columns and using this formula which is intuitive we can find out the exact location of a particular element.

So, this part is row 1 this part is sorry this part is row 0, this part is row 1, this part is row 2, you can see that this is row 1 column 0, row 1 column 1, row 1 column 2 so and so forth.

(Refer Slide Time: 06:05)

### How to read the elements of a 2-D array?

- By reading them one element at a time

```

for (i=0; i<nrow; i++)
  for (j=0; j<ncol; j++)
    scanf ("%f", &a[i][j]);
  
```

$\&a[1][1]$   
 $\&a[1][3]$

Now, given so that is how it is stored and how we can find out what the address is, but how do we read a particular array elements into a 2 dimensional array. We have seen that I can read one dimensional array the elements of a 1 dimensional array into the array by

repeatedly getting the character from the user and storing it in the proper indexes and proper positions by varying the index right. Here also we will be reading one element at a time and we will store them. So, my array is say the array that I am storing is a  $i$   $j$  having some rows and columns and  $i$  is the row index and  $j$  is the column index. So, initially in this for loop look here, there are there is a nested for loop here. So, first I am keeping the  $i$  fixed for  $i$  equal to 0. Now, I come here inside this, inside this there is a nesting of for loop say I have got 4 columns. So, for  $j$  is equal to 0 to  $j$  less than 4? What do I do? I scanf and a  $i$   $j$  now each element here each element here is identified by the row number and column number and so this one is and a 11, this one is and a 13 in that way we do.

So, now for  $j$  is equal to 0 I read the particular value and store it here suppose it is 5 next what is done, I am still inside this loop  $i$  increment  $j$ . So,  $j$  comes here note that  $i$  is fixed still how long will a  $i$  remain fixed as long as I am inside this inner loop and how long shall I be inside this inner loop till  $j$  reaches the end of the number of columns. So, next I read the other value. So, it is 2, next I read the other value it is 7, next I read another value may be it 0, then it comes to 4 comes to the end of the row because all the columns have been filled up then I come out of this loop and again go inside this. So,  $i$  is now incremented,  $i$  is now incremented to this one and I again do the same thing inside this loop. So, this inside this loop is filling up a row and this one is filling up all the rows.

(Refer Slide Time: 09:40)

### How to read the elements of a 2-D array?

- By reading them one element at a time

```

for (i=0; i<nrow; i++)
    for (j=0; j<ncol; j++)
        scanf ("%f", &a[i][j]);

```

Please take a little time to understand this. This loop, I am once again explaining this loop is filling up one row keeping i fixed, i is fixed and in a loop I am filling up the values here, here, here, here and then I increment this one. So, this part for filling up all the rows one by one, I come here again and again fill this up then I increment i and again fill this up again I increment this i, come here and fill this up all right that is how we read the elements of an array. So, that is being what is being shown here.

(Refer Slide Time: 10:32)

### How to read the elements of a 2-D array?

- By reading them one element at a time

```
for (i=0; i<nrow; i++)
    for (j=0; j<nrow; j++)
        scanf ("%f", &a[i][j]);
```
- The ampersand (&) is necessary.
- The elements can be entered all in one line or in different lines.

The ampersand is necessary here just as in the case of an array it was necessary it is necessary here as well. The elements can be entered all in one line or in different lines it really does not matter whether you type 5, blank 6, blank 7, blank 8 or you type 5 enter 6 enter like that that really does not matter because we already know that every element will be stored in a row wise fashion.

(Refer Slide Time: 11:08)

## How to print the elements of a 2-D array?

- By printing them one element at a time.

```
for (i=0; i<nrow; i++)  
    for (j=0; j<ncol; j++)  
        printf ("\n %f", a[i][j]);
```

a [0] [0] a [0] [1] a [0] [2] a [0] [3]  
a [1] [0] a [1] [1] a [1] [2] a [1] [3]  
a [2] [0] a [2] [1] a [2] [2] a [2] [3]  
a [3] [0] a [3] [1] a [3] [2] a [3] [3]

How to print the elements of an array? Again, suppose I have got an array, suppose I have got an array suppose I have got a 2 dimensional array and I will have to print the elements I cannot just print the array in one shot I cannot do that. So, again here I will be fixed and then I will print say this is a 0 1 2 3 0 1 2 3. So, first a 00 will be printed whatever was the value here I am printing it here, then j will be incremented. So, the next one that will be printed will be a 01, next j will be incremented. So, it will be a 02, a 03 so and so forth and then I will be incremented again in this way. So, it will be next printing will be a 1 and j will be reinitialised to 0 1, a 10, a 11 and like that it will be printed.

(Refer Slide Time: 12:44)

The screenshot shows a Windows desktop environment. In the center, there is a slide titled "How to print the elements of a 2-D array?". Below the title, there is a bulleted list: "• By printing them one element at a time." To the right of the list, there is some R code and its output. The code is:for (i=0; i<nrow; i++)  
 for (j=0; j<ncol; j++)  
 printf ("\n %f", a[i][j]);A handwritten note next to the code says: "– The elements are printed one per line." To the right of the note, the output is shown as a vertical list of numbers: 1, 2, 3, 5, 4, 6, 7, 9. Below this list, there is a vertical line of numbers from 1 to 9, each aligned under its corresponding number in the list above it. The desktop taskbar at the bottom shows various icons, and the system tray indicates the date and time as 1/1/2018 4:29 PM.

Now, the elements are since it is a back slash n you can see the elements have printed one per line because every time I am printing I am giving a back slash n. So, although the; my array could be 1 2 3 5 4 6 7 9, it will be printed as 1 2 3 5 4 6 7 9 because at every point I have given a back slash n.

(Refer Slide Time: 13:23)

The screenshot shows a Windows desktop environment. In the center, there is a slide titled "How to print the elements of a 2-D array?". Below the title, there is a bulleted list: "• By printing them one element at a time." To the right of the list, there is some R code and its output. The code is:for (i=0; i<nrow; i++)  
 for (j=0; j<ncol; j++)  
 printf ("\n %f", a[i][j]);A handwritten note next to the code says: "– The elements are printed one per line." Below this note, there is another set of R code and its output. The code is:for (i=0; i<nrow; i++)  
 for (j=0; j<ncol; j++)  
 printf ("%f", a[i][j]);A handwritten note next to this code says: "– The elements are all printed on the same line." To the right of the notes, the output is shown as a single horizontal row of numbers: 1, 2, 3, 4, 5, 6, 7, 8. Below this row, there is a horizontal line of numbers from 1 to 8, each aligned under its corresponding number in the row above it. The desktop taskbar at the bottom shows various icons, and the system tray indicates the date and time as 1/1/2018 4:29 PM.

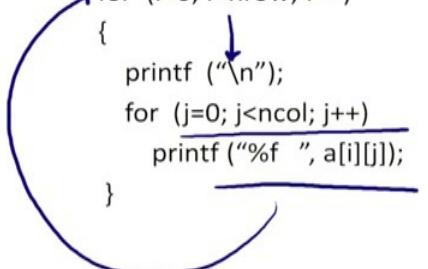
Now, here what are we doing here? In this case the elements will all be printed in the same line because I have not put the back slash n. So, it will be printed all the elements say sorry all the elements say 1 2 3 4 5 6 7 8, it will be printed in this way if that was the

matrix then this part will be printed in one line and then I go up there is no back slash n this one will be followed and the entire thing will be printed in one line, 1 2 3 4 5 6 7 8.

(Refer Slide Time: 14:14)

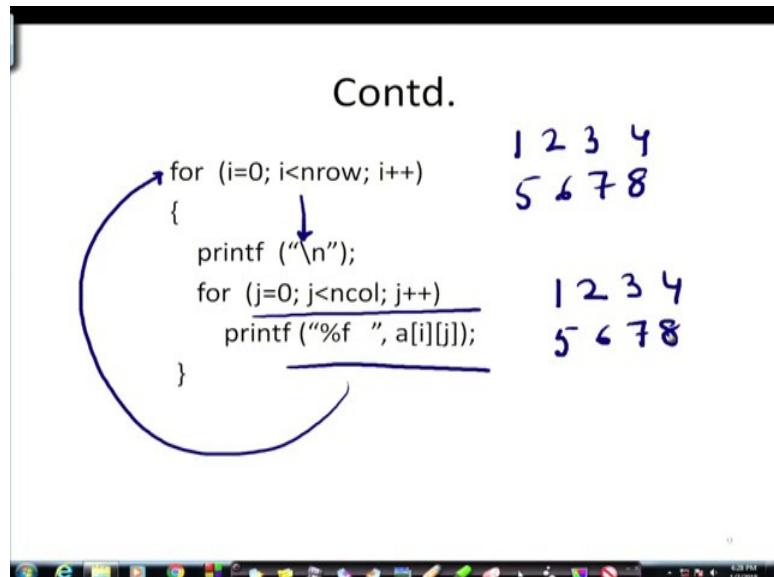
Contd.

```
for (i=0; i<nrow; i++)  
{  
    printf ("\n");  
    for (j=0; j<ncol; j++)  
        printf ("%f ", a[i][j]);  
}
```



$1 \ 2 \ 3 \ 4$   
 $5 \ 6 \ 7 \ 8$

$1 \ 2 \ 3 \ 4$   
 $5 \ 6 \ 7 \ 8$



But if I had done this that first I had this again 1 2 3 4 5 6 7 8 and I want it to be printed in this way because here I have not given any back slash n it will be printed one two 3 4 and as it comes out of this loop it goes here and comes to the next row first thing that it does is printf back slash n. So, the control will come here and then 5 6 7 8 will be printed therefore, this is a better way of printing a 2 dimensional array. So, that it looks like a 2 dimensional array.

(Refer Slide Time: 15:14)

Contd.

```
for (i=0; i<nrow; i++)  
{  
    printf ("\n");  
    for (j=0; j<ncol; j++)  
        printf ("%f ", a[i][j]);  
}  
— The elements are printed nicely in matrix form.  
• How to print two matrices side by side?
```

$N$        $M$   
 $\begin{matrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{matrix}$        $\begin{matrix} a & b \\ c & d \end{matrix}$   
 $\text{char } M [2][2]$   
 $\text{int } N [2][4]$

So, that this will make the elements now the question is you just think how can we print two matrices side by side. So, suppose there is one matrix 1 2 3 4 another is character matrix a b c d how can you print them side by side what should we do? Can you think of that? I mean if I say I have got two matrices one is 1 2 3 4 5 6 7 8. Another matrix is say a b c d. So, they have different dimensions.

So, first of all, first of all how can you represent such a matrix? Say this matrix how will you declare that simple this will be a character matrix character let us call this matrix let it be M, M is 2 by 2; that means, each elements of this matrix will be a character. And this is int, let it be N, 2 by 4, right. These are the two matrices now you can think of how you can print two matrices side by side. If I do look at this by this row by this I will first print 1 2 3 4 and then what should I do? I should go to here I should again for j is equal to j to M call might be j plus plus, I will be doing I will be printing M i j. So, this will come immediately side by side if necessary I could have given another blank in between and I could have printed this as a complete row next I go up and print this and this. So, I suggest that you try to write this program and run it and see whether you get a nice output or not.

(Refer Slide Time: 17:46)

Example: Matrix Addition

```
#include <stdio.h>
main()
{
    int a[100][100], b[100][100],
        c[100][100], p, q, m, n;
    scanf ("%d %d", &m, &n);
    for (p=0; p<m; p++)
        for (q=0; q<n; q++)
            scanf ("%d", &a[p][q]);
    for (p=0; p<m; p++)
        for (q=0; q<n; q++)
            scanf ("%d", &b[p][q]);
}
```

```
for (p=0; p<m; p++)
    for (q=0; q<n; q++)
        c[p][q] = a[p][q] + b[p][q];
    for (p=0; p<m; p++)
    {
        printf ("\n");
        for (q=0; q<n; q++)
            printf ("%f ", a[p][q]);
    }
}
```

10  
1/1/2018

Diagram illustrating matrix addition:

Matrix A (2x4):

1	2	3	4
7	6	9	2

Matrix B (2x4):

10	12	15	13
7	14	3	0

Resulting Matrix C (2x4):

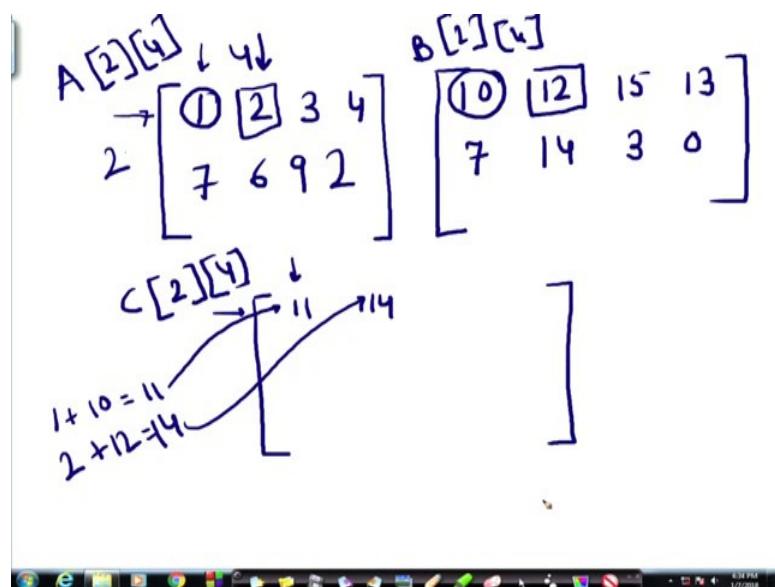
11	14	15	13

Handwritten annotations:

$1 + 10 = 11$   
 $2 + 12 = 14$

Now, we come to a very well known problem that we often encounter matrix addition. I am going to add two matrices. This is a very simple problem. So, all of you know how this can be done.

(Refer Slide Time: 18:04)



So, I have got a matrix of the same size two matrices. So, it is 1 2 3 4 7 6 9 2 and there is another matrix 10 12 15 13 7 14 3 0. Now, I want to add them and have another matrix. First of all this is a matrix of 2 rows and 4 columns 2 rows and 4 columns. So, we can say that this A is a 2 by 4 matrix and this one B is also a 2 by 4 matrix. Now, the sum I

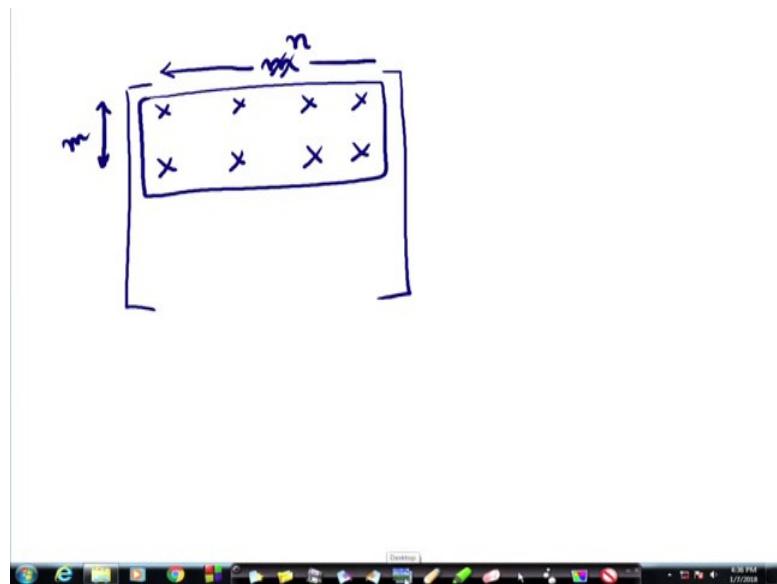
will store in another matrix whose dimension will also be two by 4 has to be. And how is matrix addition done? I will first take this element.

What is this element? I am saying this element what is this element. This element means a particular  $i j$  value 0 0 element  $i 0 j 0$  from these array. So, I take this element 1 and then take  $b i j$  of the same  $i j$  value 0 0 1 add them together 1 plus 10 I get 11, I put that 11 in the  $i j$  value remaining same 0 0 of the c matrix. So, here it is 11.

Next, I am increasing the index  $i$  here  $j$  remains sorry I am incrementing whatever I do I keep the  $i$  fixed and increment the  $j$ . So,  $i$  is 1,  $j$  is 1 is 0,  $j$  is 1. So, with that I take this value 2 and take this value with a same  $i$  and  $j$  value,  $i$  is 0 and  $j$  is 1 add them. So, that is 4teen and I store this value here in that way I go on for this entire matrix that is how I do the matrix addition. So, let us see how the code looks like for this and let us try to understand the code this much is include s t d i o dot h then my main program is continuing. What is being done?

Here I am trying to add 2, I am writing the program for adding 2 matrices of size 100 by 100 let us look at this 100 by 100, a matrix of size 100 rows and 100 columns,  $b$  of 100 rows and 100 columns and  $c$  of 100 rows and 100 columns. And I have got other integer values  $p q m n$ . I am reading the matrices, I am reading sorry I am reading  $m$  and  $n$  alright two values. Now, what is  $m$  and  $n$ ?  $m$  and  $n$  although I have here recall the difference between I mention about the size and the dimension is the maximum size the maximum size that it can have 100 and 100 are the maximum sizes of the  $a$  matrix and  $b$  matrix, but actually I can have I may not need so many locations. So, what I can do?

(Refer Slide Time: 22:01)



I have got a has got so much size, but actually I am filling up only these say this much. So, this is my m, I am sorry this is my m and this is my n ok.

So, my actual matrix is small dimension is a largest size that I can accommodate all right. So, I need that actual value m and n and then for p is an index p equals to 0 less than m p plus plus q. So, I am reading the matrix a, and here you see I am not using the conventional i j indices I am saying p is keeping account of my row and q is keeping account of my column. So, for p 0 to m and q 0 to n I am filling up the a matrix, then I come here fill up this matrix out of this outer loop.

This inner loop, with this inner loop what am I doing I am filling out particular row and the outer loop I am going to the next column, repeatedly I am saying that. So, in this way I read matrix a here, here I read matrix b then I am doing the addition here. Again my the dimensions of the two matrices will be the same when I add and the result matrix will also be the same according to matrix algebra, that is I am adding two matrices they will result in the same dimension matrix. So, again for p equal to 0 to m and q equal to 0 to m I take c p q. So, I have taken two matrices.

(Refer Slide Time: 2 4:13)

Example: Matrix Addition

```
#include <stdio.h>
main()
{
    int a[100][100], b[100][100],
        c[100][100], p, q, m, n;
    scanf ("%d %d", &m, &n);
    for (p=0; p<m; p++)
        for (q=0; q<n; q++)
            scanf ("%d", &a[p][q]);
    for (p=0; p<m; p++)
        for (q=0; q<n; q++)
            scanf ("%d", &b[p][q]);
    for (p=0; p<m; p++)
        for (q=0; q<n; q++)
            c[p][q] = a[p][q] + b[p][q];
}
for (p=0; p<m; p++)
    for (q=0; q<n; q++)
        printf ("%f", c[p][q]);
```

And now I am generating c matrix where this value this particular element is being computed by the corresponding element from a and the corresponding element from b being added I am filling it up and this is going on in a loop all right and so I fill this entire thing up. What is a last for loop? This last for loop is nothing, but printing the same array, printing the array in the proper way.

Now, here you please note that it is back slash n. So, it will come nicely after one row there will be a we will go to the new line and we will print it this way. So, this is a matrix addition which is a very useful very common and simple application of our knowledge of two dimensional array to start with.

We will other applications of this to in a future. Now, the other thing that will be discussing in the consequence lectures are next. So, we now have got an idea of 1 dimensional array and we have got an idea of 2 dimensional array. We have also seen strings and character arrays. Next, we will move to functions in the next lecture. We have already mentioned about the term functions while moving and in the course of other discussion, and we have already seen one particular function that is main function that is always there is a main body of any C program and we have also seen some library functions. But next lecture onwards we will look at user defined functions.

We have seen some stored functions in the earlier lecture like string copy, string length, finding string length, similarly a user can himself or herself write a particular function of

his or her own need. We will have to see why we need to write functions and how a function can be written from the next lecture onwards.

Thank you.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

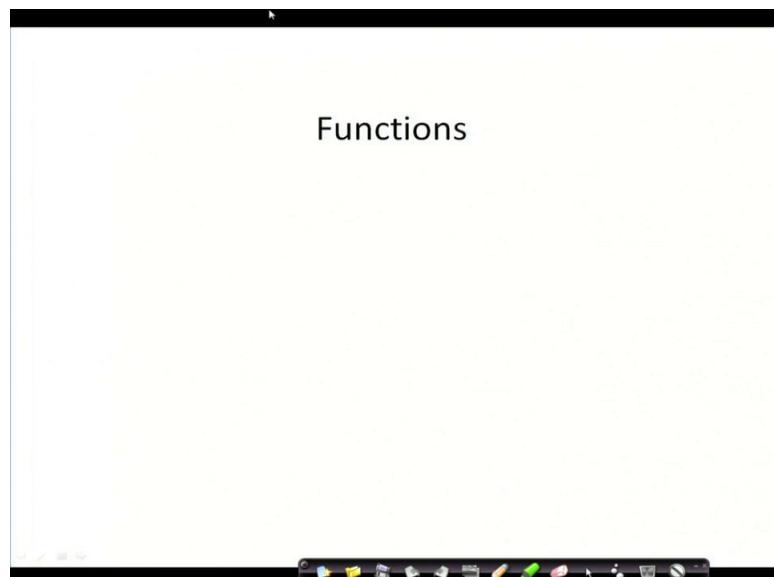
**Lecture – 35**  
**Introducing Functions**

In the earlier lectures, we had discussed about one dimensional array as well as 2 dimensional array, we have seen how one dimensional array can be stored, read and printed and as well as we saw how a linear search can be applied over one dimensional array.

Similarly we have seen for 2 dimensional arrays how it can be read, it can be stored, it is stored in row major form and how it can be printed. Now today we will start discussing on a very important and vital component of programming and also (Refer Time: 01:06) of the C language that is functions. The concept of functions is very general and we will have to first look at it from the general angle why it is required, what is a function?

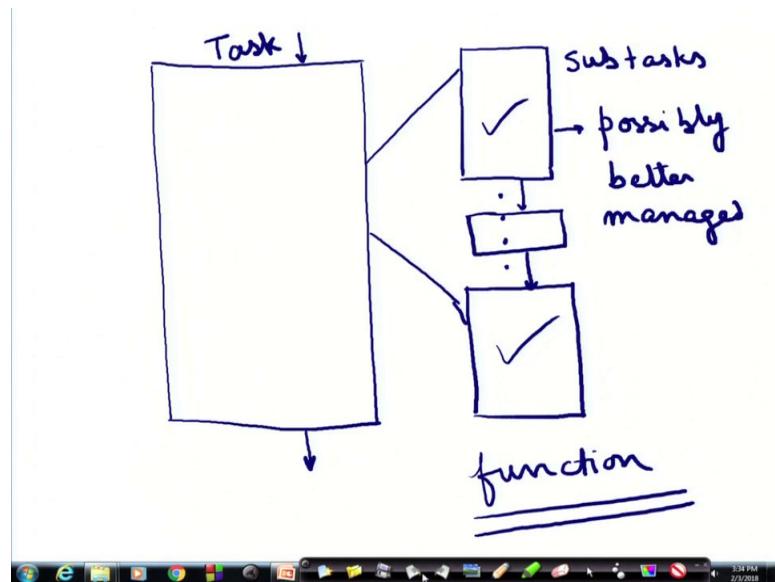
And then we can look at some of the details of this the implementation of this concept in the C language. So, first let us start with functions.

(Refer Slide Time: 01:39)



So, we will we will start to visualise a task.

(Refer Slide Time: 01:48)



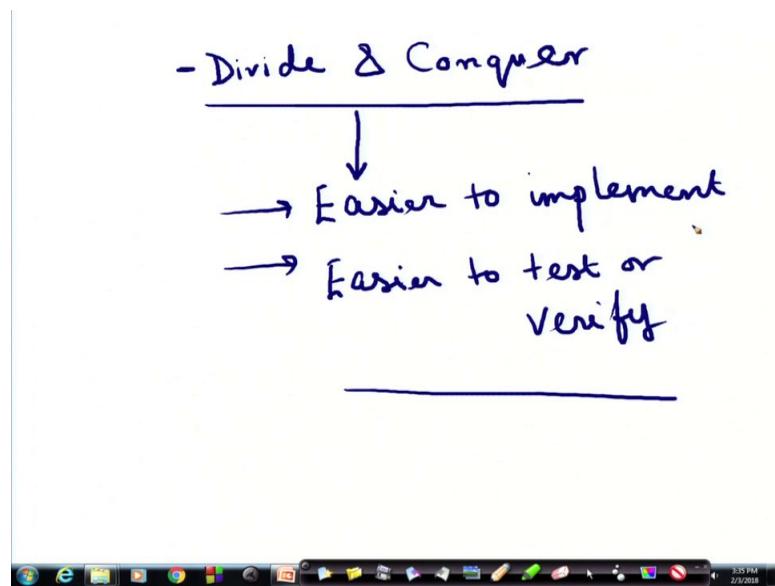
Suppose I have got a big task alright which I have to implement by programming. Now any task this is one task this task may be too complicated. And for any human programmer or any human being to solve this task from the beginning till end in one shot may be difficult. Often what we do we break up this task into a set of smaller sub tasks, different sub tasks? And each of this sub tasks are smaller enough, small enough. So, that we can manage it better each of these are possibly better managed, alright.

We can say if you think of a program segment a part of a program. If the task is broken big task is broken down into a small task a number of small tasks, then it is easier to write the program for each of these smaller tasks. And each of these tasks can be independently tested for it is correctness, whether it is working properly or not each of these can be separately tested.

And then if all these are connected together if all these are connected together one after another and then I can ultimately get this enter task done. So, instead of addressing the whole task into execute the whole task, or implement the whole task, as a single task it is often advisable to break it down into a number of subtasks.

Now let us talk about programming, now if we want to know what is a when we say the term function, if we take the English meaning of that function is means doing something. So, it is also a task therefore, this entire function can be broken down into smaller functions and each of this functions can be implemented independently.

(Refer Slide Time: 04:50)

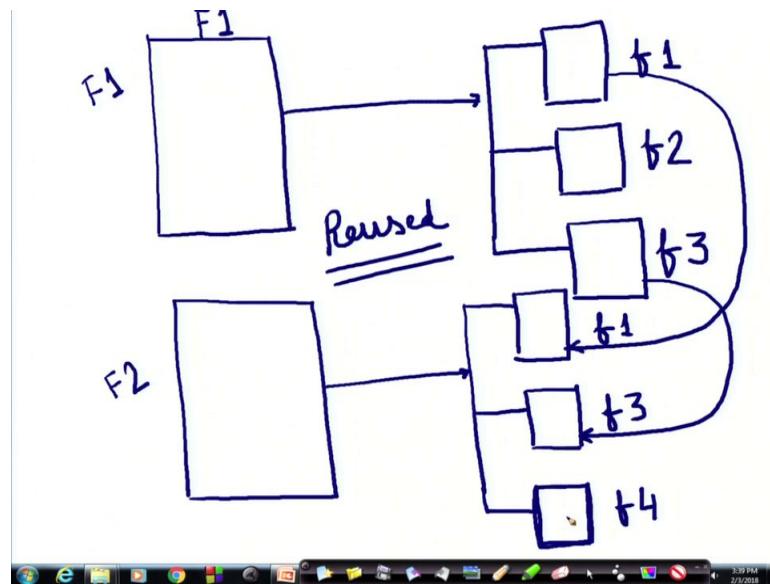


So, the advantage that we get by this means is that we can divide the problem into smaller parts and thereby we can conquer the small, we can conquer the smaller sub segments and in the process, we in the process we achieve the complete task that is the first thing. So, by this it is easier to implement and also easier to test or verify.

We can check whether each of these components are working correctly or not. Now when we find that all the components are working correctly and if they are connected together correctly, then we will have the entire problem solved correctly that is the first advantage of writing a complex program broken down into smaller functions.

Now the other advantage is suppose I had 2 implement one task.

(Refer Slide Time: 06:25)



And in order to implement this big task, I had to write a number of functions smaller functions. Say 3 smaller functions together implement this big function alright. So, let us call them  $f_1$  function 1,  $f_2$  for function 2 and  $f_3$  for function 3.

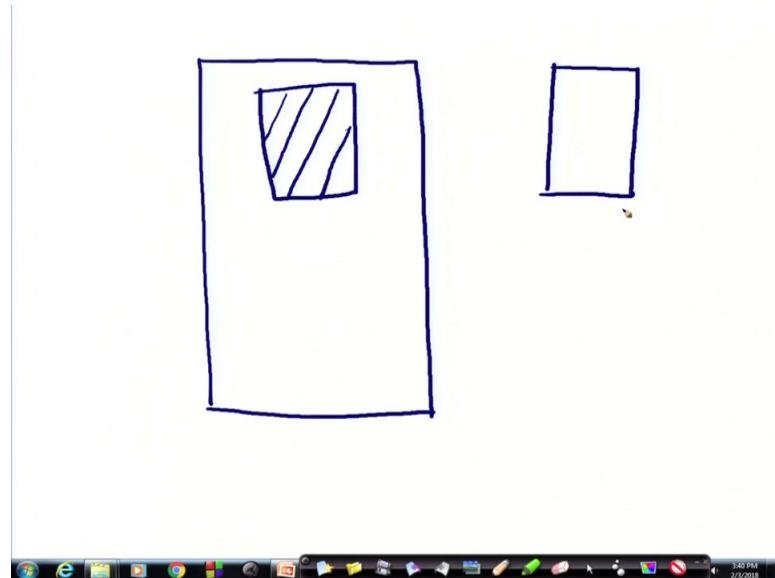
Now suppose I have now this problem has been solved by solving each of these 3 functions, each of these 3 functions have been written and tested and consequently they have been connected together and we have got the complete function. Now suppose another friend of yours wants to write another task solve another task say this was this was also a task.

So, this is also a function I am writing that as capital  $F_1$  and there is another friend of yours who is trying to complete, another task which we are naming as  $F_2$ . Now in order to develop  $F_2$ , he finds that I have to solve some problems some sub problems, which are solved by  $f_1$  and another sub problem which is solved by  $f_3$ , might be that this sub problem requires  $f_1$ ,  $f_3$  and say another function which was not written by anybody till now  $f_4$ .

Now; obviously, since  $f_1$  has been written and tested he the writer of  $f_2$  did not write  $f_1$  again, because that  $f_1$  this  $f_1$  can be used for this purpose similarly this  $f_3$  can be used for this purpose, and only effort that he has to spend is to write  $f_4$ . Thus the functions which are written already right can be reused.

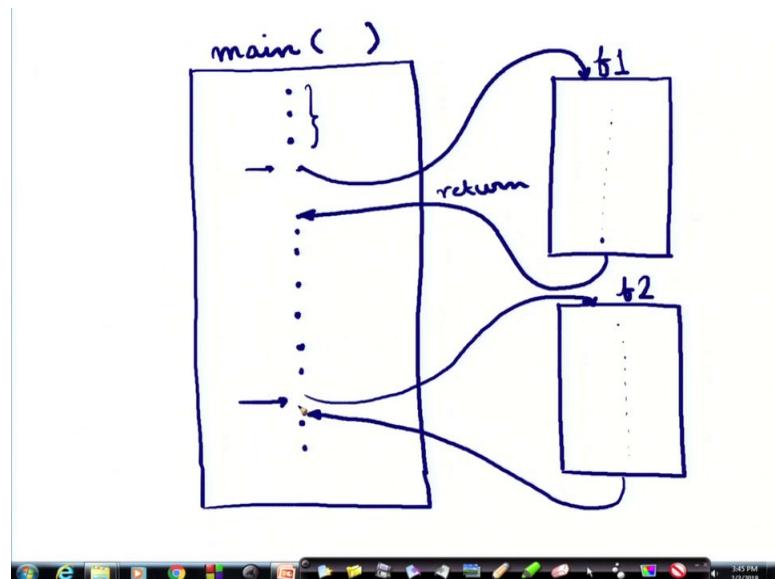
So, this reusability is another very important advantage of writing functions. Now we have said that in order to solve F 1 or F 2 we have to write we will have to independently write and test smallest functions and connect them together. What do I mean by connecting them together let us try to understand that.

(Refer Slide Time: 10:05)



So, say we have a function that is to be done developed and we find that this part has to be this part; this task has to be solved by some function. So, I write a function here for this part; let me do it again.

(Refer Slide Time: 10:39)



This is my whole problem that I have to solve. So, I am solving some things here some steps I am solving. And then I find here there is something that has to be done which is not very easy. So, for that I came at this point and for this I need to write a function or might be there is a function already existing in my library, which has been written by somebody else, which I can reuse.

Now in that case and also say here after doing that here I can do a couple of simpler tasks and then here I come to another point, which requires it to be independently solved and written. So, I need a function for that and then after that I will do some more simple things and my task will be over say this is a situation.

So, when I come here I will have to I means this program, which is a main tasks main task that I have to solve. So, since we have seen the name main in our description for C earlier, let me also name this to be main and I am just for nothing I am just putting some parenthesis here.

So, main is a main task that I have to do and at this point I am doing something's and here I need the help of this, now remember. That this function whose name is say f f 1 that can be used by my main function or somebody else main function also. Therefore, when I need this to be executed with my data, then I must whatever data I have prepared here that some of that I will have to pass on to this here.

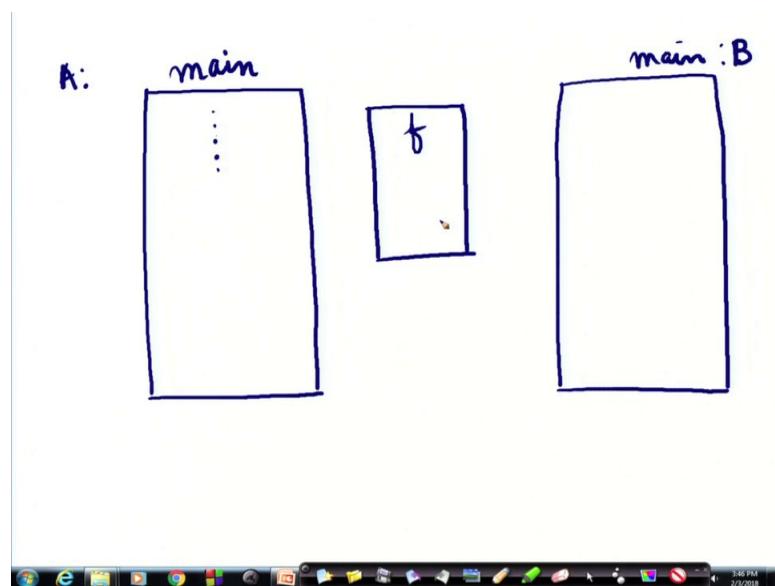
And this will solve this subtask and then I will have to; that means, a this must give back whatever it computed to the main task and the main task will continued doing simple things here. So, you see that there are 2 links; one is going in the function, another is coming out or some data that is being returned by this function. This is being returned by this function. Then I carry out some tasks simpler tasks here and at this point I find now I need help.

So, some part of the data from here will be used by the function f 2 and that has to come to f 2 here and f 2 will carry out the task and will return to the point here. Now about this return you should observe one thing, that the function was called from this point, we call this thing to be a function call function call with some data being passed on to this. And function calls or function invocation and after invocation this function works and it returns where does it return?

It returns to the point in the main task just after the point from where it was called. It was called from here. So, it comes here and then returns to the next point just after the after this call. So, this is a calling point and this a return point.

Similarly for here you see that it is executing and at this point it is being called. And after execution the control is returned into the immediately after the point from where it was called.

(Refer Slide Time: 15:56)

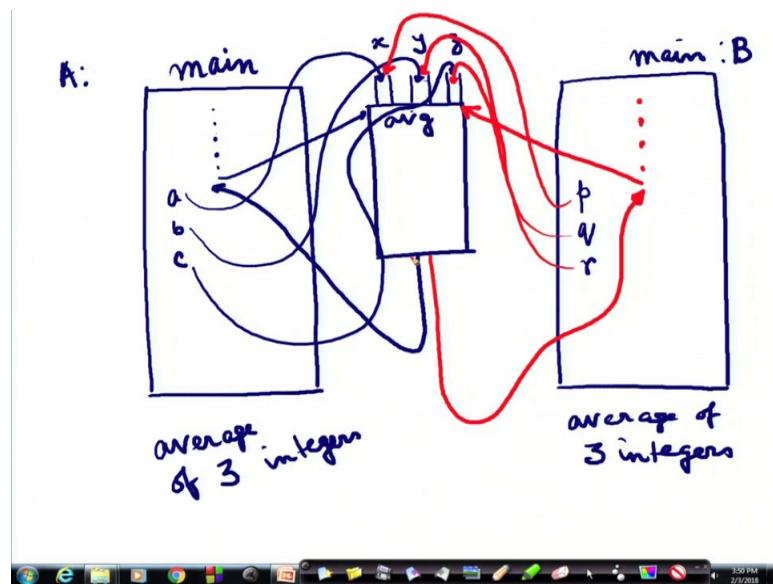


So, this is one important thing to understand the other important thing is that let me draw it again here. I have got my main task and this is my main and here is another main task, some other main task not written by me, but somebody else this is of mister A and this is of mister B for mister B there is he has written another main.

Now mister A or mister B both will require this function f. Now the task the purpose of what this task does is same for example, this task is computes the average of some numbers, 10 numbers, 5 numbers whatever it is or may be floating point numbers, integer numbers, whatever it is it computes the average.

So, let me let me quickly rename it not keeping it wage any longer let me call it, average or for a specific purpose, I want to write it inside the reason will be clear immediately this computes the average.

(Refer Slide Time: 17:21)



Now mister a wants to use this average function for some data for 3 data he wants to compute the average of 3 integers, suppose this one this one computes the average of 3 integers, but the integers can vary right. So, this also wants to find the average of 3 integers to make it simple.

But the in numbers for which A wants to compute the averages different from the numbers that B will compute the average for, but this avg is common to both therefore, suppose the numbers that some integers say are a b and c and here this person wants to find the average of 3 integers which are p q and r 3 variable names.

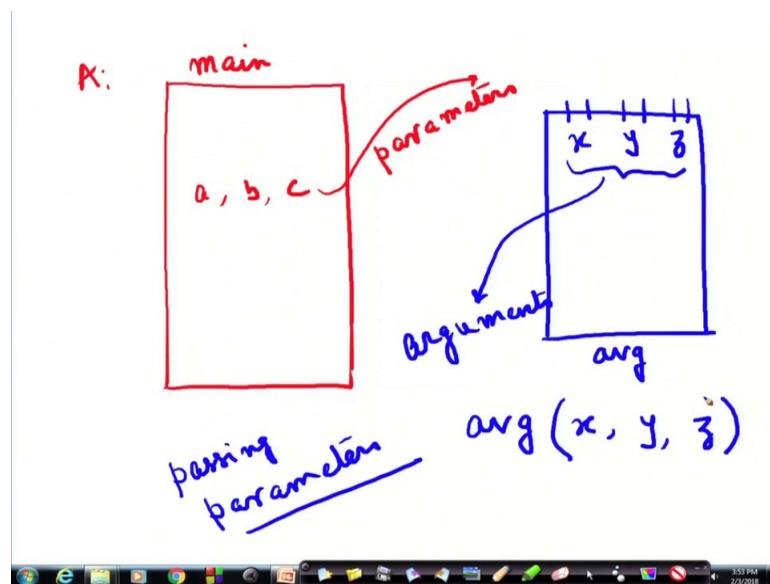
Now this average cannot remember p q r or a b c it will simply just like think of a box with 3 pipes coming in and data will come in through there. So, he just names this average for he names these pipes as x this is y and this is z. So, when this function main wants to compute the average of a b c he must send a through the pipe x b through the pipe y and c through the pipe z and you will get the average computed, and the average will be returned from the point where it was called from this point immediately after that it will go back clear.

Now when let me change the colour, now when the function main B; B wants to compute the average he wants to compute the average of p q r. So, p will be sent to the pipe x, q will be sent to the pipe y, and r will be sent to the pipe z. And the average will be computed suppose it was called from this point, from this point, this average was called.

So, then it will return the to this point with the value of the average with the value of the average whatever is computed here.

Therefore, so if this is clear then let us come to 2 more terms.

(Refer Slide Time: 21:17)

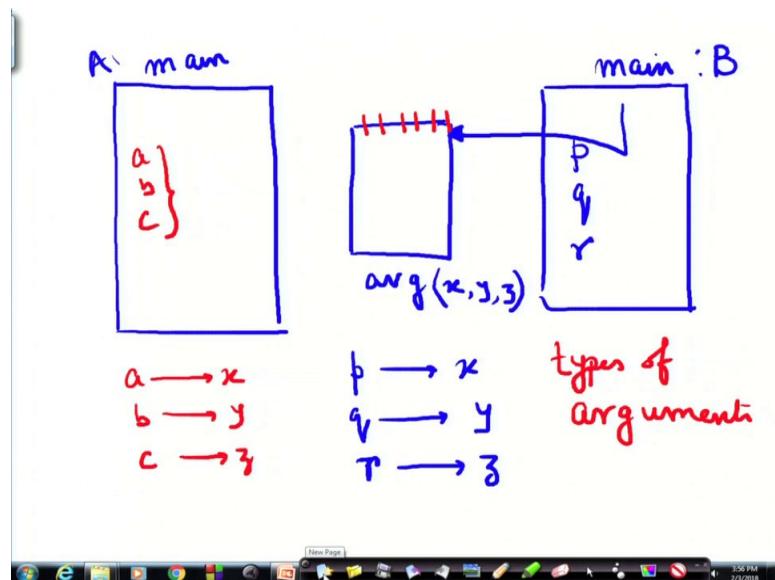


They are that this function main A, A is main was sending a, b, c to the function as parameters we call it as parameters. And this function which is our average function is not biased to a, it has got it knows that it needs 3 inputs and there are 3 input pipes, 3 input positions.

So, these x y z are arguments this we call as arguments alright; that means, this average when it is written it has got 3 input pipes. So, we can write simply like this function name average with 3 parameter arguments x y and z. Now who ever calls it we will have to establish the mapping, between the parameters that it wants to pass the parameters it wants to pass to the through the arguments.

So, the connection that we had shown in the earlier diagram which was where both the programs, were both the programs are accessing it. So, each of these parameters should the parameters should be matching with this arguments. Let us come to this once again say.

(Refer Slide Time: 24:00)

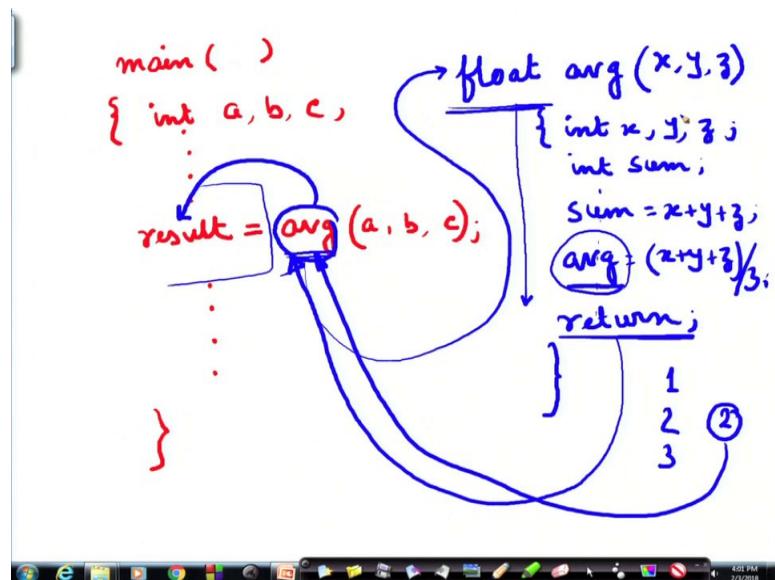


So, here is main of A and here is main of B, I repeat what I was saying till now and this has got p q r now, you must be must have realised now, what are these p q r? When I use this function average, which has got 3 arguments x y z which are nothing, but 3 pipes which are arguments and when I call from some point, when I call this function average then I have to pass these parameters p q r to these arguments x y z.

So, p will go to x q will go to y and z r will go to z, on the other hand when a is calling when a is calling, what is happening, when a is calling then a had the a b c to be passed on to this. So, a goes to x, b goes to y and c goes to z. Now since now if you just think these are 3 pipes, now the pipes are every pipe has got a width.

So, if I want to fit smaller pipe into a bigger pipe that would not fit. So, the width of diameter of both the pipes must fit together, what do I mean by that, what I mean by that is that whatever arguments are whatever are the types of the arguments that must match the types of the parameters or in other words the types of the parameters also must match the types of the arguments. So, that the pipes fit there should not be any mismatch over there. So, let us take little more deeper look here.

(Refer Slide Time: 26:55)



So, suppose I have got a main function here I am not so much bothered about the same types now and the main function is running and here I want to say result is average of a b c and etcetera, etcetera, etcetera and we end here. And let us assume that here I have declared int a, b, c both of them are integers. Now we have got the function which is average.

Now, this average of 3 integers can be a float. So, I just write I will explain it a little later float average x, y, z and I can say int x, y, z and whatever I am computing average here and. So, thus this is simple you can have int sum and sum equals x plus y plus z, and average is equal to x plus y plus z by 3. Then I write returned, why do I write returned I will come to that.

Now look at the 2 things a I will talk about this part a little later. Here I am I have declared a b c to be integers. So, in the memory for a b and c 2 bytes or 4 bytes as the system may demand that type of that much memory has been given. Now when I write here a v g; that means, I am invoking or calling this function a v g. And I am passing on a v g requires 3 parameters x, y, z and x, y, z are all integers. And I could have written there is another way of writing it that I could have written here int x int y int z that is also possible, but later we will see.

So, a, b, c there is a correspondence between a, b and c with x, y and z. And they are matching in the type then this computation. So, we come here then follow my blue line this is being computed and ultimately the value of average is computed.

Look that the value is being computed in a v g which is also the name of the function. So, in a way you can say that the name of the function is just like a variable that is holding the value, that is holding the value. So, that average any variable that is holding a value must have a type that is why since average of 3 integers can be a float therefore, we have to assign a type to the name of the function.

Designating what type of value it is returning. So, here we can see that it will compute average and then there is a statement new statement that you are encountering here is returned, where is it returning it is returning to the point from where it is called at is this point.

So, the average, suppose if the values are 1 2 and 3 then the average is 2. So, then 2 goes to this a v g and that a v g is being transferred to result. Now if I do print result here 2 will be printed. So, this is to give you an idea of what is meant by in calling a function invoking a function, what is meant by returning from a function, and what are parameters and arguments.

We will see more of this in the subsequent lectures, but you should also understand why we do this reasons at 2 fold as I said one is to divide or break down a complex problem into manageable sub problems, and the second and test them independently, and the other issue is other advantage big advantage is that once we make a thing a function, which is tested we can keep it for being reused that is a very very important thing.

And this concept is general over different languages like; we have got class etcetera which we can reuse a number of times. And this is this idea the implementations are varying from language to language, but the concept is common and is used in different languages.

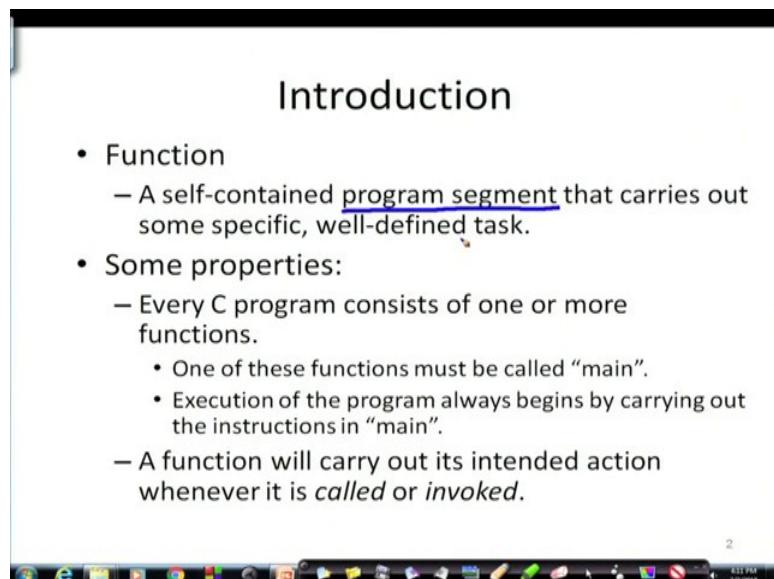
Thank you and we will continue with functions in the subsequent lectures.

**Problem Solving through Programming in C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 36**  
**More on Functions**

We have seen what a function is, and why a function is required, what are the advantages of using a function? We have also seen, what is a parameter and what is an argument, and the types of the parameters in the argument must match. Otherwise, the system will of course, give you error and the reason is because there will be type mismatch and the data will not be passed properly ok. Because this is essentially a parameter is being assigned to an argument ok. Now, in this lecture we will look at some more detail nuisance, detailed points of function. So, to start with let us revise what we have seen.

(Refer Slide Time: 01:03)



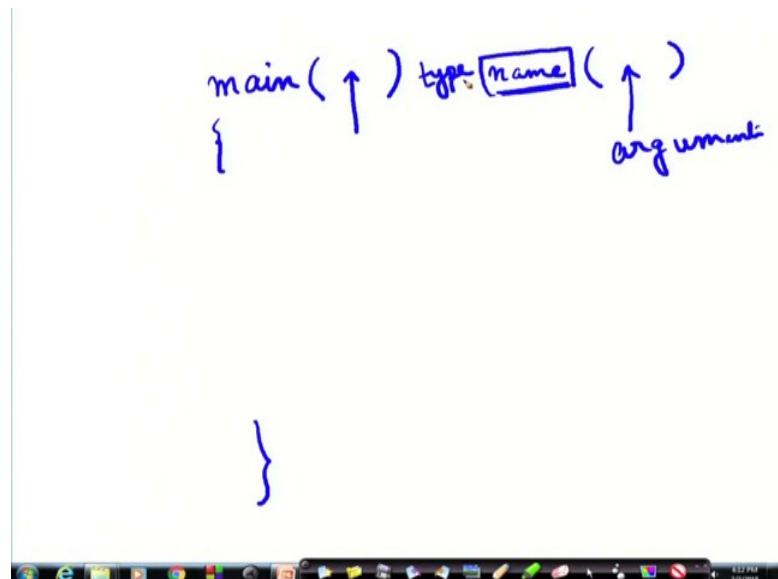
The image shows a computer screen with a presentation slide titled "Introduction". The slide contains a bulleted list describing functions:

- Function
  - A self-contained program segment that carries out some specific, well-defined task.
- Some properties:
  - Every C program consists of one or more functions.
    - One of these functions must be called “main”.
    - Execution of the program always begins by carrying out the instructions in “main”.
  - A function will carry out its intended action whenever it is *called* or *invoked*.

A function; what is a function? A function is a self-contained program segment. This is very important; this is not the whole program, but it is self-contained, and it is a program segment; part of a program. That is a main task and a part of that being executed is being implemented by this function, and this is self-contained because this can be independently tested giving data and finding out, whether the task for which it has been designed is actually being fulfilled or not ok. It carries out some specific and well-defined task. Now that is the general concept of a function, now in C, any C program can consist of one or

more functions. At least one should be there why, because we need always the main right we have seen that we need main.

(Refer Slide Time: 02:12)



Any C function must have a main. And main is nothing but a function. And why did we put here this empty bracket; because the structure of a function is always a function name and a place for the arguments right.

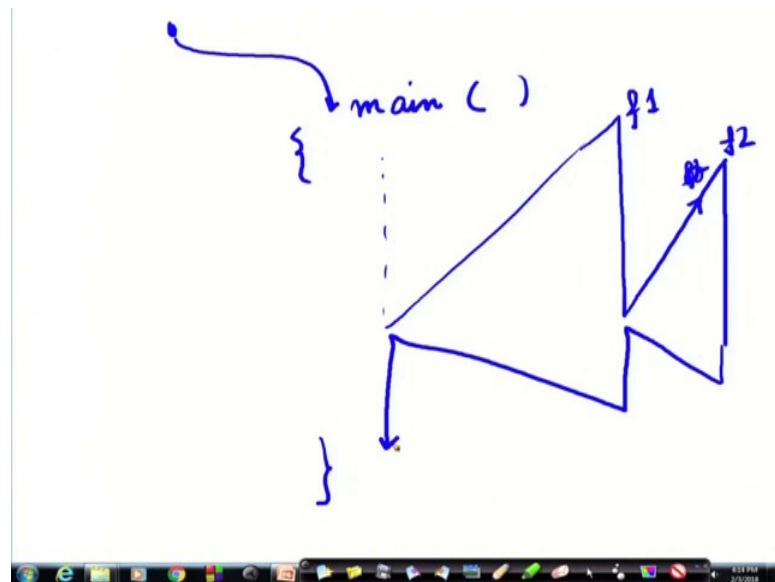
Now, you can say that well, when we are starting a main function we are writing a main function that we will do the task, it is not taking data from anywhere outside. Therefore, why there is no argument. Yes, you are right. There is no argument that is why we put 2 braces and there is an empty space here, this is empty, right? But still it looks like a function. Moreover, in the earlier lecture, we have seen that the result of the function is returned in it is name. And since this is a with like a variable, it will have a type. So, some type like for average we found float.

(Refer Slide Time: 03:37)



So, this main can also have a type. So, we will see later that this main can have a type. We will have a set of empty arguments and the body of the function, and there can be some there can be type. Say, sometimes a type can be int. If this main function; that means, this main function will return an integer and when we say that it does not return anything, we can say also that this function is of type void; that means, it does not return any value ok, void. So, we will see this later, but this is the reason why we say that every C program must consist of one or more than one function. One of these functions must be called the main; and the execution of the program always begins by carrying out the instructions in main. That is the entry point of any program. I mean, I cannot say that main will be somewhere in the middle no.

(Refer Slide Time: 03:51)



When I start, when I start, it must the gateway is the main. And then from within main, I can go out and use some function f 1, and from f 1 I can go out to another function f 2, another function sorry, another function f 2 like that.

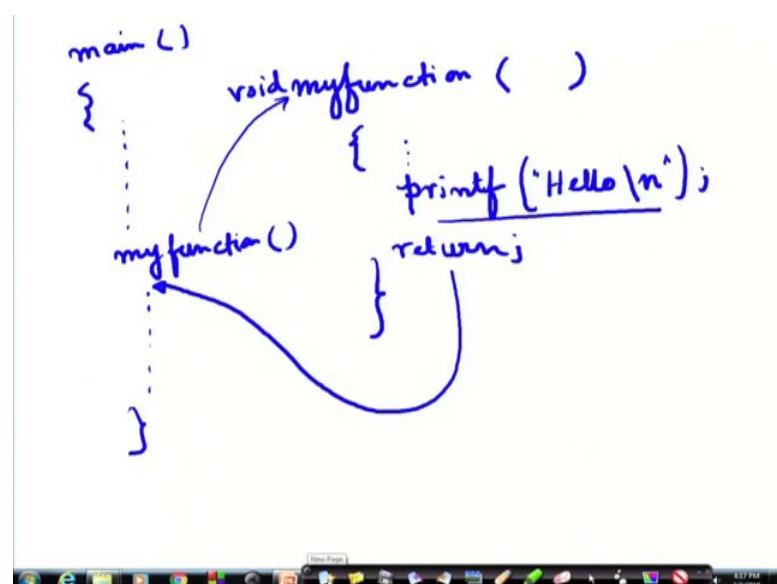
But the gateway the entry should be main; and ultimately the exit say f 2 will return here, then f 1 will return here. Ultimately the exit will also be through main, alright? A function will carry out its intended task, whenever it is called or invoked. So, we have encountered with this term called or invoked right.

(Refer Slide Time: 05:51)

- In general, a function will process information that is passed to it from the calling portion of the program, and returns a single value.
  - Information is passed to the function via special identifiers called arguments or parameters.
  - The value is returned by the "return" statement.
- Some function may not return anything.
  - Return data type specified as "void".

So, in general, a function will process information, that is passed to it, from the calling portion of the program. From, where it has from where it has been called or invoked. And then, the function will take the data; that is, if some data is to be passed, then it will be passed to it from the calling part. And it will return a single value. So, what are these points? Information is passed via special identifiers, called arguments and parameters. And the value that is returned is returned by the return statement. Some functions do not return anything. For example, let us say this. Say, I am writing a function, say my function.

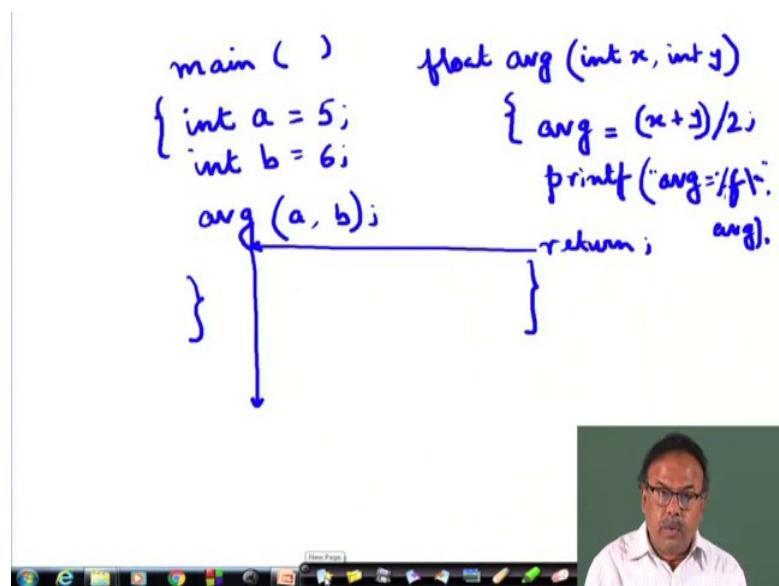
(Refer Slide Time: 07:01)



It is maybe, it is not taking any parameters. Because in the body it is simply nothing doing nothing just printing printf, hello, let us also valid function. So, in this case the main function, where it was it was just here and it just called my function here with nothing this also possible. So, my function it goes there and does not. So, I do not need to give any return, but if I give a return that is also fine it does not return anything. So, this of type void it is not returning anything. So, but the control in any case it does not return any value, but the control will come back here and from here it will be executed.

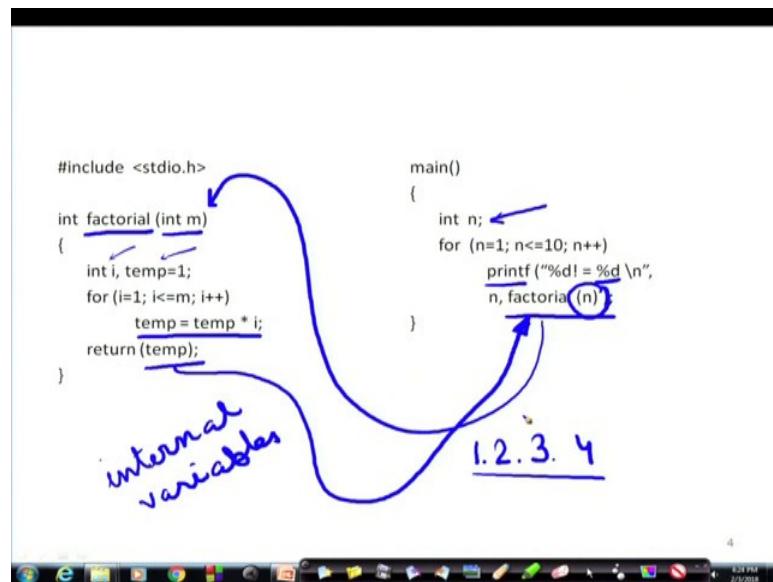
So, there may be some functions, which do not return any value like it just printing some data ok. That is possible for example; I can make it even more realistic.

(Refer Slide Time: 08:35)



Say here `main` is here, and there is `int A 5`, `int B 6`, alright? And here, I call `average` a comma `b`, right. And here I write `average int x, int y`. And here I just compute `average` is equal to `x plus y by 2`. `Printf a v g equals back slash, a v g equals`, I am sorry, percentage `f back slash n` comma `a v g`. That you understand. And I can write `return` or I may not write `return`. So, what will happen? This will come here 5 and 6, 5 will be passed on here `x` will be 5 `y` will be 6. The value will be 5.5 that will be printed here. But I have to give a type to this `average`, because here the `average` type is not defined. So, I have to say this will be `float average`. Because there is some value, some type to this value some type to this variable. Which will hold the value? So, this also possible here you can see that, I do not need to write `a return`, but there is no harm, if I write just simply `return`, that is the good practice; that means, my `return` my control will come back here. And that is the last statement that will be the end ok. So, let us move forward. So, some function may not return anything the `return` type is specified as `void`.

(Refer Slide Time: 11:21)



So, here is an example of factorial. We start with so, here first I have written the function here, and then I have written the main.

But however, in whatever way we write, the main the program while execution we will enter the main first, alright? So, so the execution will be something like this that will enter from here main. It takes the variable n, which is an integer; then n1 to less than 10 n plus, printf percentage d ok, and n and then factorial alright. So, factorial a now when this factorial n is encountered, then we the; this function is called. This function what is doing? It has got int m. So, m is the argument, and is being called with a parameter n; which has been which is being taken; first 1 then 2 then 3. Like that till it is 10.

So, I am taking it n, and then here let us see, what we are doing int i. Now, this i this variable i and this variable temp. These 2 variables are purely internal variables; that means, this variables are internal to the body of the function. Now each of the functions has got a life, each has a life right. So, factorial as long as factorial is running factorial is live, and when factorial is live, whatever variables are defined within factorial, they are live. As soon as we exit from factorial, that function is dead. And the variables associated solely I repeat. The variables associated solely with the function also dies with the death of the function. So, let us see here i and temp. Then for i equal to one to m temp equals temp times i. So, 1 times, 2 times, 3 times 4.

So, it is what is being computed, 1 times 2 times 3 times 4. So, like that whatever is the value of m that is being up to that we are computing a factorial, alright? So, if I call it, now here I have called it with n the value of n, alright? With a value of n, I am calling it, and I am getting the factorial. And the temp is being returned, where it is being returned, this temp is being returned to factorial temp is being returned to factorial, and that is being printed through this percentage d, alright?

So, this is being printed. So, return temp, because why it is different? Here I did not write return, because I have named it as factorial, and I have written temp then temp is going to this value factorial. If I had not done return then ultimately; because here I am keeping the result in temp; not in factorial has not been used as any variable inside the function. Now when I go there, and suppose if I had use some other temp here, this temp and this temp would have been different. We will come to this later, but this just an example of a function being invoked.

(Refer Slide Time: 15:58)

The slide has a red header 'Functions: Why?' and a black footer bar with various icons. The main content is a bulleted list:

- Functions
  - Modularize a program
  - All variables declared inside functions are local variables
    - Known only in function defined
  - Parameters
    - Communicate information between functions
    - They also become local variables.
- Benefits
  - Divide and conquer
    - Manageable program development
  - Software reusability
    - Use existing functions as building blocks for new programs
    - Abstraction - hide internal details (library functions)

To the right of the list is a hand-drawn diagram of a blue rectangle with a wavy line inside, representing a function or module. Two arrows point from the top towards the rectangle, and one arrow points downwards from the bottom of the rectangle.

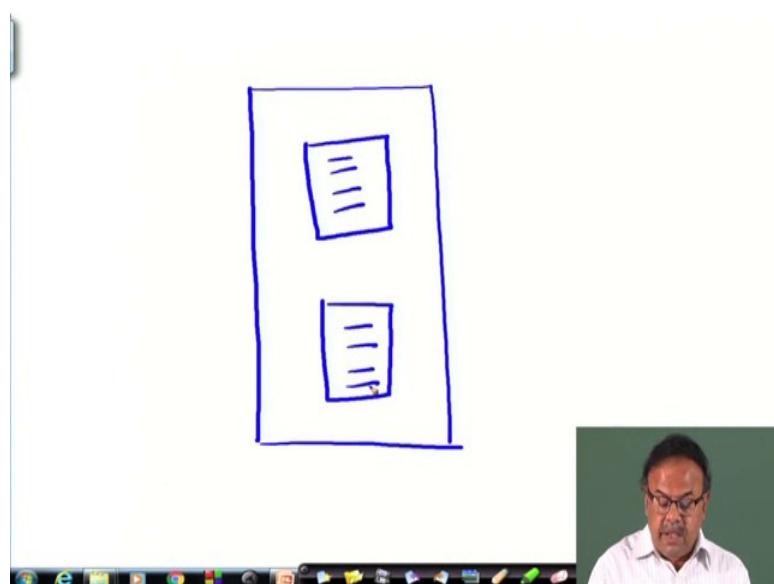
We just recapitulating, why we need functions? This term may be new to you. It is modularising a program, is breaking down a program into small parts. Each part independent part is called a module. That is why breaking down a big program into smaller parts is known as modularisation. So, it is modularising a program. All variables declared inside function are local variables. Very, very important. Which are declared inside functions are local variables; that means, they are known only as long as the

function is remaining running. As soon as the function completes its execution. They also cease to exist. Therefore, a value variable I for example, can be used as an internal variable of a function, and can also be used in some other function or in the main function. They will actually physically be mapped to different memory locations.

So, we have also seen what parameters are. The parameters communicate information between functions, parameter and argument ok. They also become local variables. So, parameter arguments are local variables the parameters means, the arguments are also local variables. The benefits are divide and conquer we know we, we have devoted one completely lecture on that; manageable program development. Software reusability; all existing functions can be used as building blocks for new programs, and this is another thing that you may realise later, that inside a function, how I am implementing or how somebody has implemented, it may be very complicated, and I need not bother about that.

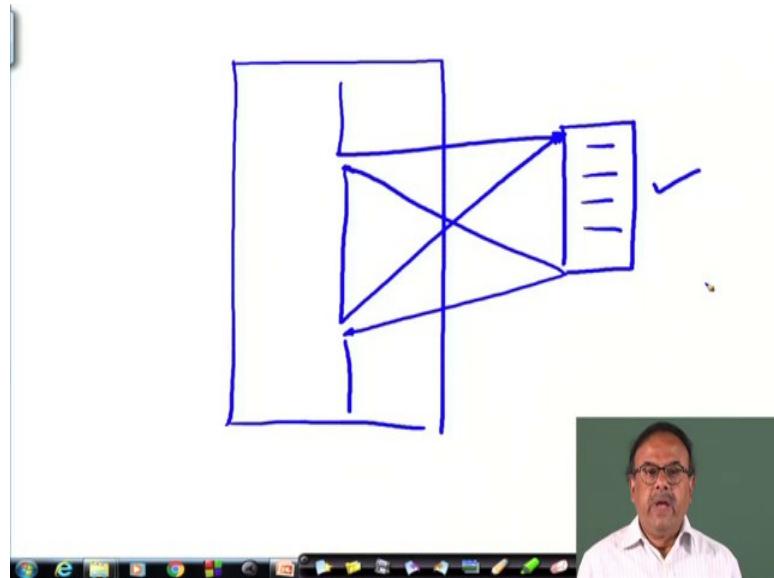
I know what it asks for, what variables have to be put in through this and what output I will get out of it? Therefore, whatever is there inside is abstracted out from it is hidden from it. The internal details are not needed; that is, another big advantage of using functions. And of course, avoid code repetition; something which somebody has written. Or say for example, in a main program. Let us take another example to understand.

(Refer Slide Time: 19:12)



This better say, I have got a program, in which I am doing some computations. Say, computing the standard deviation here, I am computing the standard deviation here, and may be. So, at both these places I write this, write this, but that is not needed.

(Refer Slide Time: 19:36)



If I have a function for computing standard deviation then, here is my main program and there is one function for standard deviation here. And from this point, I can simply call this get the value again continue, from wherever I need I can again call this, and I can get the value, and in that way, I save in the number of lines of code that I write. So, that is known as avoiding code repetition.

(Refer Slide Time: 20:16)

## Defining a Function

- A function definition has two parts:
  - The first line.
  - The body of the function.

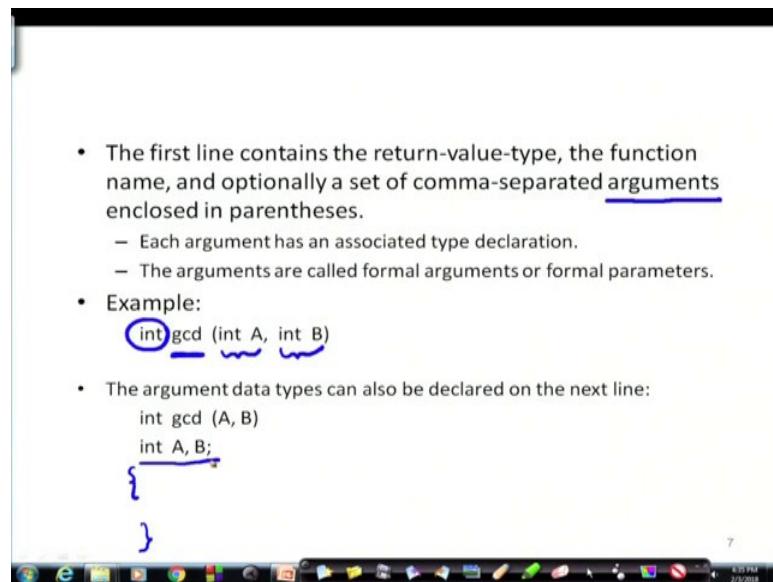
return-value-type function-name (parameter-list)  
{  
  declarations and statements  
}

argument

Now, defining a function; how do you? Now we are till now I was trying to explain you somethings, now we are looking at the syntactic details of a C function. Now that can vary from language to language, but more or less the ideas are same. So, a function definition has got 2 parts. The first line, first line and the sorry, the first line and the what is happening here? First line and the body of the function ok. Now, what is the first line typically? What can we have in the first line? First line we have got the function name. I do not know why it is not coming. In the first line we have got the function name, and the parameter list. I call it argument list, but whatever you.

So, a function has got the first line. And we can see; what the first line is? The first line will contain the function name, and a return value type, what type of value is being returned. Followed by this is the body of the function. The second part is the body of the function. Where there are declarations and statements. Some internal values can be declared internal variables can be declared here. And the other statements are also here. Now, so, that is; so, this one I actually I am calling them argument. Some people also call both of them to be parameters ok.

(Refer Slide Time: 22:22)



Now so, the first line contains the return value type. The function name and optionally a set of comma separated arguments enclosed in parentheses. Like, say here I am defining a function called gcd, greatest common deviser. It has got a type int, and has got 2 arguments. One is A and one is B, I have written the types of them in this parentheses itself int A, comma int B they are comma separated.

The argument is possible that I can declare the arguments also on the next line. For example, I can say instead of writing it in this way, I can write in this way int gcd A comma B and in the next line before the body, before the before the bracket, probably in earlier example I did a mistake, I had put the bracket inside the bracket. It should not be inside the bracket. Immediately after that it you can write int A comma B; that means, you are declaring that these are A and B. Either inside or immediately after that, both of them are fine. Now these are called the formal arguments or some people also call it formal parameters.

(Refer Slide Time: 24:04)

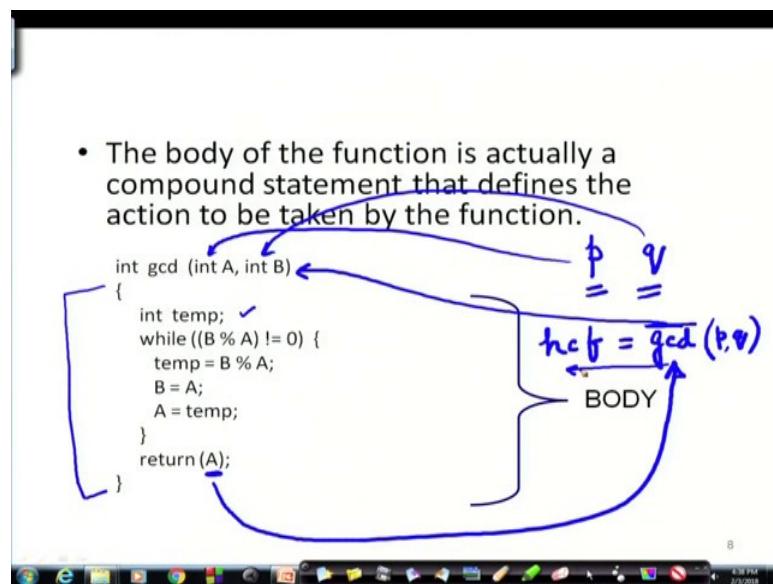
- The body of the function is actually a compound statement that defines the action to be taken by the function.

```
int gcd (int A, int B)
{
    int temp;
    while ((B % A) != 0) {
        temp = B % A;
        B = A;
        A = temp;
    }
    return(A);
}
```

BODY

Now what is the body of a function? The body of a function is actually a compound statement. I will just take just as I had statements like for loop ok, or. So, let us take or say while loop or if conditions, where I write if some condition then some statement, else some statement like that. So now, this whole thing is a compound statement. Similarly, here the entire body of the function can be assumed to be a complete compound say statement; where we are passing on to arguments and is giving us some value. Let us look at what is being done here. In gcd int A, int B. So, from the calling function must be I am supplying some values p and q, may be 2 integers of type integers, which are mapping to this p and q A and B.

(Refer Slide Time: 25:02)



Now here you know we have seen this algorithm earlier. We take a variable temp. Now that is now the body of the function is defined by these braces. Temp, while B is not divided by A, as long as that does not happen. We take we divide B by A and take the remainder, and then go on that classical way of finding the gcd, you find the reminder and make the remainder the deviser, right? So, that is exactly what we are doing in a loop.

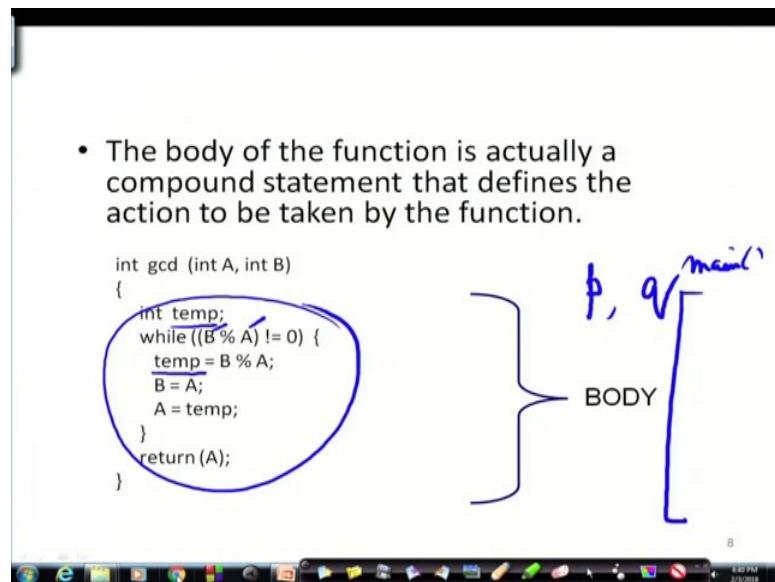
Ultimately the final deviser which divides and gives us A 0, that will be the when it that will be the gcd. So, that gcd now; that gcd is coming in this A. Therefore, I cannot here just write simply return. I have to say that return A. And may be from the main function I had something like say hcf is equal to gcd p comma q. Now, this a is being returned to this gcd. And because this g c because it was a gcd function, this was called and that is why it is going to a is coming to the here, and then by this assignment statement that is going to hcf. That is how the function is being connected ok. We also see what is the body of the function?

(Refer Slide Time: 26:59)

- When a function is called from some other function, the corresponding arguments in the function call are called actual arguments or actual parameters.
  - The formal and actual arguments must match in their data types.
- Point to note:
  - The identifiers used as formal arguments are “local”.
    - Not recognized outside the function.
    - Names of formal and actual arguments may differ.

So, when a function is called from some other function. The corresponding arguments in the function are called actual arguments or actual parameters; that means what I am calling parameter, some people call them actual parameters. So, just in the earlier example p q were actual parameters. And what was there a b or x y, whatever was there is a b are the are the formal parameters or formal arguments. Now as we had said, the formal and the actual arguments must match in their data types. The identifiers used as formal arguments are local. Not recognised outside the function. The names of formal and actual arguments may differ as we have seen here, they are differing my actual arguments were p q.

(Refer Slide Time: 28:01)



And here it is A and B of course, they are differing. Now this A and B, whatever it is here, this A and B will not be reflected anywhere outside this function. Suppose, here there is a main, the main will not get the value of A and B, this completely local here and as soon as it ends the A and B vanishes, A and B will no longer be available to you. Same is true for temp because temp has been defined internally within this function alright. So, this is a very key thing that you must understand.

So, so, when a function is called from some other function, the arguments are passed points to note that the names and of the formal and actual arguments may differ right.

(Refer Slide Time: 29:01)

```
#include <stdio.h>
/* Compute the GCD of four numbers */

main()
{
    int n1, n2, n3, n4, result;
    scanf ("%d %d %d %d", &n1, &n2, &n3, &n4);
    result = gcd (gcd (n1, n2), gcd (n3, n4));
    printf ("The GCD of %d, %d, %d and %d is %d \n",
           n1, n2, n3, n4, result);
}

gcd (int A, int B)
{
    return (A);
}
```

So, here I will conclude with this lecture I will conclude with this example. So, here we are computing the gcd of 4 numbers. In the main function, we have got n1, n 2, n 3, n 4, 4 values have been read here, you can see that they are being read as and n1 and n 2 and n 3 and n 4. They are been read. Now the result is; now gcd of n1, n 2 gcd of n 3, n 4. So, there is one function gcd; which takes int A int B, right that is what we saw. Now first, when within this parentheses, we will first compute this gcd n1 and n 2. So, a will have n1 the value of n1, b will have the value of n 2, and the function will return something, return A as we saw in the last example, return A. And so, the body of the function ultimately ends with return a. And that a will come here suppose that value was 3 say. So, 3 comes here. So, what do I have now? I have got gcd of 3 comma gcd of n 3, n 4. So now, this same function will be called with a having the value of n 3 and b having the value of n 4. And suppose I compute them, and I find ultimately that the gcd is 5. So, the 5 will come here.

Then what happens to this? Then we will compute again called gcd with 3 and 5. So, again here now a will have 3, and b will have 5 and the return of course, a 3 and 5 will be a the result the hcf of 3 and 5 is 1. So, return a that will come here, and the result will be one. That is how the gcd is computed in cascading of the function calls. The same function you see is being called time and again repeatedly from this single statement in the main function. We will continue looking at functions more.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 37**  
**Functions (Contd.)**

So, in the earlier lecture, we have seen how a function can be invoked, and it returns the values. As we had mentioned that it is not the case that always a function will return a value here is an example.

(Refer Slide Time: 00:35)

Function Not Returning Any Value

- Example: A function which only prints if a number is divisible by 7 or not.

```
void div7 (int n)
{
    if ((n % 7) == 0)
        printf ("%d is divisible by 7", n);
    else
        printf ("%d is not divisible by 7", n);
    return;                                ← OPTIONAL
```

The code is annotated with blue arrows and text. A blue arrow points from the parameter 'n' in the function header to the variable 'num' written above the code. Two blue arrows point from the two printf statements to the corresponding lines in the code. A blue arrow points from the word 'OPTIONAL' to the circled 'return;' statement.

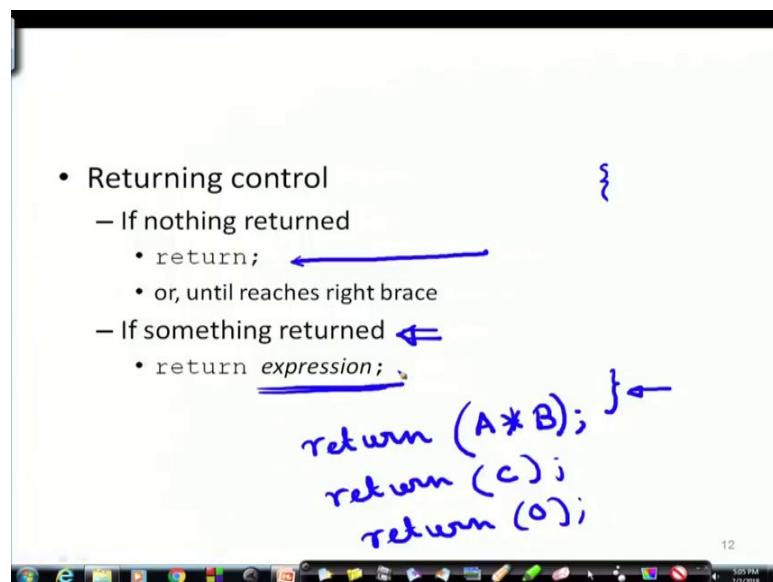
You look at this function div 7 which says; that means, the purpose of this function is to find out whether a particular number that is passed on to it as a parameter, and parameter whether that is divisible by 7 or not. So, the program the code is very simple what should we do? If n is divisible by 7; that means, if n modulus 7 is equal to 0, then we say printf n is divisible by 7 otherwise we print n is not divisible by 7.

Now, in this case we are just printing from here straight the print out is coming printing is coming out from here. The main function is only passing on the value reading scanning reading some value of n and passing on that value of n here or may be here it is n is the argument. So, might be the main program is reading a value num, and passing that value num here. Now this num is also of type integer and the rest of the things are being done by the function. It is testing whether it is divisible by 7, if it is divisible by 7,

it is printing like that otherwise its printing the other message. Now in this case putting this return is optional, because even if I did not put the put the return when I would have met these parentheses that is n bracket it would automatically return to the calling point.

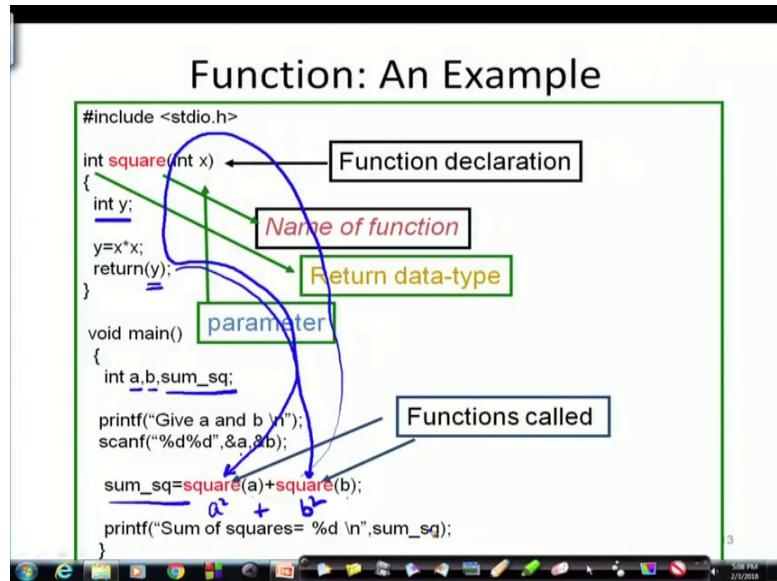
However there is no harm if I put the return.

(Refer Slide Time: 02:44)



So, returning control; we have seen that how the thing is invoked by parameter passing. Now returning control if nothing is returned then you can simply write return semicolon or we can skip that and until it comes to the right brace, that is last right brace that is automatically taken as the return. But if something is to be returned if something is to be returned then we must put the return statement with say may be return A times B some expression or it could be return C or it could be something like return 0 or return 1 whatever we have to do something, some expression and expression automatically in I mean you know we will also capture the constants so.

(Refer Slide Time: 03:57)

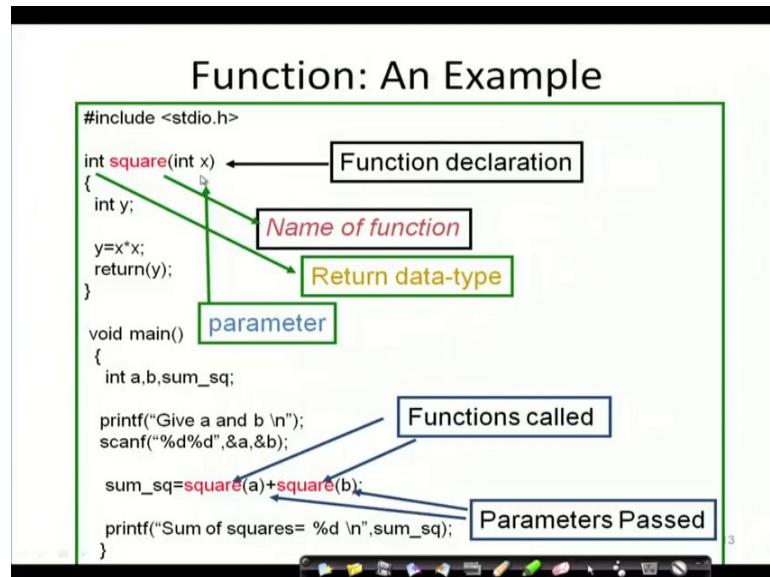


Now, let us look at this example here you see the layout is also important here we are first writing the function this is the function the function declaration is coming first square of x square of an integer x and square of an integer x will also be an integer therefore, the type of the function is int as you can see here. So, that is this whole thing is the function declaration, then starts the body of the function. So, here the function declaration consists of the name of the function, and the parameter that is x of type integer and this int, here is a return data type what the type of the data type that will be returned. So, these 3 together make the function declaration.

Next we are coming to the body of the function int y what is that that is a temporary variable temporary variable why I am calling it temporary? Because it lives as long as this function is active, as soon as the function ends the role the definition of this y is also lost. So, here you see we will come to that later that here you see that this is an internal variable, here I am computing the square y assigned x times x, and I am returning y. I am returning y and after returned that y vanishes y where is y returning to? Wherever they square has been called now here you see here is a sum of square the what is main doing now let us come to main. Main has got some variables a b and sum of square is another variable. So, printf give a and b I am reading a and b. Now I am calling this function twice first with the parameter a next with the parameter b and a and b how I mean in sequence goes to this argument x and the square of a is computed. So, return y will return first here a square. So, we get a square here then this is called and y is

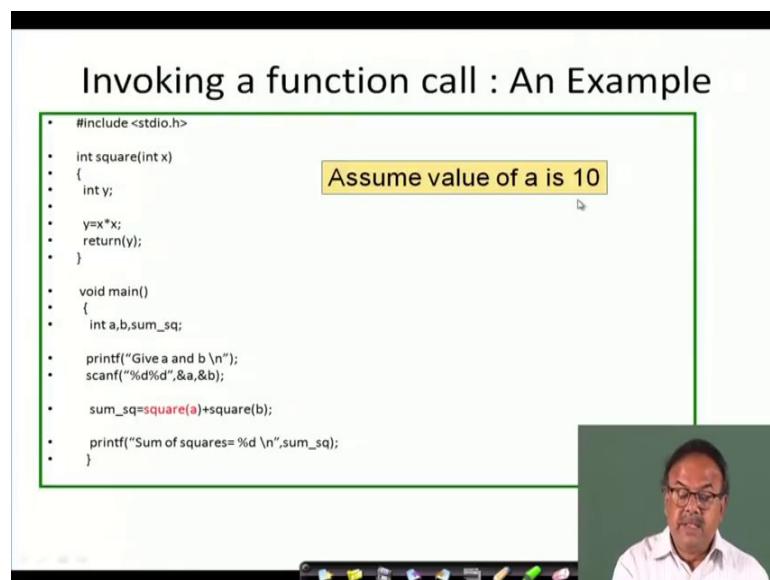
returned here. So, we get b square and then these 2 are added and we get sum of square alright and then we are printing the sum of square. So, you need to you can also try to see what is a flow of data in such cases alright let us move ahead.

(Refer Slide Time: 07:28)



So, these are the parameters passed and here are the, here is the argument you can see that.

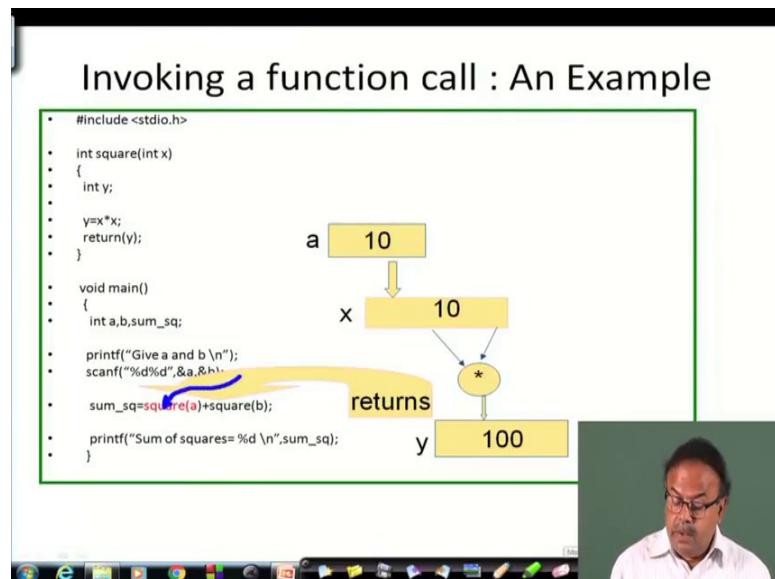
(Refer Slide Time: 07:35)



Now, invoking a function call here the same thing what is happening when a the thing that I just now explained.

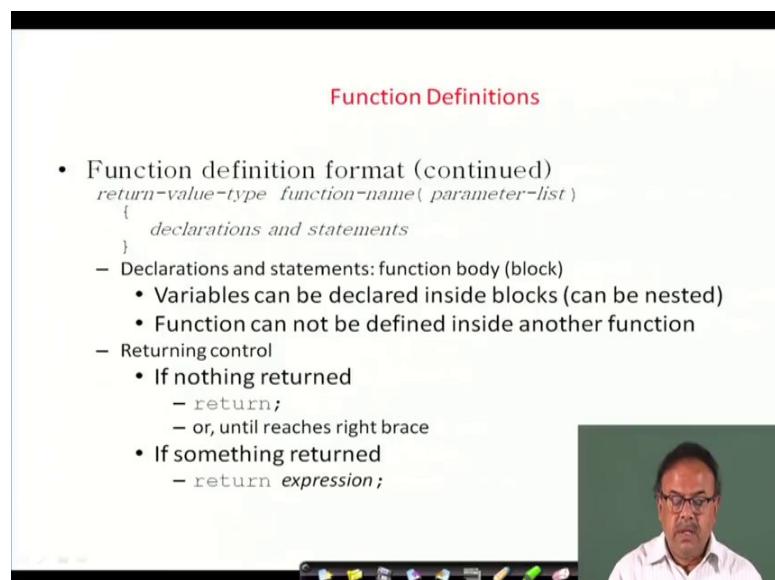
When I am saying now let us see what happens to the variables assume that the value of a that has been read here is 10 alright then square of a means square of 10. So, 10 goes to x, right.

(Refer Slide Time: 08:05)



So, a is 10 here that goes to x. So, x becomes 10, now we compute y which is 10 times 10 I am getting y. Y is becoming hundred right now this hundred is coming here actually this arrow is a little wrong here. So, it will be actually coming to this point clear next suppose.

(Refer Slide Time: 08:53)



So, in that case, similarly it will be for b, if b was some value that is the how then x suppose b was 7 then x will get 7 and then y will be 49, and 49 will come to that square of b. So, 100 plus 49 will now be added and will be kept as the sum of square. So, in the earlier example you could see that in the earlier example the first the function was return and then the main now let us look at go ahead function definition.

So, we have seen that a function name preceded by return value type and declaration statement and then the function body I am repeating certain things variables can be declare declared inside the blocks the blocks can be nested; that means, there can multiple blocks function cannot be defined inside another function this must be clearly understood a function cannot be defined within another function and returning of control the control will have to be returned as we have seen, if nothing returned then return we have already seen that if something is returned then return that expression.

(Refer Slide Time: 10:24)

### An example of a function

```
int sum_of_digits(int n)
{
    int sum=0;
    while (n != 0) {
        sum = sum + (n % 10);
        n = n / 10;
    }
    return(sum);
}
```

$$\begin{array}{r} 125 \\ + 1 \\ + 2 \\ + 5 \\ \hline 8 \end{array}$$



Here again another example of function, the more examples you do the better you will understand. Here the function as the name implies you see its always better to use meaningful names of functions some of digits of n. So, if there be number like 125 then I am trying to extract this digits 1 plus 2 plus 5. So, that will be 8 that is what my program wants to do. So, initially sum is equal to 0 while n is not equal to 0, sum plus remainder.

(Refer Slide Time: 11:17)

### An example of a function

```
int sum_of_digits(int n)
{
    int sum=0;
    while (n != 0) {
        sum = sum + (n % 10);
        n = n / 10;
    }
    return(sum);
```

10 | 125      12  
        | 120      12  
        | 5      5  
sum = 5

10 | 12 | 1  
        | 10      10  
        | 2      2

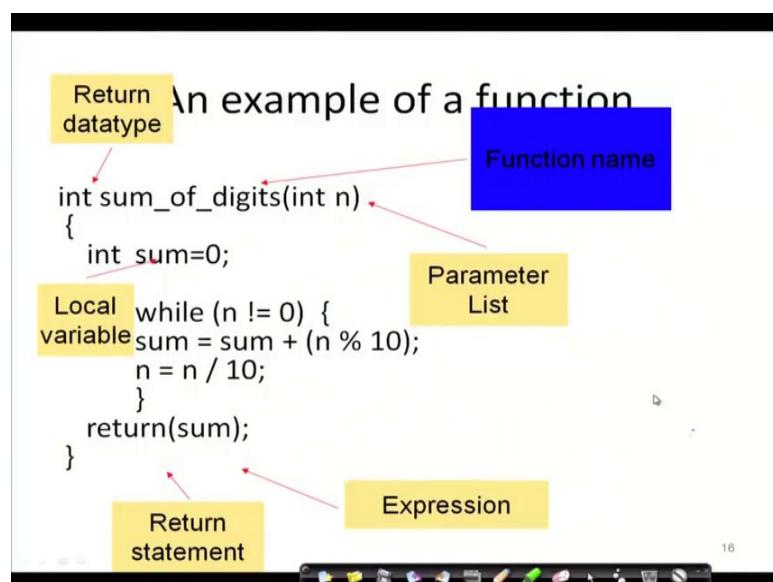
16

The screenshot shows a Windows desktop with a slide titled 'An example of a function'. The slide contains C code for a function 'sum\_of\_digits' that calculates the sum of digits of a given number. To the right of the code, there are two columns of handwritten calculations. The first column shows the division of 125 by 10, resulting in a quotient of 12 and a remainder of 5. The second column shows the division of 121 by 10, resulting in a quotient of 12 and a remainder of 1. Below the slide is a taskbar with various icons.

So, 125 if I divide by 10, then I will have 12 here and remainder is 5. So, sum plus 5. So, sum becomes 5 clear.

Now, then we find out the device dividend that is 12 and again divide that by 10, we get 2 to be the remainder. So, we take sum to be 5 plus 2 and so and so forth. Ultimately I am getting the sum. So, return sum this another example of a function; here you can see that this n is coming as a parameter all these sum; sum is a an internal variable and will not have life beyond the body of the program.

(Refer Slide Time: 12:21)



So, here you see sum of digits is a function name int is a return data type, parameter list is that local variables is sum. Sum is a local variable and return statement is return sum clear all these we have already discussed, this is merely a division; and here you can see that the return can have an expression here only a variable is an expression.

(Refer Slide Time: 12:54)

The slide is titled "Variable Scope". It contains a C code snippet and a screenshot of a Windows desktop environment.

```

• int A;
• void main()
  • {
    •   A = 1;
    •   myProc();
    •   printf( "A = %d\n", A);
    • }
  • void myProc()
  • {
    •   int A = 2;
    •   while( A==2 )
    •   {
    •     int A = 3;
    •     printf( "A = %d\n", A);
    •     break;
    •   }
    •   printf( "A = %d\n", A);
    • }
  • ...

```

A blue arrow points from the first declaration of 'A' in the main() block to the variable declaration in the myProc() block. To the right of the code, handwritten text says "global" above a box containing "A" and "100". Below it is another box labeled "Am" with a long horizontal bar underneath.

Printout:

-----

global

A [ 100 ]

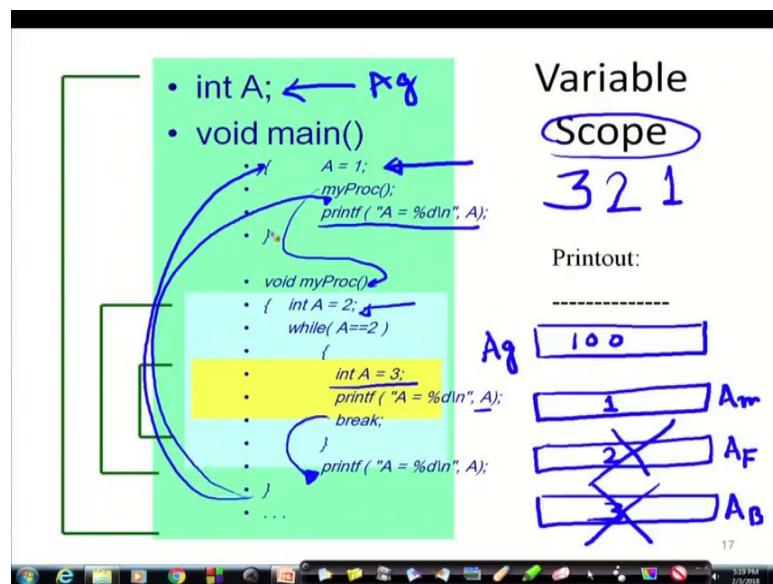
Am [ \_\_\_\_\_ ]

17

Now, we come to a very important concept, which I was mentioning in the passing that the life of an intern variable internal to a function exists as long as the function is live as long as the function is active.

Now, that formally is known is called the scope of the variable or the variable scope. So, let us look here there is an interesting program. Now here you see my entry point is the main, but even before that, I have declared a variable int A; that means, this A is a global variable that means, it is there always. Suppose it is A value 100 then it remains this this is retained. So, let us keep this and let us see what happens. Now I am entering main and I have assigned A to 1 now; that means, now inside this function inside this main I have got mains A which is another A let me write it m is it visible let me do it in a better way.

(Refer Slide Time: 14:38)



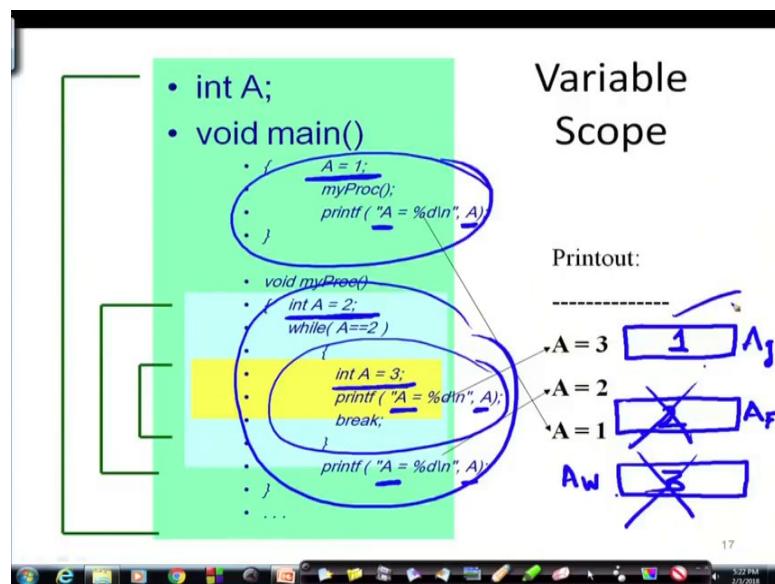
So, there is one A, I am calling that A g that is this A, is A g that is the global A and suppose that is 100. Now this is another a which is defined in the main. So, the function this will be live inside main. So, let me call it A m alright. So, now, A becomes 1 here as I come here a 1, then I am calling my proc a function my proc from here it comes to my proc and suppose my proc has got another A here. So, let me call that to be A x just for understanding that it is of my proc or let me call it let me call it of the function right; so, AF. Now that is initialised to 2 now you see how many different a's I have? Multiple a's now inside this block I initialise another A to be 3. So, that is another A that means, this one is not being disturbed A is 2 and while A is 2 I am making another a because here I am declaring you see this is a pure declaration int A. If I had just written A, if I had just written A assigned 3, then this A would be assigned 3, but here I have declared another A int A.

Therefore there is another A coming within this y while block that I am calling AB, and that is becoming 3. Now I am printing A, which here will be printed the inner most the current A that means, 3 will be printed then I break; break means what? I come out of the while loop we have learned break. So, we come out of the while loop and then I print a which is so, as soon as I break out of this this is gone no longer live.

So, I am coming here now which a is in my scope which a is in my scope, my scope is this I am within this function. So, this A. So, that will be printed here. So, 2 or there was

some back slash n I am ignoring them that will come one after another and then I come out of this its over. So, I go to this main function go back sorry not here I am sorry it should go back to this point; that means, I will now execute print which a is in my scope now I have come out of this come out of this function. So, the its scope is also gone. So, the scope of this function main is now live. So, what is the value one? So, that will be printed; that is how the things will be printed. So, we will repeat it if necessary, but let us try to see the execution.

(Refer Slide Time: 18:14)



Now so, if I first do it then this one will be printed a assigned 3 next that is gone. So, here A will be this will be assigned A assigned 2, then I will go up there and the A that is in the scope of this function that will be assigned and then. So, that will be assigned ok.

You see although I declared a global A internally when I declare some other A, this global A I have I look here a point has to be seen. I declared A, a global A here A g that was declared. Here I have assigned to A, I have not declared another A, I have not written int A I have simply written a assigned 1 that means, the a that was there is already existing globally that has been assigned one.

But when I come here and I am declaring int A internally inside this process, another A is created which is the A of the function and that is assigned by with 2 not this one, they are 2 distinct entities. Now again here I have declared another A. So, since I have declared another A, this is the A of this while loop might be and that is becoming 3 and

accordingly the corresponding whenever I say print which one will be printed, which one will be printed will be the one that is within its scope this a is in the scope of this. So, that was printed this and gone this A was is in the scope of this. So, printed and gone and this A is in the scope of this and this is printed. So, this is known as the scope of variables ok.

(Refer Slide Time: 20:38)

**Function: Summary**

```
#include <stdio.h>
Returned data type: parameter
int factorial (int m)
{   Function name
    int i, temp=1; Local vars
    for (i=1; i<=m; i++)
        temp = temp * i;
    return (temp);
} Return statement
Self contained programme
```

main()
{
 int n;
 for (n=1; n<=10; n++)
 printf ("%d! = %d \n", n,
 factorial(n));
} Calling a function

main() is a function

A video frame of a teacher is visible in the bottom right corner.

So, if we summarise functions, you can see this I do not know how much is visible. So, main function I am calling of a function factorial, and we have already seen that and then the function is having different its a self contained programme, which has got its definite name function definition whether type of the argument is also specified. Now main is a function and here I am calling a function, I am actually calling a function by name calling by name and is a returned data type repeating that the function name is there the parameter is there and the return statement.

And the other variables like temp and all those are local variables you are repeating.

(Refer Slide Time: 21:36)

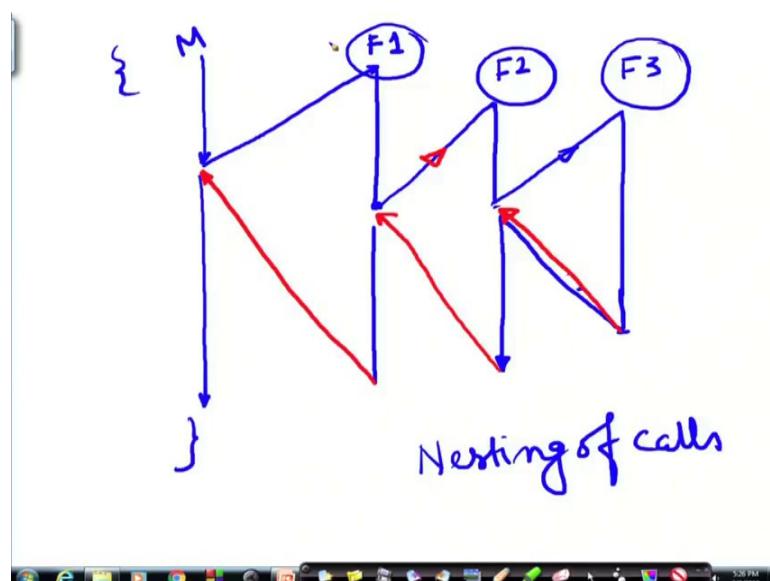
## Some Points

- A function cannot be defined within another function.
  - All function definitions must be disjoint.
- Nested function calls are allowed.
  - A calls B, B calls C, C calls D, etc.
  - The function called last will be the first to return.
- A function can also call itself, either directly or in a cycle.
  - A calls B, B calls C, C calls back A.
  - Called recursive call or recursion.

19

Now some points is a function cannot be defined within another function, which we have told, but I am repeating it again all function definitions must be disjoint; that means, I cannot define one function within another nested function calls are allowed what is meant by nested function call nested function call means that suppose here is a main program going on.

(Refer Slide Time: 22:10)



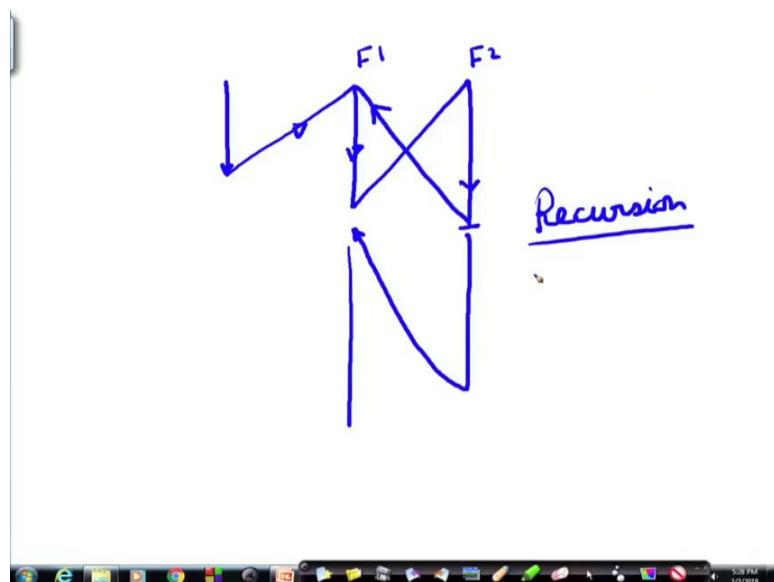
I call a function this is function say F 1, and this was my main from some point in the function in order to solve this problem F 1, I may call another function F 2 this is nesting

calling, but not defined they are defined separately F 1 defined separately, F 2 defined separately, m defined separately. Now from F 2, I may call another function F 3 and F 3 completes F 2 has called F 3 because of some reason. So, that problem reason is answered I mean for some value of a some computation that computation has been done from here it returns to the point from where it was called. So, this a return point. So, is it clear or should I use some other caller for the return point is it necessary. So, I am not getting the colour. So, let me use the existing colour whatever was there. So, let it go. So, from here now the colour has come. So, I can show the return here I am returning, but then again I am continuing with this function.

And when F 2 is over, then I again return to F 1. So, it was called F 1 called F 2 for some purpose that purpose is solved. So, I return here and then F 1 continues again in its whatever it has doing and F 1 was called by main for some particular reason when that purpose is served, then we return to this point and then main continues right and ultimately main ends with this bracket. Now here this is known as nesting; that means, I nesting of calls.

So, I have made a call, and from that call I can make another call from there I can make another call. But the point to point to note is that all this functions must be independently and separately defined they cannot be defined one among the other. So, nested function calls are allowed A calls B, B calls C as I have shown m calls F 1, F 1 calls F 2, F 2 calls F 3 like that it can happen. The function called last will be the first to return; obviously, we have seen that in our earlier slide that we go back to the F 4 from F4 I return to F 2 and like that and. So, a function can call also call itself either directly or in a cycle, we will see this separately what is meant by that. A calls B this is this can be in 2 ways one is that say A calls B, B calls C, C calls back A that is possible like say here if we see that.

(Refer Slide Time: 26:14)



Main was running it called F 1, F 1 called F 2 and then F 2 can again call F 1 and then this call ultimately for this call the return has to come here and ultimately it will have to return here etcetera. So, this part will have to see separately that is its a function one function is calling another and that can be in a cycle F 1 calling F 2, F 2 calling F 1 it can happen or recursion means say a particular function F 1 calling itself a number of times that requires a special attention and a special discussion that we have to carry out we will do that in subsequently.

But right now just it is remember that these calls can be in a cycle or it can be called to itself which is a recursion.

(Refer Slide Time: 27:30)

The slide has a red header 'Math Library Functions'. Below it is a bulleted list:

- Math library functions
  - perform common mathematical calculations
  - `#include <math.h>`
- Format for calling functions
  - FunctionName (argument);
    - If multiple arguments, use comma-separated list
  - Arguments may be constants, variables, or expressions

A handwritten note in blue ink is present on the right side of the slide, reading '#include <stdio.h>'.

The bottom of the slide shows a Windows taskbar with various icons and the text '20 5:30 PM 2/3/2018'.

Now, we have got some math library functions, which perform common mathematical calculations and I do not remember whether in an earlier class I mistakenly said that, I had mistakenly I do not remember exactly whether I did that or not that you need to include just as we include math dot s t d i o dot h, similarly we have to include math dot h. So, just as we include s t d i o dot h if we use some mathematical functions which are already available in the c library, we have to include math dot h. I do not remember whether while first introducing the square root function, I have might be mistakenly I wrote math dot leap that is a library function. So, dot leap if I had said that that you should ignore it is math dot h include math dot h and.

(Refer Slide Time: 29:11)

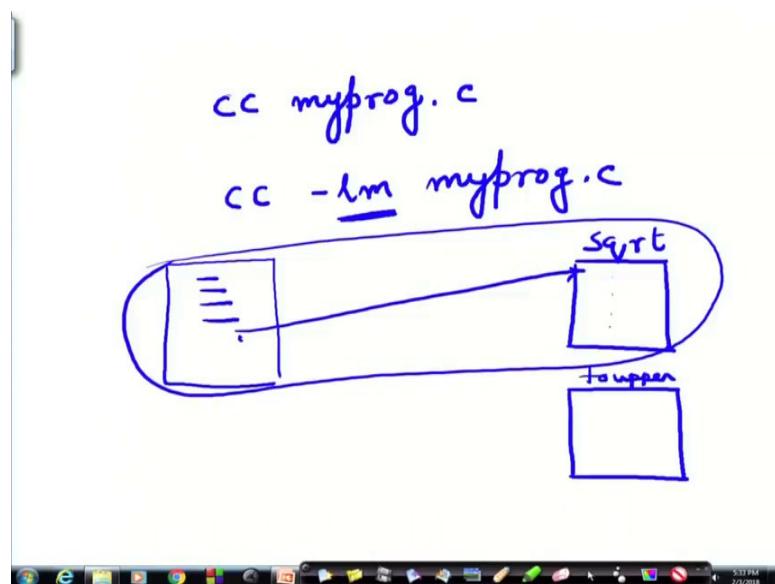
The slide has a black header bar. Below it, the title 'Math Library Functions' is centered in red text. The main content is a bulleted list:

- Math library functions
  - perform common mathematical calculations
  - #include <math.h>
  - cc <prog.c> -lm
- Format for calling functions
  - FunctionName (argument);
    - If multiple arguments, use comma-separated list
  - printf( "%.2f", sqrt( 900.0 ) );
    - Calls function `sqrt`, which returns the square root of its argument
    - All math functions return data type `double`
  - Arguments may be constants, variables, or expressions

At the bottom right of the slide, there is a small number '20'. The Windows taskbar is visible at the bottom of the screen.

So, here there is an important thing, when I compile you known any function that we any program that we write we have to compile it in order to get and executable code. Now in your exercises you must have done by now the typically you compile a c program like this.

(Refer Slide Time: 29:33)



C c myprog dot c right, but if you use some mathematical library in your function then you should write c c minus l m myprog dot c or; that means, link to the mathematical library you compile first you compile now you see.

What is happening is the mathematical libraries are here say some square root function somebody has written for you and that is in the c library alright may be some other function like to upper, which converts from lower case to upper case all these things are there. Now to upper is the separate library where square root is the math library. So, if in your function you if in your program you write you refer to the square root some mathematical library, then you must do this why because purely myprog dot c will generate some object code alright object code.

Now the code for this has to be linked to this has to linked. So, that your ultimately the fool executable code also takes into this account this code link to this code will be forming your executable code because the square root you will need anyway at that time of running it right. So, here it has shown minus l m at the later also c c program name and then link with the mathematical library format for calling the functions, forget about this let us make it for the time being ignore this just say percentage f function name. So, this is point number 1, there are many mathematical library functions in order to include in order to use them we have to include immediately after s t d i o dot h hash include math dot h and we must use this give this linking command.

Now, when we call the function, the function name argument if multiple arguments then we can use a comma separated list say for example, printf some format the square root nine or there is only one. Argument not much arguments may be constant variables or expressions all math now this is important. All math functions return the data type double this is important you should keep in mind this are for c, all the math functions are returning the data type double. So, in order to make it compatible with the variable where you except the value returned by the math function that should also be double arguments may be constants or variables.

(Refer Slide Time: 33:22)

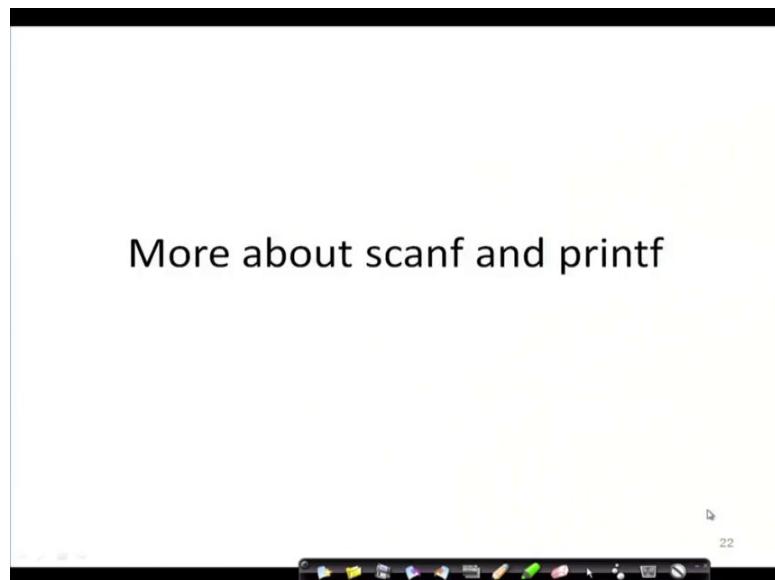
The screenshot shows a presentation slide with a black header and footer. The main content area has a white background. At the top, the title 'Math Library Functions' is centered in a small, dark font. Below the title is a bulleted list of C library functions, each followed by a brief description. The list includes:

- double acos(double x) -- Compute arc cosine of x.
- double asin(double x) -- Compute arc sine of x.
- double atan(double x) -- Compute arc tangent of x.
- double atan2(double y, double x) -- Compute arc tangent of y/x.
- double ceil(double x) -- Get smallest integral value that exceeds x.
- double floor(double x) -- Get largest integral value less than x.
- double cos(double x) -- Compute cosine of angle in radians.
- double sinh(double x) -- Compute the hyperbolic cosine of x.
- double sin(double x) -- Compute sine of angle in radians.
- double sinh(double x) -- Compute the hyperbolic sine of x.
- double tan(double x) -- Compute tangent of angle in radians.
- double tanh(double x) -- Compute the hyperbolic tangent of x.
- double exp(double x) -- Compute exponential of x.
- double fabs (double x) -- Compute absolute value of x.
- double log(double x) -- Compute log(x).
- double log10 (double x) -- Compute log to the base 10 of x.
- double pow (double x, double y) -- Compute x raised to the power y.
- double sqrt(double x) -- Compute the square root of x.

In the footer, there is a small navigation bar with icons for back, forward, search, and other presentation controls. The number '21' is centered at the bottom right of the slide area.

So, here are some examples of math library function like finding the cos of some angle x finding the sin thus all this functions are known as a sin a cos a tan inverse tan (Refer Time:33:41) tan ceiling function floor function. Floor function means it finds the greatest largest integral value that is less than x, suppose some say 200.56. So, the largest integral value is 2 hundred that is the floor function. So, cos a cos is finding the cos of an angle in degree whereas, a cos sorry cos is for finding the cosine of angle in radian now there is no point in memorising them as and when you need them look at the manual look at the book and very soon you will get a custom to the different library which are available for c and.

(Refer Slide Time: 34:35)



We will come back to this in the next function, next lecture about some more very well-known functions which you have already encountered with and then we will proceed further with recursion.

Thank you.

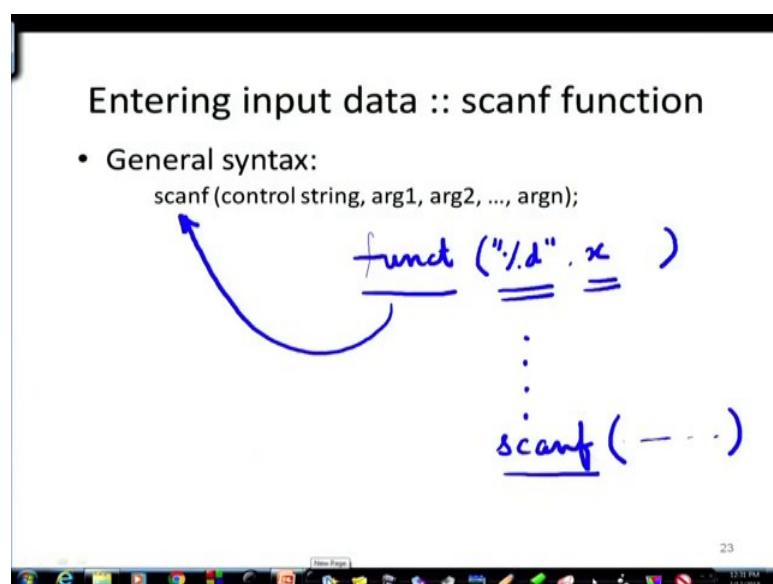
**Problem Solving through Programming in C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 38**  
**Scanf and Printf Functions; Function Prototype**

In the earlier lectures, we have looked at functions and we will still be looking at functions. We have also looked at the library functions and just to recapitulate that, whenever we are using some library functions in our program, then we have to some mathematical library functions, then we have to hash include math dot h. And while we link it, we have while we compile it, we will also have to link it with the maths library with the option minus l m as we have seen.

Now, today, since now, we have got an idea of what functions are. Let us have a relook at our old friend's scanf and printf, which we were using a number of times. Now, scanf and printf are nothing but functions. These are also some library functions and those are included whenever we use include s t d I o dot h. Now, you see scanf and printf being functions when we enter data, let us look at this structure.

(Refer Slide Time: 01:29)



We write scanf followed by some parameters. We have also seen that, a function is a function is nothing but, something like f; say, function name followed by some

parameters. Here, this scanf is that function name and the parameters are the control string; that means, say for scanf you have got something like percentage d and then x alright. So, this is the control string and these are the arguments alright. So, both of these are arguments to the functions scanf.

So, scanf is nothing but a function. So, whenever in my program, suppose I am writing a program and here I use scanf with some parameters. Like this, then it is nothing but a call to a system function. A system function which actually does the task of reading the data from the input input device.

(Refer Slide Time: 02:56)

### Entering input data :: scanf function

- General syntax:

```
scanf(control string, arg1, arg2, ..., argn);
```

  - “control string refers to a string typically containing data types of the arguments to be read in;
  - the arguments arg1, arg2, ... represent pointers to data items in memory.

23

So, a control string refers to typically the data types of the arguments. We have seen percentage d percentage f etcetera. And the arguments are pointers to data items in memory. These arguments in scanf are what are; the typical arguments in scanf? They are say, whenever I use scanf, say percentage d then and a or something like that right?

Now, this and is nothing but an address of the variable a. So, in my memory, wherever the variable a is, the variable a the compiler is allocated this memory location, for the variable a and I am passing the address of that so; that means, it is a pointer pointing to some address or a pointer that is pointing to this particular location in the case of scanf, ok. So, that is why, it said the arguments these arguments arg 1 arg 2 arg n are representing nothing but pointers to data items in the memory .

(Refer Slide Time: 04:21)

## Entering input data :: scanf function

- General syntax:

```
scanf(control string, arg1, arg2, ..., argn);
```

- “control string refers to a string typically containing data types of the arguments to be read in;
- the arguments arg1, arg2, ... represent pointers to data items in memory.

Example: `scanf (%d %f %c", &a, &average, &type);`

- The control string consists of individual groups of characters, with one character group for each input data item.

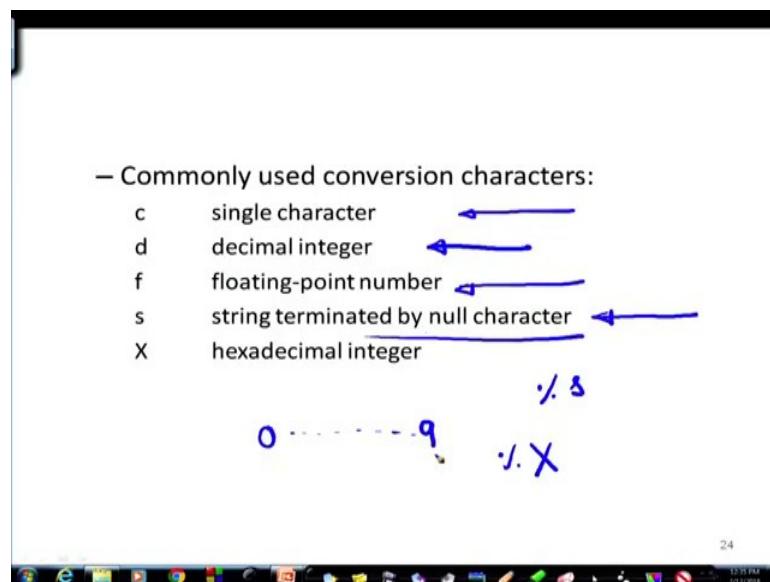
- ‘%’ sign, followed by a conversion character.

23

So, the example, a typical example is, say, percentage d percentage f percentage c to which is nothing but the control string, it is designating that and a; a is a is an integer and and a is a pointer to a. And average is a floating-point number. And and average is a pointer to average. And type is a character type variable. And and type is, why is type a character type variable? Because, I have put in here and c since I have put in here and c that tells me that this type is a character type variable and and type and type is a pointer to the variable type.

The control string consists of individual groups of characters with one-character group for each input data item. So, this is one-character group for this a type. So, percentage we have seen that percentage sign means, is a conversion character.

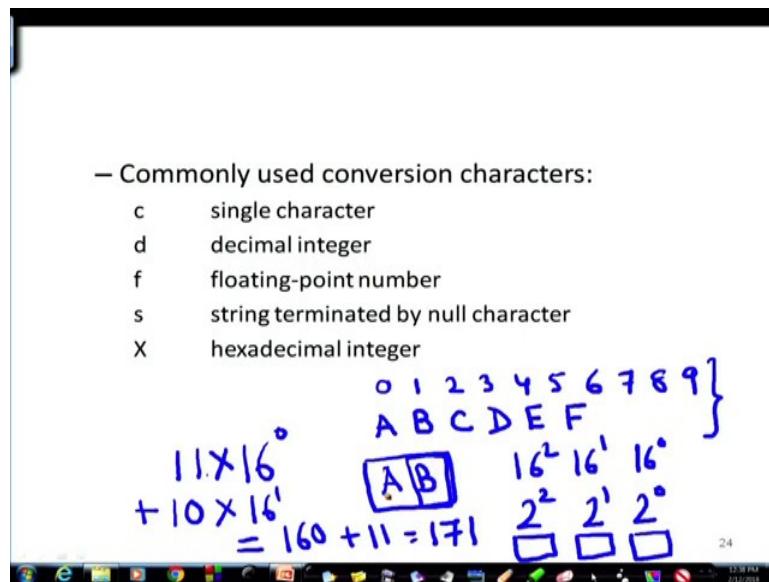
(Refer Slide Time: 05:29).



The commonly used conversion characters we have already seen some of them, these are known to us, percentage d is for decimal or integers, percentage f for floating point numbers, percentage c for single character, percentage s is a string. Whenever, I am reading something variable as a string, we have discussed that in that case, we are reading the string using percentage s and that is a string is always terminated by null character.

Similarly, percentage x denotes that, the number that I am reading is a hexadecimal character. Hexadecimal means decimal is with a base ten; that means, 0 to 9 are my elements. So, all the numbers I have, I am describing using 0 to 9. That is my alphabet set. Whereas, in the case of hexadecimal, the base is 16 and so, base being 16, I have got 0 1 2 3 4 5 6 7 8 9. These 10 numbers followed by A for 10, B for 11, C 12, D 13, E 14 and F 15.

(Refer Slide Time: 06:48)



So, up to that, 0 to 15, I can have. So, in a 16, when I work with a base 16 and that is known as hexadecimal. Now so, so you can just establishing a similarity, analogy with our binary numbers system that we have seen, we can have, say, a number A B.

Now, A B is a string. So, the position string position weights are, first position is 16 to the power 0, second position is 16 to the power 1, third position is 16 to the power 2, like that. In the case of binary, we had the first position way, it was 2 to the power 0, 2 to the power 1, 2 to the power 2, like that right. Now and each of these positions can be filled up in the case of binary by 0 and 1, but here, in the case of hexadecimal, it can be filled up with any of these 0 to a 0 to F.

So, A B, when I say; that means, B is what in decimal B is 11 12. So, 12 times what is the weight of this position 16 to the power 0 plus A is 11 sorry, sorry, sorry, sorry, I am, I am, I am sorry. this is 10. 10 to sorry let me correct that, A is 10 and B is 11. So, it is 11 into 16 to the power 0 and 10 times 16 to the power 1. So, how much is that coming to? 160 plus 11. So, that is the number 171. Now, if I had to represent 171 in binary, I would have required much more number of bits right.

And since it is hexadecimal, another so, here we using hexadecimal, I am being able to do that using 2 hexadecimal bits, which we can call hex x, but, let us see A B.

(Refer Slide Time: 09:54)

The slide contains the following text:

- Commonly used conversion characters:

c	single character
d	decimal integer
f	floating-point number
s	string terminated by null character
X	hexadecimal integer

Hand-drawn annotations on the slide:

- A blue box labeled "171" is written above a 2x2 grid.
- The grid has four cells:
  - Top-left: A
  - Top-right: B
  - Bottom-left: 1010
  - Bottom-right: 1011

At the bottom of the slide, there is a standard Windows taskbar with icons for various applications like File Explorer, Internet Explorer, and others. The date and time are also visible at the bottom right of the screen.

A being 10 and B being 11, in binary what would that be B is 11; that means, 8 2 1, this is 11 and 10 is 8 2 that is all. So, I would have required 4 bits for representing 171, which I am representing using hex x alright. So, that is a hexadecimal integer. So, if I print something in hex in this in this format, then it would be printed like 171 will be printed as A B. Otherwise, it will be printed as in the binary number or in the decimal number. If I do it with percentage d will be printed like this.

So, it was a brief introduction to hexadecimal numbers, but that you can read up yourself. So, let us proceed.

(Refer Slide Time: 11:03)

— Commonly used conversion characters:

c	single character
d	decimal integer
f	floating-point number
s	string terminated by null character
X	hexadecimal integer

— We can also specify the maximum field-width of a data item, by specifying a number indicating the field width before the conversion character.

Example: scanf ("%3d %5d", &a, &b);

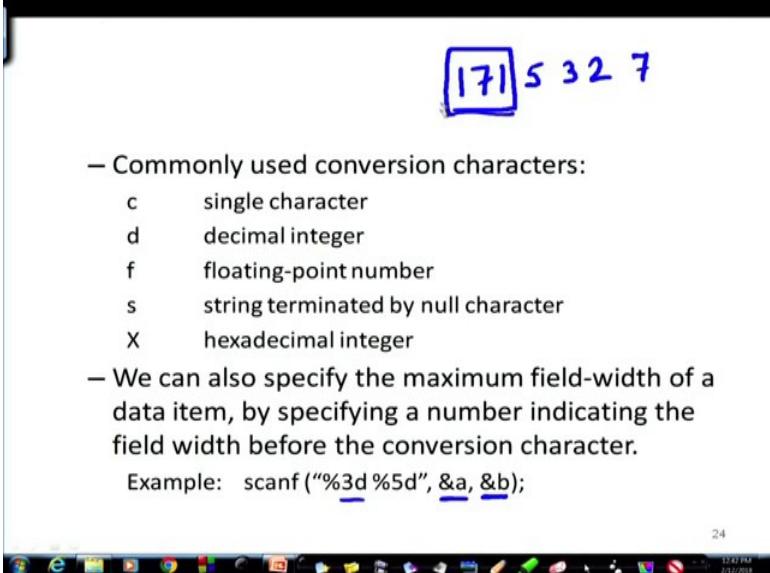
24

We can also specify the maximum field width of a data item. Now, this is something I had purposefully evaded till now, just to not to make the things complicated. I want that you first get a custom to the normal features of c. But now, we are in a now, I think you have you are in a position where you can write programs. Here, I am introducing the notion of field width. A data will be printed within a width, right? So many positions will be given to that number. data item the number indicating the field width before the conversion character.

For example, let us make it clear. For example, percentage 3 d, percentage 5 d for A and B; that means, A will be expected to be of sorry; A will be expected to be of 3 positions; say, percentage it is 3 d decimal 171 whereas, B will be 5 d. So, there will be 5 positions for this 1 2 3 4 5. So, may be 52732. Now, if I want to print 171 or read 171 using the format 5 d, then what should I print? I should print a sorry I should provide the data for the 5 fields. I should put it blank, blank or 0 0 then 171 assuming, it is right justify alright.

So, this is the width. This is the width that we are talking about. We can specify how many what is the when I am reading what is the format of the data? How many widths, how many positions it is taking for as the data, right?

(Refer Slide Time: 13:32)

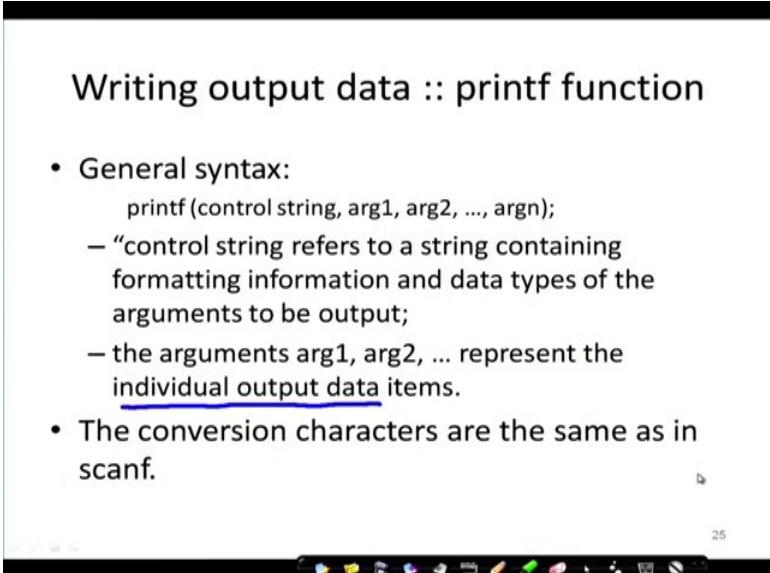


1715327

- Commonly used conversion characters:
  - c single character
  - d decimal integer
  - f floating-point number
  - s string terminated by null character
  - X hexadecimal integer
- We can also specify the maximum field-width of a data item, by specifying a number indicating the field width before the conversion character.  
Example: `scanf ("%3d %5d", &a, &b);`

So, say for example, if I have 171 5 3 2 then 7, then, if I read it, the first data if field is feed is this, I type to the keyboard and if it is in percentage 3 d it will be fine ok. So, I will read that.

(Refer Slide Time: 13:56)



## Writing output data :: printf function

- General syntax:

```
printf(control string, arg1, arg2, ..., argn);
```

  - “control string refers to a string containing formatting information and data types of the arguments to be output;
  - the arguments arg1, arg2, ... represent the individual output data items.
- The conversion characters are the same as in scanf.

Again, now coming to writing now, this printf is also a system function. Just like scanf printf is also a system function and here also, since you know already, how do you write printf? I write printf using a control string within the double code and then followed by

this number of arguments. So, the same thing will hold here. Control string refers to the string containing formatting information and data types of the arguments to be output .

So, now, here while printing the arg 1 arg 2, these are the representing the individual data items. In the case of scanf, they were pointers to the data items. Here, they are individual data items that I have told you earlier. That is why, we use ampersand a ampersand b in the case of scanf. But in the case of printf, we just write a b, ok. Because, this a b are directly referring to the variables the data items. The conversion characters are the same as scanf.

(Refer Slide Time: 15:16)

- Examples:

```
printf ("The average of %d and %d is %f", a, b, avg);
printf ("Hello \nGood \nMorning \n");
printf ("%3d %3d %5d", a, b, a*b+2);
```

1 | 7 | 1 | 6 | 3 | 2

1712

So, let us look at some examples. Printf the average of a and b is avg and that avg is percentage f, you can see that right. So, avg is a float.

Now, I could have also done this that, so, this is nothing but, there is a control string , here you see, I have said that 3 d 3 d 5 d. So, a will be printed. So, when do you printed a will be printed, there is no new line here you can see.

So, a b and a times b plus 2 will be printed side by side. So, a will be printed on 3 fields like, say 171, b b will be printed on 3 next 3 there is no space given. So, this will be b 6 3 2 and the expression. Here a times b plus 2 will have have 5 spaces 5 positions given now; obviously, if my data for result for a was 1712, then only this much will be printed. Then the rest will be left out, ok.

(Refer Slide Time: 17:02)

The slide contains the following text:

- Examples:  
printf ("The average of %d and %d is %f", a, b, avg);  
printf ("Hello \nGood \nMorning \n");  
printf ("%3d %3d %5d", a, b, a\*b+2);  
printf ("%7.2f %5.1f", x, y);

Hand-drawn diagrams illustrate the printf format specifiers:

- A diagram for "%7.2f" shows a horizontal box divided into 7 fields. The first 5 fields represent the integer part (labeled 101) and the next 2 fields represent the decimal part (labeled .52). A vertical line separates the integer and decimal parts.
- A diagram for "%5.1f" shows a horizontal box divided into 5 fields. The first 4 fields represent the integer part (labeled 0101) and the next 1 field represents the decimal part (labeled .5).

At the bottom of the slide, there is a Windows taskbar with icons for Start, Task View, File Explorer, Edge, Mail, Photos, OneDrive, and others. The date and time are also visible.

So, here you see for the percentage f.

Now, for this, I think it is not very difficult to understand. You can read them in any textbook. It is clearly discussed in most of the textbooks. So, I recommend you to go through these formats and as you practice, you will keep that in mind, but just to tell you this. So, floating point number will have 2 parts, right? One is the integer part and the other is a decimal part. So, suppose here, when I say 7.2 f x is a floating-point number, which is being printed in the formats 7.2 f. So, this entire width is 7 positions and the 2 positions here are kept for the decimal. So, these 2 are kept for the decimal and we imagine, we will have a decimal point here and the rest 5 will be 5 will be for the integer part.

For y what will happen? For y the total field is 5, out of which, so total field is 5 out of which only 1 position is kept after the fraction. So, here, I assume the this point.

And so, there are 4 positions to represent an integer and one position to deserve represent a fraction. So, if my result was 101.52 in this format, if I print that, then I will get 0101 or this 0 can be may not be printed.

(Refer Slide Time: 19:09)

The slide contains a list of bullet points and some C code. The bullet points are:

- Examples:  
printf ("The average of %d and %d is %f", a, b, avg);  
printf ("Hello \nGood \nMorning \n");  
printf ("%3d %3d %5d", a, b, a\*b+2);  
printf ("%7.2f %5.1f", x, y);
- Many more options are available:
  - Read from the book.
  - Practice them in the lab.
- String I/O:
  - Will be covered later in the class.

26

Many more options are available. You can read from the book and we will do this later.

(Refer Slide Time: 19:13)

The slide has a title "Function Prototypes". A bulleted list states: "Usually, a function is **defined** before it is called." To the right is a handwritten diagram of a C program. It shows a header file inclusion, a function prototype for `f1()`, a brace for the function body, a brace for the entire function, and a call to `f1()` from the `main()` function. Below the code is a video feed of a teacher speaking.

Now, coming to a very important concept ah, that is, function prototypes. This is possibly new name that you are getting. What are these prototypes? Now, we are going to write functions and those functions, one of the functions which must be there is the main function, right? Typically, people write first the main function and then the first the functions and then the main function, but you are free not to do that also. So, usually a function is defined before it is called. So, typically, what we do is, when our main our

programs starts, so, we start the program starts from here, we define of we write a function with some parameters here and the body of the function is here, then, we write, say, main. And inside main, I call f 1 this assigned to some variable x like that. So, this is the typically some what we do alright. After I did, this include s t d I o dot h. I started with the function first; that is, one option now, but always, that is not done.

(Refer Slide Time: 20:58)

**Function Prototypes**

- Usually, a function is **defined** before it is called.
  - main() is the last function in the program.
  - Easy for the compiler to identify function definitions in a single scan through the file.

$x = \underline{f1}(a, b)$

The hand-drawn diagram shows a vertical stack of three rectangular boxes. The top box contains the label 'f1' with a checkmark. The middle box contains the label 'f2' with a checkmark. The bottom box contains two labels, 'f1' and 'f2', each with a checkmark. A blue bracket on the right side of the diagram groups the top two boxes ('f1' and 'f2') together, while the bottom box stands alone. This illustrates that the function definitions (f1 and f2) are grouped together, while their individual bodies (the boxes themselves) are separate.

So, main in that case, if I define the functions first the functions, that I am going to use, I have thought about that I have designed them. So, first I write those functions and after that, I write the main function. In that case, the main is the last function in the program. So, in that case, the compiler, so, I have got my program here, entire program here. And I have defined the function f 1 here. I have defined function f 2 here. And when I write main here and in main, I refer to f 1 f 2 whichever whenever is this required.

Now, let us look at the compiler think of yourself to be the compiler. Now, if you had started compiling the main first and you would have come to f and f 2, then you would wonder what is that f 1 and f 2. No variables are defined right? You would have taken just like, if you had written x in the main, some function f 1 some function like f 1 a b right. Then, this f 1 is not recognised; I mean you do not know about that here, but if you write it before, the compiler will actually start looking from the beginning of the page, beginning of the program. You will understand, a function has been defined here, I understand that function. So, I know that and the I know what this function does it

compiles that and f 2 also. So, when it comes to in this reduction, when it comes to the main and finds f 1 and f 2, it already knows that. So, there is no problem. So, easy for the compiler to identify the functions, when it scans through the file this a file right, the whole thing is a file

(Refer Slide Time: 23:22)

## Function Prototypes

- Usually, a function is **defined** before it is called.
  - main() is the last function in the program.
  - Easy for the compiler to identify function definitions in a single scan through the file.
- However, many programmers prefer a top-down approach, where the functions follow main().
  - Must be some way to tell the compiler.
  - Function prototypes are used for this purpose.
    - Only needed if function definition comes after use.

However, always that is not done many programmers prefer a top down approach, where I will first define main and then I will put the functions, but then what will happen to the compiler? The compiler if it starts with main, it is starting from the beginning and if it encounters f 1 or f 2 in the body of the main function, then it will get confused what is it just like, if I had got a variable x being used and it is not defined there been error because, I do not know what this x is, what type it is right? So, I have to declare that before there. Similarly, for functions there must be some way to tell the compiler that is the function and for that function prototypes are used function prototypes are used.

(Refer Slide Time: 24:32)

## Function Prototype (Contd.)

- Function prototypes are usually written at the beginning of a program, ahead of any functions (including main()).
- Examples:  
`int gcd (int A, int B);  
void div7 (int number);`

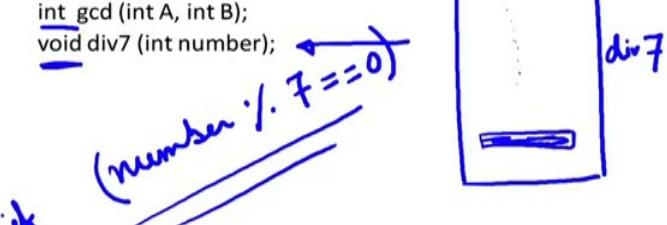
Let us look at this needed we the function comes later. So, function prototypes let us look at . Are usually written at the beginning of the program ahead of any other functions including main, say for example, this is a prototype. This is a prototype, int g c d int a int b ok. So, a and b are the parameters and g c d is a function of return type int. So, I know just by this prototype by this, I know the basic whatever is a body whatever is a body of g c d, I know it is interface, I know it is input output. What is it is input output? There will be some a coming some parameter.

There are 2 parameters internally, they are named as a and b. I also know that, this is an integer and this is also is integer and I know that it is name is g c d and I also know that, it is output is an integer. Whatever is in between, I am not bothered about right now this much I know. So, I will expect whenever g c d will come, I will allocate 2 integers space for that and one return facility should be there.

Similarly, let us look at the second one div 7 int number.

(Refer Slide Time: 26:24)

## Function Prototype (Contd.)

- Function prototypes are usually written at the beginning of a program, ahead of any functions (including main()).
- Examples:  
`int gcd (int A, int B);  
void div7 (int number);`  


28

So, the prototype tells that, it is a function. Its name is div 7. You can guess div 7 means, is testing the divisibility by 7 and there is one integer that is being fed and that integer's name is number and void is a type; that means, it is not returning anything. Maybe, it is printing something from here, it just checks takes a number; it sees whether it is divisible by 7 and prints it here divisible by 7, not divisible by 7, whatever you know. How to find divisibility by 7? So, you take the number and find the mod of that with 7 and if this is equal to 0, then divisible by 7, otherwise not. So, that is in the body of the function and here, we just do the printing. So, I am not returning anything to the calling function. That is why, this is known as void. Void is a valid type.

(Refer Slide Time: 27:43)

## Function Prototype (Contd.)

- Function prototypes are usually written at the beginning of a program, ahead of any functions (including main()).

- Examples:

```
int gcd (int A, int B);  
void div7 (int number);
```

int x;  
;

- Note the semicolon at the end of the line.
- The argument names can be different; but it is a good practice to use the same names as in the function definition.

28

Note the semicolon now here this very important. Whenever for hash include the std I o dot h, I did not give a semicolon. But whenever I am declaring a function or function prototype, I have to give a semicolon just as I had given a semicolon when I declare something like int x semicolon.

Similarly, here, this semicolon is very important. It is just like a declaration. So, the argument names can be different, but it means it is a good practice to use the same names in the function. Now ah, so, what is being said here is that, although I am using here, showing that A and A, but when I am actually writing the function later say, sorry.

(Refer Slide Time: 28:46)

## Function Prototype (Contd.)

- Function prototypes are usually written at the beginning of a program, ahead of any functions (including main()).
- Examples:

```
int gcd (int A, int B);  
void div7 (int number);  
int gcd (int X, int Y);
```

  - Note the semicolon at the end of the line.
  - The argument names can be different; but it is a good practice to use the same names as in the function definition.

28

When I am writing the function later, when I wrote later on int gcd, I can make, let us space, sorry I can write int gcd int x comma int y that is also possible ok. Because, my prototype just said that, 2 integer places. Now actually, I can change the name, change the name of the place, but that is not that important. I mean that is not I mean, usually it is better if we can keep both of them.

(Refer Slide Time: 29:33)

## Function Prototype: Examples

```
#include <stdio.h>  
  
int ncr (int n, int r);  
int fact (int n);  
  
main()  
{  
    int i, m, n, sum=0;  
    printf("Input m and n \n");  
    scanf ("%d %d", &m, &n);  
  
    for (i=1; i<=m; i+=2)  
        sum = sum + ncr (n, i);  
  
    printf ("Result: %d \n", sum);  
}  
  
int ncr (int n, int r)  
{  
    return (fact(n) / fact(r) / fact(n-r));  
}  
  
int fact (int n)  
{  
    int i, temp=1;  
    for (i=1; i<=n; i++)  
        temp *= i;  
    return (temp);  
}
```

29

So, here is a function prototype example. I will continue with this in the next lecture. I will discuss this and we will move to some other important feature of function like parameter passing in the next lecture onwards.

**Problem Solving through Programming in C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 39**  
**Parameter Passing in Function Revision**

So, in the last lecture, we have discussed about function prototype. Before moving to some other topic let us look at an example of function prototype and so that we can understand it better.

(Refer Slide Time: 00:26)

### Function Prototype: Examples

```
#include <stdio.h>
int ncr (int n, int r);
int fact (int n);

main()
{
    int i, m, n, sum=0;
    printf("Input m and n \n");
    scanf ("%d %d", &m, &n);

    for (i=1; i<=m; i+=2)
        sum = sum + ncr (n, i);

    printf ("Result: %d \n", sum);
}

int ncr (int n, int r)
{
    return (fact(n) / fact(r) / fact(n-r));
}

int fact (int n)
{
    int i, temp=1;
    for (i=1; i<=n; i++)
        temp *= i;
    return (temp);
}
```

*nCr = fact(n) / fact(r) / fact(n-r)*

Here you see here is a program where I have got a function  $nCr$ ,  $n$  choose  $r$  and another function  $fact$ . Now, the functions are actually little later here you see  $n$  choose  $r$  is like this standard  $n$  choose  $r$  that you do. Here that is a factorial  $n$  by factorial  $r$ , so factorial  $n$  divided by factorial  $r$  into factorial  $n$  minus  $r$  that we know from our school right. So, that is one function that is one function.

The other function is the factorial. So, here is another function, this is one function this another function. Now, look at the beauty of this function this function is in a simple one return statement I have written everything. So, here you see the body of the function is merely a computation of an expression  $fact(10)$  divided by  $fact(5)$  divided by  $fact(5)$ ,  $fact(10)$  minus  $5$  and this will be computed and that will be returned as an integer. Now, what will be the input the input will be  $n$  and the  $r$  both of them are integers.

Another point to note here is this function itself this function itself when is running first say I get into this function and I tried to compute the return value, first I find fact n and from here I am calling another function that is the factorial function and the factorial function is taking only one input one integer n and based on that it is computing the factorial. Computing the factorial you must be remembering by now, or knowing by now, that is I am starting with 1 and 1 times 2 times 3 etcetera I am going on doing and that is what is being done here temp is 1 and temp is equal to temp times i, this is wrong this should be small i, temp is temp times i, whatever the i value is, 1 times 1, 1 times 2, 1 times 3 it goes on till i reaches n and then i return temp.

So, the point to that you know, this will this thing you know. So, what is being done here is when I am computing this function then I come to this I call this function return temp, I return here and then I move here and again I find again I call to this fact again, this time with the different parameter r and so again fact r is computed after this is computed second time when I return back here. And then I again call this fact using n minus r, so just to make the things clear. So, the first time I am calling this from here and returning back here, second time I am calling this from here the same thing and returning back here and third time I am calling from here and returning back here.

(Refer Slide Time: 04:29)

### Function Prototype: Examples

```
#include <stdio.h>
int ncr (int n, int r);
int fact (int n);

main()
{
    int i, m, n, sum=0;
    printf("Input m and n \n");
    scanf ("%d %d", &m, &n);

    for (i=1; i<=m; i+=2)
        sum = sum + ncr (n, i);

    printf ("Result: %d \n", sum);
}
```

```
int ncr (int n, int r)
{
    return (fact(n) / fact(r) / fact(n-r));
}

int fact (int n)
{
    int i, temp=1;
    for (i=1; i<=n; i++)
        temp *= i;
    return (temp);
}
```

Now, for the third call I am, so, here is one n and that this n and here what I am passing I am passing n minus r, I am first computing this and that is being passed here and that is

being computed. So, here you see it is an example of nested call this was called from the main and this in turns call I mean call another function and in this way it goes on. That is about the beauty of these two functions. But now, here is my main, main function here is my main function. So, the compilers comes from in this way and recognises at this point that ok, n c r is a function how does it know it looks like int. But how does it recognise that it is a function? It recognises n c r to a function because of this parameter argument list.

(Refer Slide Time: 05:40)

### Function Prototype: Examples

```
#include <stdio.h>
int ncr(int n, int r);
int fact(int n);

main()
{
    int i, m, n, sum=0;
    printf("Input m and n \n");
    scanf ("%d %d", &m, &n);

    for (i=1; i<=m; i+=2)
        sum = sum + ncr (n, i);

    printf ("Result: %d \n", sum);
}
```

```
int ncr (int n, int r)
{
    return (fact(n) / fact(r) / fact(n-r));
}

int fact (int n)
{
    int i, temp=1;
    for (i=1; i<=n; i++)
        temp *= i;
    return (temp);
}
```

*<sup>n</sup>C<sub>1</sub> + <sup>n</sup>C<sub>2</sub> + <sup>n</sup>C<sub>3</sub> + ...*

29

Here it is said int r there are two integers coming in as parameters next it understands that fact is also a function. So, its expecting to encounter n c r and facts somewhere down the line and proceeds here it takes m and n, reads m and n, then it is in a loop where it is calling n c r. So, it is series you can understand n c r is computing some with some value i initially 1. So, it is basically n c 1 plus n c 2 plus n c 3 plus etcetera it will go up to m all right. So, that is what is being computed here and so from here a call is made to this.

And as we have seen earlier while this is being executed this one is calling here this is returning back here then again this one. Now, we are making a journey in this line. So, this one is calling this and we are returning back here as we have seen just now. Now ultimately when this entire thing is computed then we will come to this, last bracket here sorry I should not this bracket here; that means, it is a end of n c r. So, I had called n c r

from here. So, I will return back here and will whatever n c r value is that will be added to sum and that will go to this sum and then will proceed in this way I hope this clear.

(Refer Slide Time: 07:42)

## Function Prototype: Examples

```
#include <stdio.h>
int ncr (int n, int r);
int fact (int n);

main()
{
    int i, m, n, sum=0;
    printf("Input m and n \n");
    scanf ("%d %d", &m, &n);

    for (i=1; i<=m; i+=2)
        sum = sum + ncr (n, i);

    printf ("Result: %d \n", sum);
}

int ncr (int n, int r)
{
    return (fact(n) / fact(r) / fact(n-r));
}

int fact (int n)
{
    int i, temp=1;
    for (i=1; i<=n; i++)
        temp *= i;
    return (temp);
}
```

int ncr (int, int)

Now, here I have not written n c r. So, I could have I could have written this earlier in that case this prototype would not be needed, but since I decided that I will be writing it later. So, I decided to just introduce them as prototypes here. Now, you should be very careful about the syntax of the, of writing the prototypes it should have the function type just like a function name and the parameters.

Now, you will see later that it is also possible that I could have written something like int n c r, int comma int that will also mean that this n c r takes two integers as (Refer Time: 08:45) I mean arguments, that will also that is also allowed, but it is nice to write this in this way.

(Refer Slide Time: 09:06)

## Function Prototype: Examples

```
#include <stdio.h>
int ncr (int n, int r);
int fact (int n);

main()
{
    int i, m, n, sum=0;
    printf("Input m and n \n");
    scanf ("%d %d", &m, &n);

    for (i=1; i<=m; i+=2)
        sum = sum + ncr (n, i);

    printf ("Result: %d \n", sum);
}
```

```
int ncr (int n, int r)
{
    return (fact(n) / fact(r) / fact(n-r));
}

int fact (int n)
{
    int i, temp=1;
    for (i=1; i<=n; i++)
        temp *= i;
    return (temp);
}
```

29

Now, next will move to a very important concept; let us quickly have a look at this the prototype declaration and the function definitions are actually here.

(Refer Slide Time: 09:10)

## Parameter Passing: by Value and by Reference

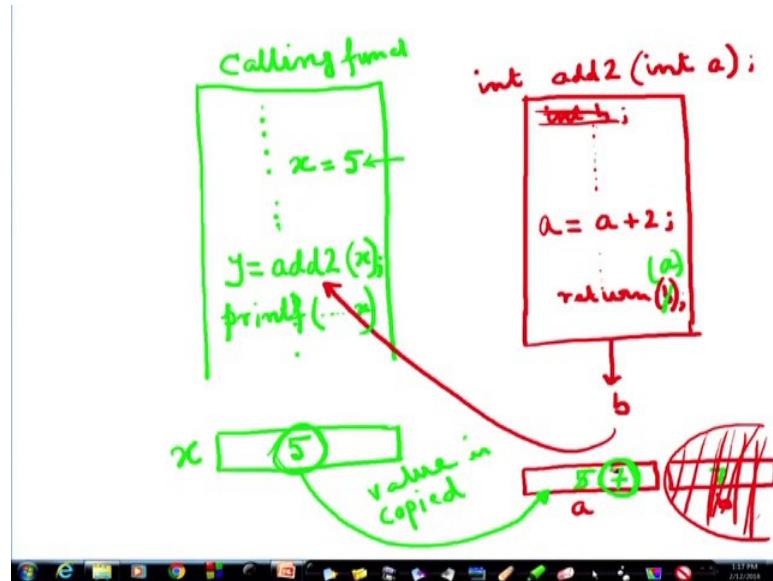
$x = 5$

- Used when invoking functions
- Call by value
  - Passes the value of the argument to the function
  - Execution of the function does not change the actual parameters
    - All changes to a parameter done inside the function are done on a copy of the actual parameter
    - The copy is removed when the function returns to the caller
    - The value of the actual parameter in the caller is not affected
  - Avoids accidental changes

30

Now, we are moving to a very important concept of passing the parameters, how do we pass the parameters from the calling function to the called function. There are two distinct ways in which it can be done one is calling by value another is calling by reference. So, right now, let us think of calling by value all right. Now, let us try to understand it in a simple way.

(Refer Slide Time: 09:57)



Say, here is my main function or the calling function let me not call it main function as we have seen they can be nested functions. So, this is the calling function. Now, inside the calling function I have got some variable x and which may have the value says somewhere 5 it is an integer. Now, somewhere here I am calling a function let us call a simple function decrement or no decrement add 2 or say add 2; that means, whatever is a value we will add 2 to that simple thing. So, here I say y is add 2 x and then semicolon and I go on. And here is my function add 2 int a all right and add 2 is also type integer.

Now, so a we know is a local variable to the function. So, here I will take. So, what will happen? Suppose it was 5, this will result in a will get 5 and here may be here I declare another variable say b although it was not necessary, but I am just saying b is a plus 2 because its task is to add 2 and then I say return b. So, this b which is an integer is being returned. So, b which will be in this case it was 5 it will be going back here. I hope this is clear.

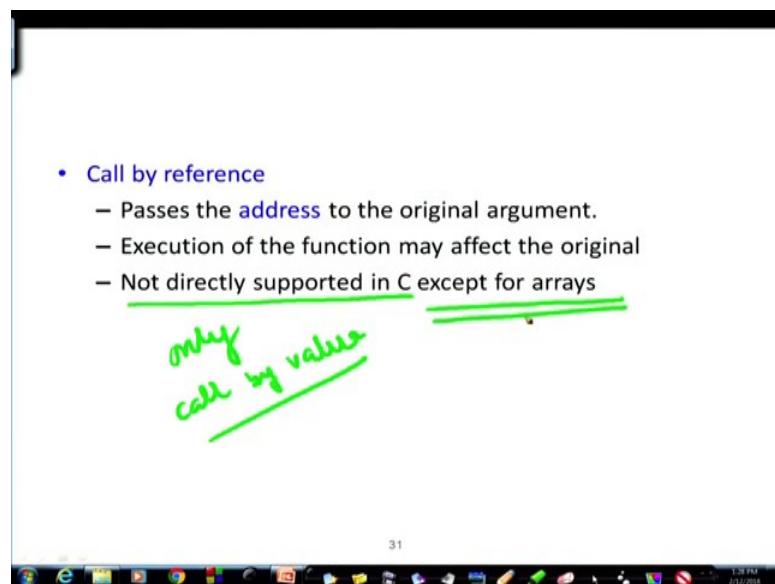
Now, when you know the scope of variables that b is a variable or a or b whatever the life of those or the validity of those are restricted only during the life of this function. So, but; however, in the case of a looking from the point of view of compiler I have got a variable a and I have got another variable b and here I have another variable x. Now, this variable x was having the value 5. So, here in x I had 5 by this statement, this statement made it 5. Now, since x is 5 and this one is expecting the value a, now, this 5 will be

copied in a; what is being copied? Not x, but the value of x. So, this one will be will be getting 5 right. So, the copy value is being copied, the value is copied in this variable 5. Now, it takes b was something, but here. So, b becomes 5 plus 2 b becomes 7, 7 comes out.

Now, suppose I change this program a little bit. I erase this b and make it sorry I make it a. So, this b is no longer there this is not required all right this variable has not been defined I say that this line is also not there, int a and I just do a plus, a assigned a plus 2. So, now, when it was called add x then from here from here this x was copied here and a will be incremented to 7. So, this will be incremented to 7 and obviously, this will be return a not return b because b is not there anymore, so 7, that 7 will be returned here. But you see if here now, 7 has been returned and 7 will be y. Now, suppose if I say I am not following the syntax I am just saying because there is no space here printf something x. What would be printed? For x what would be printed? 5 will be printed because x is still 5 x has not changed. I have simply copied the value to the argument variable and have played with that changed it whatever I wanted to do I have done.

So, take a little time to understand this. This means that the variable that I have in the calling function the value of that will be copied to the argument of the called function. So, there are several advantage is to this one is, so it passes the value of the argument execution of the function does not change the actual parameter like the actual parameter was x which was 5, it remains as 5 although the function added two to that and it came back all changes to a parameter done inside the function are done on a copy of the actual parameter not the original parameter. The copy is removed when the function returns to the caller that entire variable location that was given for the variable a in our example is returned back to the pool of memory locations. The value of the actual parameters in the caller is not affected. Consequently it also saves us from some accidental changes programming that can come copying due to programming errors.

(Refer Slide Time: 17:59)



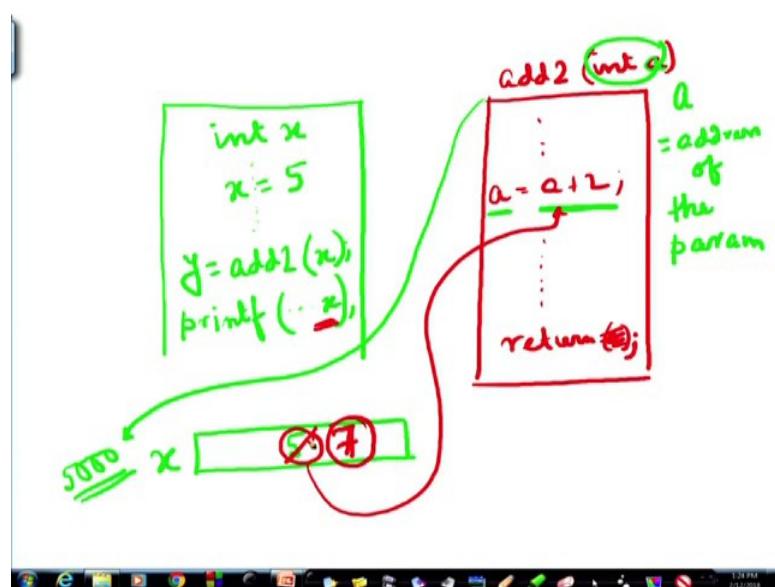
31

1:39 PM

1/12/2018

On the other hand the other thing that is another type of parameter passing is known as call by reference, call by reference. Here we are not copying we are not copying the variable, we are passing the address of the original argument.

(Refer Slide Time: 18:35)



1:39 PM

1/12/2018

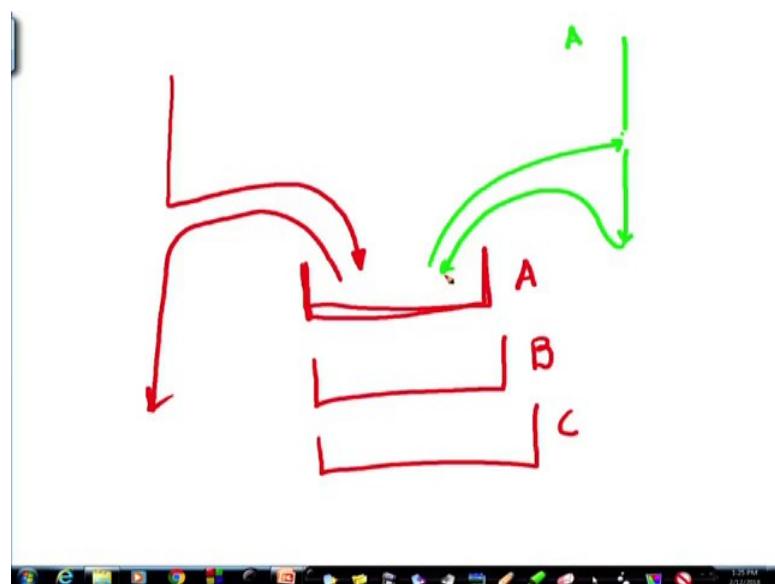
So, let us take the earlier example again if I had my main function here and I had similarly int x and here y was add 2 x here printf something x, and here I had that function add two int a and here in the body I did a assign a plus 2 and return a, return a. Now, here is a variable x and that has got an address say that address is whatever 5000.

So, now, its value here somewhere x was 5. So, its value is 5. So, here what I am in the case of reference I am not copying the value of 5 to a instead in the parameter passing I will write it in a different way not in this way which I will discuss later.

If I assume that this one a, a is not taking the value, but the address of the parameter, address of the parameter. Means what? Means that this time this a here I am passing not the value 5, but I am just simply saying that whatever data you want to work with that is the data is in this location 5000. And it expects that reference that the data that I am working on my a is actually staying 5000. So, what it does? It takes the when it computes a assigned a plus 2 this is not looking nice. So, let me write it a, a assigned a plus 2 it. Now, knows that it is not the value a is here I have to get the value from this location. So, it gets the value from this location, but no other variable location has been allocated because I know where there is only one common place, only one common place you have done given something here and you have told me where you have kept the data and I am also working in that vessel itself.

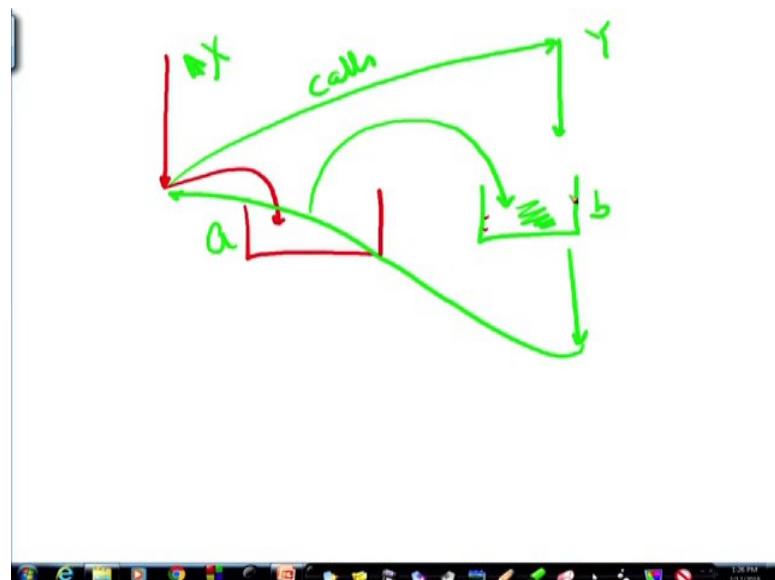
So, what is happening here is this 5 will be changed to 7, here only and return I need not return a it was just return would be sufficient. So, changes have been done here. So, in this case when I come to this printf x what will happen what will be printed 7 will be printed because the actual data has changed here. Let me show it you another example.

(Refer Slide Time: 23:03)



Here is my calling function and I think this vessel analogy will be fine. Now, it says that here is the data where I have poured it only tells you where it has poured, the name of the vessel, suppose the name of the vessel is *a* there are many other vessels A B C and only this vessel name has been passed and when this one is doing something it knows the vessel name *A*. So, it comes here and takes the data from here does something and returns the data from here and here this program when it is computing when it returns and then ultimately it returns there and while it returns it takes this value this value the change value and continues. So, in the case of calling by reference we are not copying the value. On the other hand what would happen in the case of call by value using the same vessel analogy?

(Refer Slide Time: 24:15)



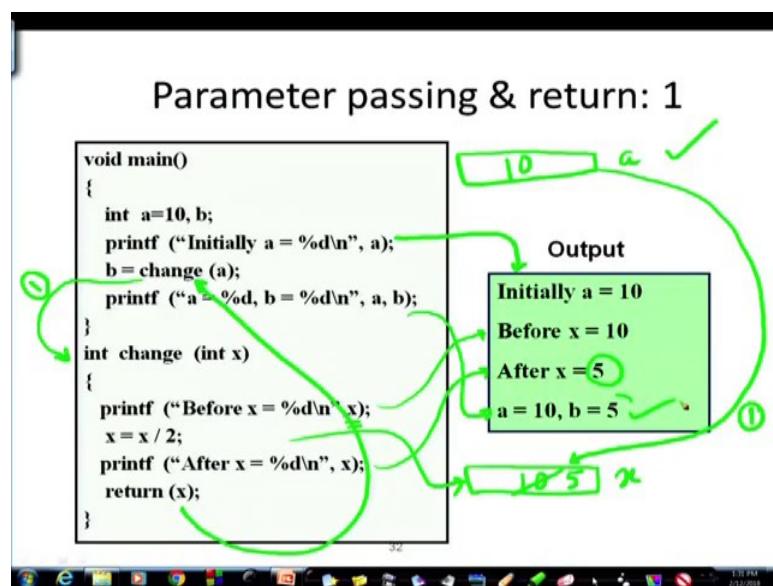
Say, this program was running the value was in a vessel the value was there in a vessel, but sorry, but my function do not bother about the inter change of colours; my function was also having its own vessel and when the function has been called the main function calls this *a* calls or say *x* calls *y* right, then also *x* copies *y*, copies the value of the variable suppose that was *a* that is copied in the vessel that is belonging to the function it may be *b*. So, this one does whatever it does here and returns this value over here. So, this one is not disturbed whatever changes are being done are, done here.

So, that is a very fundamental concept in parameter passing. There are two types of parameter passing one is call by value another is call by reference. Now so, here you see

execution of the function may affect the original because I am sharing the same vessel. Now, this in C, in C we actually carry out only call by value, only call by value except for the case of arrays except for the case of arrays there is a reason for that you will understand and for arrays we are not passing the values we are passing by reference otherwise it's always call by value.

So, let us have a look at some of the examples here, first example.

(Refer Slide Time: 27:02)



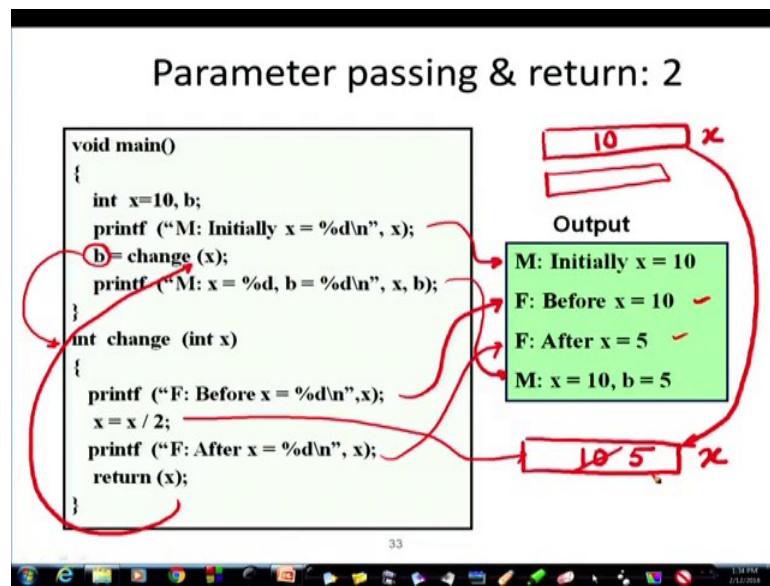
We have got the main let us see what is happening, a has been initialised to 10 and b is not initialised, printf initially a, a. So, what will be printed? Look at this, this will print first line initially a equals the value of a which is 10. So, here, here there is a variable a which has got the value 10. Then b is being assigned change a what is change a change a is a function.

So, one mistake is here I should have declared this change prototype here; however, I should have the function prototype should have been defined earlier. Now, I come here printf before in the; what does the function do prints before x x. So, before, so this point before x equal to x whatever x was, it has got the copy of that x, it has got the copy of that x. So, this it was a, a and that has been copied here for its own x, x is also 10 copied because when it called you actually copied this. When this call was made, when this call was made first then before that, before that it was also copied here and then only this was done. So, it was printed then x divided by 2. So, x becomes 5 by this statement done.

So, what is being printed? After here this line is being printed here after x equals x. So, what will be printed? I am still inside the function please remember I am still inside the function. So, 5 will be printed and then I return return x. So, what is being returned? 5 is being returned where it is being returned here, x has been a has been changed and that is going to b printf a assigned. So, this printf is this printf a sorry a equals here it I am printing a, a is a, so a is not changed. So, it is being printed as 10 and b is being printed as 5.

So, you see a has not been changed the reflection of the change has been reflected in this function and is being assigned to b, clear. Now, let us take another example and you will yourself try to look at this example and a little change has been done let us all together try to follow this example. So, a starting again a mistake is I should have declared the function prototype here. So, those things I have not shown here so, but you should do it.

(Refer Slide Time: 30:59)



Now, let us see here. Again int x equals 10. So, x is a variable to the main function x is having 10 printf M here main function, printing initially x is 10, there is calling the function. So, here the function is being called. And whenever this function is being called there is a local to the function there is an argument x where this 10 is being copied.

Now, printf here, this printf in the function it prints before x was x before changing. So, x was x. So, what will be printed? 10 will be printed. Then I change x here. So, at this

point x is becoming 5, all right. Then I am saying print after the change this is this printout after that change x is what it has been changed 5 and then I return to the main function with the change value of x, so b gets 5. Now, I am printing in the main function x is 10 and b is 5. So, it has been changed that is being reflected in this printf and b is also 5 b was here, b is in the main function. So, that b has been assigned after this change and that is also 5. Now, the distinction I think will be clear. So, so these are the two cases that we have shown.

(Refer Slide Time: 33:38)

### Parameter passing & return: 4

```
void main()
{
    int x=10, y=5;
    printf ("M1: x = %d, y = %d\n", x, y);
    interchange (x, y);
    printf ("M2: x = %d, y = %d\n", x, y);
}

void interchange (int x, int y)
{ int temp;
    printf ("F1: x = %d, y = %d\n", x, y);
    temp= x; x = y; y = temp;
    printf ("F2: x = %d, y = %d\n", x, y);
}
```

**Output**

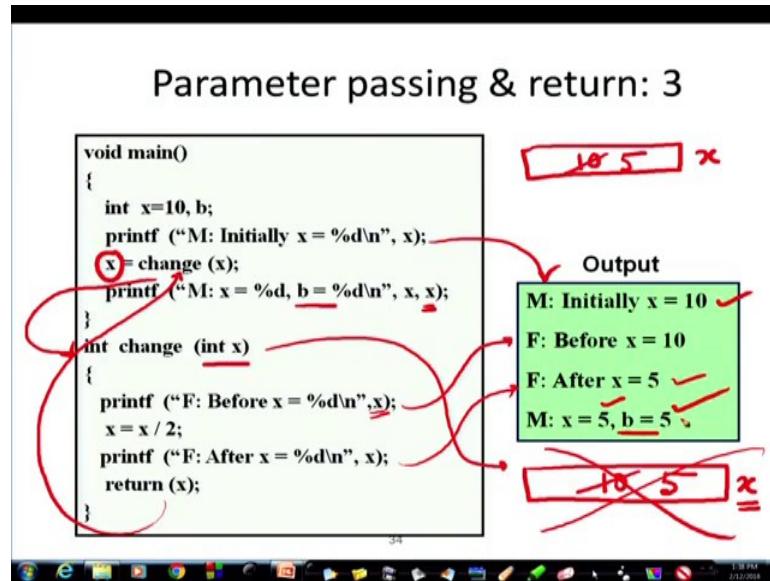
```
M1: x = 10, y = 5
F1: x = 10, y = 5
F2: x = 5, y = 10
M2: x = 10, y = 5
```

35

How do we write an  
interchange function?  
(will see later)

So, now, here let us see this is another example.

(Refer Slide Time: 33:49)



Here the slight change that has been done is that the change value I am keeping in x. Now, this was x, and x was 10. Now, note x was 10. So, initially x is 10 fine, from this line. Now, I have called change x, so it is coming here. But its parameter is also x, but that really does not matter. These x from here I will create another x. This who is the owner of this x, the owner of this x is only this function as long as this function is running these x has got a meaning it is existing other after that it is not there. But since I called it the value 10 was copied in this, but you see these two are two different memory locations. Consequently what is happening? When I print it here before x was 10. Now, see which x is being printed? This x is being printed because this x is not known to this function this function only knows its own x then x is changed to 5 that is also done to be done locally here and is being said here after that the x is 5 because this x is known then I return x. So, 5 is returned and 5 is assigned to x.

Now, whenever I have gone out of this function this x is no longer existing vanishes; that means, the compiler returns it to the memory pool now. So, which x? Is this x? This x. So, this value that is 5 that is being changed will come here. Now, here I am doing printf x x; that means, both b is, so x will be printed as 5 and b equals x. So, x is also 5. Here you see that the same value is the variable is x. So, both will be 5.

So, I hope you could understand this difference. Let us take the another example or we will come back to this in the next class and we will start with a new example and

continue with parameter passing. So, what we learnt in today's lecture is a very important concept of call by value and call by reference and we could see that in C in general call by value is adopted except for arrays and what is call by value call by value means the calling function copies the value of the parameter to the variable corresponding to the argument. Whatever change is done is done locally inside that argument and when is returned it goes back to the main function, the value goes back to the main function. Whereas, in case of call by reference there is only one vessel one variable and I am not creating another variable, I am simply passing the address of that variable to the called function and the called function changes in that particular variable and whatever changes are happening they are reflected in the main function, calling function as well.

We will continue with this.

**Problem Solving through Programming in C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 40**  
**Parameter Passing in Function Revision**

So, we had looked at parameter passing and we have looked, we have seen that the difference between call by value and call by reference and we have seen quite a few examples to be specific 3 examples on call by value how. And also another thing that was supposed to be noted I hope you have noted that, that is the scope of variables. That whenever there is an *x* in the main function and the *x* in the called function then these two *x*'s are different. The *x* that is defined in the called function is a separate location than the *x* in the main function and that the life of that variable ends with the end of the function, right.

(Refer Slide Time: 01:15)

**Parameter passing & return: 4**

```

void main()
{
    int x=10, y=5;
    printf ("M1: x = %d, y = %d\n", x, y);
    interchange (x, y);
    printf ("M2: x = %d, y = %d\n", x, y);
}

void interchange (int x, int y)
{ int temp;
    printf ("F1: x = %d, y = %d\n", x, y);
    temp=x; x=y; y=temp;
    printf ("F2: x = %d, y = %d\n", x, y);
}

```

Output

M1: x = 10, y = 5	✓
F1: x = 10, y = 5	
F2: x = 5, y = 10	
M2: x = 10, y = 5	

How do we write an  
interchange function?  
(will see later)

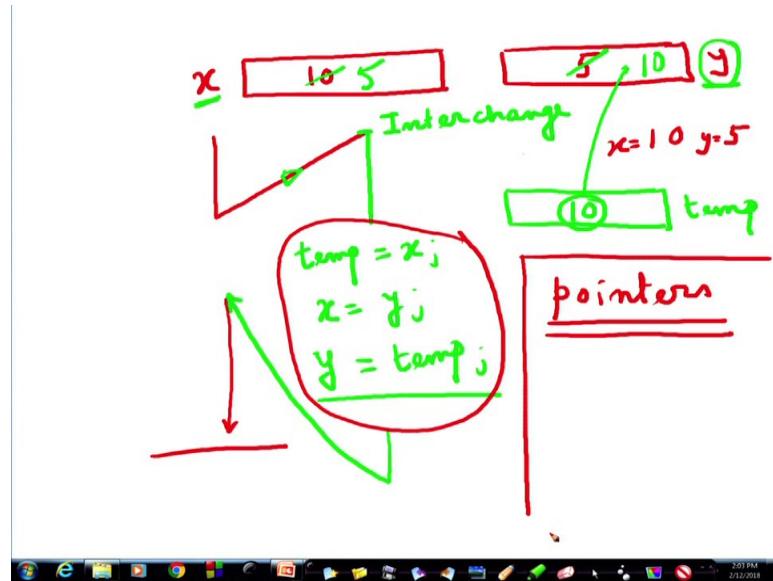
Now, let us look at these example, you yourself will be able to trace this. So, I give you some time to trace this through and then we will continue. Look at this function carefully and do not look at this green part, do not look at the green part yourself and try to without looking at that try to find out what the values will be for the different printf statements.

Let us start. So, we have got x assigned to 10 and y assigned to 5, x and y, x is 10 and y is 5. Now, so when the I printf do this printf x is printed to be 10 y is printed to be 5 fine, no issue. Then I call a function interchange x y. Look at this the type of this function interchange the type of this function interchange is void because that immediately tells us that it is not going to return anything it is going to do something as the name implies it is going to interchange x and y. And what are the parameters? Parameters are x and y this is the argument. So, here I have got another set of x and y.

Again note that this x and y are different. Now I initialize temp, so temp is another variable here temp. Now, what I am doing here? The first printf what is being done here printf x equal to x and y equal to y which x and which y the x this x and this y. Now, when I entered this function these values have been copied here. So, the printf will simply print 10 and 5 no issue now temp is getting x, x was 10. So, I put 10 here and then x assigned y sorry y assigned x; that means, the value of y I am still within the function I am within the function. So, this value is being assigned. So, by this statement what happens this becomes 5 and this remains 5 right and then what is being done here y assign temp; that means, this temp is becoming 5 right temp is becoming sorry, sorry I am sorry absolutely sorry this was 10.

Now, for this statement temp is going to y; that means, y will be changed to 10. So, here I find 5 and 10. So, it is a when it entered here I could see 10 and 5 and so when I come and print from here x will be 5 and y will be 10, I come out of the program and printf x and y. What will be printed? My god, what I find is x is being printed as 10 and y is being printed as 5 as it was here. So, no change has actually taken place. Why did it happen like this? You can immediately see the reason that whatever change took place took place here inside the function and this actually tells you the importance of the scope of variables. These variables scope ended with this function and their change was not reflected in the main here that is why although I did it here it would not change. However, suppose let us do some intellectual exercise here.

(Refer Slide Time: 06:43)



Suppose  $x$  was 10 and  $y$  was 5 and there was a  $\text{temp}$ , now  $\text{temp}$  is inside the function. So, I will use different colours for that I will, now although I remind you that C does not allow call by reference, but just to see whether you have understood call by reference properly. Suppose we are allowed to use call by reference then my main program has got this  $x$  and  $y$ . So, it comes here prints say prints  $x$  and  $y$ . So, 10 and 5 are printed  $x$  equal to 10  $y$  is equal to 5 and it calls interchange all right, it calls interchange.

And suppose just suppose that this call has been done by call by reference. So, what has been passed here? The address of this and the address of this not the values, so no other copies have been made. So, now, I have got the local variable  $\text{temp}$  and here I do  $\text{temp}$  is assigned  $x$ . So,  $\text{temp}$  gets 10 then  $x$  assigned  $y$ . Now, what is my  $x$  and what is my  $y$ ? The content of this address which have been passed will go to the content of  $x$ . So, this will be 5 and then  $y$  will be assigned  $\text{temp}$ . What will happen?

This  $\text{temp}$  this value will go to the suppose I can make call by reference then this will go to this address, but if I just simply write in this way that does not mean it really does not tell me whether I am saying that I am copying the from the address of actually copying the value, but I am just taking telling a hypothetical case. So, this 10 will go there and this 5 would be changed. In that case when I come back from here and I print here then I would have got the change scenario reflected, but the mechanism writing simply like this

in C means it is called by value not called by reference. So, what is being passed is actually the value.

So, if I could, if I could do that then it was possible to have that interchange and there is a mechanism for passing the addresses and not the actual values by using a concept called I mean structure called pointers which are nothing, but addresses we will have to we will see that later if time permits. But in general remember that this interchange has not been possible by call by value in the way we wrote the program. So, with this we complete our discussion on parameter passing normally, but a distinction between parameter passing by call by value and call by reference.

(Refer Slide Time: 11:23)

The slide has a black header bar. The title 'Passing Arrays to Function' is centered in a large, bold, black font. Below the title is a bulleted list of three items:

- Array element can be passed to functions as ordinary arguments
  - IsFactor ( $x[i]$ ,  $x[0]$ )
  - $\sin(x[5])$

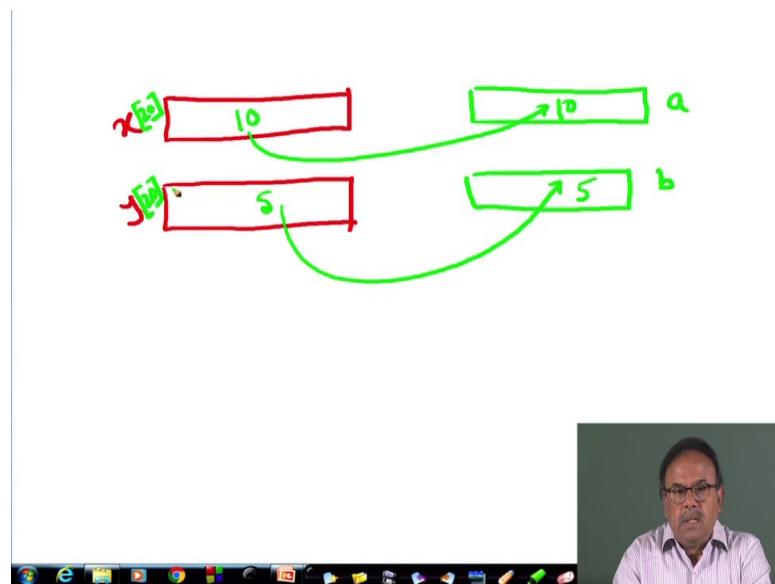
Hand-drawn annotations include:

- A green arrow pointing from the text 'IsFactor ( $x[i]$ ,  $x[0]$ )' to a green box containing the expression ' $x[i]$ '.
- A green box containing the expression ' $x[0]$ '.

In the bottom right corner of the slide area, there is a small video frame showing a man with glasses and a striped shirt, likely the lecturer. The slide also includes a standard Windows taskbar at the bottom with icons for various applications like Internet Explorer, Google Chrome, and Microsoft Office.

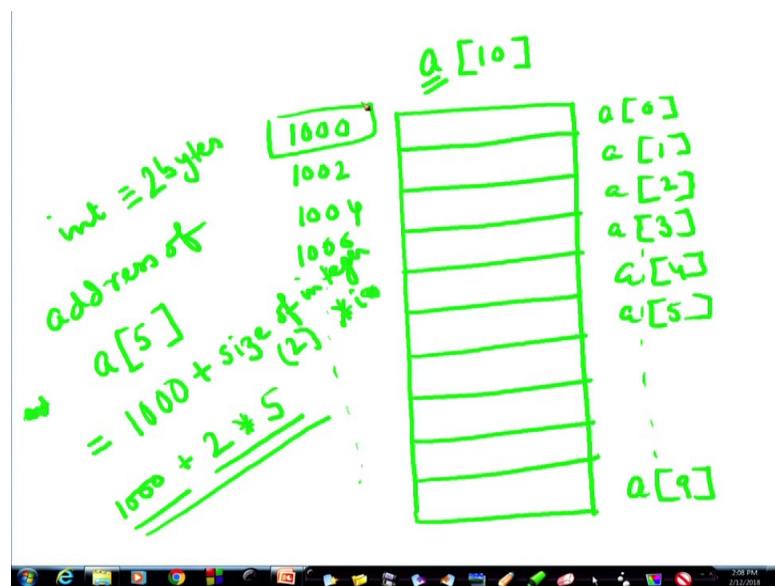
And we now start another very important point that is passing arrays to functions.

(Refer Slide Time: 11:45)



Now let us try to think. Here when we were having a variable *x* another variable *y* in the called function, in the caller function and I was having in my called function two other variables *a* and *b*, then the value of this would be copied here and the value of this was copied here. But suppose *x* is not an integer *x* is an array of 20 elements then all those 20 elements have to copy here and suppose this is an array of another 20 elements, so another 20 elements I have to copy here right. C allows only for arrays the parameter passing by difference. Now in order to understand that let us try to look at the structure of an array.

(Refer Slide Time: 12:56)

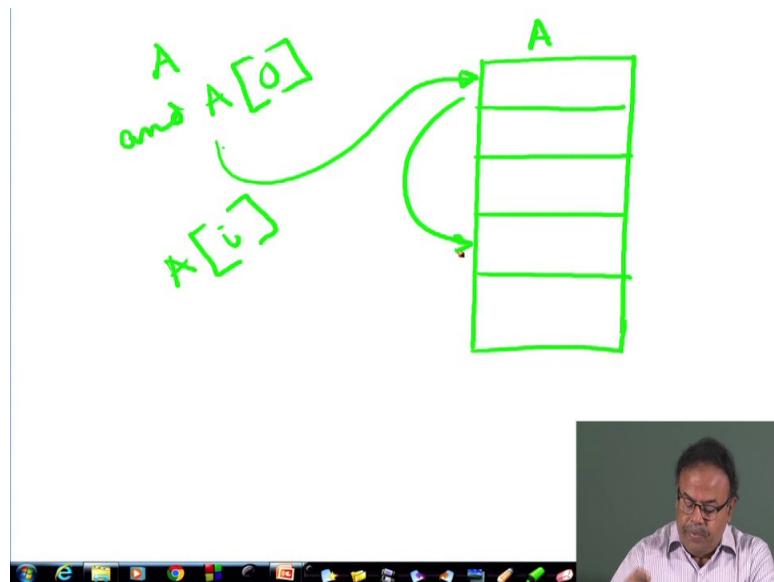


Suppose I have got an array a 10 now I am not making a distinction between the size of the array actual size of the array and the dimension of the array I am assuming that the array a has got 10 elements. So, I have got 1 2 3 4 5 6 7 8 9 10 all right and these are a 0 a 1 a 2 up to a 9, right these are the locations we know that. We also know that an array is allocated contiguous memory locations by the compiler.

So, all these are contiguous therefore, it is sufficient to know the address of the starting location, suppose this is 1000. If this is 1000 and if it be an integer and I assume that an integer, integer takes 2 bytes say 16 bits and then this will be 1002 this will be 1004 etcetera I can compute any particular address, any particular address of this. So, if I say a 5 the address of this a 5 address of a 5 can be easily computed as the starting address thousand whatever is a starting address plus the index is 5; that means, and the size of integer which I know in a particular machine say 2 bytes times how much will be 106, 108. So, what will be 2 into whatever is a index minus 1; sorry a 5, a 5 will be the 6th element right. So, 5 times whatever this is suppose this i, i times i right size of the integer times i.

So, it will be 1000 plus 2 times 5, 1010. So, this is a 0 a 2 a 0 will be 1006, a 4 will be 1008 and a 5 will be 1010. Since it is contiguous it is sufficient for me to know the starting address of the array, therefore, it is good enough to establish the correspondence between the name of the array and the starting point of the array. What I mean by that is again I draw this.

(Refer Slide Time: 16:46)



Now, I named the array differently A and suppose it has got 5 elements all right. Now A and A 0 are treated to be synonymous. When I say a; that means, I am actually referring to this element, this address, not this element this address. And since I know that therefore, A i depending on the value of i, I can compute where the actual location will be this is the fundamental concept we need to understand, all right. Therefore, we can pass the an array to a function as ordinary arguments.

For example, is factor whether  $x \mid i$  is a factor of  $x \mid 0$ , suppose I want to do that. So, you see is factor earlier I did x or y here I am writing  $x \mid i$ ,  $x \mid 0$ . So,  $x \mid i$  is what? Suppose x is an integer array. So,  $x \mid i$  is an integer  $x \mid 0$  is another integer. So, I can simply pass this, sin of a particular angle. Where is that angle? In an array x an array x is there and in that the fifth, sixth element I am taking. So, this element is coming as the parameter. So, let us proceed a little further.

(Refer Slide Time: 18:38)

A[0]

## Passing Entire Array to a Function

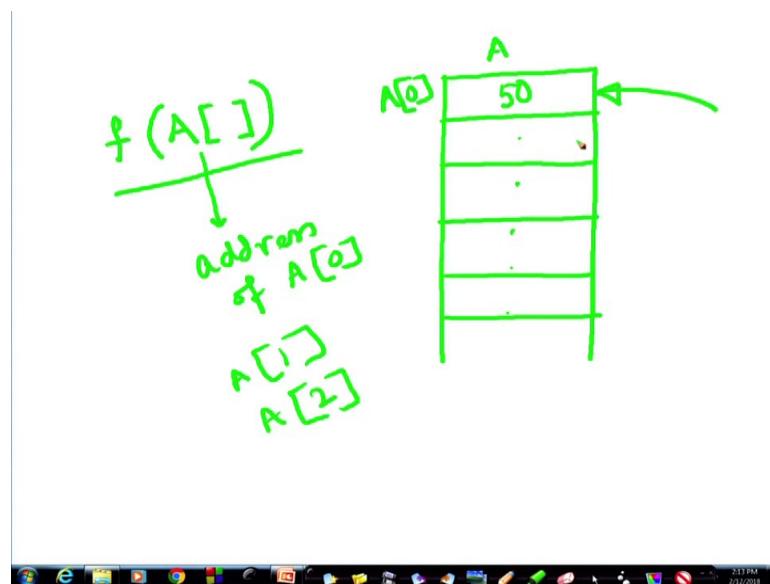
- An array name can be used as an argument to a function
  - Permits the entire array to be passed to the function
  - The way it is passed differs from that for ordinary variables
- Rules:
  - The array name must appear by itself as argument, without brackets or subscripts
  - The corresponding formal argument is written in the same manner
    - Declared by writing the array name with a pair of empty brackets

37

2:11 PM 7/12/2018

Now, that is for the individual elements  $x_i$  or  $x_5 \times 0$  I am just passing an element and if it is an integer array then an integer is being passed if it is a floating point array then a float will be passed, but what if I want to pass the entire array to a function. That is what I just now explained. That is an array name like  $A$  can be used as an argument to a function because  $A$  essentially means  $A[0]$  and if I can pass  $A[0]$  the address of  $A[0]$  is known then all the elements are known because they are contiguous.

(Refer Slide Time: 19:46)



Now, the way it is passed differs from that of the ordinary variables. Why? Here when I have got an array and array A with A 0 when I am passing the array suppose somewhere in my function f I am passing the array A and just to say that it is an area I just do something like this. This is not the correct syntax, I will show you the syntax a little later.

Now, you see I write it in this way all right. Now that means, I am passing A 0, but if I pass the value of A 0 which might be a 50 I have got no clue about the other values I am not passing all the values. So, what do I have to pass? if I pass the address of A 0 address of A 0. So, this is what is being passed is address of A 0 then I can get access to all these elements A i, A 1, A 2 anything because I can compute the address very easily as I have shown just now, all right, I can do that.

So, this is an example this is a case where we call by reference we actually will pass the address and not the value of A 0. We are passing the address of A 0 and the address of A 0 is the same as the name A this is a very fundamental concept and let us try to understand this. How is it passed? The array name must appear by itself. So, now, we are talking about the some syntaxes, the array name must appear by itself as argument without brackets or subscripts the corresponding formal argument is written in the same manner. Let us look at an example.

(Refer Slide Time: 22:01)

## Whole Array as Parameters

```

const int ASIZE = 5;
float average (int B[])
{
    int i, total=0;
    for (i=0; i<ASIZE; i++)
        total = total + B[i];
    return ((float) total / (float) ASIZE);
}

void main ()
{
    int x[ASIZE] ; float x_avg;
    x = {10, 20, 30, 40, 50};
    x_avg = average (x);
}

```

Only Array Name/address passed.  
[] mentioned to indicate that  
is an array.

B

(float) X  
Typecasted

Called only with actual array name

Say here, I am going to pass the whole array as a parameter. So, I have declared constant int a size 5. So, something some variable a size is assigned 5. Now, float average is a function which will which is taking as parameter, sorry which is taking as parameter an array B all right, just the name of the B only array name or address of B is passed. Now, this symbol is mentioned to indicate that this is an array right. Now, let us see what is happening int i total 0, for i equals 0 i less than a size; that means, less than 5 i plus plus total equals total plus b i. So, what is happening? So, here is an array B, here is an array A, a size the array size is 5 and all these elements are being added.

Now, all these elements are being added in this loop. So, B i only B 1, B 2, B 0, B 1, B 2, B 0, B 4 they are being added to total and then I am returning what am I returning I am returning total divided by a size now there is a new thing here also its better to look into that. Now, the array now average will be a floating point number and B is an integer array. So, sometimes, when i, so what has been done here is float a size when we do this float some variable x; that means, this variable is being type casted is being made to be represented as a float. So, if it was 5, 5 suppose then float A size. So, since is 5 float A size will make it 5.0.

(Refer Slide Time: 24:41)

```

const int ASIZE = 5;
float average (int B[ ])
{
    int i, total=0;
    for (i=0; i<ASIZE; i++)
        total = total + B[i];
    return ((float) total / (float) ASIZE);
}

void main ( )
{
    int x[ASIZE] ; float x_avg;
    x = {10, 20, 30, 40, 50};
    x_avg = average (x);
}

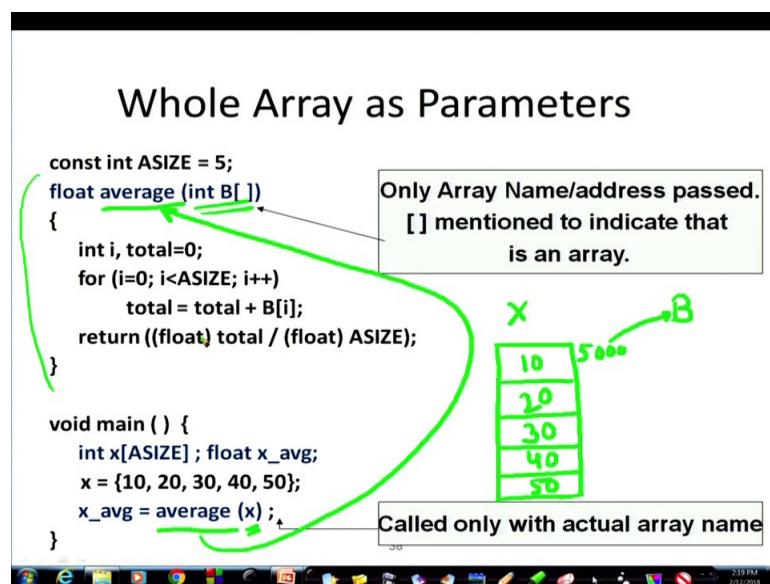
```

And suppose the array was 1 2 0 4 5. So, 9 to 11, 12 total was 12. So, float total will make it 12.0. So, I am casting twelve forcing it to be represented as a float. So, then 12.0 is being divided by 5 with 5.0 and I am getting the average and that average is being

returned here. So, that is that was not our main contention here that is the purpose of this float. But the main contention here is that I have passed this array B, but who passed it the main function. Let us study the main function.

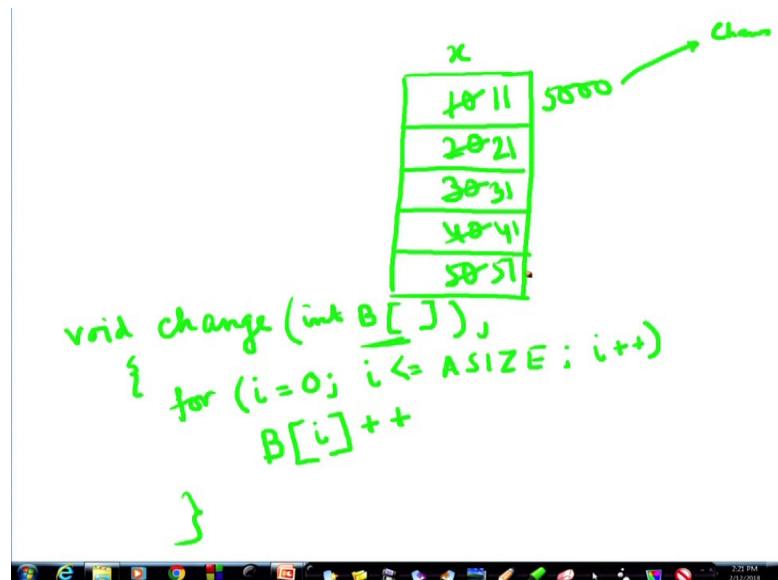
The main function x is an array of size A size that is 5 and x average is of some variable and x has been assigned as 10 20 30 40 50; x average I will compute, but I am calling this function average from here and I am passing x. What is x? x is an array and that is appearing just as a variable here and that is being accepted as B. So, what is a correspondence between x and B, x is somewhere suppose starting with location 5000 and there are 5 elements as I been said here 10, 20, 30, 40, 50.

(Refer Slide Time: 26:24)



Now, when this x is passed to B this function now knows that B is an array because it had this thing and this 5000 is passed to B. So, now, B knows where is B, B is the same array. Now, whatever change I do here, suppose in this function instead of computing the total if I had written this function in a different way for example let us do that and suppose I have got this function x, I have got this function x this array, sorry I am sorry x where 10 20 30 40 50 are there and in my function change, I can also say void change int B this and in the body of the function what I do is for please try to understand i assign 0 to this, i less than equal to A size or 5 whatever it is, i plus plus, B i plus plus say this. What will happen?

(Refer Slide Time: 27:29)



Inside this function change I will take the first one change it to 11, this one will be 21, this one will be 31, this only 41, this only 51 and when I return I made a change in B, but B and x are the same because this address 5000 was passed to the function change and this b got 5000 and so whatever change has been done here will be reflected in the main function when I return, clear.

So, that is the importance. So, here I have called it with the actual array name here of course, I computed a total and return some other value, but if I had just change it inside this function I could have made it void function and the change would have been automatically reflected in the name.

(Refer Slide Time: 29:47)

**Contd**

```
void main()
{
    int n;
    float list[100], avg;
    :
    avg = average (n, list);
    :
}

float average (int a, float x[])
{
    :
    sum = sum + x[i];
}
```

We don't need to write the array size. It works with arrays of any size.



39

2:21 PM 7/12/2018

So, here you see we do not need to write the array size it works with arrays of any size. For example, here void this etcetera where list is an array of 100 elements and average I am calling here float average int a is an integer, and float x is map to list all right, x is an array because here how is the correspondence done. This list, list is being mapped to this same address is being passed and n is being passed to a how many elements are there, all right. So, I need not specify the size because the size is already specified here. I have got 100 element array, this is my list, this is list right. So, I have got 100 element here I know. What I have passed to x is just the address. So, whatever it is 100 or 150 that be true for x also right. So, we do not need to write the size of the array.

(Refer Slide Time: 31:12)

```
void VectorSum (int a[ ], int b[ ], int vsum[ ], int length){  
    int i;  
    for (i=0; i<length; i=i+1)  
        vsum[i] = a[i] + b[i] ;  
}  
  
void PrintVector (int a[ ], int length) {  
    int i;  
    for (i=0; i<length; i++) printf ("%d ", a[i]);  
}  
  
void main () {  
    int x[3] = {1,2,3}, y[3] = {4,5,6}, z[3];  
    VectorSum (x, y, z, 3);  
    PrintVector (z, 3);  
}
```

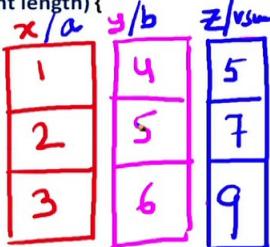
Now, similarly array is used as output parameters. So, suppose I am going to do vector sum; that means, I have got two arrays a and b. So, what I am doing here? I have got two arrays a, an integer array another integer array b and I am adding them. So, I will add this element say 5 with this 7 and I have got another array v sum, v sum where the sum will be stored so this will be 12, if it was 6 and this was 5 here we will store I will add these two and I will store 11 ok.

So, let us see how the vector sum. Now, vector sum will be void you know because it is doing the addition and the addition is remaining in this vector sum. So, I have got let us start with the main function x is an array. So, this is actually x a or let me say let me put the main function first. So, it will be x in the main function and a in the this function y slash b. I am writing slash b because these two was the same because the address is shared all right and vector sum will be z. So, z or vector sum. So, x is 1 2 3, y is 4 5 6 and z, z has not been initialized because, so let me let me; so that you are not confused, let me do it like this.

(Refer Slide Time: 33:28)

### Arrays used as Output Parameters

```
void VectorSum (int a[ ], int b[ ], int vsum[ ], int length) {  
    int i;  
    for (i=0; i<length; i=i+1)  
        vsum[i] = a[i] + b[i];  
}  
void PrintVector (int a[ ], int length) {  
    int i;  
    for (i=0; i<length; i++) printf ("%d ", a[i]);  
}  
  
void main () {  
    int x[3] = {1,2,3}, y[3] = {4,5,6}, z[3];  
    VectorSum (x, y, z, 3);  
    PrintVector (z, 3);  
}
```



The diagram illustrates the state of three arrays at the end of the VectorSum function's execution. Array x (red border) contains values 1, 2, 3. Array y (purple border) contains values 4, 5, 6. Array z (blue border) contains values 5, 7, 9. Handwritten labels above the arrays indicate x/a, y/b, and z/vsum.



I have got an array that is x having values 1 2 3, another array y having values 4 5 6 and another array, another array z which will have the final value and z all right it is 0 elements, but we do not know the value. Now, I am calling vector sum and what am I passing x is being passed to this x goes to this. So, this is known as a for the function y goes to this. So, this is known as b. So, x and a locations are shared z and v sum the locations are shared, all right.

Now, I am calling vector sum. So, all these are being added in a loop 5 7 9, then I come out I come out. Now, you see when I come out of this function these are already reflected because it was passed by reference. Now, let us look at the print, print vector what is happening here. Void print vector that is another function that I am calling here z 3; that means, 3 elements will be printed from that z array z array was here which had that elements like 5, 7 and 9.

(Refer Slide Time: 35:29)

Arrays used as Output Parameters

```
void VectorSum (int a[ ], int b[ ], int vsum[ ], int length) {  
    int i;  
    for (i=0; i<length; i=i+1)  
        vsum[i] = a[i] + b[i];  
}  
void PrintVector (int a[ ], int length) {  
    int i;  
    for (i=0; i<length; i++) printf ("%d ", a[i]);  
}  
void main () {  
    int x[3] = {1,2,3}, y[3] = {4,5,6}, z[3];  
    VectorSum (x, y, z, 3);  
    PrintVector (z, 3);  
}
```

vector-sum.c

Now, with that print vector it is just in a loop  $i$  equal to 0,  $i$  less than length. What is length? Length is 3 and what is the array  $z$  is  $a$ . Now, this  $a$  and this  $a$  are different again, this  $a$  and this  $a$  are different. This  $a$ , this is this  $a$  was this functions  $a$  and this  $a$  is this function  $a$ . So, they are different. So, here they are printing 5, 7, 9. So, that is as an output parameter.

So, you see actually whenever I am reflecting on an array I need not pass it because essentially automatically passed right. So, that is what is very important. So, I hope you have understood this.

(Refer Slide Time: 36:29)

## The Actual Mechanism

- When an array is passed to a function, the values of the array elements are **not passed** to the function
  - The array name is interpreted as the **address** of the first array element
  - The formal argument therefore becomes a **pointer** to the first array element
  - When an array element is accessed inside the function, the address is calculated using the formula stated before
  - Changes made inside the function are thus also reflected in the calling program

41



The actual mechanism is when an array is passed it is the array elements are not passed, but what is passed is an address and the argument becomes a pointer to the first element or a pointer to the first element; that means, the address of the first element. And when an array element is accessed inside the function the address is calculated I have already explained the formula that is used for finding the array. So, this is known as call by reference.

(Refer Slide Time: 37:04)

## Contd.

- Passing parameters in this way is called **call-by-reference**
- Normally parameters are passed in C using **call-by-value**
- Basically what it means?
  - If a function changes the values of array elements, then these changes will be made to the original array that is passed to the function
  - This does not apply when an individual element is passed on as argument

42



So, I think you have understood this. We will continue our discussions in the future lectures on some other important issues.

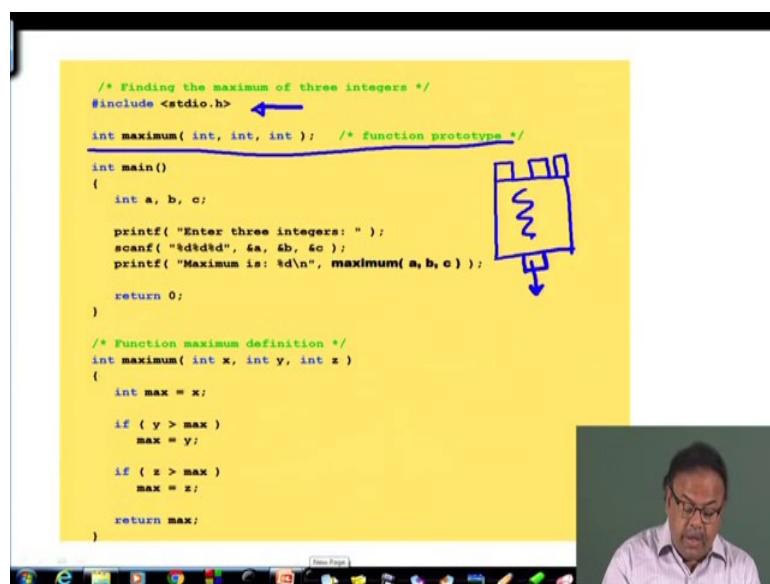
**Problem Solving Through Programming in C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 41**  
**Substitution of # include and Macro**

We have discussed about passing arrays as parameters. And for passing arrays as parameters in C to, C functions we take the course to call by reference. Otherwise for all other parameter passing we take the course to call by value. And just to summarize the call by value and call by reference. Call by value we copy the value of the actual parameter to the formal parameter; formal argument and in call by reference we do not copy the value, but we just pass on the address of the pointer and reference to the actual parameter ok.

So, as a result; what happens? That if any change to that variable is resulted in the function itself, in the case of call by value that will not be automatically reflected to the main program, but in the case of call by reference since there is no other separate copy that will automatically be reflected. Just to conclude that part we will see a number of applications of functions just to come to conclude that.

(Refer Slide Time: 01:44)



Let us have a quick look at 1 function, but there is no idea, but just vision of the prototypes that you can see here, finding the maximum of 3 integers.

So, we have the header file, here include studio dot h. Then here we declared a function prototype. The actual function maximum has been written later. So, here you see that what does this line tell us? This line tells us that maximum is a function and it consists of 3 integer arguments, or 3 integer parameters will come. So, as if this is a function which has got 3 input doors and each door is wide enough to accommodate an integer ok. So, that much is told. And whatever is inside this I am not aware of that, and at this stage I am not aware of that. And the output will also be another integer that is told by this.

Then so, here the salient point to see is that here I have just it was sufficient to just specify the types, no names have been put in. Now comes the main program that is very simple a, b, c are 3 integers.

(Refer Slide Time: 03:19)

```

/* Finding the maximum of three integers */
#include <stdio.h>

int maximum( int, int, int ); /* function prototype */

int main()
{
    int a, b, c;

    printf( "Enter three integers: " );
    scanf( "%d%d%d", &a, &b, &c );
    printf( "Maximum is: %d\n", maximum( a, b, c ) );

    return 0;
}

/* Function maximum definition */
int maximum( int x, int y, int z )
{
    int max = x;

    if ( y > max )
        max = y;

    if ( z > max )
        max = z;

    return max;
}

```

So, they are local to this function, enter the integers abc are interred here. So, you know; what a is? what b is, and what c is, and then you are calling inside c. Here inside the print f I can call maximum with the parameters a, b, c and we come here. And here I have defined here I did not name here I did not name the variables; here I have defined the names of the variables int x, int y, int z. So, this x, y, z again property of this maximum alright.

So, xyz are here. And the algorithm of the logic is simple. Initially I make max to be x; if y is greater than max and y is the max, otherwise z is the max and then I am returning max. I am returning max means I am coming here, and then that max is being printed.

And returns 0 now my main function, where is my main? Main is here, main has got a type int. So, I am returning 0. So, int will if at the end of, the main function a 0 is returned, I will assume that the program has successfully completed ok. The reason I brought up this example is just to illustrate this case that you need not at the prototype level you can escape specifying the parameters.

(Refer Slide Time: 05:06)

The screenshot shows a C program with three blue callout boxes on the right side:

- Prototype Declaration**: Points to the line `#include <stdio.h>`.
- Function Calling**: Points to the line `maximum( a, b, c );` in the `main()` function.
- Function Definition**: Points to the `maximum()` function definition below.

```
/* Finding the maximum of three integers */
#include <stdio.h>

int maximum( int, int, int ); /* Function prototype */

int main()
{
    int a, b, c;

    printf( "Enter three integers: " );
    scanf( "%d%d%d", &a, &b, &c );
    printf( "Maximum is: %d\n", maximum( a, b, c ) );

    return 0;
}

/* Function maximum definition */
int maximum( int x, int y, int z )
{
    int max = x;

    if ( y > max )
        max = y;

    if ( z > max )
        max = z;

    return max;
}
```

So, here is a prototype declaration, function calling, and function definition you know that.

(Refer Slide Time: 05:11)

The slide title is **Calling Functions: Call by Value and Call by Reference**. The content is a bulleted list:

- Used when invoking functions
- Call by value
  - Copy of argument passed to function
  - Changes in function do not effect original
  - Use when function does not need to modify argument
    - Avoids accidental changes
- Call by reference
  - Passes original argument
  - Changes in function effect original
  - Only used with trusted functions
- For now, we focus on call by value

So, this are already repeated I am not going into that.

(Refer Slide Time: 05:18)

An Example: Random Number Generation

- **rand function**
  - Prototype defined in `<stdlib.h>`
  - Returns "random" number between 0 and `RAND_MAX` (at least 32767)
    - `i = rand();`
  - **Pseudorandom**
    - Preset sequence of "random" numbers
    - Same sequence for every function call
- **Scaling**
  - To get a random number between 1 and n
    - `1 + ( rand() % n )`
      - `rand % n` returns a number between 0 and `n-1`
      - Add 1 to make random number between 1 and n
    - `1 + ( rand() % 6 ) // number between 1 and 6`

46

(Refer Slide Time: 05:23)

```
1 /* A programming example
2  Randomizing die-rolling program */
3 #include <stdlib.h>
4 #include <stdio.h>
5
6 int main()
7 {
8     int i;
9     unsigned seed;
10
11    printf( "Enter seed: " );
12    scanf( "%u", &seed );
13    srand( seed );
14
15    for ( i = 1; i <= 10; i++ ) {
16        printf( "%10d ", 1 + ( rand() % 6 ) );
17
18        if ( i % 5 == 0 )
19            printf( "\n" );
20    }
21
22    return 0;
23 }
```

Algorithm

1. Initialize seed
2. Input value for seed
  - 2.1 Use `srand` to change random sequence
  - 2.2 Define Loop
3. Generate and output random numbers

48

So, instead I will be talking about a new thing here, that is hash include.

(Refer Slide Time: 05:26)

## #include: Revisited

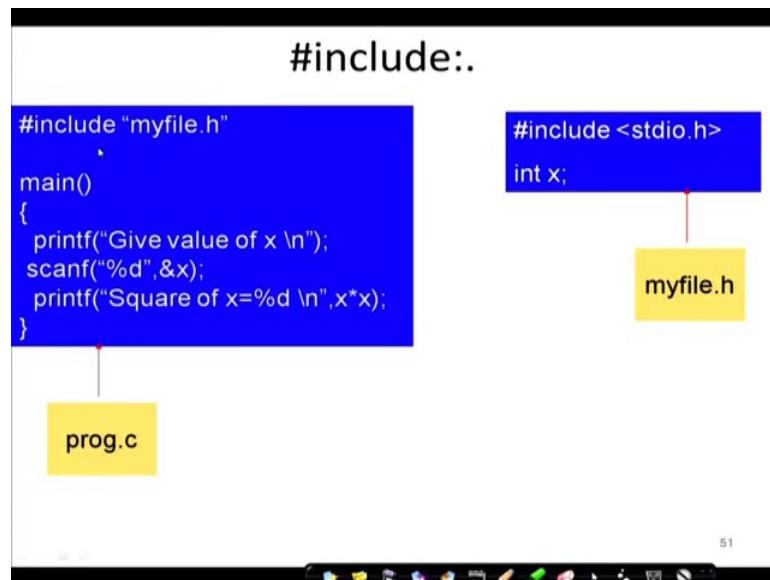
- Preprocessor statement in the following form  
`#include "filename"`
- Filename could be specified with complete path.  
`#include "/usr/home/rajan/myfile.h"`
- The content of the corresponding file will be included in the present file before compilation and the compiler will compile thereafter considering the content as it is.

50

We have talked about hash include, we have used hash include. And we mentioned that hash include is nothing but a preprocessor statement. A preprocessor statement how? Say hash includes filename. We have seen hash include studio dot h, but it can be for that matter any file name, hash include file name.

Say for example I can specify the full path, say slash usr in a Linux environment slash home rajan my file dot h. I want to include this file; I want to include this file in mind for serving. This file in my program, all right; the content of the corresponding file will be included in the present file, before compilation is d1 before compilation is d1. And the compiler will compile there after considering the content as it is. So, it is a preprocessor statement. That is, I put that inside that hash include that file is included and then the compilation is d1 ok.

(Refer Slide Time: 07:01)

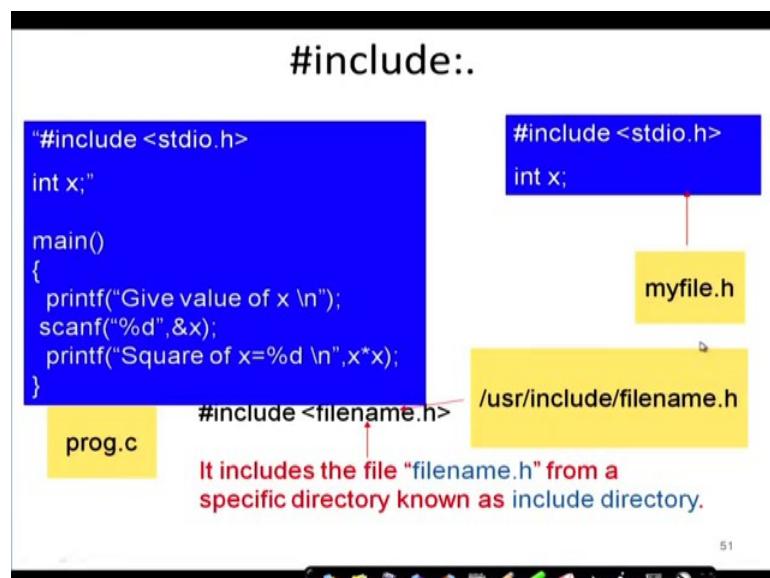


51

Now, let us look at this hash include example. Look at this program segment which is my source program, which is known as proc dot c. So, this is my source program. This source program has got hash include no stdio dot h, but it has just said that in hash include my file dot h, and then a main program follows ok. Now, this my file dot h is nothing but another file, which is which has got hash include stdio dot h and index.

So, when I include my file dot h, then this particular file will be included this particular file that is this segment will be included here.

(Refer Slide Time: 07:59)



51

So, consequently what will happen is when I say in general hash include file name dot h it includes a file name file, file name dot h from a specific directory which is known as the include directory. When I say hash include it takes from the include directory, and in the include directory I have kept my file dot h, and consequently when that is included the ultimately the thing that looks like is, it will be it will look like this.

Because that hash include I am sorry, there should not be an inverted comma here, it should be just hash include stdio dot h followed by intex. So, this whole thing whole thing has been has replaced my file dot h. And where was my file dot h? My file dot h was under user usr include file name dot h. So, that is how the include I can have other files included in my program. Now another point of critical discussion a is macro definition. We have seen the macros like hash define.

(Refer Slide Time: 09:06)

#define: Macro definition

- Preprocessor directive in the following form  
`#define string1 string2`
- Replaces the *string1* by *string2* wherever it occurs before compilation, e.g.  
`#define PI 3.14`



So, let us so, that is again a preprocessor directive. We have seen that hash include is a preprocessor command; that means, even before the compilation is d1; that is, that is copied there on the other hand, you see the preprocessor directive of hash defined string 1 string 2; that means, string 1 should be replaced by string 2. So, how does it happen? It replaces string 1 by string to whenever it occurs, where ever it occurs, where ever it occurs before compilation.

So, for example, hash defined pi to be 3.14. So, wherever it will find this pi, wherever it will find this pi, it will replace that with 3.14.

(Refer Slide Time: 10:19)

```
#define: Macro definition
```

```
#include <stdio.h>
#define PI 3.14
main()
{
    float r=4.0,area;
    area=PI*r*r;
}
```

```
#include <stdio.h>
main()
{
    float r=4.0,area;
    area=3.14*r*r;
}
```

52

So, here for example, you see, we have got hash include stdio dot h, then we define pi to be 3.14 as the hash define preprocessor command. Now in my main program I have got float r 4-time 4; r is 4, and area is another variable both are float. Now area is pi times r times r, r is already initialized. So, what will happen is this will be translated to this pi will be replaced. So, the body will be float r assigned 4 and area.

And area is so, r is 4, and area will be 3.14 times r times r. So, before come even before compilation, this pi is being replaced by 3 of the value 3.14 as has been stated in the hash defined command.

(Refer Slide Time: 11:36)

## #define with argument

- #define statement may be used with argument e.g.

```
#define sqr(x) ((x)*(x))
```

53

We can also give hash define with an argument. Till now we have seen hash define pi as a constant hash define some constant k to be 5 or whatever. Here we can also define some functions. Like you see hash define square x is x times x. So, wherever square x will appear, that will be replaced by x times x ok.

(Refer Slide Time: 12:17)

## #define with argument

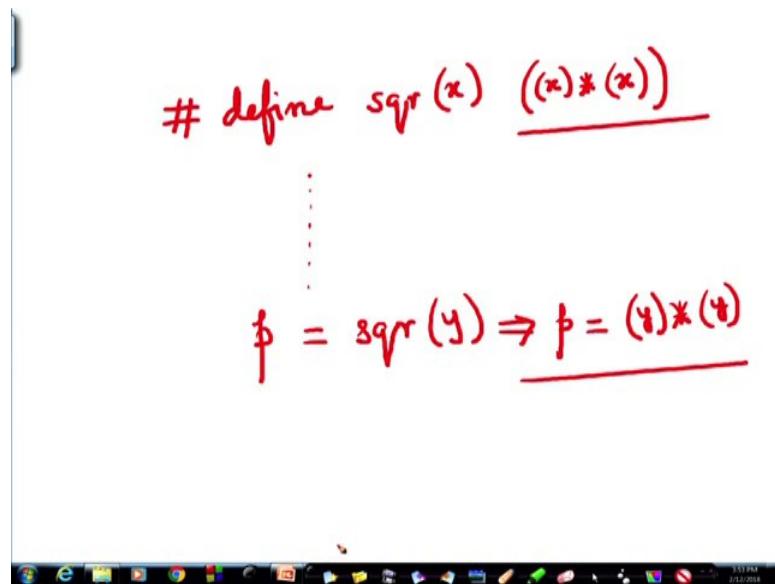
- #define statement may be used with argument e.g.

```
#include <stdio.h>
#define sqr(x) ((x)*(x))
main()
{
    int y=5;
    printf("value=%d\n", sqr(y)+3);
}
```

53

Say for example, here let us look at this program again. I think you are not being able to read this. So, let me go back say let me have if I have, declared say, hash define square x is x times x.

(Refer Slide Time: 12:46)



Then in my program, wherever I will get say, p is assigned square y. So, this will be translated as p assigned y times y ok. So, this; what should be d1 that has already been told here that square x is this. So, this will simply replace it ok. So, that is the purpose of hash define here, we can see that and so, let us look at this example again.

(Refer Slide Time: 14:01)

#define with argument

- #define statement may be used with argument e.g.

```
#include <stdio.h>
main()
{
    int y=5;
    printf("value=%d \n", ((y)*(y))+3);
}
```

sqr(x) written as macro definition      Which one is faster to execute?

sqr(x) written as an ordinary function?

Main y is 5, print f value is y times y; that means, this is being repeat this is what has been generated after the square x has been square here it was square x plus 3. So, that has become y times y plus 3. Now, which 1 is faster to execute? You can think of I can if I

write square x written as a macro definition, then I am pasting as if I am pasting the body of the code inside this in the main program replacing that. So, I do not need to call any function. I can write it as a macro definition and I as a result even before compilation.

Even before compilation, I can get this look. And so, this can be straightaway applied without requiring to call a function. Now if square x is written as an ordinary function, not like a macro definition, in that case I have to call a function, and calling a function has got some overheads which I am not discussing here. So, if it be a very simple thing like this it is better to make it a macro. So, that it automatically gets pasted and it becomes faster.

(Refer Slide Time: 15:40)

The screenshot shows a presentation slide with a black header bar. The main title is '#define with arguments: A Caution'. Below the title is a bulleted list:

- #define sqr(x)  $(x*x)$ 
  - How macro substitution will be carried out?  
r = sqr(a) + sqr(30);  $\rightarrow$  r = a\*a + 30\*30;  
r = sqr(a+b);  $\rightarrow$  r = (a+b)\*(a+b);

At the bottom of the slide, there is a small image of a man with glasses and a blue shirt, likely the speaker. The Windows taskbar is visible at the very bottom of the screen.

However, there is a word of caution for example, if I had defined square x, now it is now when I am defining as a macro it is I have to ensure that it is correct. Therefore, if I define square x as x times x. Now, how the macro definition substitution will be carried out? Let us look at this. If there be something like r assigned square of a plus square of 30.

So, it will be a times a plus 30 times 30. That is how it will happen ok, now what about r square of a plus b? It will be simply pasted as a plus b times a plus b. Consequently, the result will be wrong, because in the during execution b times a will be d1 first, and then that will be added with a and with b. So, this is not what I intended. Therefore, if I had d1 this, define this as like this, and like this, this problem would not have occurred.

So, unless I do that this a plus b should will not be I need to take if I define it like this then it becomes this which is the correct 1. But unless I do that it will lead to a wrong result.

(Refer Slide Time: 17:16)

#define with arguments: A Caution

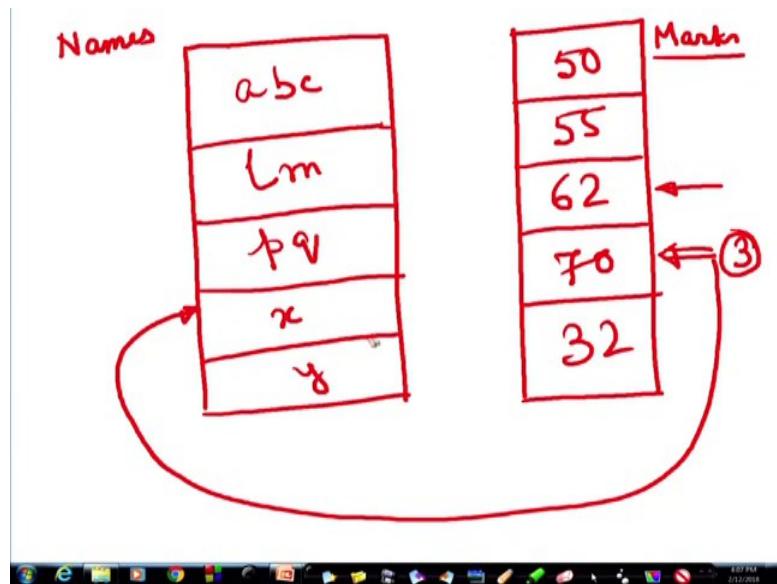
- `#define sqr(x) x*x`
  - How macro substitution will be carried out?  
 $r = \text{sqr}(a) + \text{sqr}(30); \rightarrow r = a*a + 30*30;$   
 $r = \text{sqr}(a+b); \rightarrow r = a+b*a+b;$  WRONG?
  - The macro definition should have been written as:  
`#define sqr(x) (x)*(x)`  
 $r = (a+b)*(a+b);$

So, you must be very careful about it, this is wrong. And the macro definition should have been written as I have shown square x is x times x, in that case we would have got the correct result. Now before moving to anything else, we will now look at some applications of what we have learned. We are now at a position where we have learnt all the fundamental tools, without the sophistications, that are required to write a program ok.

So, suppose I want to solve a problem like this. That, I have got 5 students in the class; and the 5 students have got 5 names of course, and each of them have got some marks, total marks, right. Now how do I represent that? And I want to find out which student I want to print the name of the student who has got the highest marks, all right? Is the problem clear let me repeat the problem, I have got 5 students in a class? And I want to find out which student has got the highest marks. And I want to know the name of that student ok.

How can I solve this problem? This is a simple problem. Here what I will need is, first I will need an array of the names of the students. So, first I need an array of the names of the students.

(Refer Slide Time: 19:13)



So, here 5 names will be stored very, very disproportionate array. And let me call this array to be name or names. And on this side, I have got another array, which holds the marks order of the different students. Corresponding to the roll number maybe 1, 2, 3, 4, 5 the marks are stored here. Somebody got 50, somebody got 55, somebody got 62, somebody 70, somebody 32. Say, and their names are a, b, c, l m p q x and y; suppose, these are names, these are the different names. Now why do I need to add is, these are this; the array called marks. Why do I need 2? 2 arrays, first of all the arrays should be holding same type of data.

So, what is the type of data that the array names are holding? It is a array of character or character string. Now how can I represent these names? I can represent them as an array of an array of strings, or each of these names is what each of these names is a character array. Therefore, if I just look at names, names will look like a 2-dimensional array. Now what would be my algorithm first of all come to the representation.

Later, what would my; be my algorithm? First of all, I will have to go to this mark array which is simpler it is an array of integers, assuming all integer marks are given. Then I will have to carry out find out the maximum of this. And you can write a function to find the maximum of an array, right. So, let us try to write the function first so, I have got an array marks; which has got 5 elements, all right.

(Refer Slide Time: 22:22)

```
int maximum (int M[]),  
{ int i=0; maxindex;  
int max = M[0];  
for (i:=1; i<=4; i++)  
{ if M[i] > max  
    max = M[i];  
    maxindex = i;  
}  
return (maxindex);  
}
```



So, I can call a function max, what let us first of all think of this maximum value what will it return me? Say, if we will find out say if I come to this, it will find out which 1 is the maximum, and will return me an index, right or this is not the maximum sorry this 1 is the maximum.

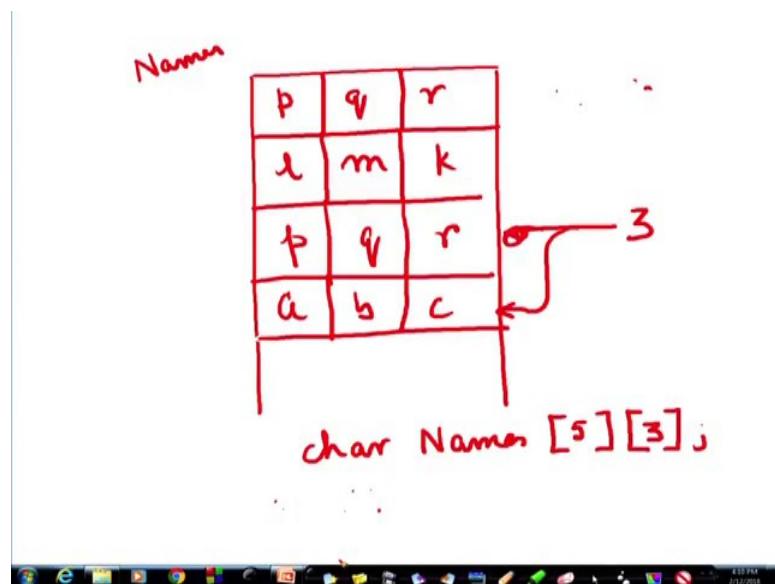
So, it will return me an index 0 1 2 3 so, it will return me an integer. Therefore, I can start with like this that int maximum of I can call an array, integer array int suppose that is m, this and the main function will call it with marks. And inside this what will happen? You know, initially let me call a value int this is a local max assigned m 0. So, I initialize a variable max with m 0. Now for I assign some index i.

So, I have to declare I also here. I assigned 1 i less than equal to 5 less than equal to 4, sorry, less than equal to 4, i plus

. Here if mi is greater than max, then if I then max is mi. And that will go on. Now ultimately what shall I turn from this? Should I return the max, no, I am returning the value of i. Because, where I found the maximum now? So, for that now this I is changing. So, here I will also have to make say. So, let me just make this I to be 0, let me make this i to be initially i is 0. So, I could have d1 away with this. And wherever I get this max I am changing this i, whenever I am getting this max I can say that there is another max index. And I am making max index equals i. So, I am remembering where I found the max and I will be returning the max index, all right.

So, my maximum function in this way, what it will do is it will find out the maximum so, it will return 3. So, the first thing is to find max. Now I have got an array names, how does that represent it? The representation can be as strings or it can be as an array of characters. If I consider this to be an array of characters, then remember that it is a 2-dimensional array ok. And I will take in that case I will take. So, this 3 and I will come to this 3 this, and I will print out this particular element from that array ok. So now, what about these names let us see. Names is an address a p, q, r or something.

(Refer Slide Time: 27:59)



So, I can have this as an array of characters. So, p q r l m whatever it is, and this part is blank. So, what is my declaration of names? Names will be there are 5 names, 5 rows and each row is a character.

So, it is a character array char names, let me write it here. Char names 5 rows, and each row may have 3 columns, right. At best like this lm p lm k pq r like that say. Now I have supposed come with that maximum index was this rows so, that is 3. Now how do I print this pq r name here, if it is kept as a string I could have straight with percentage s format. But here how would I do with this here I can, I will have I will print which row I will print. I am not writing the entire code ok. Print f which row would I print I had print the names 3, but there are 3 characters. So, names 3 in a loop I have to print right. So, print f so, how would the printing be d1? How do I print? 1 rows of an array a 2 dimensional so,

here I come. So, I will print in a loop names 3 0 3 1 3 2 ok. Actually, this is 3 should be somewhere here, it is confusing 3 should be somewhere here say a, b, c.

So, 3 is actually this row. So, names I will be printing 3 0 3 1 3 2 and that I can print in a for loop. So, I leave it to you to try to print this in the character format. You can also try it in the string format. So, what we have d1 here is, we have seen the how the macros are replaced, and we will see further examples of functions all through in the future lectures, where we will be applying functions and arrays consequently.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 42**  
**“search” as a function**

We have seen functions in the earlier lecture. And we have also seen we tried to solve a problem with where there were names of the students in one array, and the marks in another array. And we tried to find out try to print out the name of the student who got the highest marks so, for that we had 2 different arrays. Now the same thing I can do today say I have 10 students.

(Refer Slide Time: 00:51)

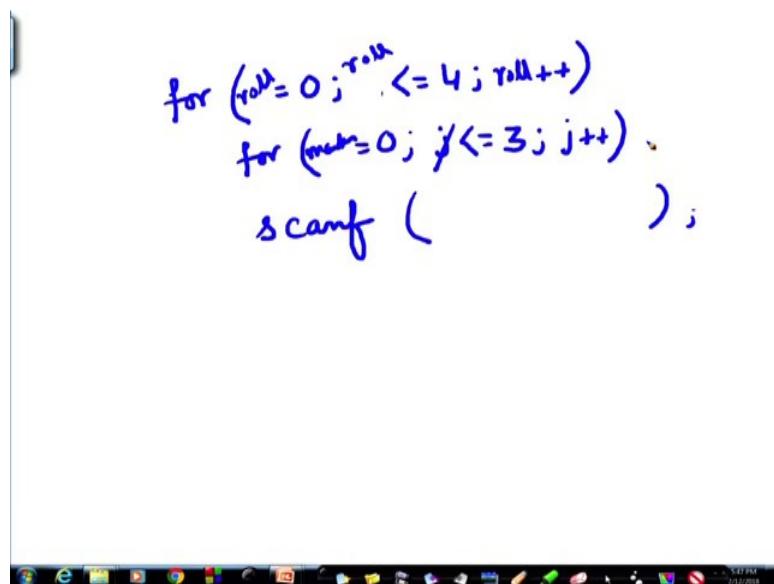
Roll.Nr	Marks	L	S	M	H	T
1		50	25	70	35	~
2		70	20	90	15	~
3		35	75	90	80	~
4		20	25	60	30	~
5		10	20	30	40	~

And each of them have appeared for, I am so sorry, each of them has appeared for 4 subjects, all right. Or let us say there are 5 students, and each of them has appeared for 4 subjects. So, how do I, what would be a convenient way of representing them? One convenient way of representing them would be to have a 2-dimensional matrix, 2-dimensional array, where on this side I will have the roll number of the students, which is an integer and I have got 5 students. So, 1, 2, 3, 4 and 5, 5 students, and the marks as the columns right, for any problem solving we will have to think of how can we represent the data so that our computation is made easy. So, at our computation is facilitated.

So, suppose there are 5 students say roll number one, roll number 2, roll number 3, roll number 4, roll number 5, and for each of them we have got 4 columns for the 4 subjects, right. Suppose the subjects are say language science, mathematics and say history, 4 subjects right. And so, how would I read the marks? How would I first of all store the marks of the students, you can very easily understand now, now I am not writing the program I am leaving the writing of the program to you.

But let us try to understand how it can possibly be solved. So, I will have a function, I can write a function of acquiring data. What will that do? For on this side, I can have an index i, that will talk about the row that I am talking about; that means, which student I am talking about and another index maybe j or instead of i j you can say roll and marks, that is also fine ok. So, keeping I fixed, and then you can fill up the row of the marks of the student. So, how would that program segment look like? That program segment will look like for i assigned 0, i less than equal to 4, because there are 5 students, i plus plus.

(Refer Slide Time: 04:04)



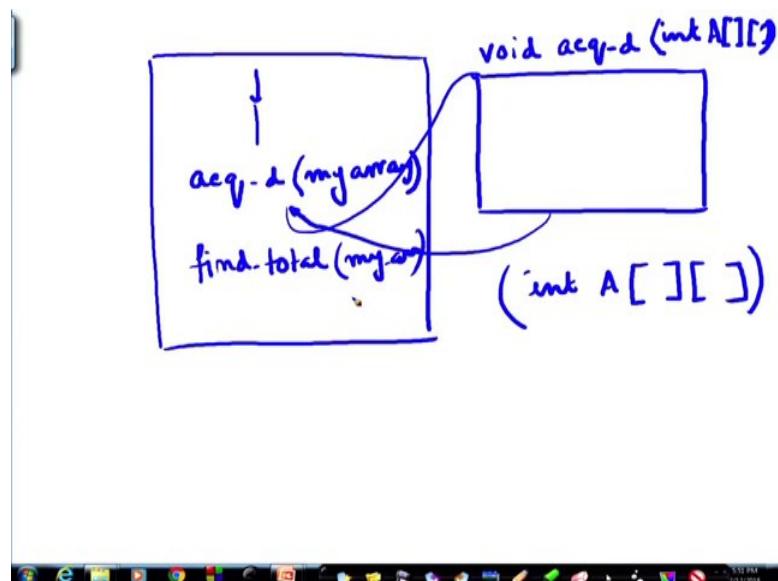
That is for one row for j equals 0 , j less than equal to 3, because there are 4 columns j plus plus , scanf now you know what to write within the scanf ok.

So, in that way I can read the marks. What I was saying is that instead of this I can see for, I can be very clear about it, for roll equal to 0 roll less than equal to 4 roll plus plus, sorry , roll plus plus. And for j similarly I can make marks. So, marks 0, marks less than 3, marks plus plus it is a train that way I can fill up the table ok. So now, in that way, I

can fill up the table and say, I get the marks as 50, 25, 70, 35 like that some numbers. Now I can have another function that will find out the total marks of each student. So, right now I do not have a space for that.

So, I can keep that in mind when I am preparing my table, then I can have the fifth column for the total marks, right. Now I will pass on I have read this. So, first of all, I in from my main program, I went to a particular function I called a particular function, which acquired the data and filled up all these. Say, 70, 20, 90, 15, 35, 75, 90, 80, 20, 25, 60, 30, 10, 20, 30, 40. Like that that it has been prepared. So, quickly let us see what we did.

(Refer Slide Time: 07:14)



We had we entered into a main function. And from the main function, I called invoked a function, which is acquired data, and what should the type of that function be acquired data, is it returning anything? No.

So, it should be something like void acquire data d. And where is it acquiring the data? In the integer array, int say let us call that array A and there should be it is a 2-dimensional array. So, I should have something like this. This time writing clearly int a if that array was a, that was the parameter. Now so, here that was array A, and here it has been called like acquire data, say, my array. And also the size has been specified. The size is suppose globally specified.

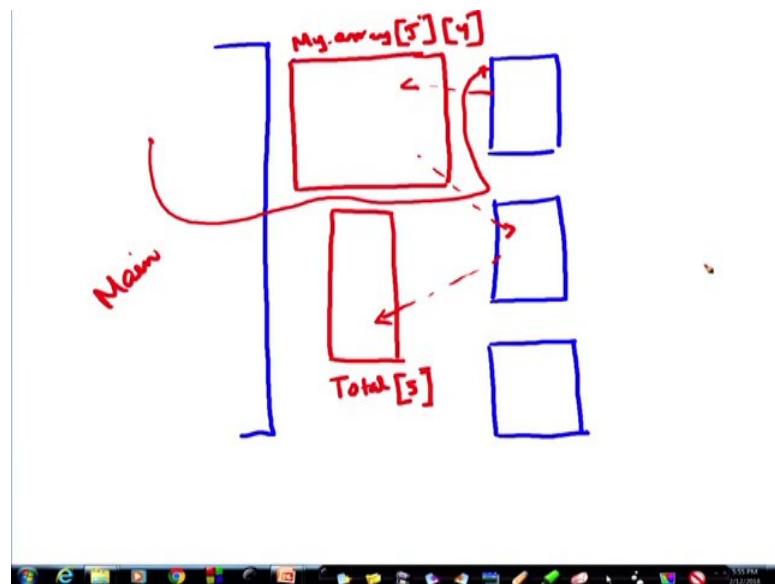
So, I get that array size. Suppose it is defined that row roll size is 5 and subject size is 4. So, I know, but this says subject size plus 1. So, I needed one more. So, I can I can go here and from there, I returned here. And then I can call another fine total of my array. Then what will you do? Essentially, I come to this, and what does that function do that piece of function; that is, finding out the total of each student.

So, that will again be a for loop like the one that we had done earlier, and I will add keep I to be fixed, and add these and write the total here. Now I could have kept it here, or I can have another array total let us do that, suppose I am not having I am just having the marks here. And the total array is not here, but a separate array all together, all right.

So, I have another array which is total. So, I can have the sum of each of them being passed to this. So, in that case when I am calling, when I am calling fine total then I am passing on this array as well as the total array 2 arrays because I want to put the sum on the total array ok. With that I can find the sum of this sum of this, sum of this, sum of this, sum of this. And then I can make another function, where it finds the max or whatever I want to do.

So, accordingly the point is that for each of the activities, I can make small small functions. One is for acquiring data; one is for finding the total. One is for finding the maximum number, or maybe some other things also sorting and all those. I can do, I can find out the failure list another function. So, I can have a complete set of small small functions.

(Refer Slide Time: 11:36)



Each independently done, say did acquiring data, finding total, finding max, all these and they are communicating through a main program, and the main program my data is one array. So, let me let me draw the data using red.

So, here I have got my array, which was called my array, array which had the size 5, 5 and 4. And I had another array total, total which is a one-dimensional array how many? 5 students so these 2 so, the main function is invoking from different points for different tasks these functions. And this function is working on this array, shared array. This function is working on this array and generating the total, maybe this function is doing something else.

So, that these arrays are being shared by the main function and the small functions. That is how using functions I can divide a task into smaller sub tasks and can do it very nicely. Now with this, now let us move to another problem one a version of which we had encountered earlier; that is, searching an array. When we have got an array for example, here when he found say suppose, I found the total here. I found the total, and from there I find out which one is a maximum or is it that if there is any student whose total is less than 100, I want to test that.

So, I have to search this array. Similarly, I can search if there is any student, who has got in science more than 90. I can search that along this array, all right? I find if there is any 40 is the pass mark, if there is any student. Who has not passed in maths? I find here

there is a failure case. So, for that I need to do a search. So, search is very fundamental, search is absolutely fundamental to computation. And so, first we have already seen linear search, but I once again have a journey through that ok.

(Refer Slide Time: 14:46)

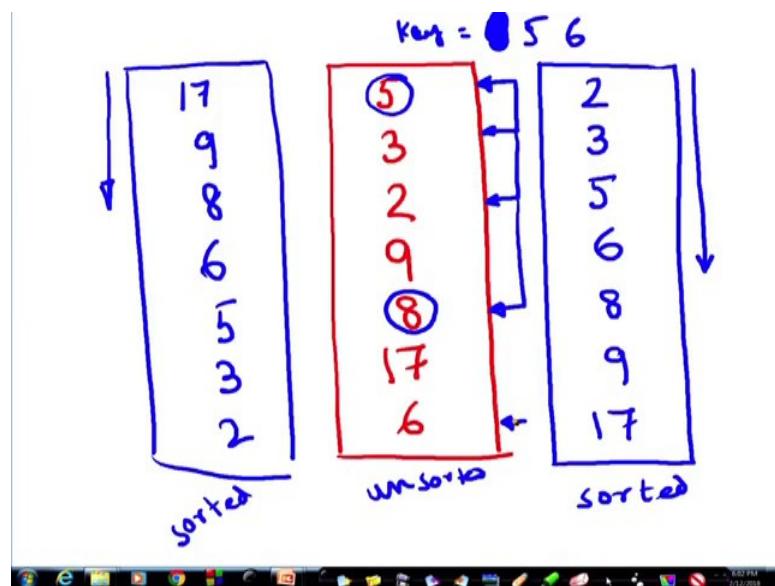
## Searching

- Check if a given element (**key**) occurs in the array.
- Two methods to be discussed:
  - If the array elements are unsorted.
    - Linear search
  - If the array elements are sorted.
    - Binary search

So, we know that in a search we are given a key which we are searching for a particular key that we want to find out, if the key is present in the array, then we have to say that the key is present and also the position where the key is present, all right. Now we will consider 2 cases. One is that the array is unsorted.

What do I mean by unsorted? By unsorted I mean that the elements are not in any particular order.

(Refer Slide Time: 15:22)



For example, think of this array where the elements are 5, 3, 2, 9, 8, 17, 6. Now this array is not in any increasing order, not in some decreasing order. It is decreasing here decreasing here. Again, increased here, decreased here, again increased here, decreased here. On the other hand, if the array was something like this; say, 2, 3, 5, 6, sorry, 2, 3, 5, 5, 6, 8, 9, 17. Then this array is sorted in an increasing order. That as I go from top to down it is in an increasing order, it is sorted in increasing order. It could be sorted, in the decreasing order also. Like, say for example, I start with 17, then 9, then 8, then 6, then 5, then 3, then 2.

So, this is as we go down the numbers are decreasing. So, these are ordered or sorted. And this is unsorted. These also sorted, but in a different order. So, I may have to search from an array that is either unsorted or sorted. So, first let us think of dealing with unsorted arrays, all right. And we will look into the linear search algorithm for doing that. And if the elements are sorted we have got a more efficient such algorithm called the binary search.

(Refer Slide Time: 17:46)

The screenshot shows a presentation slide with a black header and footer bar. The main content area has a white background. The title 'Linear Search' is centered at the top in a large, bold, black font. Below the title is a bulleted list of points about linear search:

- Basic idea:
  - Start at the beginning of the array.
  - Inspect every element to see if it matches the key.
- Time complexity:
  - A measure of how long an algorithm takes to run.
  - If there are  $n$  elements in the array:
    - Best case:  
match found in first element ( $1$  search operation)
    - Worst case:  
no match found, or match found in the last element ( $n$  search operations)
    - Average:

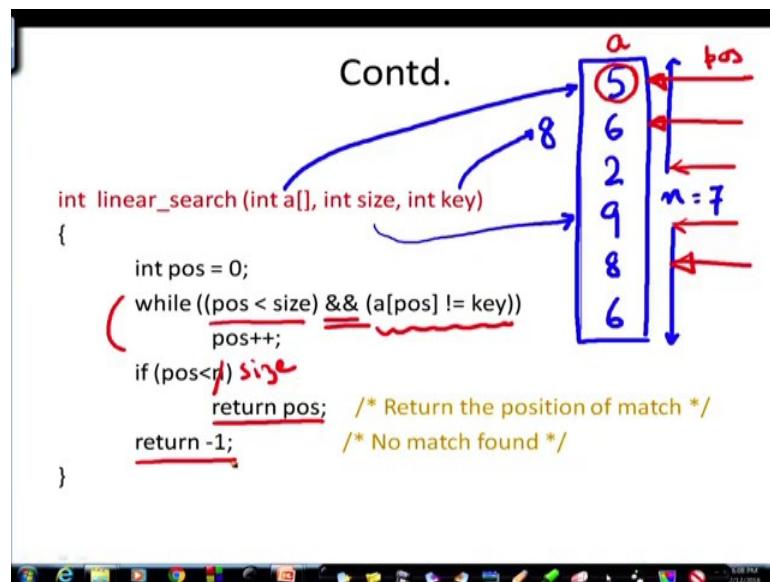
So, linear search we have already seen in an earlier lecture. So, the basic idea is we start at the beginning of the array. So, if we come here, if we come here, sorry, we start say I want to search this. I start at the beginning of the array. I start here. And I have got a key. A key is there, say key is 8. So, I start from the top.

Look for the availability of a look for the element matching the key, 5 is not matching 8, then I go on increasing this index, and go on comparing till I find either if 8 is not there, I will come to the end of the array all the elements have been checked I have not found the match, or when I get the match I will say that at this position, I have got the key matching, all right. So, we will start at the beginning of the adding, and inspect every element to see if it matches the key. Now if I want to do it in this way, we often talk about time complexity, it is a measure I am not going into the formal way of the measure, the measure of how long an algorithm takes to run.

So, as you can understand, if there are  $n$  elements in the array, then the best case would be let us you will find out what it will be say, I start with this. It may be supposed, the key was 5, suppose the key was 5. In that case, that is the best case with one match one comparison I find it how many comparisons how many inspections I needed only one. So, that is the best case, the worst case could be if my key was 6. If my key was 6, then I would have to inspect every element. And since there are 7 elements here, I had to carry out 7 comparisons, 7 inspections till I found the match right.

So, the best case would be first element one inspection one search operation. And the worst case would be no match found. Either the last element or not at all found even after that. In that case I need  $n$  search operations. So, the best case is one, the worst case is if the size of the array is  $n$ , then  $n$ . So, the average would be  $n$  plus 1 by 2 search operations,  $n$  plus 1 by 2 that is a cost how much time that gives a measure of the time the algorithm will take to run.

(Refer Slide Time: 21:23)



So now we are trying to write it we had explained this algorithm also earlier. But now that we have done functions, let us write a function for linear search. Why am I writing the function? Say again if I go to this case. Now I may like to say my task is to find out whether I any student who has got more than 80, all right. So, I will need to write a function, I have to search this in a linear way in a sequential way one after another and for that I have to write a function.

So, how is that function, how will that function look like? Let us see what are the things I need I have got an array. So, I need to know which array I will be working with, and that is this array A. And what is the size of the array n whatever that is 7 maybe in our case the example that I was showing was 7. So, that is so, after I compare 7 elements, if I do not find the key, then I am unsuccessful in finding the search is not yielding any result. The other thing is key the element for example, 8 which I am finding out in this array of numbers, right. Say something like this I am trying to find out 8.

So now let us look at the algorithm. Int pos equal to 0 pos means position. So, initially I am in this position. That is this array is a so, a 0, while pos is less than size it is, right now 0 it is less than size; that means, the pos is not has not exceeded the last element; that means, I have not yet checked the last element. That is why this is said, and the this pos the element a pos a pos, pos is the index is 5, and suppose my key is 8, if a pos is not equal to key.

Then I will increment pos, I will check for the next element. Now look at this. This is an end here. If either of these conditions fail; that means, if I have not found it fine, but I have exceeded the key exceeded the array size, I mean limit then I will stop. Or if I have found the key 8 here, although I have not reached the end I will stop ok. So, that is the condition, then I go on increasing pos; if pos is less than n; that means, when I come out of this while loop, when any of these conditions are not satisfied.

So, if pos is less than n; that means, actually it should have been pos is less than sighs ok. If I have come, if still pos is let us it should be pos is less than size. If pos is less than size; that means, I have not come to the end of the array then; obviously, why did I come out? Because this condition was false this condition was false means what? I have already found the key, right then I will return the position.

So, pos increments from 0 to 1 to 2, to 3, then to 4, when I do that the size was 5, I have not exceeded 5 is still less than 5. And so, pos could be in 2, but I have come out I have come out because I found this. So, at pos pos position 4, I am getting the key. So, pos is returned, otherwise if this is not true, then I will return minus 1, minus 1. If it is returned; that means, that I have not found the key ok. So, this is the look of the linear search function, very simple. Now so, the key appears in if the key appears within this limit, then I will return the pointer pos otherwise I shall return the key.

(Refer Slide Time: 27:04)

**Contd.**

```
int x[ ] = {12, -3, 78, 67, 6, 50, 19, 10};  
          ↑↑↑↑↑↑↑↑  
          pos=4
```

• Trace the following calls

Linear search (x, 8, 6);      Returns 4  
Linear search (x, 8, 5);      Returns -1

So, here is an example say, this is the array x, 12 minus 3, 78, 67, 6, 50 etcetera. I want to trace the following calls. I call search so, if you go up the linear search, let me call this is linear search. It should be written down as linear search. I am calling with the array x all right the size of the array 8, and the key is 6 ok. Here what will happen? Size of the array is 8.

So, I will start from here 12, no minus 3, no my key is 6 right, I increment pos from here to here, no, it is not matching pos is not the key, a pos is not the key here a pos the key. So, I come out with 0, 1, 2, 3, 4, pos value is 4 here. What will happen for this? I the same array x size 8, but my key is 5 you can see that I will go up to this and after that I will increment pos plus plus. So, it will be more than 8. And so, the size is 8. So, I will not be able to I will say that I have not got the key, all right.

So, that is so, this one will return 4, and this one will return minus 1. That is how the linear search algorithm works. Now let us stop here for today. Next, we will look at another more efficient search, but it require that demands something more from us that the array must be sorted in that case we can apply the binary search algorithm to make it more efficient ok. So, today what we saw is how we can write as small small functions to solve a big problem after a break down that big problem into smaller parts. And also, we found how we can the concept of linear search which we learnt earlier, how we can write a function for that ok.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

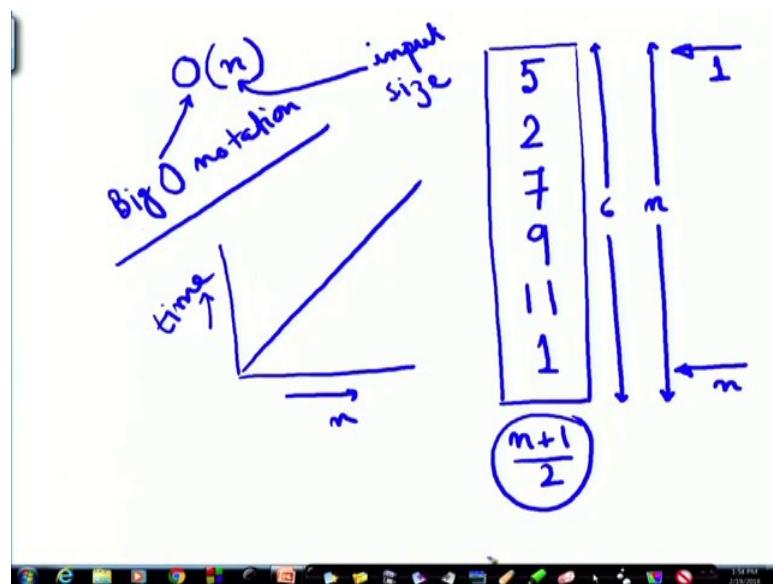
**Lecture – 43**  
**Binary Search**

In the earlier lecture, we have seen, how we can search in a list of items, may be list of integers or may be list of names or list of real numbers in a linear way; that means, we start from one end of the list and we have got a key and we check for every element whether the element in the list matches the key or not. In that way, we go on from top to bottom. And ultimately, if we get the match of the key to an element of the list, we declare it is found. And we also say where it is found.

Otherwise, if it, if we exhaust the list and still do not find the element, then we say that the list is not found. And the way in which we have find searching from the top to bottom, that process is known as linear search. We have also seen that, the time that time that is required to search for an element in the best case, it would be of order 1; that means, at the very beginning, we can find the element. Otherwise, we have to show many comparisons, we have to do? The minimum is 1 if we get the match at the beginning of the top of the list. Otherwise, we have to we may have to exhaust all the elements and whether it is found or not found. We can say only after we have compared all the elements.

So, if there is a list of  $n$  elements, the maximum number of comparisons that I may have to do will be  $n$ . Therefore, on an average, it will be  $n$  plus 1 by 2 that we say in complexity of algorithms parlance. In that parlance, in that terminology, we call it order of  $n$  ok. That is not the main issue with us, but with we can say therefore, that in a list of elements, if I have got  $n$  elements and the list is not sorted, so, may be 5 2 7 9 11 1, say, this is the list and I want to search for a key.

(Refer Slide Time: 02:44)



So, this in this case, it is 6. But, in general, I can say that there are  $n$  elements right. In general, I can say number of elements is  $n$ . So, the best case I can get a match here, the number of comparisons I require is 1 and the worst case I have to come up to this and compare all these  $n$  elements.

So, the number of comparison on an average will be  $n$  plus 1 by 2, all right. That is a average number of comparisons that we denote in computer parlance as of the order of  $n$ . This  $O$  has got name called big  $O$ . I am sorry let me write it in this way. This called big  $O$  notation big  $O$  notation. So, where it just denotes the how much time a computer can a program will take to run, in terms of the input size, what is this  $n$ ? This is the size of the input data, we call it input size.

So, obviously, you can see that, as these list increases, if it becomes 100, then the time average time that will be taken average. This is average complexity all right, will be more than this. If it is with 1000 even more and as increases, the time will increase in a linear fashion. As  $n$  increases, the time will increase in a linear fashion. However, the way we are searching is known as linear search. One thing to note is that, in this case, we are not making any assumption. This is time. We are not making any assumption about the way in which the data items are organized.

(Refer Slide Time: 05:37)

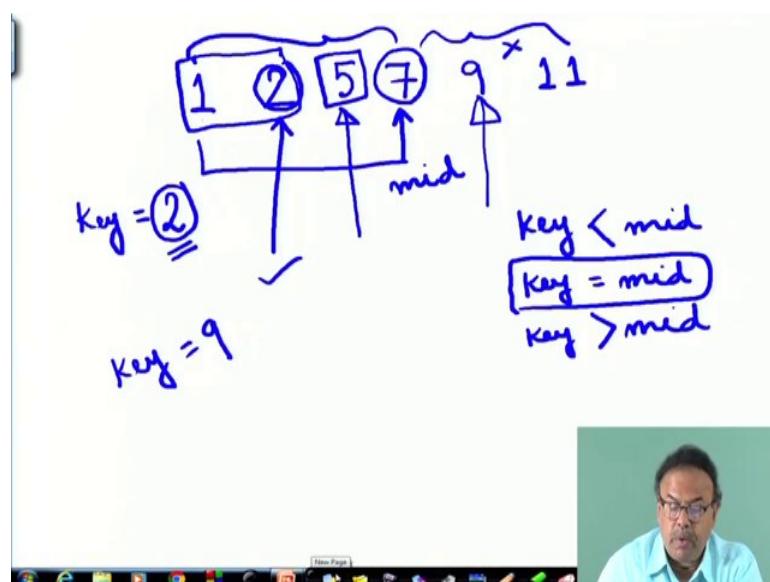
## Binary Search

- Binary search works if the array is **sorted**.
  - Look for the target in the middle.
  - If you don't find it, you can ignore half of the array, and repeat the process with the other half.
- In every step, we reduce the number of elements to search in by half.

Now, in an endeavor, to see, if we can make it better, we today discuss another very important type of search algorithm, which is known as binary search. Why the name binary is coming very clear in a moment, but the important thing to note is that, in this case, the list or the array must be sorted; that means, it is organized in some particular way; either in an ascending order or in the descending order ok.

So, that is why, it is called binary. Let us let me just try to explain it with an example. Say, I have got the array, but in a sorted way.

(Refer Slide Time: 06:31)



So, I have got 1 2 the elements where there in the earlier case one 2 five seven nine eleven right 1 2 5 7 9 11. So, 1 2 5 7 9 11. Now suppose my key is well, let the key be 2. Now, we will start at middle of this array first we will look at the middle. Now, since is this is an even number, even sized array, 6 elements middle can be somewhere here. Say, let me take this, this is the midpoint.

Now, I compare the key with the middle element 2 and 7 are being compared. 2 things can happen; either 2 the key I I will rather say, either the key is less than the mid element or the key is equal to the mid element or the key is greater than the mid element. 3 things can happen. Now, if the key is equal to the mid element, then my search immediately stops. Yes, I found it and where did I find it? The index is mid.

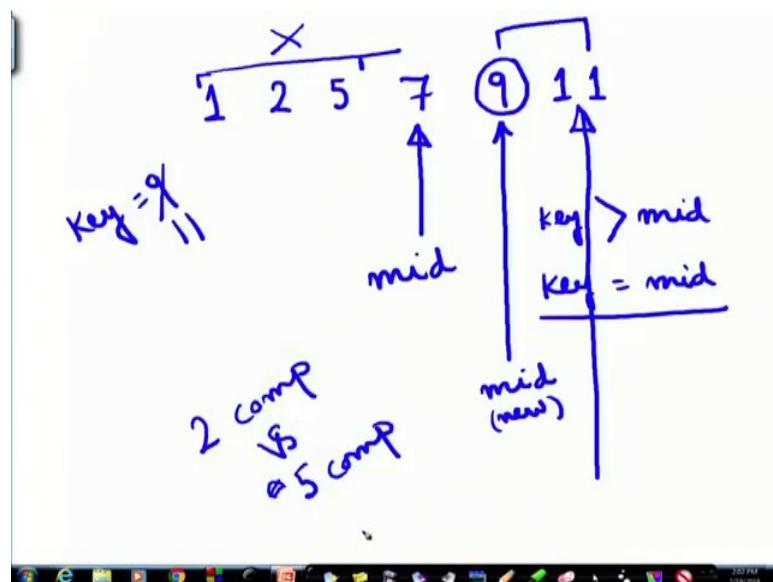
If the key is less than the mid element, as is the case here, then what can I say? I can immediately say that, the I still do not know whether the key is there in this array or not, whether the key is here or not I do not know as here. But the thing that I say is, that the key, since it is less than the mid, it cannot be on this side, it cannot be on this side. Not possible. It must be if at all on this side, because, the key is less than the mid.

Therefore, I can restrict my search within this period. Suppose, within this zone, now what I will do? I will again take the middle point of this array, right? So, between here it is becoming very simple. So, the middle point of this part is suppose 5. I could have had it from 1 1 to 5, but I am just taking from 1 to 7.

I come at this point and again now I compare this element and still it is less therefore, am sure that it is not in this area it must be in this zone. Now I again, only for this part , I apply again, I find mid element and this is here and I find that the key and the element are matching. So, it is found. So, how many comparisons I needed here? In this case, I need it 1 2 3 comparisons. In the case of linear search of course, you could have got it luckily here in 2 comparisons all right.

But, what would have happened if my key was 9? In the case of linear search, suppose, the key was 9 in that case, in the case of linear case what would what would have happened I would have started from here 1 comparison, 2 comparison, 3 comparison, 4 comparison and the 5th comparison I would have got it right, on the 5th comparison I would have got it. But, let us see what would have happened in the case of the new search technique that we are looking at.

(Refer Slide Time: 11:04)



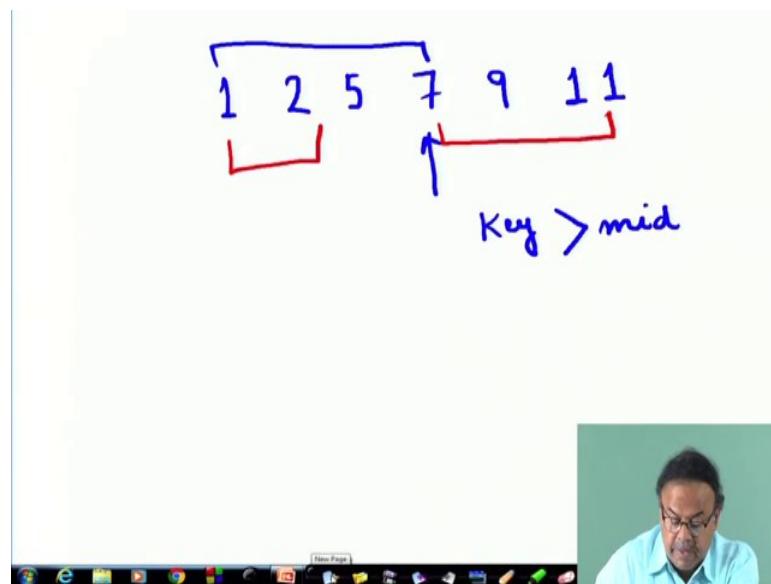
So, again I have 1 2 5 7 9 11 and my key is 9. I know that in case of linear search, I needed 5 comparisons. What will happen here? I will come to the middle point here. I find that the key is greater than mid. This is mid. The key is greater than mid. Therefore, I know that the key if at all cannot remain in this zone. This part is ruled out this part is ruled out. It must be in this zone. So, again within this I find the new mid. So, the new mid comes here. And I find this new mid mid new this and I check this, now I find that the key is equal to mid.

Therefore, I find my search completes here with this index as output. Now how many comparisons I needed here? 1 2 only 2 comparisons.

So, 2 comparisons verses 1 2 3 4 5 comparisons in the case of linear search. Now why was it reduced? The same thing would have happened with if my key was 11. Let us have a look if the a key was 11. In that case, suppose, the key was 11, in the case of linear search, I would have required 6 comparisons here to reach at 11.

However, in this case, I would be sure if key is greater than is, then my next iteration will be between these 2 and I would have found mid new mid and I would have got it with 3 comparisons. Now why is it becoming so? Why is the number of comparison being reduced? The reason is we have got this array and at every stage, what I am doing is, am looking at the key and depending on whether the key is greater than the mid.

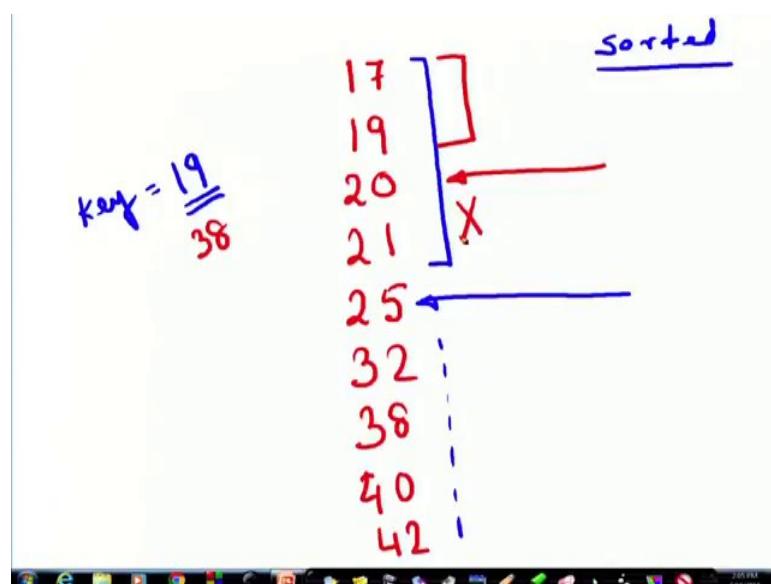
(Refer Slide Time: 13:44)



I am concentrating on only one half of the array either this array or this array. And then, depending on the key value, I take suppose it is greater, then I will concentrate on these zone. If it is less, I will concentrate on this zone. And iteratively, I will be reducing my search to a smaller array. Let us have another example.

Let us have another example. Let us have it a little bigger all right. Let us have 17 19 20 21 25 32 38 40.

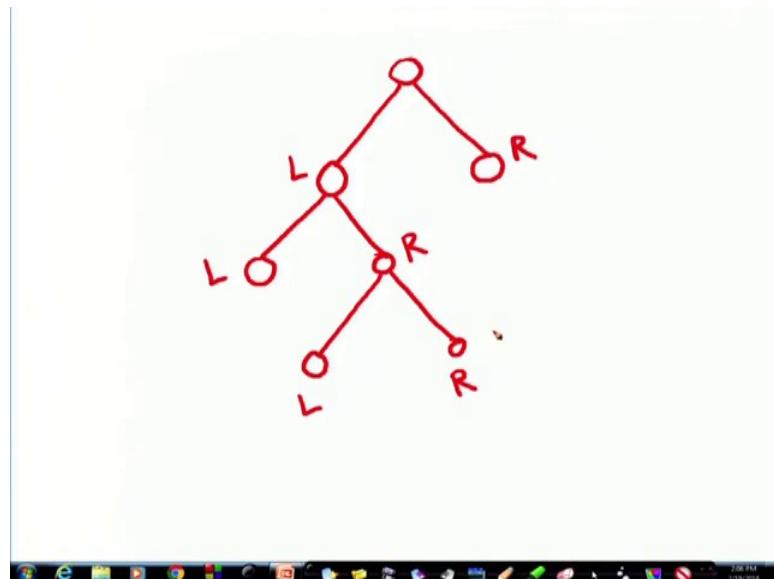
(Refer Slide Time: 14:45)



So, now I have got again even number let me have 42. Say odd number arrays 1 2 3 4. Now I have got 9 elements right. Now I will start with the mid element. What is the mid element? Here, between this, the mid element is this. And suppose my key is 19 , then I know that, my key cannot lie, since it is less than 19, it cannot lie in this part of the array. If at all, it will lie on this part of the array. Why? Why do I say that? I can say that because, this array is sorted.

Otherwise, I could not have said. Since it is an increasing order, I can say that, since the key is less than the mid element, therefore, it must be in this zone. And so, I come to this zone and find out the mid element again may be in this zone or up to this. Say I come here and find out the mid element this again. So, am looking, I am immediately reduced my list to half, right? This half, it could have been if my key was 38, then I would have restricted to this half not this half. But for 19, am restricted to this half. Now again, 19 is less than 20, immediately I will restrict myself to this half and I will not consider this part, gradually am going to the lower half or the higher half at every stage.

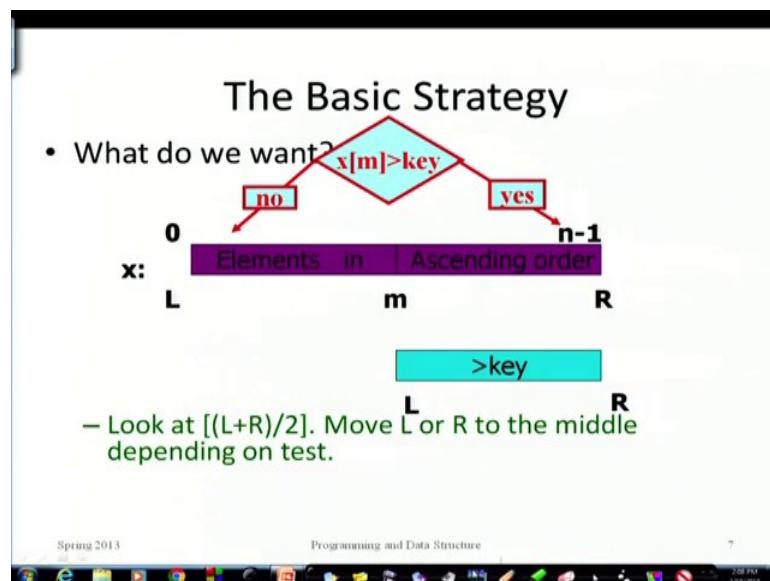
(Refer Slide Time: 17:10)



So, what is happening is something like this. Am starting with the whole array, then am concentrating either in the left half or on the right half. Then again, depending on that, either on the left half or on the right half if it is in the right half, then among them, this am coming to left half and right half. So, at every stage, am dividing the array to half. That is why this is called binary search. Either it is here or it is not here not in this zone.

So, in that way, I carry on. So, con consequently, the number of elements that I restrict my search to gets reduced at every iteration. Let us look at this in a little more detail. So, in every state, we reduce the number of elements by half. I think this is clear now, that this statement. Now, if you do not find it, you can ignore the half of the array and repeat the process.

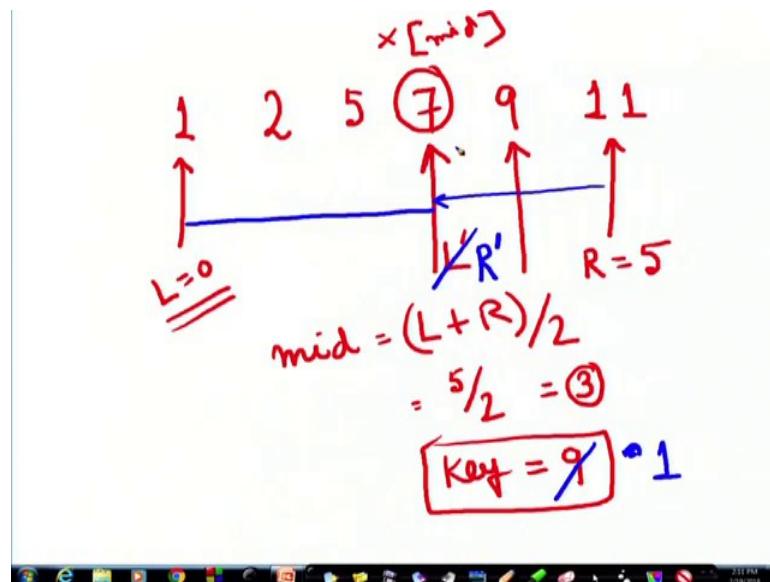
(Refer Slide Time: 18:24)



So, now, let us look at this basics strategy. What do you want? The array is from 0 to n minus 1, n elements. Here, I was showing mid zone only. So, this is the mid m and this is the left end. This is the right end. So, there are 2 indexes, indices all right? Now, am taking the key and I am checking this part m. The element m this element; that is, xm. And depending on whether if it is greater than the key, if xm, if the key is greater than xm, key is greater than xm is less, then I will be concentrating here. If xm is greater than the key, then I will be concentrating on this half on this half or on this half. That is what I was explaining till now ok.

So, here, if it is no, then I will. If it xm is less than the key , then the I will come to this half. Otherwise, if it is less I will come to this half greater than that all right. So, will first given this L and r, what is my mid element? How will I compute my mid element or m? This L plus R divided by 2. And then, depending on whether it is less or greater, we will we will move the left or L or R till the middle depending on the test. So, again let me show it with a example that we are showing.

(Refer Slide Time: 20:21)



So, something like this 1 2 5 7 9 11 what is being said is, 9 11 ah. So, L is L equals 0 and R is equal to 1 2 3 4 5 6 0 1 2 3 4 5. So, R is equal to 5. So, mid is L plus R divided by 2. So, that is 5 divided by 2. So, we can take 2 or we can take 3 depending on. So, if I come to if I take 2, if I take 3, then am coming to mid. So, what I will do is, if 3 is my midpoint, if I take 5 by 2 is 3, then, that means, it will be in case of odd I will add 1 to that. So, I can move. Now if the key is suppose, key is 9 suppose the key is 9. So, my mid was the 3rd.

It is 0 1 2 3 or let me 0 1 2 3. There is my mid. Now, since the key is greater than the element x m x mid, since it is x mid is less. I will be restrict my search in this area. Therefore, what I will do? I will move this L to mid. So, this will be my new L. And I will find the mid with. Now, next mid will be this new L plus R by 2. So, it will be this element if my search was on the other side; that means, if the key was not 9, but the key was 2 or 1, let us make it 1.

Then, at this point, I find that the key is less than x mid, then I will be restricting my search in this area. In that case, I will move my R, I will shift this R over here, and this the mid will be R, R will be the mid. So, next, I will be restricting my search in this half and forget about this half. So, that is how we at every iteration, we break down the entire array into halves. So, if you have understood this, let us proceed. Repeat the search

operation in the reduced interval. So, what we are doing is, we are looking at this binary search algorithm and we are trying to design a function.

(Refer Slide Time: 23:35)

Contd.

```
/* If key appears in x[0..size-1], return its location, pos s.t. x[pos]==key.  
If not found, return -1 */  
  
int bin_search (int x[], int size, int key)  
{  
    int L, R, mid;  
    while ( _____ )  
    {  
        _____;  
    }  
    _____;  
}
```

Diagram illustrating the binary search process:

- A blue box labeled "Bin Srch" contains the pseudocode for the binary search algorithm.
- Arrows point from the parameters "List", "Key", and "m" to their respective variables in the code.
- The variable "m" is also shown pointing to the "mid" variable in the code.



And finery binary search is a function, whose name am just keeping as bin search all right. This is the name of the algorithm. And what are the parameters? Let us see. The parameters are one is the array that is being passed the list that have to search. I also need the size of the array and I need the key.

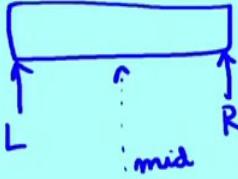
So, these are 3 things. So, binary search is a function. What are it is inputs? The list or the array which is x, the size or what we are calling about right now n and the key and what will the output be, bin search we will tell us whether it has been found or not ok. But, so, it will be a 0 or 1. Found or not. 0 or 1 or it can written as an index also ok. So, now, if we proceed with this idea, then let us develop the algorithm step by step inside this function. I have got the search. I have got the list the list is given to me.

(Refer Slide Time: 25:39)

**Contd.**

```
/* If key appears in x[0..size-1], return its location, pos s.t. x[pos]==key.  
If not found, return -1 */

int bin_search (int x[], int size, int key)
{
    int L, R, mid;
    _____;
    while ( _____ )
    {
        _____;
    }
    _____;
}
```



Spring 2013      Programming and Data Structure

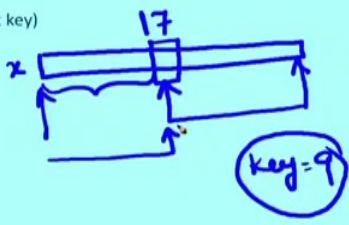
This array is known to me. I declare internally L R and I have to use another index called mid. Now, this has got no meaning outside. This function next step what should I do?

(Refer Slide Time: 26:04)

**The basic search iteration**

```
/* If key appears in x[0..size-1], return its location, pos s.t. x[pos]==key. If not  
found, return -1 */

int bin_search (int x[], int size, int key)
{
    int L, R, mid;
    _____;
    while ( _____ )
    {
        mid = (L + R) / 2;
        if (x[mid] > key)
            R = mid;
        else L = mid;
    }
    _____;
}
```



Spring 2013      Programming and Data Structure      9

Next step would be while some condition mid, I will have to find out L plus R by 2. As you know, if this is the array x, if the mid element is greater than the keys So, the key, say this mid element is 7 and my key is 9. So, this element is greater than I know that this element is greater. So, this is 17. This is greater than the key. Then I will have to

keep my search within this zone. So, this R will be updated and the R will come here. Otherwise, L will be moved here. This much is clear. You think over this and build upon this algorithm we will take it up in next step again, we have to decide on how to build this things up.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 44**  
**Binary Search (Contd.)**

We were discussing about the binary search function or binary search procedure.

(Refer Slide Time: 00:25)

The basic search iteration

```
/* If key appears in x[0..size-1], return its location, pos s.t. x[pos]==key. If not
   found, return -1 */
int bin_search (int x[], int size, int key)
{
    int L, R, mid;
    _____;
    while (_____)
    {
        mid = (L + R) / 2;
        if (x[mid] > key)
            R = mid;
        else L = mid;
    }
    _____;
}
```

A hand-drawn diagram illustrates the binary search process. It shows a vertical array of elements with indices L, mid, and R marked. The array has a 'Size-1' label at the top right. A blue 'X' marks an element at index L. A blue 'Key' label points to a specific element in the array. Arrows show the movement of pointers: L moves up (to mid), mid moves down (to R), and R moves up (to mid). The array elements are represented by small circles.

So, here we can see that binary search is of type integer is a function of type integer that will be returning an integer. And it has got the parameters the list or the array which is being designated as x here, the size of the array and the key. Now if the key appears say, the array is here, the array is this, or the list is this, this is called x. And it starts with 0, and the last one is therefore, whatever is in size minus 1.

So, if the key there is a key, and if the key appears anywhere here, it will return the position of the point where the key is, if this be the position that be returned. And if not found it will return minus 1, so either it will find the position send the position. So, that is also integer, or minus 1 hence the height is integer, alright. Now we have got 2 pointers, one is L the left pointer left index and the right index of this. And from there we are finding out the middle. So, we are finding out midpoint by this, like, now if the key is smaller than the middle element.

If the key is smaller than this middle element, then obviously, our search will be on this side. Therefore, this R will be moved to this point. L will remain here and my search will be restricted in this zone. Otherwise L will be moved here, R will not be disturbed and the search will be restricted to this zone. This much has been achieved by this piece of code, what else do we need? What else do we need to do? Now how long will this go on?

(Refer Slide Time: 02:43)

### Return result

```
/* If key appears in x[0..size-1], return its location, pos s.t. x[pos]==key. If not found, return -1 */

int bin_search (int x[], int size, int key)
{
    int L, R, mid;
    while ( L+1 != R )
    {
        mid = (L + R) / 2;
        if (x[mid] <= key)
            L = mid;
        else R = mid;
    }
    if (L >= 0 && x[L] == key)  return L;
    else return -1;
}
```



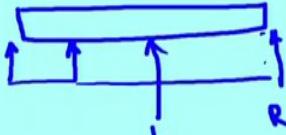
So, in this way, what will happen is, I will move either L or R, either this or this, and when will this be completed, how long will this loop go on? If you think a little bit we will find that it will go on as long as L and R are not crossing over. So, if I shift and search this part R is here.

(Refer Slide Time: 03:15)

## Initialization

```
/* If key appears in x[0..size-1], return its location, pos s.t. x[pos]==key. If not found, return -1 */

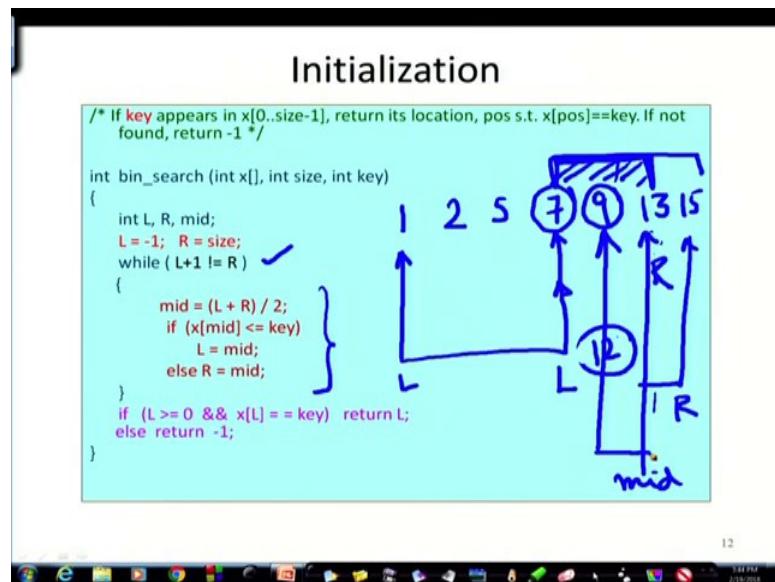
int bin_search (int x[], int size, int key)
{
    int L, R, mid;
    L = -1; R = size;
    while ( L+1 != R )
    {
        mid = (L + R) / 2;
        if (x[mid] <= key)
            L = mid;
        else R = mid;
    }
    if (L >= 0 && x[L] == key) return L;
    else return -1;
}
```



Suppose I do, I initially start with L equal to be minus 1 and R to be the size. So, I am shifting from so, here is my array and L value of L is minus 1. So, it is here and R is equal to size; that means, pointing here size plus 1; that means, sorry this point is size minus 1. So, that is actually here, and while as I share, I shift L and I shift between this part etcetera.

Then my shift L, R is here; as soon as L and R crosses over, if R will R moves this side then also a L and R can cross over. If they cross over then that is the; that is my point where I should stop.

(Refer Slide Time: 04:18)



So, I initialize this; so, and I go on shifting this, depending on either L or R. That is exactly what we are doing when we took the say 1, 2, 5, 7, 9, 13 like that 15. So, I start it with the middle number, and when the key was suppose my key is 16, all right? Or say.

Let me erase this, my key is say 8; 12. Suppose, my key is 12, what will happen? My L is here, there is L, this is R, I start with mid is 7 so, key is greater than this. So, I will have to restrict my search in this zone I take the middle of this. So, R becomes this, this becomes R, and I am sorry, am sorry R remains the same. I have to search between this zone; so, what I do? I shift L I have to search between this zone. So, I shift L to this. So, this is my 12, and this is R I have to search in between this. I will find this to be the mid, fine? And I find that the key is less than the mid; so, I will restrict my search within this zone, this zone, within this zone right.

Since it is less so, then what should I do? If since am moving on this zone, I will shift R keeping L fixed I will shift R here. So, this will be R now; now between this n L 1 and R I find mid, this is my mid ok. I find 9 to be less than 12; that means, it should be on this side. So, I move L, as I move L; L is crossing R; that means, now I have come to a position where I have ex exhausted everything, and I could not find the key, all right?

So, if L is greater than 0, and x L is equal to key, L is greater than equal to 0. And x L is the is a key then I return to the point, otherwise return minus 1. So, that is the binary search algorithm ok. This is how am carrying out the algorithm through a C code.

(Refer Slide Time: 07:27)

Binary Search Examples

Sorted array

-17	-5	3	6	12	21	45	63	50
-----	----	---	---	----	----	----	----	----

-17 -5 3 6 12 21 45 63 50

Trace :

binsearch (x, 9, 3);  
binsearch (x, 9, 145);  
binsearch (x, 9, 45);

So, here is an example am writing down an array. Suppose, sorry am writing down a sorted array, suppose it is minus 17, minus 5, 3, 6, 12, 21, 45, 63 and 50. 63 it cannot be 50, say this one is say this up to 63. That is my array.

Now if I carry out so, the size is 1, 2, 3, 4, 5, 6, 7, 8. I must have one another one. So, let me have 75, and if 3 be my key this is the key. So, what will happen? I will start L here, R here, and I will find the midpoint here. So, the midpoint will be here, 12 and 12 is greater than the key. So, my search will be within this zone, this R will be shifted here. So, this is my now R, and this is L, I will find out midpoint 3 here now. As soon as I come to this mid, and I find that this mid is matching the key my search is found, all right? I get I get my key in a particular position.

(Refer Slide Time: 09:24)

## Binary Search Examples

**Sorted array**

-17	-5	3	6	12	21	45	63	50
-----	----	---	---	----	----	----	----	----

**-17 -5 3 6 12 21 45 63 75**

**Trace :**

binsearch (x, 9, 3);  $\rightarrow$  L=1; R=4; x[2]=3;  
binsearch (x, 9, 145); L=2; R=4; x[3]=6;  
binsearch (x, 9, 45); L=2; R=3; return L;

**We may modify the algorithm by checking equality with x[mid].**

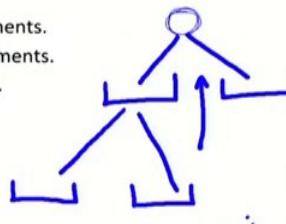
So, you can write it in different ways. So, so, in this way, if I do that, then I will get different results. So, minus 17 I will just miss out minus 5, 3, 6, 12, 21, 45, 63 and 75. So, if for this, my L was at minus 1 R was at 9, and my mid was this x 4 12 ok. So, then x 1 is minus 5. I carry on doing this. L is still 1, R is 4, R has been shifted to here. And x 2 is 3, I found it.

So, that is my result, in that way I will carry out and you can find out that L and R and then I returned. L I go on shifting. So, that is what it has happened. I have got the mid value, and I have got the value, but then still am going on shifting. You can you can also, what I have suggesting is, it is also possible to check the equality with the x mid as soon as I find that the element has matched with the mid then it is found.

(Refer Slide Time: 10:46)

## Is it worth the trouble ?

- Suppose there are 1000 elements.
- Ordinary search
  - If key is a member of x, it would require 500 comparisons on the average.
- Binary search
  - after 1st compare, left with 500 elements.
  - after 2nd compare, left with 250 elements.
  - After at most 10 steps, you are done.



14

I think you have understood it you can try writing out the algorithm yourself these are binary search we will see later that it can be also done through recursion.

Now why are you doing so much? Earlier, linear search was very much simple, right? Here suppose we had thousand elements, if we had thousand elements, then the ordinary search if a key may key is the member of x, it would on an average require on an average it would have required 500 comparisons on an average. But what will happen in the case of binary system search? After the first compare 1000 elements.

So, we are left with only 500 elements, because I start it from I start it with an array, and then based on that I have either gone on this half of the array or this half of the array depending on whether the key is less than or greater than the midpoint this is the mid. So, on an average first compare I am left with 500 elements. Next compare this part is again divided if it is on this side am left with 250 elements, right. After at most 10 steps, we are done in that way I go on dividing it and after at most 10 steps I am done.

So, I have thousand elements ok. So, I can so, in that way I get 250 elements, and 125, 125; then half of that 60, 65 or something like that.

(Refer Slide Time: 12:42)

**Time Complexity**

- If there are  $n$  elements in the array.
  - Number of searches required:  $2^k = n$ ,  
 $\log_2 n$  Where  $k$  is the number of steps.
- For  $n = 64$  (say).
  - Initially, list size = 64.
  - After first compare, list size = 32.
  - After second compare, list size = 16.
  - After third compare, list size = 8.
  - .....
  - After sixth compare, list size = 1.

$O(\log_2 n)$

$O(n)$

15

In that way, number of steps that I will be following will be around 10. In general, if there are  $n$  elements in the array, number of searches required is  $2$  to the power  $k$  should be  $n$ , where  $k$  is number of steps ok. For  $n$  if there be 64. Initially, the list is 60 the size is 64, after first comparison list is 32. After second it is 16, then 8, then 4, then 2 and then 1 so, 6 steps. So, basically at every step am breaking it down into half. Therefore, if I need the  $k$  steps.

(Refer Slide Time: 13:30)

$$2^k = n$$
$$k \log_2 = n$$
$$k = \log_2 n$$

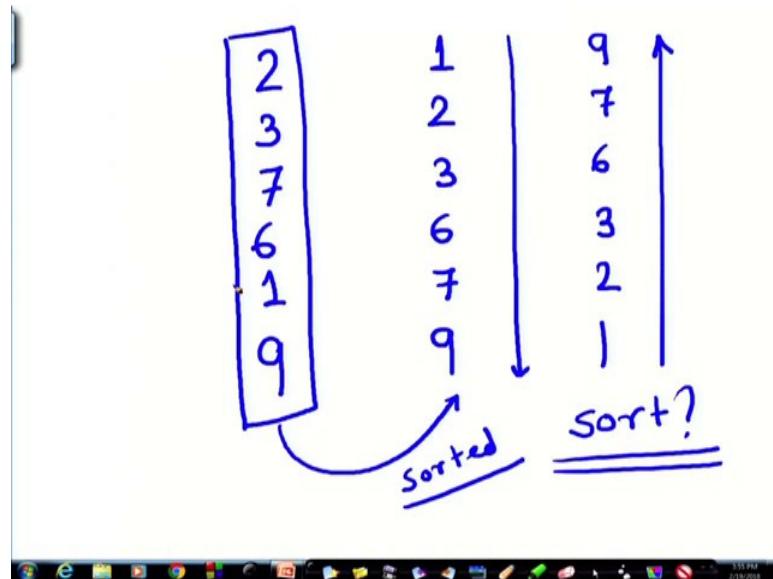

Then, if I need k steps, then  $2^k$  should be equal to n; that means, how many steps are there? How many, what is k if I take  $\log_2 n$  and  $k = \lceil \log_2 n \rceil$  is equal to n. Therefore, k will be  $\log_2 n$  to the power base of log 2. So,  $\log_2 64$  to the base 2 is 6. We will be completing in 6 steps. Whereas, in the worst case I would have needed 64 comparisons in the case of linear search. Now given this idea of binary search.

So, what have you seen? We have seen binary search is a search algorithm, that enables us to search for an element much faster, than linear search the approach being that we divide the array to be searched in half every time. So, we divide it into half and restrict our search in half, and in the next step I further divide it into 2 half and restrict it to even a smaller steps.

So, every time am halving whatever array am checking. So,  $2^k$  days to the power k time by half date where k times have half date  $2^k$  days to the power k should cover the entire n. And so, from there we get the complexity to be of  $\log_2 n$  to the base 2  $\log_2 n$  to the base 2 in general it will be  $\log n$  if 64 is replaced with n (Refer Time: 15:43) then it should be  $\log n$  to the base 2, that is my time complexity compared to. So, I can say this order of  $\log n$  to the base 2, compared to order of n, that was the case in the case of a in the case of linear search.

Now, we will move to another very fundamental problem in programming. We often need that say for example, here we have done one thing that is, I had an array like say 2, 3, 7, 6, 1, 9.

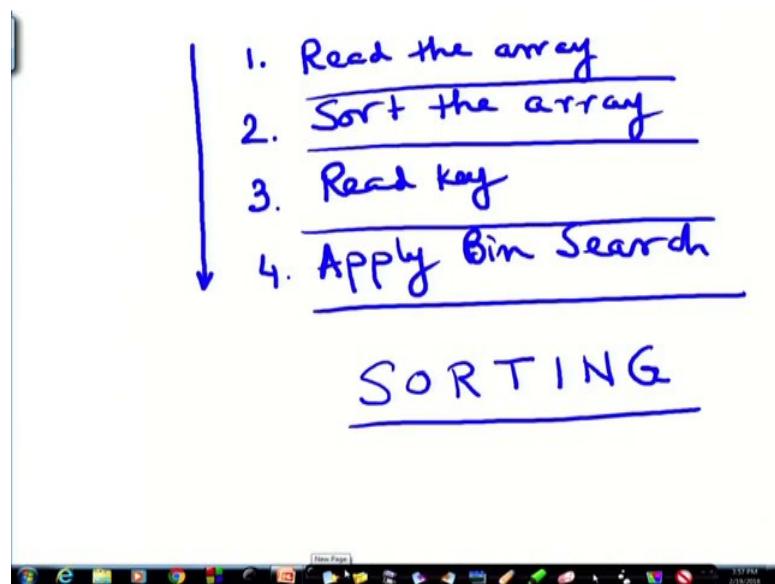
(Refer Slide Time: 16:23)



Now this was an array, on which I cannot run binary search. Why I cannot run binary search? Because binary search can be done only on sorted array. So, if I sort it, either in increasing or decreasing order. Say, for example, it becomes 1, 2, 3, 6, 7, 9. This is a sorted array, or the other thing could be 9, 7, 6, 3, 2, 1. These also a sorted array. In the other direction, it is descending order this is the ascending order ok.

So now another problem is, to arrange the things arrange the objects, suppose they are number of numbers like this. We have to arrange them, sorted or arranged in a particular way. The question is, how to do sorting, how to sort? So, suppose I have given this array to my program, this is the input. Then if I want to apply binary search, then I cannot directly apply it apply binary search on this. So, what I can do? I can first take the array say I read the array let me write.

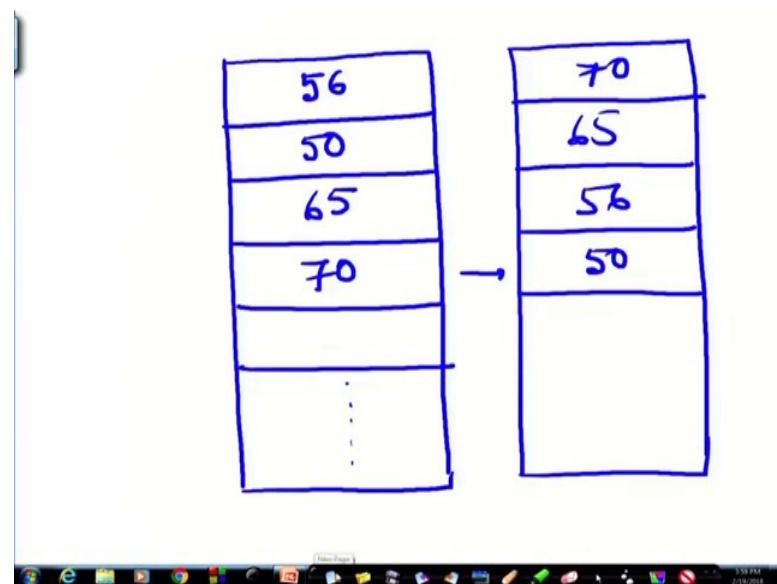
(Refer Slide Time: 18:00)



Then sort the array, and then read key apply binary search. Now binary search takes for n number of elements it will be faster. But how do we sort the array? So, it is faster than linear search, but is it advisable to approach in this way if the array is not sort, is it advisable to sort the array and then apply binary search? In order to understand this, we have to first see how an array can be sorted, how an array can be arranged in either increasing or decreasing order.

That is also a very important problem that is called the sorting problem. And this has got very important I mean application, in many of many of objects whenever I so, for example, for example, you have got the marks of the students, all right? Then you have got a marks array hm.

(Refer Slide Time: 19:50)



Say, you have got 2-dimensional matrix may be where, all right, let us take one dimensional one dimensional array; where, the roll numbers are here, every row is corresponding to a roll number. And here I think, I will I will just do it again. I have got the marks here; 56, 50, 65, 70 etcetera, etcetera.

Now, I want to find what is the highest marks. One thing is that I can find that, out or I want to attribute ranks, first second third like that. In that case I can sort this array, and from here I want to have an array like this; where the first element will be 7, the second element will be 65, then 56, then 50. This also sorting the marks. In a decreasing order, and along with that I can also sort the roll numbers, and can publish which roll number got marks right. So, what we want to see next is sorting.

(Refer Slide Time: 21:50)

**Sorting:** the basic problem

- Given an array  
 $x[0], x[1], \dots, x[\text{size}-1]$

reorder entries so that

$$x[0] \leq x[1] \leq \dots \leq x[\text{size}-1]$$

2  
5 |  
5  
6  
6  
7  
9

16

Let us define the basic problem first. Sorting means given an array like 0 to some size minus 1 reorder the entries so that they are in this way; that  $x_1$  is greater than equal to  $x_0$   $x_2$  greater than equal to  $x_1$ . So, if it be this sorry, why is this greater than equal to coming up? If for example, 2, 5, 5, 6, 6, 7, 9; this also sorted array. So, both these elements could have taken any order, but since they are same they can be assumed to be sorted. That is why this greater than equal to thing that is coming up.

(Refer Slide Time: 22:45)

**Sorting:** the basic problem

- Given an array  
 $x[0], x[1], \dots, x[\text{size}-1]$

reorder entries so that

$$x[0] \leq x[1] \leq \dots \leq x[\text{size}-1]$$

2 |  
5 |  
5  
6  
6  
7

- List is in non-decreasing order.
- We can also sort a list of elements in non-increasing order.

16

The list here is in non-decreasing order, non-decreasing. I cannot say increasing order because it is not increasing 2, 2, 5, 5, 6, 6, 7. So, here of course, there was an increase, but here there is no increase. So, this is rather instead of calling it increasing order, we can call it non-decreasing order. A list of elements in non-increasing order it is not decreasing order. So, I can also sort it in the other way as it shown earlier.

(Refer Slide Time: 23:30)

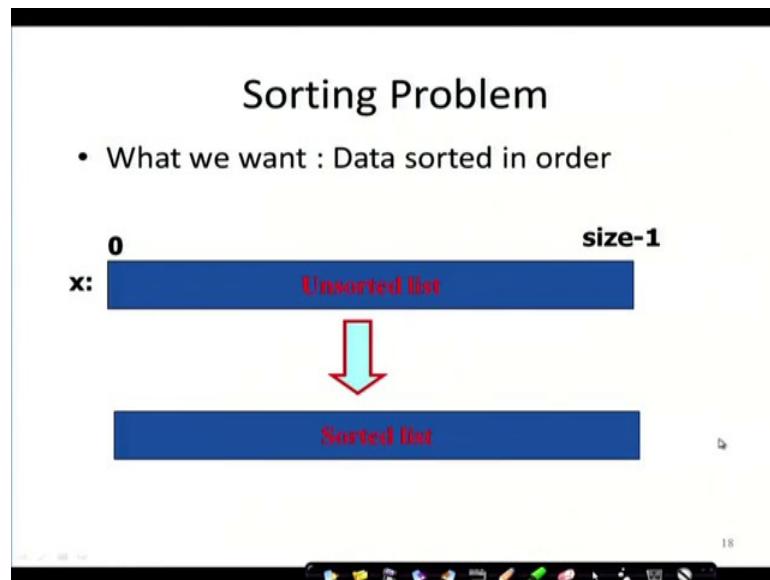
## Example

- Original list:
  - 10, 30, 20, 80, 70, 10, 60, 40, 70
- Sorted in non-decreasing order:
  - 10, 10, 20, 30, 40, 60, 70, 70, 80
- Sorted in non-increasing order:
  - 80, 70, 70, 60, 40, 30, 20, 10, 10



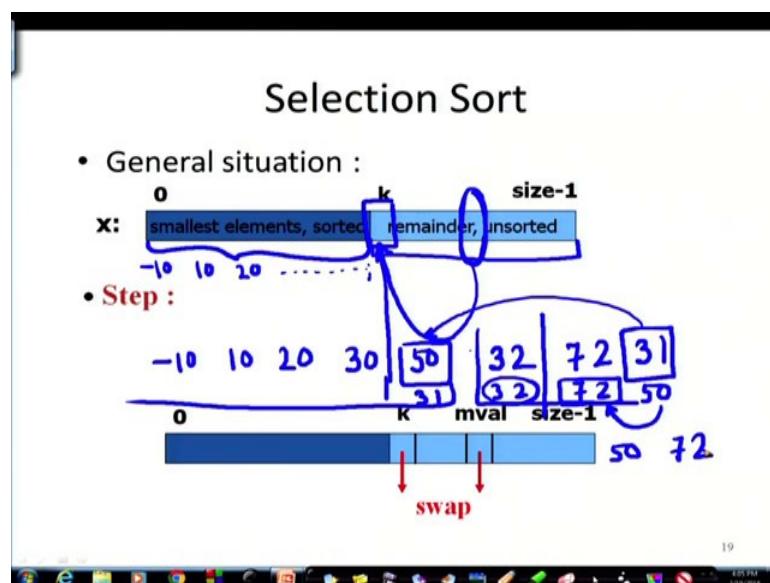
So, here is an example, this is sorted in non-decreasing order, you can see this array. 10, 30 now this original list. Now if I sort them in non-decreasing order it will 10 minus 10, 10, 20, 30, 40, 60, 70, 72, 70's, 80. And if I do it in a non-increasing order, then minus 80, 70, 70 in this way. This is not increasing in this way, here it is not increasing, all right? Here it is not decreasing if I go in this direction. So, these are the 2 possibilities.

(Refer Slide Time: 24:08)



So, the sorting problem is, if we have an array from 0 to size minus 1, I want to sort it and get a sorted list, all right?

(Refer Slide Time: 24:25)



That is what I want to have. So, the first algorithm that will be talking about is selection sort. Selection sort is something like this. That suppose we have got a list here and this part of the array is already sorted, this part of the array is sorted in a non-decreasing order. This already the smallest element is here, the smallest element minus 10, then 10, then 20. This part is somehow as sorted. Now am less left with sorting this part only this

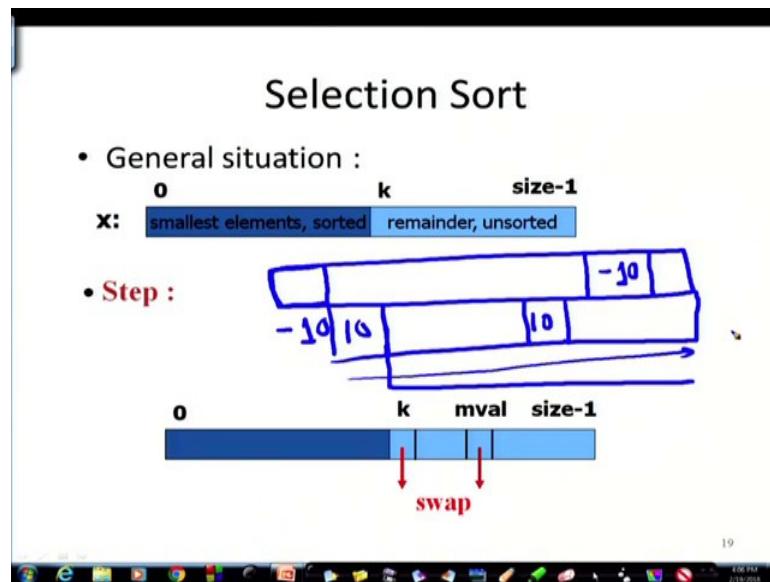
remainder. What should I do? I am going to select the candidate for this position, right. Am going to select the candidate for this position. For that what shall I do? I will start since part is sort, all the elements of this side or either equal to this or less than this.

So, I will have to look at this part, and find out minimum element that is here. This element that is minimum, since this part is sorted, this element this element must be greater than this elements. So, among these, only for these people this one is the smallest. So, I select that I select that, and put it in this position I swap how do I do that? Suppose here it was, it was minus 10, 10, 20, 30. And here it was 50, this point, this part is sorted. 50, 32, 72, 31. Now I remember this position, and search from here and find out the minimum. So, this 31 comes I swap it here.

So, what happens to my array? This part, let us see what it will happen. It will now become from here it was 30 it will be 31, 32, 72, 50. So, up to this part it is sorted now. Now form here I will try to find out the minimum element. I find this is the minimum element. And I sort out the minimum element. I swap this minimum element with itself. So, up to this is sorted. Out of these now I have to find out the candidate for this position.

So, I find out the minimum I come here and swap this. So, it becomes 50, 72. So, here in this selection sort the approach is that we have to we have to we are going position by position. And I am selecting the element that belongs to that particular position. So, whenever am starting with the search then at the beginning I am having am starting with the if this be my array.

(Refer Slide Time: 27:57)



Then am first searching for this position. And this position will have will be the minimum of all these. I search all through here, and find the minimum suppose the minimum must 10 was here.

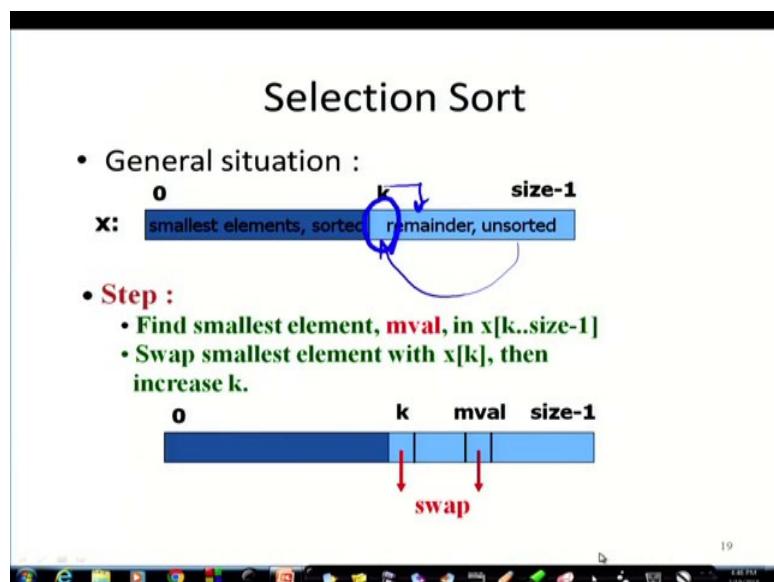
So, I take and bring this minus 10 here. So, it becomes minus 10 I know this is the minimum. Now am sorting am now looking for this position. I select the element, minimum element in this zone. I find here something 10. So, this part is done. So now, 10 is swapped here. So, 10, 10 and then this part is unsorted. This part I will sort. In this we way we go on. This is one of the sorting approaches. We will deal with it further in the next lecture.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 45**  
**Sorting Methods**

So, we were discussing about sorting. And in particular we were discussing about, one sorting technique which is a selection sort. So, in selection sort what do we do we have got an array.

(Refer Slide Time: 00:28)



And of size, so, that means, the index is 0 to size minus 1. So, what we do? We some parts we assume that some parts are sorted. So, first thing that we do is find the smallest element. So, if from 0 to  $k$  is sorted already in the order. Then my job is to sort from  $k$  to size minus 1. So, what we do the step is, that we find the smallest element minimum value in the list  $x$  from  $k$  to size minus 1, all right. I find the minimum value, and then swap the that minimum value with the  $k$  th position; that means, this position, with this position I carry out here in this position I bring the minimum value. And these goes on and then increase  $k$ ,  $k$  will be increased to the next position. And in this way, I will be going on.

(Refer Slide Time: 01:41)

The diagram shows a horizontal array of elements represented by blue rectangles. A vertical line labeled 'k' marks the current position being checked. To the right, a circle labeled 'mval?' indicates the current minimum value found. A bracketed section of the array from index k+1 to size-1 is highlighted in blue, with handwritten notes explaining the task: 'find the minimum in between k+1 to size-1'.

```
Subproblem
int min_loc (int x[ ], int k, int size)
{
    int j, pos; /* x[pos] is the smallest element found so far */
    pos = k;
    for (j=k+1; j<size; j++)
        if (x[j] < x[pos])
            pos = j;
    return pos;
}
```

20

So, the sub problem is the sub problem is to find the minimum element. So, if I go here as we have seen in the, I mean in this array, my task is to find the minimum position ok. So, where is say this part, this part is already sorted, I am sitting here, and this is my k th position, right. This my k th position, and I want to find the minimum where is the minimum value ok. Where is the m value, minimum value where? So, for that, this is the function to find the minimum location in the array x, and starting from this location k, up to the loc up to the location size minus 1.

So, I need to needs know size, I need to need need to know k. That is how we are giving the parameters, as we had explained in functions. So, at every point, you look at how the functions are written that will help you revising this ok. Now the body of the function, x pos will be the smallest element found so far. So, initially I am making pos to be k.

So, x pos is this point, now from k plus 1 to up to size I go on checking, whether this x j this position is less than this position. So, I find out in this way, here I find the minimum right, just see if you can understand the code. And find the minimum in between k plus 1 to size minus 1. In between this am trying to find the minimum. So, whenever I get the minimum am I will be returning that particular position. So, this is the position that I will be returning this position ok. So, what would be my full algorithm therefore? So, I have got a function, I have got this function, min loc, which will return with the minimum location.

(Refer Slide Time: 04:38)

## Subproblem

```
/* Yield location of smallest element in x[k .. size-1];*/  
  
int min_loc (int x[ ], int k, int size)  
{  
    int j, pos; /* x[pos] is the smallest element found so far */  
    pos = k;  
    for (j=k+1; j<size; j++)  
        if (x[j] < x[pos])  
            pos = j;  
    return pos;  
}
```

Now so, it will yield with the location of the smallest element.

(Refer Slide Time: 04:42)

The main sorting function

```
/* Sort x[0..size-1] in non-decreasing order */
int selsort (int x[], int size)
{ int k, m;
  for (k=0; k<size-1; k++)
  {
    m = min_loc(x, k, size);
    temp = x[k];
    x[k] = x[m];
    x[m] = temp;
  }
}
```

So now you see a selection sort, main sorting function will be this. It takes the array and the size ok.

So, this is my array  $x$ , and this is the size. So, I will have to work between 0 to size minus 1. I have got 2 intermediate variables  $k$  and  $m$ . For  $k$  equal to 0 to  $k$  size minus 1, what am I doing? First am starting with this, this element; that is  $k$  equal to 0. And what am I doing? I am finding out from  $k$  in the array  $x$  up to size I am finding the minimum

location. So, suppose here is the minimum. Then I am what am I doing here? Temp is going to  $a_k$ . So, this  $k$  is going to temp. And this  $a_k$  is being returned by  $a_m$  this is the minimum is coming here, and then this temp is going here. So, here what am I doing is, I am swapping  $x$  my array was internally I called the here there is a small mistake I can find. It should be  $x_k$ , all this should be  $x$ , all right because I am dealing with  $x$ .

So,  $x_k$  will get  $x$  minimum  $m$  because and vice versa and this  $y$ . So, what is happening here let us see.

(Refer Slide Time: 06:55)

### The main sorting function

```
/* Sort x[0..size-1] in non-decreasing order */

int selsort (int x[], int size)
{ int k, m;
  for (k=0; k<size-1; k++)
  {
    m = min_loc(x, k, size);
    temp = a[k];
    a[k] = a[m];
    a[m] = temp;
  }
}
```

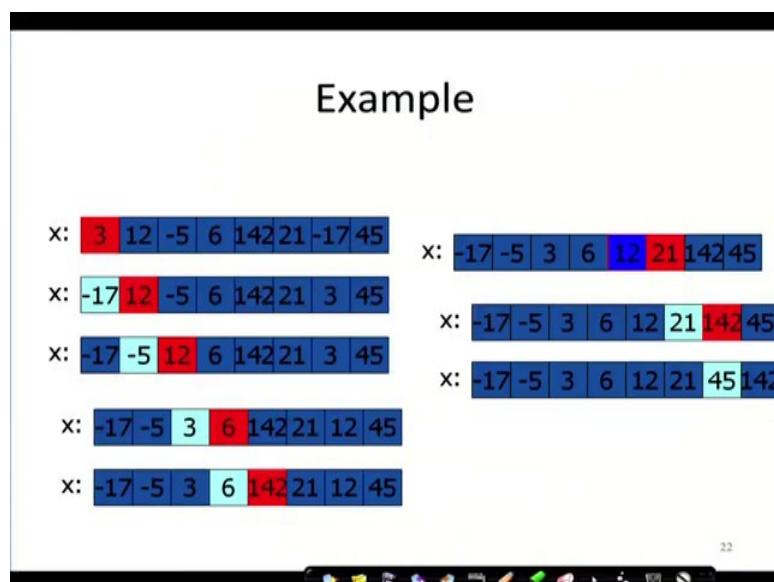
Suppose my array was something like this 50, 20, 70, 10, 100. So, I am first this is my  $k$  equal to 1. I want to bring the minimum at this position. So, what I do? I from here I start searching for the minimum. I find tenth with the minimum, then I swap here is this piece of code, I swap I bring 10 here, and 20, 70 remains and 50 I take here. And then what do I do?  $K$  is incremented so now  $k$  is here. Now I find min loc if what is the minimum here compared less than this or not there is none so, fine next.

So, this remains next I increment this. So, this is also in position, this is in position, because it need not be exchanged next my  $k$  is increased here at this step,  $k$  plus plus it has been increased here. I find the minimum in this area; I find this to be the minimum. So, what is happening? 10 was in place, 20 was in place, I swap 50 and 70 here. So, 50 comes here I find the minimum, 70 remains here, 100 remains here. Now I change my  $k$

and see if there is any minimum left here at the min location is not returning me anything. So, therefore, I get this to be my sorted array ok.

So, this is what so, you can see that so in this way we can, sorry, here is an example you see here.

(Refer Slide Time: 09:07)



The example you see here, I start with this. So, 3 is the minimum, I can see here 12 minus 5 6 etcetera I can see, minus 17 is the minimum. So, I find out minus 17 here I swap that. So, minus 17 comes here, and 3 goes to the place of minus 17. So, these 2 now 12, 12 and I find the minimum from 12 onwards, I find minus 5 to be the minimum. So, that one will come here. So, minus 5 comes here, and 12 is swapped there. Now within this part I find the minimum, minimum is 3. So, 3 will be swapped with 12, and 12 goes there in the place of 3 you see. Now so, I start with 6, I start finding the minimum. 6 is the minimum fine. So, it remains I increase k. So now, 142, in this way, I go on from here, who is the minimum for starting from 142? 12 is the minimum.

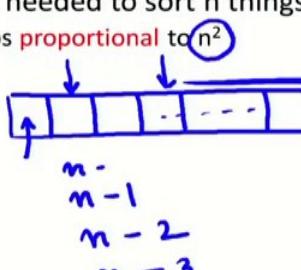
So, 12 and 142 will swapped 12, and 142 has been swapped. Next 21, 21 within this 21, 142, 45, 21 is the minimum. Therefore, I keep it in place. Next is I take 142 and 45, I find that 45 is minimum I swap that, and 142 goes there. Next is 142 is the, that is the sorted array that we get. This is what is known as selection sort. So, the philosophy of selection sort is, that am selecting the minimum element among the unsorted elements

and placing it in the position proper position where that minimum element should be; that is why we are selecting and putting it so, that is why it is known as selection sort.

(Refer Slide Time: 11:27)

## Analysis

- How many steps are needed to sort n things ?
  - Total number of steps proportional to  $n^2$



23

So, how many steps are needed to sort n things here? If I do this, how many steps am I taking. You see, how many steps am I taking? All through here if you look at this, how many steps am I taking? If the list size of the list was n then the total number of steps that are required is proportional to  $n^2$ , because for every element why it is why it is so? Let us try to understand this.

The reason is that for every position, how many positions are there? 1 2 3 like that, there are n positions. Now for every position, for this position, how many comparisons I need to find out the minimum?  $n - 1$  comparisons  $n$  comparisons for all these, next I increment it here. So, how many do I need? I need for each of these for  $n$ , for  $n$  such elements for each of them I need  $n$  comparisons then  $n$  plus for this I need  $n - 1$  comparisons, then I need  $n - 2$  comparisons, then I need  $n - 3$  comparisons in this way for example, for this  $n - 1 - 2 - 3$ . So,  $n - 4$  comparisons I need in this part, I have to compare between these things. So, in that way I go on. So, for every so, for each of them I need to compare this, right for  $n$  such things, I have to do this,  $n \cdot n$  plus  $m \cdot n$  plus  $1 \cdot n$  minus times  $n$ . So, that will be the order of  $n^2$  if I find the product, right.

(Refer Slide Time: 13:49)

## Analysis

- How many steps are needed to sort n things ?
  - Total number of steps proportional to  $n^2$
  - No. of comparisons?
$$(n-1)+(n-2)+\dots+1 = n(n-1)/2$$

Of the order of  $n^2$
  - Worst Case? Best Case? Average Case?

23

And number of comparison total number of steps and number of comparisons is again,  $n$  into  $n$  minus 1 by 2 like as I said  $n$  minus 1 comparisons plus  $n$  minus 2 comparisons, last one is one comparison. So, if you add this one to  $n$  minus 1 is  $n$  into  $n$  minus 1 by 2, you know that, and that is of the order of  $n$  square, if I break it up it will be  $n$  square minus  $n$  by 2.

So, that is again of the order of  $n$  square of the order of we write it as of the order of  $n$  square all right.

(Refer Slide Time: 14:24)

## Insertion Sort

- General situation :

**x:**  $\begin{matrix} 0 & \dots & i & \dots & \text{size-1} \\ \text{smallest elements, sorted} & \text{remainder, unsorted} \end{matrix}$

Compare and Shift till  $x[i]$  is larger.

24

Now next we come to another sorting algorithm, which is known as insertion sort. What is the general situation? Again, just like the selection sort, we have got the smallest element sort; it from 0 to this point is sorted. The remaining is unsorted, all right. Now what we do is we compare and shift till  $x_i$  is larger ok, I go on shifting.

So, what I do is I start with this element, and I put this in the proper position, I insert this in the proper position I. So, I move this in this side or in this side, and I will come to the proper position of this element, this element is being put since it is smaller the smallest elements is sorted. So, I am taking this, and I am putting it in the place where it should belong because I have not seen up to this.

(Refer Slide Time: 15:42)

```

Insertion Sorting

void InsertSort (int list[], int size)
{
    for (i=1; i<size; i++)
    {
        item = list[i];
        for (j=i-1; (j>=0)&&(list[j] > i); j--)
            list[j+1] = list[j];
        list[j+1] = item;
    }
}

```

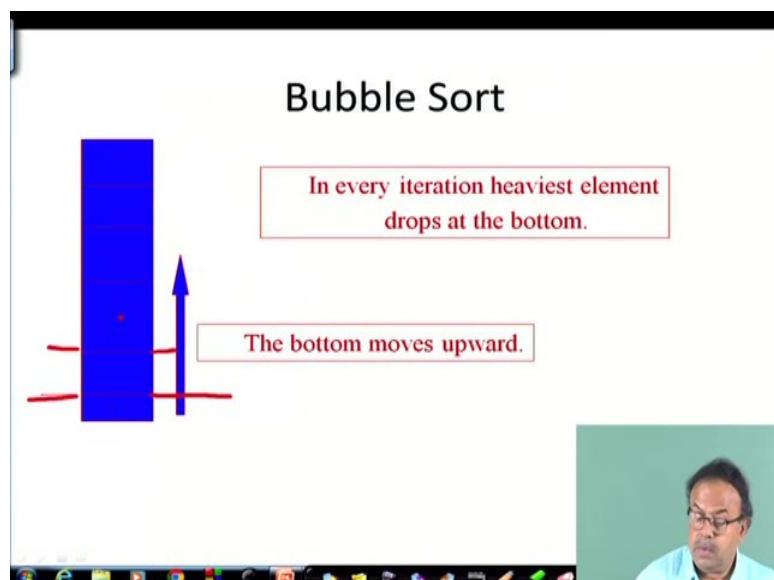
So, so, insert sort is something like this let us see if I can show you a example first.

So, what am doing is, I am comparing and shifting, till  $x_i$  is larger. I am continuously shifting the elements, in this way I go on shifting this. Ultimately  $x_i$  will be larger than all these elements. So, therefore, my sorted position will be up to this, this is larger right. So, I am sorry, I just went the other side other way. So, let us study this algorithm. This algorithm is void, why it is void? Because it is not returning anything it is taking the array and is sorting this is taking this array and this sorting this. So, its type is void it is taking a list and it is size. Now for I equal to 1 to size, I am taking item to the list I. So, I have got elements like say 12, 7, 5, 17, 9. So, first I am taking I is one so, am taking this element. This is the item. For js now I am looking at you see j equals I minus 1, and j is

greater than 0 and list. So, it is not am I am not going into this side. I will have to go in these direction, what is this mean? I start with  $j = 1$ , and go till  $j$  is greater than equal to 0, and the list is greater than  $I$ .

So, all right so, my  $I$  is here; that means, I am not I am going till the end of this array on the other side, I am going on the other side. And I shift what was my  $j$  here,  $j + 1$  and the list are being swapped. So now, it will become 7, 12, 5, 17, 9. Now and then in that way I go on shifting, all right. So now, if I come here, I increment  $I$  after one item so, I will come 5 and 5. I will go on checking, what is the proper place of 5? So, it will be 5 will be swapped with this 12, 7, 17, 9. In that way I will try to find out every each of these is for proper position. So, I think we can go for a more popular sort.

(Refer Slide Time: 19:07)



Let us go to the more popular sort, I think that will be better selection sort you have learnt insertion sort is another form. But let us learn the more popular one that is the bubble sort. Bubble sort as the name implies. It starts from the bottom, and the minimum element is pushed up to the top, how is that done? Just as a bubble floats up from the bottom and goes to the top the minimum element should be pushed up the minimum element should be pushed up to the top.

So, let us see how it happens. So, I first compare between these 2 elements, then these 2 elements, and in this way, I go on comparing between each of them and push it up ok. in every iteration, the heaviest element drops at the bottom. So, what is happening is so, if I

go here from the top, I try to find out if this element, if this particular element, I am sorry this particular element is heavier than this particular element. Then this particular element should go at the top and this should come down here, all right? Next, I compare sorry, next I compare what is happening? Next, I compare between this so, this was shows the heavy element. And I find that this is not an heavy element. It is this heavier than this. So, this may be more heavier if much heavier than this. So, this is not pushed out ok, remains. Now next I compare between this, and every time I am making the same mistake may be this is the heavier, lighter element. So, this lighter element will go up, and this heavier element will come down.

So, it will be something like this, all right. So, in this way in one direction ultimately what I will do? I will have the heaviest element at the bottom, and all the elements here are lighter than this, this element lighter than this element, but these elements are not sorted. So, what I will do? I will restrict my next iteration within this zone, and again start between this elements, and see whichever is heavier that will be pushed down. Am I clear?

You see, every time I am comparing between these and am pushing that down. I think it will be clearer in a moment. Every iteration the heaviest element drops at the bottom and the bottom this part is done and the bottom moves upward. So, next time onwards my search will be restricted between this zone, this zone to the top and the heaviest will come to the next time my search will be restricted to this zone like that. So, the heaviest here, next heaviest here, next heaviest here, heaviest here, next heaviest here, next heaviest here, like that it will go on. So, I want to see if I can give an example first.

(Refer Slide Time: 23:15)

The screenshot shows a Windows desktop environment. In the center, there is a terminal window titled "Terminal" with the command "ls" entered. The terminal output displays the contents of a file named "output.txt". The file contains the following text:

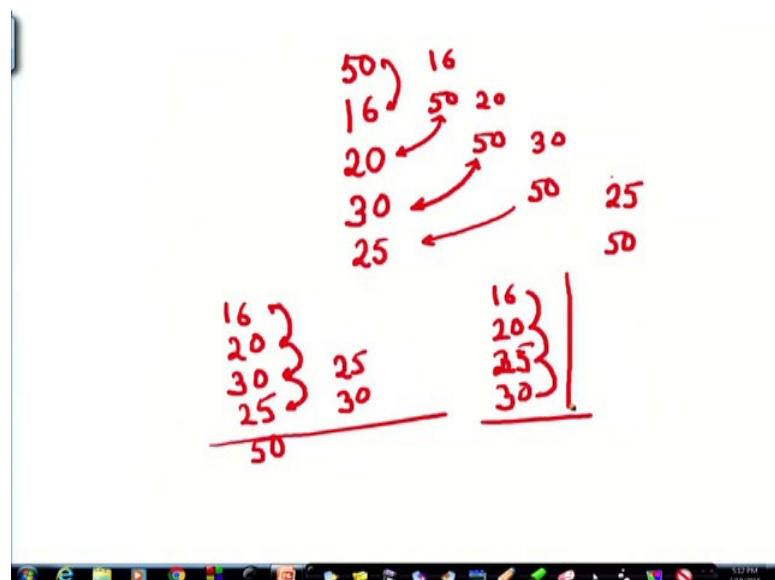
```
Contd.  
int main(int argc, char *argv[]){  
    int x[ ]={-45,89,-65,87,0,3,-23,19,56,21,76,-50};  
    int i;  
    for(i=0;i<12;i++) printf("%d ",x[i]);  
    printf("\n");  
    bubble_sort(x,12);  
    for(i=0;i<12;i++) printf("%d ",x[i]);  
    printf("\n");  
}  
-45 89 -65 87 0 3 -23 19 56 21 76 -50  
-65 -50 -45 -23 0 3 19 21 56 76 87 89
```

The terminal window has a black background with white text. The desktop background is a light blue color. The taskbar at the bottom shows various icons for programs like Internet Explorer, Google Chrome, and File Explorer.

Yeah, so, what is happening here is, say this was an unsorted array. The heaviest one has been first pushed down this 89 was the heaviest one.

So, by piece by piece comparison I have pushed it down. Next time I was restricted to this part of the array 89 has been pushed down. So and let me try to show you.

(Refer Slide Time: 24:10)

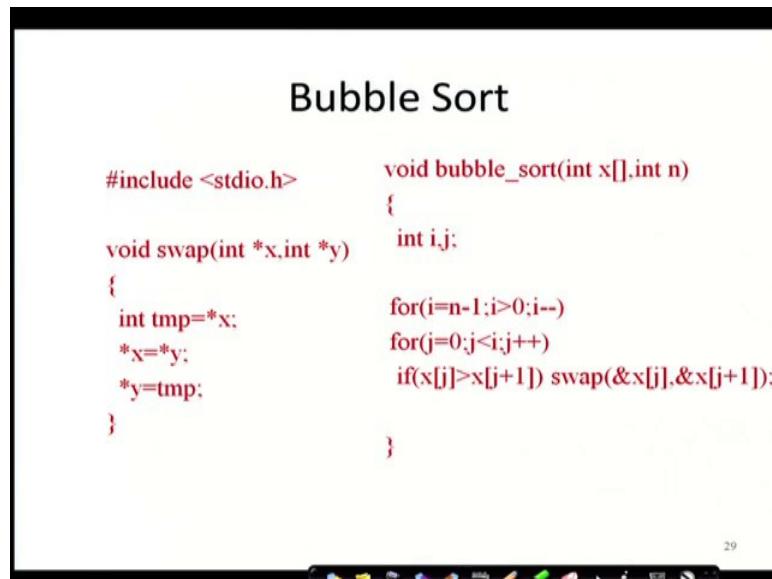


That say, suppose I have got an array like this 50, 16, 20, 30, 40. So, first I compare between these 2, 16 50 is heavier. So, I come here next, I compare between these 2 50 and 20, 50 is heavier. So, 20 comes here 50 comes here. Now 50 and 30, 50 is heavier.

So, 30, 50 and then 50 and 40 so, I compare this. So, it becomes 450. So, in one turn what do I get now I get 16, 20, 30, 40, 50 now it was search the many things are sorted only one was in out of order so, that is been done. But suppose this was let me let me just change it, it becomes became too easy. So, suppose it is 25, then after this swap, this became 25 and 50. So, I have this array. Now I again check between this 2. Heavier is in place, I check between this 2, no issue heavier in place, I check between this 2.

So now what will happen? 25 will come here, and 30 being heavier will come here. And 30 and I need not compare any further. I can because this is this I know is the heaviest. So, what do I have now? 16, 20, 30 not 30, 30 has been swapped 25, 30. I again compare between this 2. So, there is no change, as soon as I find that, there is no interchange; that means, all this things are in order. This is the idea of the selection sorry the bubble sort. Every time the minimum is shifting going to the bottom. Or I could have done it in the other way I could have justified the name of the bubble sort, if I had taken the lowest element, and pushed it up ok. So, quickly if we look at the algorithm a little bit.

(Refer Slide Time: 27:14)



## Bubble Sort

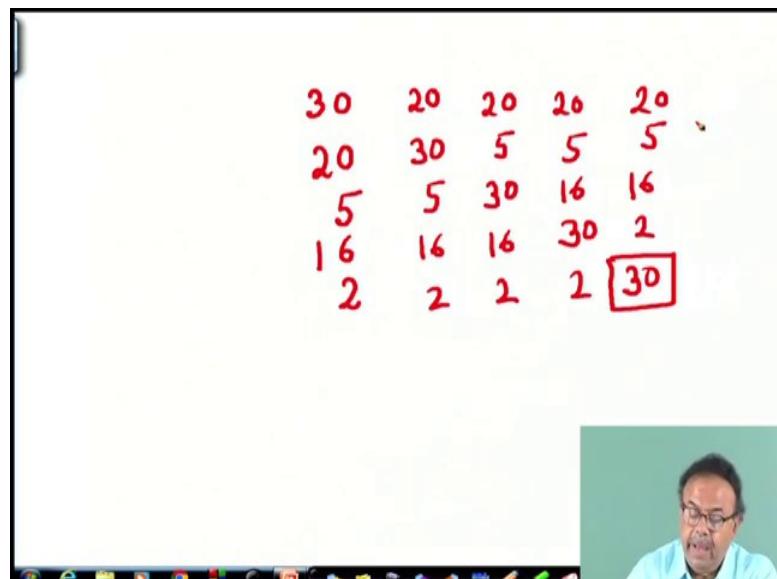
```
#include <stdio.h>
void swap(int *x,int *y)
{
    int tmp=*x;
    *x=*y;
    *y=tmp;
}
void bubble_sort(int x[],int n)
{
    int i,j;
    for(i=n-1;i>0;i--)
        for(j=0;j<i;j++)
            if(x[j]>x[j+1]) swap(&x[j],&x[j+1]);
}
```

29

So, we have got this is a bubble sort algorithm. Let us see, now in order to explain this, I will need sometime, because here we are using some new ideas, star x and star y. I think I will take it up in separate lecture, and discuss this separately, all right. But just as I had explained now, the way the bubble sort works, I will just give you one more example to

show how a bubble sort works. And then in the next lecture, I will show you the algorithm because that will require some more explanation.

(Refer Slide Time: 28:02)



Let us say, I have got this 30, 20, 5, 16, 2. So, what will be hap happening in the first iteration? 30 will be compared with 20, 30 is heavier. So, it would 20, 30, 5, 16, 2. Now 30 will be compared with 5, what will happen and this will be swapped. 30, 16, 2. 30 will be compared with 16. So, what will happen? 25, 16, 30 2 and 30 is again greater than 2. So, 25, 16, 2, 30. So, 25, 16, 2, 30 by that I get this 30 at the bottom, I now I start with again; 25, 16, 2 right.

(Refer Slide Time: 29:05)

A computer monitor screen showing a whiteboard. The whiteboard has a 4x4 grid of numbers in red: Row 1: 20, 5, 5, 5; Row 2: 5, 20, 16, 16; Row 3: 16, 16, 20, 2; Row 4: 2, 2, 2, 20. Below the grid is a box containing the number 30. To the right of the whiteboard is a video feed of a man with glasses and a blue shirt, sitting in front of a green wall.

So now, I have got 30 is at the bottom 25, 16, 2. And 30 is already in proper place, here I will swap. So, what will happen? 5, 20, 16, 2 then 20 will and 16 so, it will be 5, 16, 20, 2, then 5, 16, 2, 20, 20 is in the proper place. So, 30 and 20 are in proper place. And 5, 16, 2 I will have to handle.

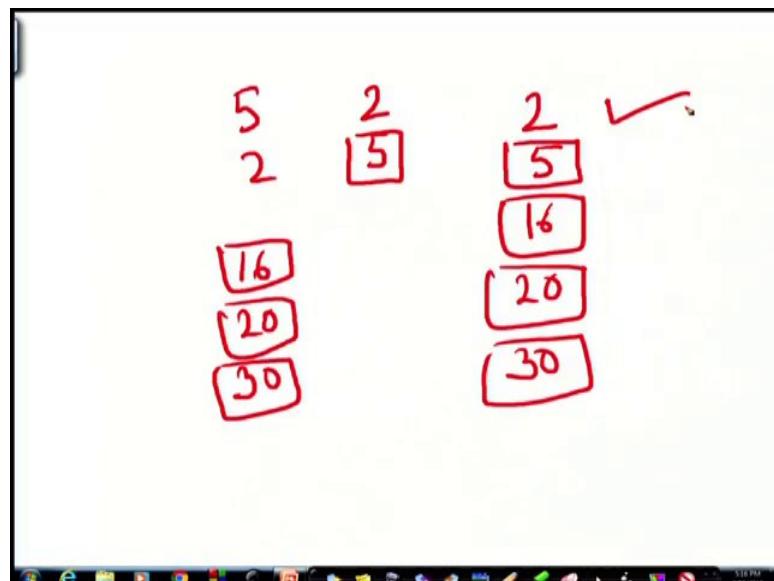
(Refer Slide Time: 29:44)

A computer monitor screen showing a whiteboard. The whiteboard has a 4x4 grid of numbers in red: Row 1: 5, 5; Row 2: 16, 2; Row 3: 2, 16; Row 4: 20, 30. Below the grid are two separate boxes, one containing 20 and another containing 30. To the right of the whiteboard is a video feed of a man with glasses and a blue shirt, sitting in front of a green wall.

So, what will be there? 5, 16, 2. So, 5, 16, 2, and here I have got 20, and 30 already in the proper place sort it. So, here 5 and 16 no change, 16 and 2 there will be some change. So,

5, 2, 16 so, 16 is in the proper place. So now, between 5 and 2 so, my thing will be 16, 20, 30 are in place.

(Refer Slide Time: 30:10)



16, 20, 30 are in place, and 5 and 2 I have to deal with, these are fine. So, my pointers are changing, right, between these 2 now 5 is heavier. So, it will be 2, 5, 5 is in the proper place. So, 2 5 in the proper place, 16 proper place, 20 proper place, 30 proper place. So, only 2 there is no question of swapping. So, I get the complete sorted file ok. So, this is the principle of bubble sort, I will be explaining the algorithm the code in the next class. In the meanwhile, you can think of you can read book and you can try to see look at this code and the algorithm yourself. I will discuss it in the next week lecture.

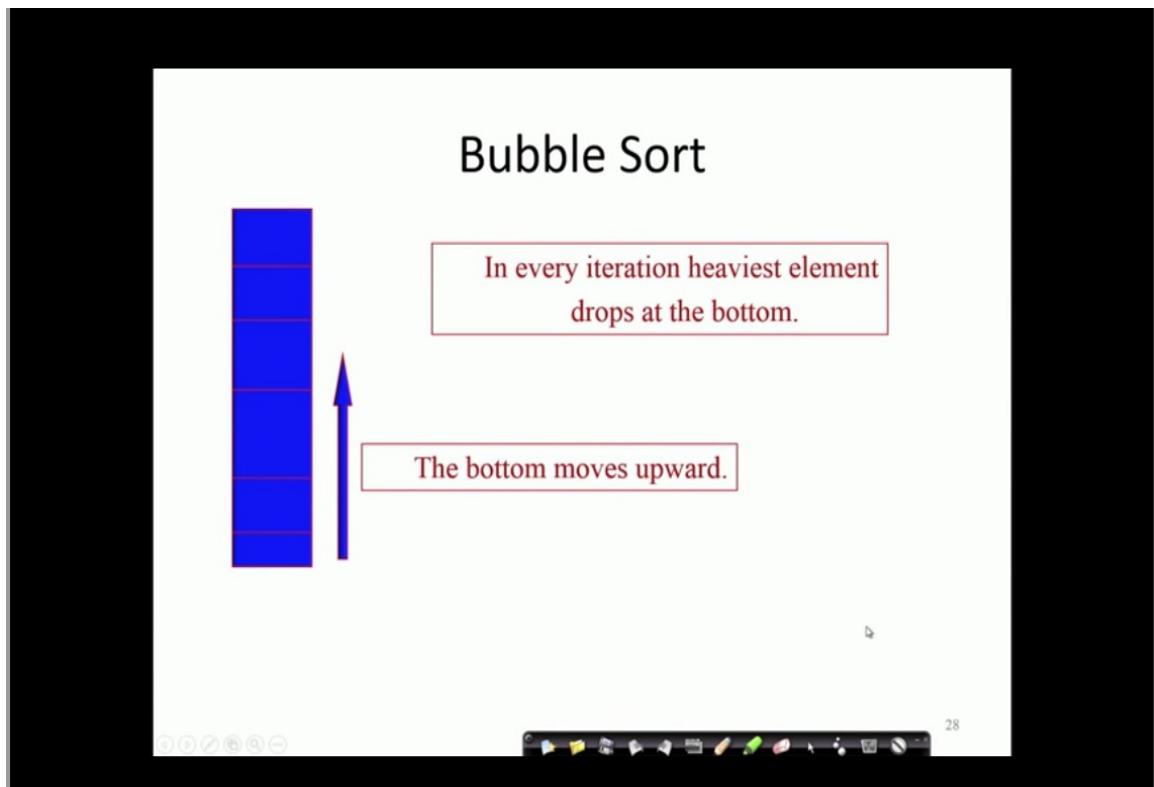
Thank you.

**Problem Solving through Programming in C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture-46**  
**Bubble Sort (Contd.)**

So, we were discussing about the Bubble Sort Algorithm and the approach as was discussed in the last class was that every element, every iteration pushes the heavier element at the bottom and the lighter elements therefore, goes up.

(Refer Slide Time: 00:33)



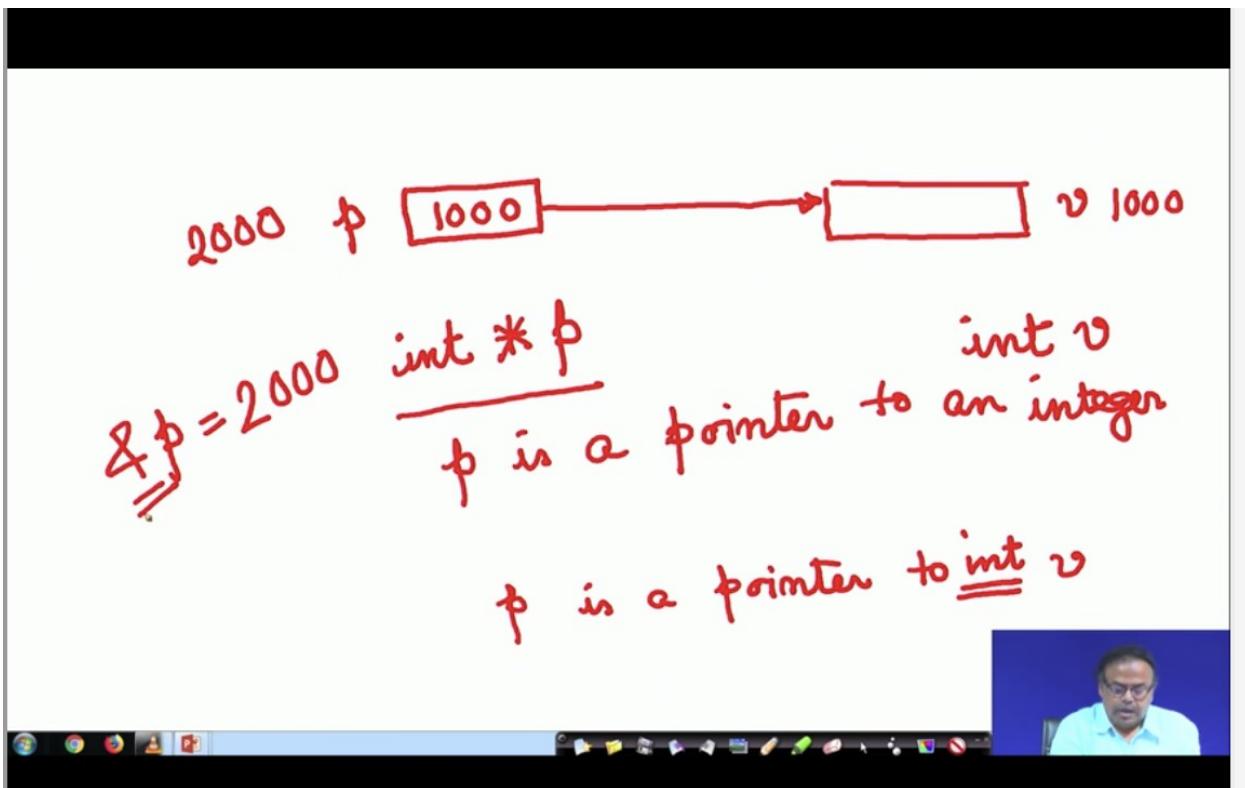
So, we had shown an example a couple of examples by which we can show that the heavier elements go down and the other lighter elements go up , we had given a number of examples to that effect.

So, if there be 50 here I do not know this visible, let me write it here 50 here and 6 here and say 12 here and 7 here then what will happen is 50 will be swapped with 6 because 50 is heavier than 6. So, larger than 6. So, first it will be 6 and 50 and then 50 will be compared with 12 again there will be a swap. So, there will be 12,50. 50 will be compared with 7. So, it will be 7, 50 in that way ultimately the 50 if it is heaviest will come at the bottom.

So, we swap only if the element at the top is heavier there by heaviest element is coming to the top and the others which are less than this heaviest element will be going at the top, the bottom moves upward. Now, we are going to write the algorithm for this or the program, for this before writing the program we need to understand a very interesting and fundamental concept of C programming that is pointers.

We will not formally introduce pointers here, but we will be talking about pointers in a different way in a context of this bubble sort. Now, what is a pointer? Pointer is something that points. Now suppose there is a variable here, suppose the variable is named v and till now we have seen that we can write declare the variable as int v. So, v has got a particular address may be that address is 1000, 1000 is the address of v. Now, if I say int v equals 50; that means, in the address 1000 50 is written.

(Refer Slide Time: 03:43)



Now, suppose we have got this address v and which is an integer.

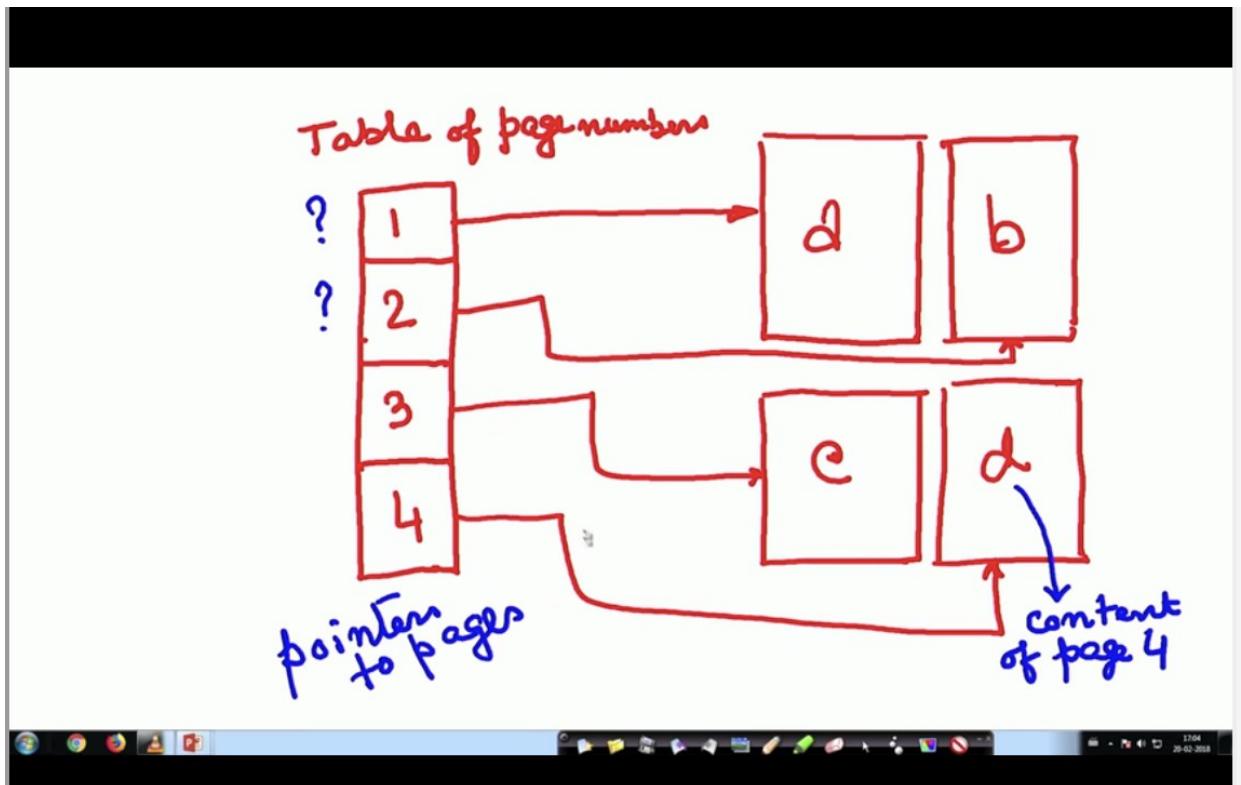
But I am not declaring v as such, but there is another variable p, which will always point to some integer. So, p will point to this v. P is pointing at this v. In that case I can say p is a pointer to v now. So, therefore, what did I say? I said that v is stored in the location 1000, now pointer p this variable p therefore, stores the value 1000.

Now this v is of type integer. So, v is a pointer to an integer v that we know because I have earlier said that v is an integer. So, I can very well say int v, but what is the type of p , p is a pointer to v, which is an integer. So, we write that as int star p, what does it mean? It means that p is a pointer to an integer, if I write it ,it is not concerned with v only. It says that p is a pointer to an integer.

So, what is the value of p now? Value of p is 1000. What is the address of p the address of p? Now where is this p ?This is in location 1000 and where is this ?This might be in location 2000. Therefore, what is the address of p? &of p is 2000, we know that this and operator gives us the address, we have encountered this during scan f when we were discussing about scan f right.

So, let us have this diagram to understand the concept of pointer suppose there are different things written in different pages of the book.

(Refer Slide Time: 07:08)



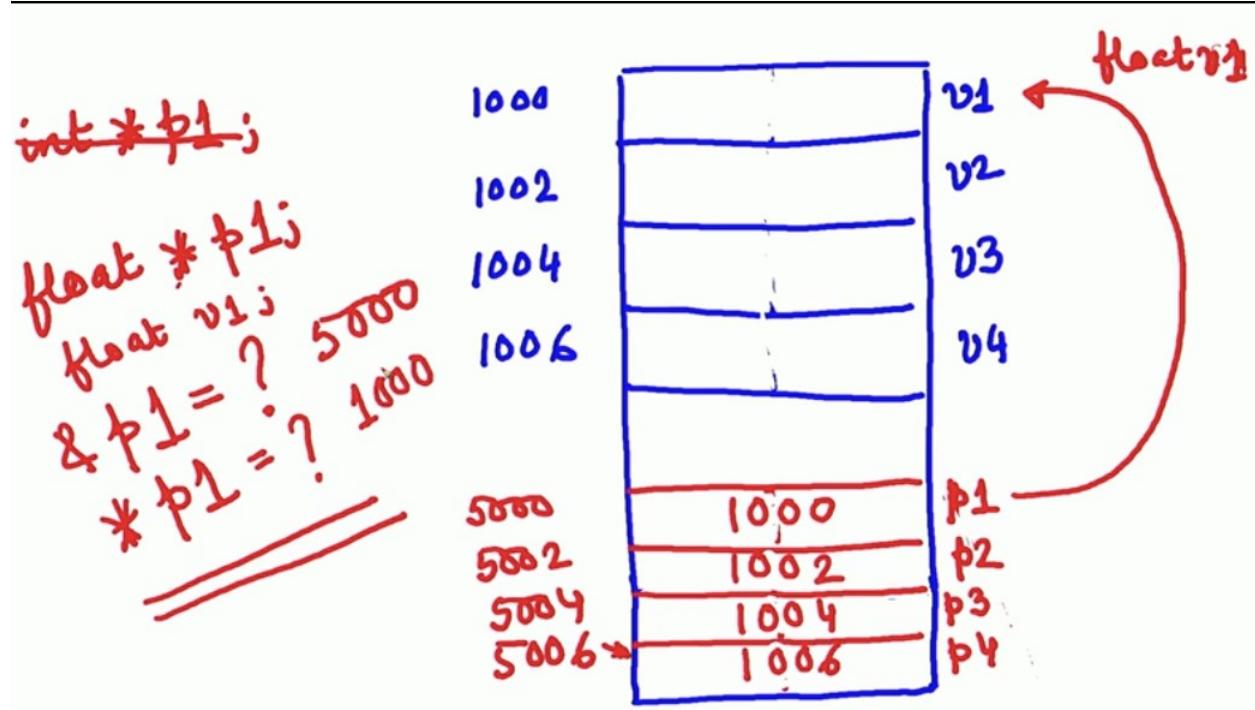
This 1 page, this is another page ,this is another page and this is another page, each of these pages have got a page number. Now I have got a table where suppose- this has got some content a, this has got some other story b, this has got some other story c, this has got some other story d.

Therefore, the content of each of this pages are the stories a, b, c, or d. But which page I am looking at ? Suppose here is a table of page numbers. So, I have got a 4 pages. So, the pages are 1 2 3 4. Now, suppose I just want to come to this page number. So, this page number does not tell me what story is there, but this page number tells me that I have to go to this page. This entry is telling me, this particular page number is telling me that, I have to go to this page. This is telling me that I have to go to this page and this is telling me that I have to go to this page.

So, what are these? What are these things? These are nothing, but pointers to pages and what is 'a' is a content of the page. So, a b c or d say is content of page 4 and these are pointers to the pages; that means, if I want to go to page 4- where should I go? Because this page may be located at some other table may be, suppose this pages of a book are torn out and have been kept on different tables ok.

So, page number 1 is giving me the table address. So, that it is on this particular table. So, it can be on table t 1, this can be on table t 2, may be the third page is on table t 4 and this is on table t 5 it is possible. If I want to read the first page I will come here -the first entry this is the point in to table t 1. So, and I go to table t 1 and to find this particular page and read the content in that . So, these are pointers to pages, but these pointers also have to be in some memory location -let me draw the diagram in different way now ,suppose this is my memory and I have got 4 variables.

(Refer Slide Time: 10:57)



Suppose I have got 4 integer variables and each integer variable takes 2 bytes. I am showing 2 bytes here one after the other all right, this is the variable v 1, this is the variable v 2, this is the variable v 3 and this is the variable v 4 .

Similarly let me just change the colour. Let this be in locations- v 1 is in location 1000 assuming that an integer takes 2 bytes, this one is in location 1002, this one is in location 1004 and this is in location 1006.

Let me extend my table bit, now suppose I have got here 4 integers. So here I have got p 1 is a pointer variable- since it is a integer let me say- p 2 p 3 and p 4. I am not bothering about how many locations it is taking.

I am now not bothering about this division here 2 bytes am just showing those 2 bytes together that will be easier for me to write all right. So, here the p 1 is storing the value 1000 why? p 1 is a pointer to this variable. So, this p 1 tells me not the value of v 1, but it tells me where is v 1 located.

So, 1000 is a position where v 1 is located. So, p 1 is storing that pointer and p 2 is say 1002, p 3 is 1004 and p 4 is 1006. Now these locations also have got address- suppose these addresses are 5000, 5002, 5004 and 5006 this 5006 ok.

So, please try to understand when I say p 1- what is p 1? p 1 is a pointer to an integer. So, int star p 1; that means, that p 1 is an integer all right and p 1 is a pointer to an integer variable . It could be a floating point variable. So, had it been a floating point number.

Now suppose each of them is a float. So, float v 1. So, v 1 is a floating point number. So, then if p 1 points to v 1, then I should write float star p 1; that means, p 1 is pointing to a floating point number .

So, since it is a floating point number ,if I say now &p1 what will I get? what is the address of p 1? what is the address of p 1 address of p 1 is 5000 and what is star p 1? What is the meaning of this? Please do not be confused. This star and this star has got two different meanings, this star

means it is just saying that p 1 is a pointer, pointer to what type of variable? Pointer to a floating point type of variable.

So, for v 1 am just writing -float v 1 so; that means, that v 1 is purely a floating point number not a pointer, but when I say float star p 1, then p 1 is a pointer to some floating number. Now, what is star p 1 in this case what is a content of p ?Content of p 1 is 1000 So, let me just repeat it once again. So, we have seen in declaration something like int star p 1 int v 1.

(Refer Slide Time: 18:47)

## Bubble Sort

```
#include <stdio.h>
void swap(int *x,int *y)
{
    int tmp=*x;
    *x=*y;
    *y=tmp;
}
void bubble_sort(int x[],int n)
{
    int i,j;
    for(i=n-1;i>0;i--)
        for(j=0;j<i;j++)
            if(x[j]>x[j+1]) swap(&x[j],&x[j+1]);
}
```

70

Now; that means, p 1 is a pointer ,suppose located at location 5000 and p 1 is a pointer to a variable v 1, which is also an integer- v 1 located at location thousand.

This two lines are just meaning these two things- this two pictures, this 5000 or 1000 this part is not known to us, because that is known by the compiler not by us, but logically , this is the picture, leave aside this 5000 and 1000 ,only these 2 parts are declared by this. Now if I say star p 1; that means, the content of p 1, whatever is the content of p 1 -now if I make p 1 to point to v 1 in that case if I make p 1 point to v 1.

So, what will that be? Content of p 1 should be assigned to &of v 1. The &of v 1 will give me the address of this and the content of p 1 will get that. Therefore, this 1000 will be written over here; that means, this pointer is pointed to this v 1. I will not go in to further details of this right now, we will come to this as an when the context comes later. Right now we this in mind let us go back to the algorithm of bubble sort, what is being done on the algorithm of bubble sort.

(Refer Slide Time: 20:59)

Let us try to understand each of the functions -first thing is we have got this standard equation. Now, first I will start with this function void bubble sort -bubble sort is void, because it is not returning any value ,it is just sorting some element -what array? x array and the size of the array is n, int ij these are 2 local variable because they are using used as indexes of these two loop. In this loop what is happening?- for I assigned n minus 1 to 0; i minus minus; what does that mean? That means, if I have an array here this is my array I have got I pointing here.

(Refer Slide Time: 22:07)

# Bubble Sort

```
#include <stdio.h>
void swap(int *x,int *y)
{
    int tmp=*x;
    *x=*y;
    *y=tmp;
}
void bubble_sort(int x[],int n)
{
    int i,j;
    for(i=n-1;i>0;i--)
        for(j=0;j<i;j++)
            if(x[j]>x[j+1]) swap(&x[j],&x[j+1]);
}
```

Because, that is the size of the array minus 1. So, i is pointing here and it will go up to 0. So, gradually it will go in this direction up. That is what the first for loop is saying. First for loop is saying that I is starting from the bottom and going up upto 0 as long as it is greater than 0 and then for j equal to 0 to j less than i ;j plus plus.If xj is greater than xj plus 1; that means, what if xj is greater than xj plus 1.

So, let us again go back to this j is starting from 0 and I am comparing this value 50 with the value 2 with j plus 1.This is j plus 1 if xj is greater than xj plus 1, what would I do I would swap. Let us go into here if xj is greater than xj plus 1 I am swapping xj and xj plus 1, I will come to the swap function later am assuming that the swap is working correctly- that means it is swapping them.

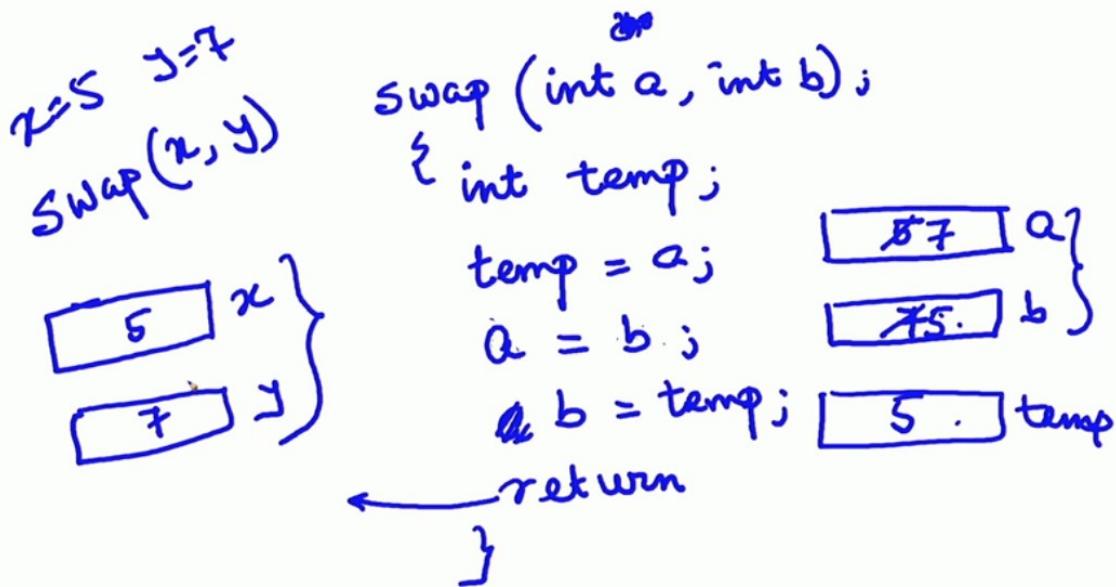
So, what is happening? So, this loop will go on- j equal to 0 then j will be 1 and in that way it will go on. So, next when I swap it becomes 2 and 50. Now, I have got other values now j is being incremented. So, now, this is becoming j and this is becoming j plus 1- in that way this swapping will be done. Suppose this is 70. So, there will be no swap no swap.

So, let us see here, if this condition does not hold -swap is not taking place.I am going back to this loop .Now, 70 remains here . So, now, j changes and j plus 1 is pointing here now there will be a swap here. So, this is 12 say -12 and 70 will be swapped. So, 2,50,12 70 in this way I will go on up to this point and next.

So, ultimately in this iteration ultimately I will have this heaviest element say 70 is the heaviest element, then I will be decremented and I will come here and then from this zone again I will do the same thing as I did till now . Exactly that is being done here. So, I will be doing this and then I will decrement n little bit and I will go on doing this. I will be then looping- doing the same loop again for 1 level less I mean the last element is already having the heaviest element. So, I will do it among the remaining part.

So, this part this part is already covered. So, I have decremented it. So, I will be now playing in this zone and once that is done this part will also be coveredI will be playing in this zone .Every time am deciding on the zone , n, n minus 1 n minus 2 like that it is going on and every time I am doing this loop for each of the pair wise element comparison and they are by am pushing the heaviest element at the bottom. That is the code for bubble sort how it works this swap keeps a I have not saying this swap. Earlier we have seen that if we do swap in any arbitrary way.

(Refer Slide Time: 27:48)



For example, if I want to swap 2 variables say I write swap int a, int b- let me do it here , if I just do swap int a, int b and inside I use another variable temp, temp gets a , a is stored here a gets b and then temp has been remembered there. So, b gets temp and so, this was my function return.

So, here suppose a (when it came) it was called with x and y s. So, main program called swap with x and y and x was 5, y was 7 right. So, x was a location -here 5 and y was a location 7. Now when it came here 'a' was a local location and I know that the functions are called by value so, 5 was copied in a and 7 was copied in b and then by this temp became 5, b became 5 and b was stored in a. This was stored in this -these became 7 and this again became 5 -all right. That is how it was done.

So, a was 5 so, temp was 5. 7 went over here to a and here 5 was copied again here- this was fine. So, this swapping was done by a by b and then when I returned no change is reflected in x and y -that was the problem.

So, we will now see how the swap function can be implemented. Another point is I have just done return and a function can return only 1 value. I could not return a and b to the main function. So, how can the thing be managed. So, that after the swapping the result is reflected back in the main function. We will see that in the next lecture.

**Problem Solving through Programming in C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture-47**  
**Use of Pointer in Function: Context Bubble Sort**

So, we are discussing about bubble sort and we had looked at some notion of pointers. So, you have now got the preliminary idea about pointers.

(Refer Slide Time: 00:33)

The slide is titled "Bubble Sort". It contains two columns of C code and a diagram of an array. The left column shows a swap function:

```
#include <stdio.h>
void swap(int *x,int *y)
{
    int tmp=*x;
    *x=*y;
    *y=tmp;
}
```

The right column shows the main bubble\_sort function:

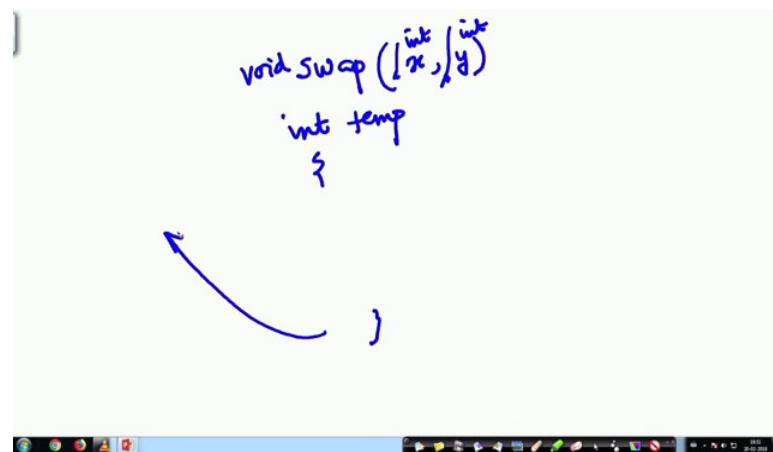
```
void bubble_sort(int x[],int n)
{
    int i,j;
    for(i=n-1;i>0;i--)
        for(j=0;j<i;j++)
            if(x[j]>x[j+1]) swap(&x[j],&x[j+1]);
}
```

Below the code, there is a diagram of an array with indices i and j. The array elements are labeled 1000, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 100A. A blue box highlights the element at index 1005, which is labeled "50".

On this segment we have seen this part of the code, we have seen - this is the bubble sort function on this side, we have got a function which is swapping 2 variables. I want to swap x and y.

So, this swap function is taking x and y. So, we have seen earlier that in the normal case if we if we swap in a function (suppose I have got a void swap xy)

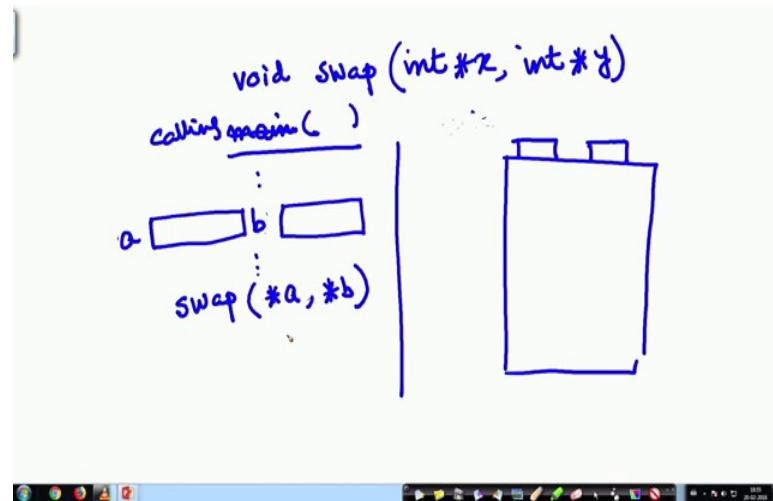
(Refer Slide Time: 01:26)



Then we are shown (we used another variable int temp) that whatever swapping I am doing here that is not being reflected in the main function, that we saw. Now what we are doing here let us see here I am writing void swap, instead of writing x and y I am writing int star x comma int star y; that means, what I am passing here is my swap function, here is my swap function and it has got 2 input parameters a function has got always one output, there can be a number of input parameters earlier it was x and y.

So, some variables x and y were having actual parameters being copied into them, but now I have not used x and y instead I am using pointer to x and pointer to y.

(Refer Slide Time: 03:30)



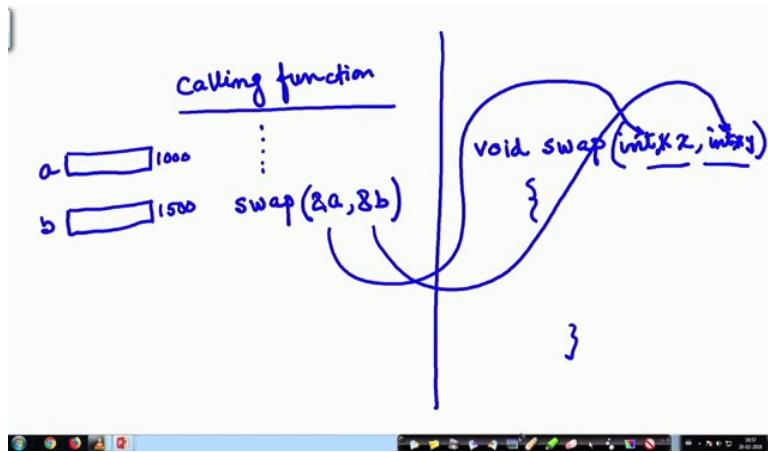
That means, now I am in my main function -here is my main function or the calling function not necessarily the main function. In this case it was called from bubble sort. So, the calling function -I should not say main it should the calling function. The calling function has got 2 variables x and y.

Now when I am calling this function void swap. So, I am actually having the function here -the swap function. I am not passing the values of x and y, to be to be more natural let us make this actual parameters are a and b.

So, somehow I am calling here swap in my calling function I am calling swap a comma b. If I had done this then this value of a and the value of b would have been copied there, but instead what I am doing now- I will not write this instead I will write int star a and int star b. So, I can say here I am passing star a comma star b; that means, what? I am passing the pointers.I will I will do it again, let us look at this say.Here I am passing the address. So, again let me do it in a nicer way.

So, here is the calling function.

(Refer Slide Time: 06:09)



The calling function has got the variables a and b and inside the calling function I am saying swap- what should I pass on I instead of saying a and b, I am saying and a comma and b . You know what is and a and and b? And a means the address of a and b means address of b right. So, I am actually passing the addresses of a and b. So, the address of a was say 1000 and this was maybe 1500 .

Now here is my called function void swap int, now a and b have been declared- int star x comma int star y. So, what is happening- what is being passed? Here and a is being passed and here and b is being passed; that means, this address and this address are being passed 1000 and 1500 are being passed.

Also I know here that what I am getting are the pointers to the variables that had to be swapped it was not int x or int y-let us go back to this once again. So, here you see I am calling  $x_j$  and  $x_{j+1}$ , in my bubble sort algorithm. In my bubble sort algorithm I had the array and j was here and  $j+1$  was here- this was already filled something and I am comparing. So, here is my i. So, I will work in this zone and I am looking at two elements  $j$  and  $j+1$  and I want to swap them.

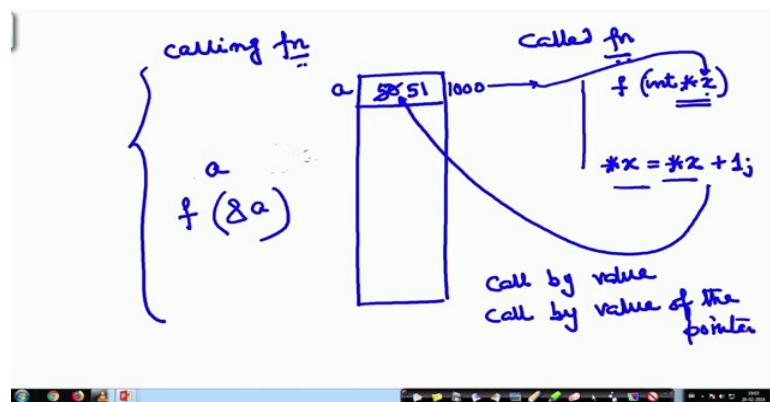
So, what I am doing is I am passing on the address of this here. So, suppose this address was thousand and this address was 1002. So, this is being passed here. So, this is 1000 and this is 1002. Now what is happening here -temp is a local variable, see temp is not a pointer, temp is a local variable. Suppose here the point was this was 50 and this was 2.

So, temp is getting the content of x right? So, 1000s content thousands content was 50, content of the pointer, content of x. So, the 50 is being copied here and then what is happening content of y is coming to content of x. So, content of y is coming to content of x.

So, this is becoming 2, this is being swapped and is becoming 2 and temp is coming as the content of y. So, this is becoming 50. So, what is happening all the changes were reflected here in the main body of the data. So, that is how we could really reflect the change on to the calling function.

So, stated in another way it would be like this -let us say I have got a data here and here is my calling function and here is my called function.

(Refer Slide Time: 10:32)



Now, there is some variable that is being used by the calling function say a; that means, this a is somewhere here, this a has been referred here. It is not proper to show it here again. Now the calling function is using the variable a somewhere here alright. So, when this variable is being used as a parameter to this called function, instead of sending the variable -say some other function I call and in that I pass on the address of that variable a.

So, that address whatever this address is -this is 1000, that is being passed on to this called function. So, the called function on the other hand knows that its input parameter is say called function is f -is int what I am getting is int star x; that means, what I am accepting as a called function is nothing but a pointer to an integer variable.

Now so, the pointer to the integer variables -the 1000 comes up and here suppose it does something with the content. Now here when I take this say star x; that means, x is a pointer x is a pointer to an integer and when I take star x asterix x; that means, it's a content of this and suppose I do it star x plus 1; that means, the content of this address same address x was 1000 and the content of that 1000 is being added. So, here it was 50. So, that is being added ,where it is being added here. So, this is changing to 51.

So, this if you recall we had talked about two things when the functions are called we said that usually c always calls by value except for the case of arrays, but here is an example where I can also say this has been called by value of the pointer. So, in this case it is an example - call by value of the pointer. That means, what I have copied it is only the pointer. Therefore, the change that is being reflected is being reflected in the main program.

So, in that way it is also a call by reference. So, in that way we can reflect the change of the function. So, if you ever faced the crisis that well I am writing a function, but I know a function can only return 1 value, but I have got multiple values to return - this is a nice way in which you can do it .

So, this is this we have a new concept we have learnt the use of pointers the notion of pointers and how it can be used in the context of just learning another language. There is a bubble sort . Once again we revise the bubble sort -let us go to the next one and will come back.

(Refer Slide Time: 14:42)

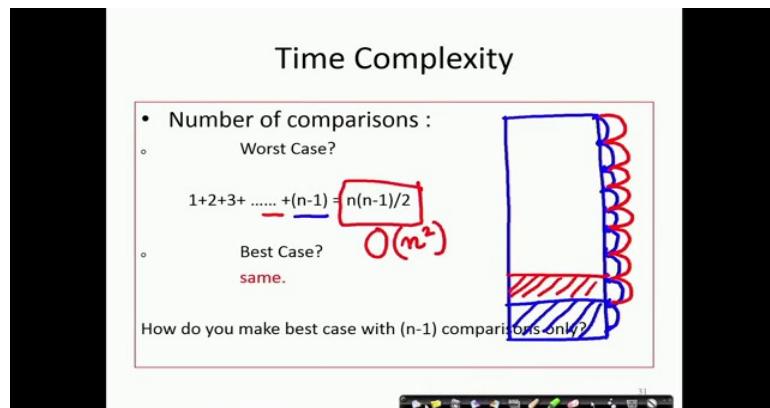
Contd.

```
int main(int argc, char *argv[])
{
    int x[ ]={-45,89,-65,87,0,3,-23,19,56,21,76,-50};
    int i;
    for(i=0;i<12;i++) printf("%d ",x[i]);
    printf("\n");
    bubble_sort(x[12]); ←
    for(i=0;i<12;i++) printf("%d ",x[i]);
    printf("\n");
}
-45 89 -65 87 0 3 -23 19 56 21 76 -50
-65 -50 -45 -23 0 3 19 21 56 76 87 89
```

So, the main program (is taking a main function) is reading an array. This array is being read. I is an integer. Now for i 0 to less than 12 there were 12 elements here- print the array , it is calling bubble sort with the array being passed and the size of the area being passed here. Now we go to bubble sort; that means, I go to the earlier slide. So, what I had seen earlier.

So, there we go and call the bubble sort and the bubble sort in turn calls, swapping is done inside the bubble sort and then we print the ultimate data all right.

(Refer Slide Time: 15:50)



So, what will be the time complexity. We have seen the time complexity in terms of the big o notation at times. So, what is the worst case - let us once again recapitulate think of the algorithm that for something like this -first I had this size n.

So, initially the number of comparisons were 1 then worst case 2; I first compare between these two, then I compare between these two, then I compare between these two. Now, depending on the comparison I may or may not exchange all right may not swap, but I have to do n minus one comparisons. So, first I do n minus one comparisons and if along with that I have done the swapping. So, the first position is now filled up with the largest element.

Then again I have to start from here compare again twice, thrice. In that way I go on comparing the second time and by now this number of comparisons is n minus 2. So, the second one will also be filled up with the heaviest element and in that way I will go on ultimately last time is 1.

So, the overall if I add them the number of comparisons is this and that is of the order of  $n^2$ , because it breaks down to  $n^2$  minus  $n$  etc. What is the best case? What will be the best case? The best case is if you do not have to do any swap it is already sorted, but still you do not know that it is sorted you will have to compare them anyway.

Therefore, the best case will be the same. Can you make the best case with  $n - 1$  comparisons only? That we can think of later, So all these searching and sorting we were doing was using arrays.

(Refer Slide Time: 18:22)

The screenshot shows a presentation slide with a white background. At the top center, the title 'Common pitfalls with arrays in C' is written in a small, dark font. Below the title, there is a bulleted list of two items. The first item is 'Exceeding the array bounds'. To its right, there is a code snippet with red annotations: a red arrow points from the word 'Exceeding' to the number '10' in the line 'int array[10];', and another red arrow points from the word 'bounds' to the condition 'i<=10' in the loop 'for (i=0; i<=10; i++)'. The second item in the list is 'C does not support array declarations with variable expressions.' Below this list, there is a code snippet for a function:

```
void fun (int array, int size)
{
    int temp[size];
    ...
}
```

Now, just a word of caution about some common pitfalls with arrays in C - is that often we can exceed the array bound without really noting of that. For example, array size has been declared to be 10; that means, my indices are from 0 to 9 right. So, if I write a for loop like this and I try to initialise it.

So, it will initialise to 1 0, then 1, then 2, then 3, till it is less than equal to 10. So, the tenth element is here -that will also be put to 0, but that may be destroying some other memory location therefore, that check is not there and c does not support array declarations with variable expressions.

(Refer Slide Time: 19:20)

**Some Efficient Sorting Algorithms**

- Two of the most popular sorting algorithms are based on **divide-and-conquer** approach.
- Basic concept:

```
sort (list)
{
    if the list has length greater than 1
    {
        Partition the list into lowlist and highlist;
        sort (lowlist);
        sort (highlist);
        combine (lowlist, highlist);
    }
}
```

So, quickly if we summarise what we learned here. We are not talking about this in this course, but what we have learned here is searching- we have seen and we have seen two very efficient methods of searching. One is the linear search which is simple and the better method is saw a binary search, but binary search is effective when the array is already sorted.

We have learnt about how to sort an array, we have particularly looked at two sorting algorithms- one is selection sort, another is the bubble sort and while discussing about bubble sort we have also looked at a very interesting way of communicating between a function a caller function and the called function, when I have to communicate or pass on or return more than 1 variable to the calling function. There we are using the pointers.

So, we introduced the concept of pointers which are nothing, but the addresses of the variables. So, instead of passing on the value I am passing on the address and whatever I am doing I am doing on the address. So, that is another important thing that we have covered in this lecture. Next, I think we will go to some other concepts of arrays .

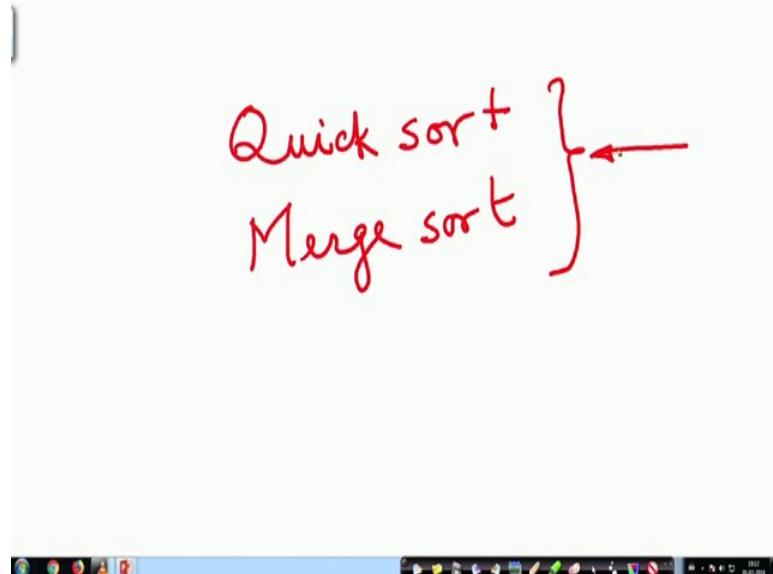
(Refer Slide Time: 21:12)

## Quicksort

- At every step, we select a **pivot element** in the list (usually the first element).
  - We put the pivot element in the final position of the sorted list.

Now, just to mention here we have seen that the complexity of these algorithms are of order of  $n^2$  both for selection sort and for bubble sort, but can we have it better? We have got two very nice sorting algorithms; one is known as the quick-sort another is known as the merge sort .

(Refer Slide Time: 21:41)



Quick sort and Merge sort, which reduces the complexity of sorting and are very popular algorithms, but we are not discussing it in this course for those of you interested can learn it later.

Next we will be moving in the next lecture to discuss about how strings can be handled as arrays -that will do in a separate lecture. So, up to this for now and will continue in the next lecture ok.

**Problem Solving through Programming in C**  
**Prof. Anupam Basu**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture-48**  
**Arrays at Strings**

---

Today, we will be having a look at how the techniques that we have learnt can be applied to different Problem Solving and this will continue from this lecture onwards for a few more lectures. We have learnt quite a few things we have seen the overall structure of the C language.

We have seen the if then else if l structure the iteration structure, like the loops we also know how the variables are defined, we have seen arrays how data can be stored in 1 dimensional array and 2 dimensional arrays, we have also seen functions. And the most important thing about functions is that it enables us to take a complex problem and divide it into different parts and solve each of the parts separately using functions. A very important thing about functions is passing of the parameters and how the function returns of value returns a value to the calling function.

We have also seen that there are two ways in which parameters can be passed from the calling function to a called function, one is called by value another one is called by reference in c language in general always call by value is adopted.

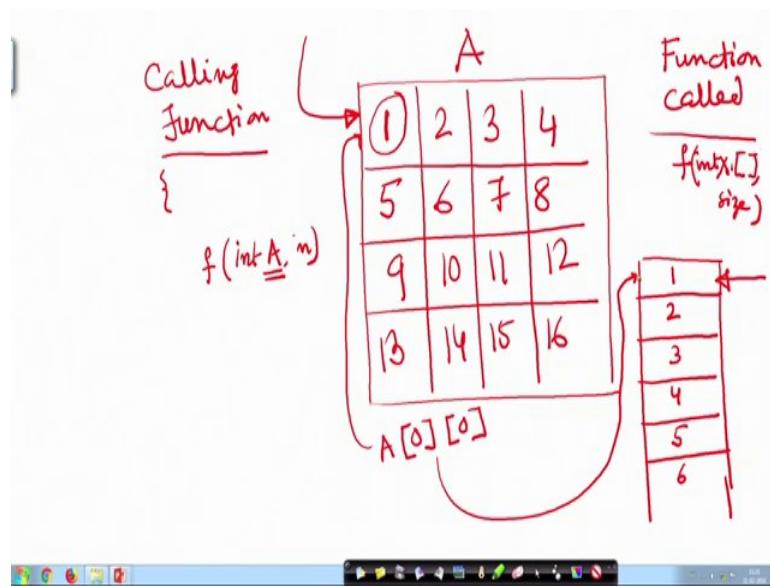
However, in call by value we have to copy the value of the data the parameter value the parameter value to the argument value. However, for the case of arrays whenever I want to pass a particular array at that point an array can be a 100 500 array or 10 by 10 array if it be a 10 by 10 array then I will have to copy hundred values. So, that is not advisable for that we just passed the array name.

(Refer Slide Time: 02:49)

## Array of Strings

And let us quickly have a look that the array name is essentially an address right and array whenever we pass suppose there is an array or 2 dimensional 10 by 10 array all right 2 dimensional array like this.

(Refer Slide Time: 03:09)



Now, all the elements I am not passing. So, it is a essentially it happened to be a 4 by 4 area. So, that is all the 16 elements I will not be passing. So, here is my called function the function called function and on this side is the calling function, the calling function

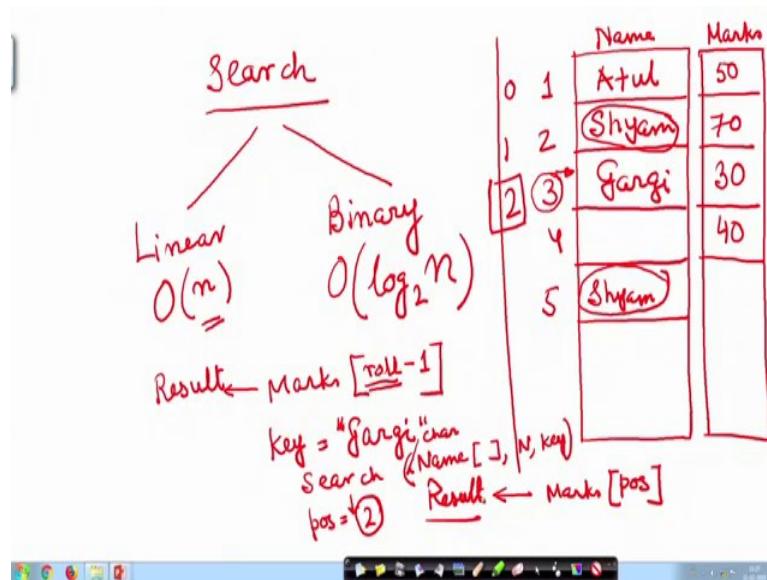
and the array name is say a. So, the calling function refers to suppose let this function be called f.

So, in from the body of the calling function whenever I am calling f, I am passing the array say if it be an integer array into A. And here I just passed the array A and here in this function body we have got say int A and just saying that the size of the or if it may not be a it can be x also, int x something and the size of that and here int A and maybe n. So, here I am actually passing the name of the array and the name of the array is nothing, but the beginning of the address of the first element of the array. So, here I am actually copying the address.

So, it is a call by reference in the case of an array. And in the last lecture we have also seen that we can also use pointers to sort of establish call by reference. Another major thing that you have to keep in mind is that suppose this array is 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16. Now, these are organized in C in a row major form. So, it is stored in the memory like 1. So, the rows first 2 3 4 then 5 6 like that. So, when I refer to the array A I actually pass on the address of the first location of the array ok. So, A 0 0 in the case of A 2 dimensional array points to this element and that is this element that is how we pass on arrays.

So, for functions and we have also seen some applications of all these techniques that we have learnt in particular we have seen how we can carry out search.

(Refer Slide Time: 06:40)



And we have seen two search techniques one is linear search another is binary search. Now in linear search the complexity the time is proportional to the number of elements whereas, in binary search it is reduced it is faster it is log of the number of elements to the base 2. And search is a very fundamental problem solving technique in technique, because suppose you have got the names of students in the class all right.

Here are the names Atul, Shyam, Gargi in that way the names are there and each of them have got a roll number 1 2 3 4. Now you want to find out how many Shyams are there in the class. So, you have to search this array and find out where Shyam is ok, maybe here and there is another Shyam here. So, that is searching and counting both the things, but you cannot avoid searching this array all right. It may be that along with that there is an array of marks obtained by roll number 1 roll number 2 etcetera.

So, maybe this is 50 this is 70 this is 30 all right. Now here tell me I want to say now they are in 2 arrays this is 1 array called name and this is another array called marks of the corresponding students. Now I want to know how much what is the marks that Gargi has obtained. How do you go about it what would be the first step in solving this problem what is the marks that Gargi has opted, in order to do that I have to first I do not know what is Gargi is roll number.

If I had known what is Gargi's roll number suppose Gargi's roll number is 3, then I could have gone in the marks array and I could have taken marks roll number what if I knew the roll number, roll number minus 1 that marks I could have taken as a result all right, but I do not know the roll number if I had known the rule number 3 then I would have gone to the second element. So, it is marks 2 would have been would have given me the roll number, but suppose I do not know the roll number then what should I do?

First I will have to search this area and what would be my key what is my key will be the screen Gargi, because I want to know the marks obtained by Gargi and what should I do with this key I should carry out a search, it can be a linear search or it can be a binary search let us for the time being assume that it is a linear search. So, we will be searching and I will be finding Gargi here I am assuming for simplicity that there is only one Gargi in the class.

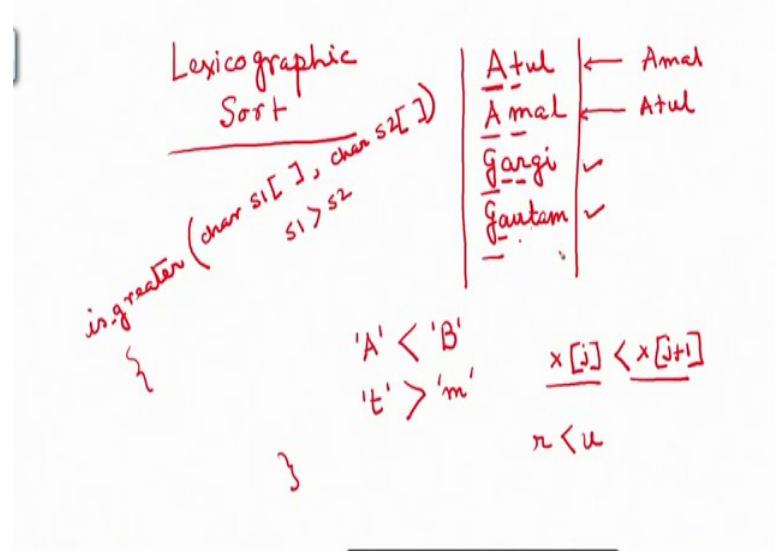
So, I will take this and I find what will the search, if you recall what will the search algorithm return me the search algorithm will return me the index of the position where

Gargi is located. And that position will be actually it will automatically come since this positions will be marked as 0 1 2 I will get this 2 returned. So, I will get. So, I get the key I search the array name this is a character array A care name size whatever size is say n and the key.

And this search will return me the value position 2 pos equal to 2, then I will simply take the value of marks I go to the array marks pos, which will be my result right. So, you see we needed the searching. Now, while I was talking of searching this array I said that the searching I am assuming this searching to be a linear search.

Would it be possible to sort this array using binary search, if you just think it over would it be possible to search, it using binary search what does binary search require binary search requires that the array be sorted right array be sorted.

(Refer Slide Time: 13:00)



So, an array like this, Atul, Amal, Gargi and Joy this is an array of 4 elements can I sort it the type of sort if, I want to sort it I will have to organize it in increasing or decreasing order let us assume that I am trying to do it in an increasing order. Now, this sort will be this character array can be sorted in an way which is called the lexicographic sort.

That means I go I sort the character array according to the alphabets or the words as they written; obviously, A comes first. So, therefore, A character A is smaller than character B all right.

So, here between these two elements both of them has got A, but similarly t is greater than m. So, in this sorting I will first check between these say a they are same think of bubble sort I am trying to compare them in normally what we did we found that say array  $x_{xj}$  is less than  $x_{j+1}$  and then we did something.

If it was if it is the case no issue if it was the other way round I wanted to do a swap, but in this case it will not be. So, simple I can take a string and compare these two strings as to which one is whether they are same or not or whichever is greater, but another way you can write a small function to compare I leave it to you I leave it to you that you now know sorting techniques.

So, you should try to write a function that will sort a string of characters all right. So, e and e are same, if they are same the function what will do it will now look at t and m now m is smaller than t therefore, Amal is smaller than Atul.

So, Amal will come here and Atul will come here now between Atul and Gargi you can see a is greater than smaller than G. So, Gargi is in a proper space please. Now if say for example, I do not have Joy here I am having another name Gautam here. Now when I am comparing between Gargi and Gautam G and G are the same.

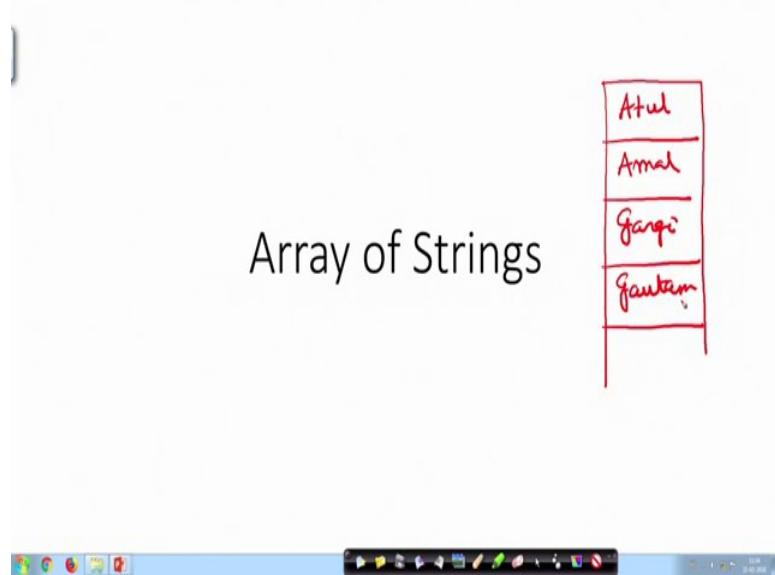
So, they are equal no change next a and a are also same no issue. So, there is no change, but r r is smaller than u, because r appears earlier than u, but in terms of our computer code technology the Ascii code of r is less than the ascii code of you.

So, in both the ways it is meaningful. So, r is less than you therefore, Gargi is smaller than Gautam. So, in that way I can carry out with another small function compare string is greater say is you can write a function is greater 2 strings ok, care S 1 comma care S 2 2 strings will be passed on this function and this function will find out, which character if S 1 is greater than is 2, it will just return if S 1 is greater than is 2 it will return A 1 if S 1 is greater than S 2 try to write such a function ok. Now, that is that I am giving you as a home assignment all of you should try to do that and with a little effort I am sure you will be able to do that.

So, I could have just as I had done in the earlier one, here in this thing I said that I will be doing linear search, I could have done binary search also provided I had provided, I had sorted them provided I had sorted them in a proper way and then with the sorted array I

could have done binary search. Now, will be also doing some problem solving within this course will be doing some problem solving in the field of numbers or equations mathematical problem solving, but before that in order to do this today let us have a relook at the strings, we had looked at strings earlier as an array of characters.

(Refer Slide Time: 18:55)



So, let us have a relook at the whole thing today how can I represent an array of strings like the one that, I was showing here and array of strings all these names Atul, Amal, Gargi, and Gautam all these are nothing, but this is a string this is a string all these are different strings. So, how can I represent them let us have a look at that.

(Refer Slide Time: 19:30)

- char ch\_arr[3][10] = { { 's', 'p', 'i', 'k', 'e', '\0' }, { 't', 'o', 'm', '\0' }, { 'j', 'e', 'r', 'r', 'y', '\0' } };
- char ch\_arr[3][10] = { "spike",  
"tom",  
"jerry" };

spike  
tom  
jerry

s	p	i	k	e	\0				
t	o	m	\0	-	-				
j	e	r	r	y	\0				



Say I can I have got 3 strings one is spike and there is tom and there is jerry. Now these 3 strings I have to store and I am saying that each string can be at best 10 characters at most 10 characters. And I am looking at 3 such names. So, it is an array 3 rows and each row having 10 characters right. So, the definite that definition would be I named them as char I could have done character array and it is 3 rows and 10 columns now I can initialize that as spike as character S character p.

So, s comes here, p comes here, i comes here, k comes here, e comes here and that is the end. So, it is backslash 0 I also put that and so, 1 2 3 4 5 6 4 places are still vacant 4 places are still vacant. Similarly for the next 1 is tom so, t o m back slash 0 and the rest are rest 6 are all free all right, but my string has ended here. So, 1 way I is that I can represent them in this way where, I specifically talk about the characters in the array and terminate them with backslash 0. The other way I can do it is simply I initialize them.

(Refer Slide Time: 21:35)

The slide contains two code snippets:

- `char ch_arr[3][10] = { { 's', 'p', 'i', 'k', 'e', '\0' }, { 't', 'o', 'm', '\0' }, { 'j', 'e', 'r', 'r', 'y', '\0' } };`
- `char ch_arr[3][10] = { "spike", "tom", "jerry" };`

To the right of the second snippet is a hand-drawn diagram of a 3x10 grid. The first row is labeled "spike" with characters s, p, i, k, e, and \0. The second row is labeled "tom" with characters t, o, m, and \0. The third row is labeled "jerry" with characters j, e, r, r, y, and \0. Dashed lines extend from the end of each row.

A small video player window in the bottom right corner shows a person speaking.

As  $3 \times 10$  the same array with 3 strings spike comma, tom comma, jerry. Now internally even if I do that it is essentially becoming the same thing internally it is being stored in the same way with 3 rows and each row having this characters s p i k e backslash 0 and so and so forth, there can be more t o m back slash 0 and so and so forth here.

Also j e r r y and so, on I am sorry backslash 0 and so and so forth it is being stored. So, both of them are equivalent I can do it in either of these ways alright. So, a string can be represented in any of these ways.

(Refer Slide Time: 22:45)

The slide contains a list item:

- We already know that name of an array is a pointer to the 0th element of the array.

To the right of the list item is a hand-drawn diagram. It shows a pointer variable `ch_arr` pointing to the memory location of the first character of the first string, which is 's'. A curly arrow also points from `ch_arr` to the label `ch_arr[0][0]`. To the right of the array, the three strings are written as `spike\0`, `tom\0`, and `jerry\0`.

A small video player window in the bottom right corner shows a person speaking.

So, we already know this again the name of the array is appointed to the 0th element of the array. So, if I have this array s p i k e backslash 0, t o m backslash 0, j e r r y backslash 0 when I am calling that char the name itself is pointing to the first element of this array. So, ch arr 0 0 it is pointing to that.

So, this and ch arr 0 0 are the same they mean the same thing ok. So, this you just like an array this is also true here.

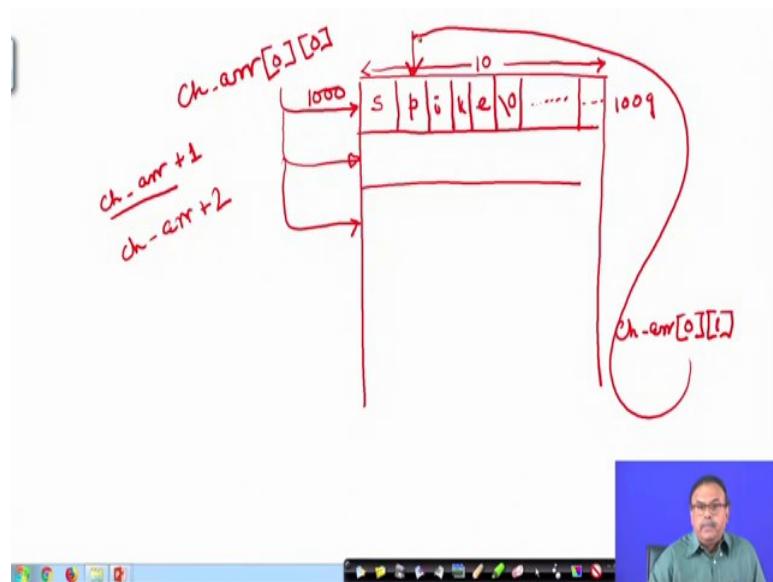
(Refer Slide Time: 23:43)

- We already know that name of an array is a pointer to the 0th element of the array.
- Therefore if ch\_arr points to address 1000 then ch\_arr + 1 will point to address 1010.
- ch\_arr + 0 points to the 0th string or 0th 1-D array.  
ch\_arr + 1 points to the 1st string or 1st 1-D array.  
ch\_arr + 2 points to the 2nd string or 2nd 1-D array.
- In general ch\_arr + i points to the ith string or ith 1-D array.



Therefore, if ch\_arr points to the address 1000 ch\_arr plus 1 will point to address 1010, why the reason is again I have got this array.

(Refer Slide Time: 24:02)



And ch\_arr is pointing ch\_ar arr is pointing to the 0 element of the first row, if this is 1000. Then 1002, 1009 this array this location is 1009, because the size is 10 size is 10. So, although s p i k e backslash 0 etcetera is there, then when I go to the next one ch underscore arr plus 1; that means, whatever this pointer arrays the entire thing is taken 0 to 10. So, I am taking here it is taking all these 10 elements as 1 array it will be pointing to the next one; that means, it will be 1010 ok.

Again if I make it ch-arr plus 2 it will come to this 1 all right that is how the pointer works, but how can I get this character this will be ch-arr 0 1 that is pointing to this element ok. Now this thing must be clearly understood. So, ch-arr points 0 th string or the 0 th 1 dimensional array ch-arr 1 points to the first string or the first 1 dimensional array I mean number 1 0 at first ch-arr 2 points the second 1 dimensional array.

So, each of them are 1 dimensional arrays and ch ar is pointing to each of these 1 dimensional arrays ch-arr is pointing to 1 one dimensional array. And as I increment the pointer this pointer is going on increasing alright in general ch-arr I points to the I th string of the yet 1 dimensional array clear.

(Refer Slide Time: 26:42)

The following program demonstrates how to print an array of strings.

```
#include<stdio.h>
int main()
{
    int i; char ch_arr[3][10] = { "spike", "tom", "jerry" };
    printf("1st way \n\n");
    for(i = 0; i < 3; i++)
        { printf("string = %s \t address = %u\n", ch_arr + i, ch_arr + i); }
    // signal to operating system program ran fine
    return 0; }
```

**Expected Output:**

```
string = spike address = 2686736
string = tom address = 2686746
string = jerry address = 2686756
```

*1st way*  
*string = [ ] .*



The following program let us look at the program and the following program demonstrates how to print an array of strings? Please take 1 minute to read this array a program yourself and try to understand. It you can see here that I am starting with the main function and although it is written in a 1 in a very cryptic way. So, my program starts from this point and ends at this point, i is an integer ch-arr is again an array of 3 strings and each can be of maximum length 10. So, spike tom and jerry are there.

So, I can print I just print first way next line I go to next line and to next line. So, I just put first way and then what do I do. So, I will print first way and then 2 lines have been given blank, then for i 0 2 less than 3 i plus plus printf string, I am printing the string. So, what will be printed string equals a placeholder for the string and then there will be some tab 1 space and then.

(Refer Slide Time: 28:34)

The following program demonstrates how to print an array of strings.

```
#include<stdio.h>
int main()
{ int i; char ch_arr[3][10] = { "spike", "tom", "jerry" };
printf("1st way \n\n");
for(i = 0; i < 3; i++)
{ printf("string = %s \t address = %u\n", ch_arr + i, ch_arr + i); }
// signal to operating system program ran fine
return 0; }
```

**Expected Output:**

```
string = spike address = 2686736
string = tom address = 2686746
string = jerry address = 2686756
```

A video player interface is visible at the bottom of the slide.

So, the first will be string here some space and then tab this is tab.

So, I give some space and write address and then sorry string equal to string equal to address equal to and there should be some space for the address and that format is unsigned integer. So, it will become longer alright.

Because unsigned and signed integer a whenever I am putting suppose I have got 16 bits for representing an integer. If it be assigned if I distinguish between plus 5 and minus 5 and keep 1 bit kept for the plus or minus then for the 16 bit I will be actually having 15 bits for this size, but if I make it unsigned then I can have all these 16 bits. So, I can represent more numbers all right more numbers.

So, that apart so, string and address and then what am I printing here care I care plus i; that means, 0 care 0 the first string and then I am again putting the care this, now what is the difference between this care this and this is being printed in percentage arrays; that means, the string will be printed and here in an unsigned integer what I am printing here is the address of that string.

So, this string spike was here when I print this just the name of the array; that means, I am in this format if I do I am printing the address of it therefore, the output will be somehow like this ok. So, this is something to be understood very clearly that this array

when I am actually looking at this I am taking the content of the array otherwise I am taking the array itself.

(Refer Slide Time: 30:36)

- The following program demonstrates how to print an array of strings.
- #include<stdio.h>

```

int main()
{
    int i; char ch_arr[3][10] = { "spike", "tom", "jerry" };
    printf("1st way \n\n");
    for(i = 0; i < 3; i++)
        { printf("string = %s \t address = %u\n", ch_arr + i, ch_arr + i); }
    // signal to operating system program ran fine
    return 0;
}

Expected Output:
string = spike address = 2686736
string = tom address = 2686746
string = jerry address = 2686756

```

So, here is a program.

(Refer Slide Time: 30:53)

This program asks the user to enter a username. If the username entered is one of the names in the master list then the user is allowed to calculate the factorial of a number. Otherwise, an error message is displayed.

```

#include<stdio.h> ✓
#include<string.h> ✓
int factorial(int );
int main()
{
    int i, found = 0, n;
    char master_list[5][20] = { "admin", "tom", "bob", "tim", "jim" },
    name[10]; printf("Enter username: ");
    gets(name);
    for(i = 0; i < 5; i++)
        { if(strcmp(name, master_list[i]) == 0)
            { found = 1; break; } }

    if( found==1) { printf("\n Welcome %s !\n", name);
                    printf("\nEnter a number to calculate the factorial: ");
                    scanf("%d", &n);
                    printf("Factorial of %d is %d", n, factorial(n));
                }
    else { printf("Error: You are not allowed to run this program.", name); }

    // signal to operating system program ran fine
    return 0;
}

int factorial (int n)
{ if(n == 0) { return 1; }
else { return n * factorial(n-1); }

name [ jim ]
stack [ admin
          tom
          bob
          tim
          jim ]

```

This program asks the user to enter a username if the username entered is one of the names in the master list then the user is allowed to calculate the factorial of a number otherwise an error message is displayed.

So, we have got sorry a master list of names now if you are a valid user your name is there in the list. So, first what is doing it is including stdio dot h it is including string dot h; that means, some string functions are also included I am defining the factorial prototype the factorial function has been written later. Suppose this part this part is later than the main function ok.

Now, the main is here and what is the main doing it has got some integer I and found is 0 initially the name is not found here and some n number for which I have to compute the factorial, master list 5 by 20; that means, each of these each of these can be 20 character long and there are 5 search. And I put the valid names as admin tom, bob, tim, and jim.

Now I am asking a name is so, I am and I am asking from the user enter user name now the user gives the name. So, get s here you are getting a new function being introduced here that is get s what is get s get care get ch, you know and here the get s means get the string. So, in 1 short it is getting the string as the name.

So, name I am getting from the user some name say it is jim suppose the user enters jim. Now for I equals to 0 to 5 why 5 there are all the 5 names are here if string compare name and master list I, if this character this string and this string matches then found is 1, otherwise I am again going in the fall loop.

So, at every stage of the loop I am trying to compare this name with this string here there is no match. So, I come here no match no match in that way I come and find jim. So, here ultimately I will get found and then I will break. Now when I come out suppose it was not jim, it was say jimmy alright then I do not get a match here. So, I can come out with found equal to 0 or found equal to 1, he found equal to 1 then welcome then welcome exclamation and then the name it is printing welcome jim.

And asking please enter the number is entering the number for the factorial and then here you see in this printf I have called the factorial, that factorial of the number is this n is this so, I call factorial and this part you know. Now that this n and this n and may not be the same I mean they are two different locations, but they same it could have been x also in that case internally had it had to be x here.

So, this is being passed over here and the factorial is being computed all right. Once the factorial is computed, now how the factorial is computed I will be discussing that later,

because here there is some trick forget about that trick now I will be discussing that later that is recursion, which you have not yet discussed and after this I am coming back here and I am printing.

So, the purpose of this program was to show how we can handle strings. So, these are very common very common thing I mean you try to login in a machine right. And you type in your login name and your password, if they do not match the system says login failure now how does it do it internally.

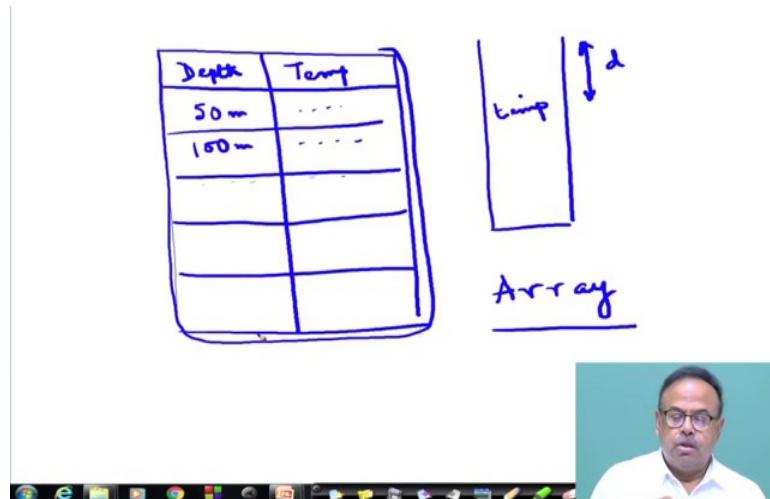
Internally it has got a list of just like this it has got a list of all the users and it takes your username and checks it with all these. That is a problem that is being solved a sub problem that is being solved in the operating system and that is the basic principle is depicted in this program alright we will continue with other problem solving approaches in the future lectures.

**Problem Solving through Programming in C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture-49**  
**Data Representation**

Today, we will be looking at some real engineering applications of programming. We will see how, whatever we have learnt till now can be applied to solve some interesting problems like solving equations. One very common thing that we need in any engineering or science is to represent data. One way of representing data is by in the form of a table right.

(Refer Slide Time: 01:04)

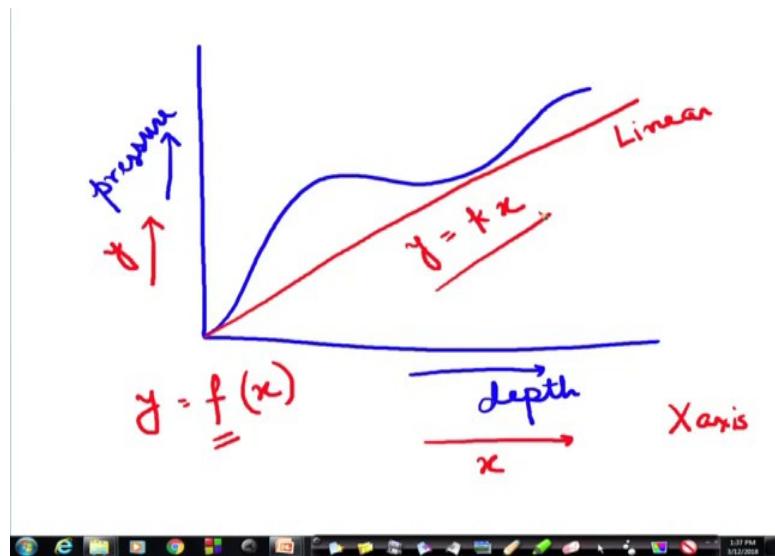


So, I can always represent some data as a table where say on one column it can be height or depth and maybe as we go down, we go through some point and at different depths, we put some sensor and get the temperature.

So, that I can represent in the form of whatever is a depth- 50 meters, some temperature, 100 meters, some temperature etcetera. So, if we need to represent any data in this form we know how to do it? We know that as it looks here -we can immediately say we will be representing it as an array. Whether I will be representing them using as 1 array 1 2 dimensional array or two different array - will depend on the format of the data the type of the data.

For example if the depth is integer and the temperature is real, then up to the knowledge that we have acquired till now will be representing them in the form of 2 arrays, 1 is an integer array another is a separate floating point array . So, that is one way of representing data as we have done also in the case of students roll number and marks earlier. Another very interesting and important thing is representing graphs.

(Refer Slide Time: 03:06)



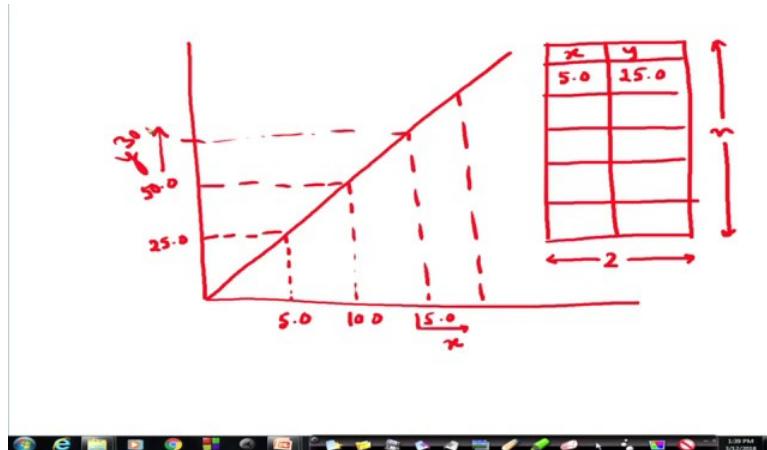
For example, I want to represent a graph say this is some pressure and this is depth and as this graph shows that it is not increasing in a uniform way with depth. There could be some graph, which would be just linear. We call it linear, where along with depth this increases in a linear fashion, here it is non-linear .

So, the question is how do we represent this sort of graph? Now you know that in this graph I can also state that this is the X axis and my independent variable is x and the dependent variable is y, then this graph essentially represents a function, which is y, a function of x. Now depending on the nature of variation of the dependent variable with the independent variable, the nature of the function will vary. This one will be a linear function right.

For example it can be y equals to some k x. This one is more complex I could have had a quadratic function also this is a linear function. Now the question that we would like to first

address is that it is so very nice to draw a picture on a piece of paper; however, a computer will not be able to just interpret this picture as we do then in that case how do we represent this graph in a computer. So, let us do it again.

(Refer Slide Time: 05:40)

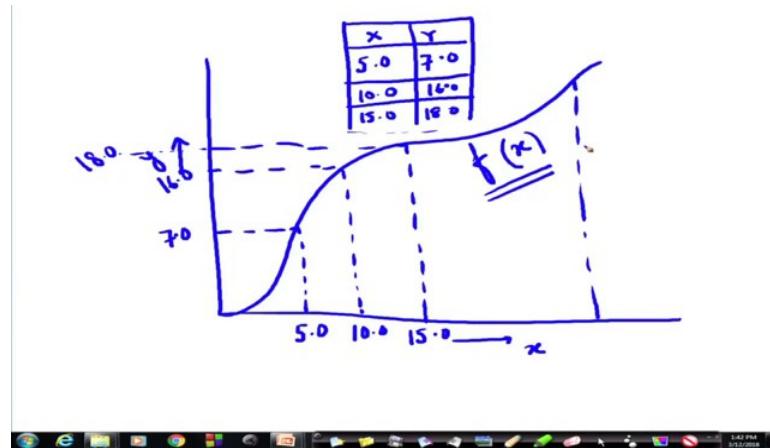


So, I have got a graph let us say a linear graph and I can say that for different values of  $x$  what is the corresponding value of  $y$ ? Now this is a linear one. If  $x$  and  $y$  are the same type I can also represent them in the form of a table; that means, a 2 dimensional array, where one side, one dimension is the variation of  $x$  on one column we have got the  $x$  and the corresponding  $y$  all right.

What will be the number of rows? The number of data points of  $x$  that we take will be the number of rows. How many?-  $n$  and this one is fixed to be 2. I can suppose that this is 5.0 and this is 25. So, I have 5 here I have 25 here - this array is of type float now here it is ten. So, it is 50, it may be 15.

So, this one will be 30, because I am talking of a linear curve it need not be linear all the time.

(Refer Slide Time: 07:40)

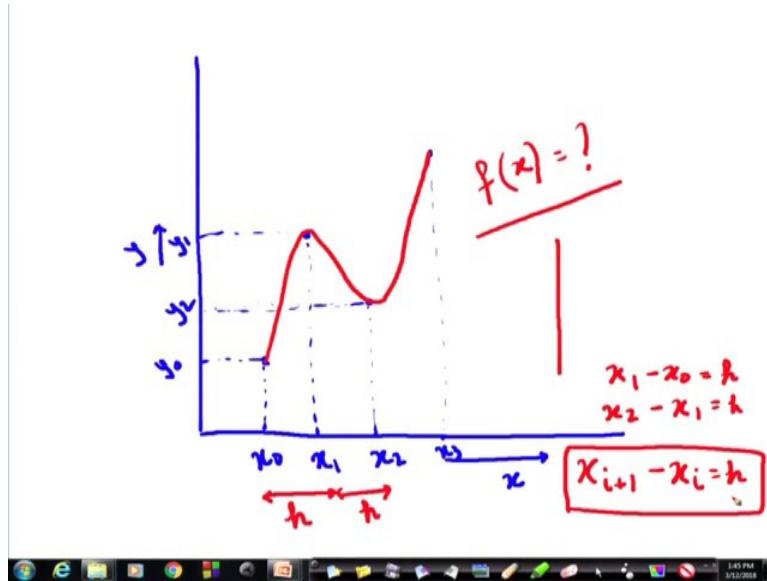


So, I can also have a curve like this, say something like this and here again I have got the y and here I have got the x. So, with a particular value of x say 5, maybe this 1 is initially. Let us say it has risen a little sharp so, maybe it is 7 and if I go here. Now, please note that we usually take the samples at equal distances. So, I take it as 10, but here it will not be doubling. it is becoming a little more than that. So, 7 ,it may be 16. I go here now, you see the slope of this curve has reduced right the gradient has reduced.

So, at 15 it will not increase that much, it will be 18, because it is still flattening out and here there is a sharp change etcetera . So, this one also I can represent in the form of a table like say I have a 2 dimensional table, where I have got 5. This is my x and this is my y. So, 5 corresponding to that 7 first row that is the first point. I could have taken 0 0 also that would be another one.

So, 10 corresponding to that 16 then 15 corresponding to that 18. So, in that way it could go on; that means, whatever is this function  $f(x)$ , a function can be represented as an array , a 2 dimensional array so, that all as a table. So, this we have to remember either as 1 array or as 2 arrays, because if this was integer this was float then I would have needed 2 different arrays. So, here is a function now let us look at another aspect.

(Refer Slide Time: 10:24)



I do not have the function, but I have got some data right. So, here  $x$  and here is  $y$  and I have got some data points for  $x$  equal to this is the  $y$  point. So, if I call it  $x_0$  this is  $y_0$ . Next one could be here -  $x_1$ , it is  $y_1$ . Here maybe it has come down for  $x_2$  it is  $y_2$ . So, I do not have a function right now, but I have got different data points and I have to find out the function. So, that is also another very interesting thing finding out the function that can represent this distribution of data points, might be here again there is  $x_3$  and it is again gone up here.

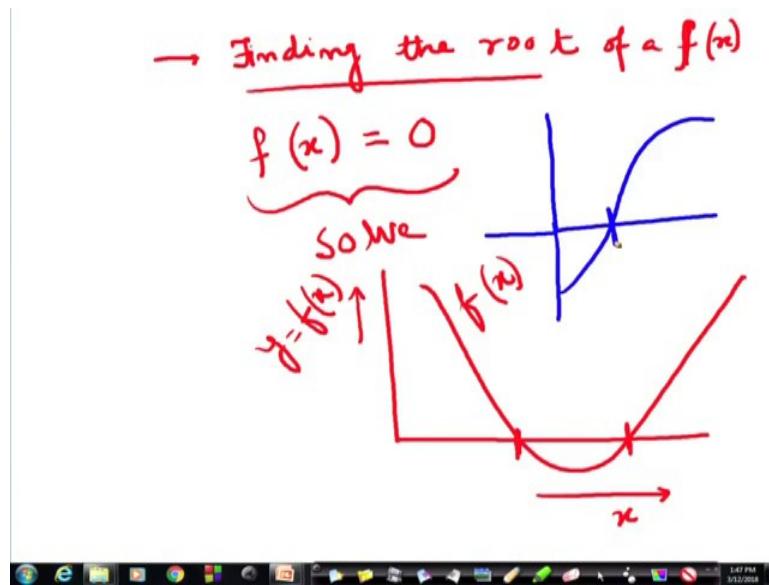
So, it is not a straight line if I want to fit a curve to meet that how will that curve look like? The curve will be something like this right and I have to find out what function is this curve what is the function. I do not know as it that is another challenging point.

So, first of all graphs or data points graphs can be represented as table and the distribution of data points can also be represented as table, because here in this scenario I do not have a function, but certainly I have got the  $x$  and  $y$  values known to me. Therefore, I can represent that as a table. Another point to note here is that usually this independent variable the sampling points that I am taking our usual equidistant.

So, I can say  $x_1$  minus  $x_0$  is some value  $h$   $x_2$  minus  $x_1$  will be the same  $h$ . Usually we represent that in order to see a fixed incremental increase of the independent variable, how much does the dependent variable vary?. So, I can say in general  $x_i + 1$  minus  $x_i$  is equal to  $h$  right.

So, this is typically how we write now. We have got a number of problems to solve using data.

(Refer Slide Time: 13:52)



So, for example, let us start with one problem finding the root of a function some function  $f_x$ . Now finding the root means essentially it is solving the equation. So, if I have got a function  $f_x$ , then my equation is  $f_x$  equals 0 and I want to solve this right and I want to find out that root, at least 1 root.

Now what do I mean by the root? Say there is a function like this. So, for what value of  $x$   $f_x$  equal to 0 is the equation? So, this one is  $f_x$ ,  $y$  or  $f_x$ . So, we are to find out for what value of  $x$ , for which value of  $x$ ,  $f_x$  is 0. So, that is the root. Now this particular function if this be an  $f_x$ , then it has got 2 distinct roots, another function could be something like this could be just like this where I have got one root, this is the point where  $f_x$  equal to 0 all right.

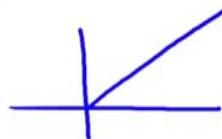
So, one of the major problems is finding out the root of a function or we will say in general the root of a polynomial. Why are we saying a polynomial? Because you know that any function say  $f_x$  is equal to  $3x^2 + 2x + 3$  is a polynomial of degree 2 or this is  $f_1 x$ .

(Refer Slide Time: 16:00)

$$f_1(x) = 3x^2 + 2x + 3$$

$$f_2(x) = \underline{4x + 3}$$

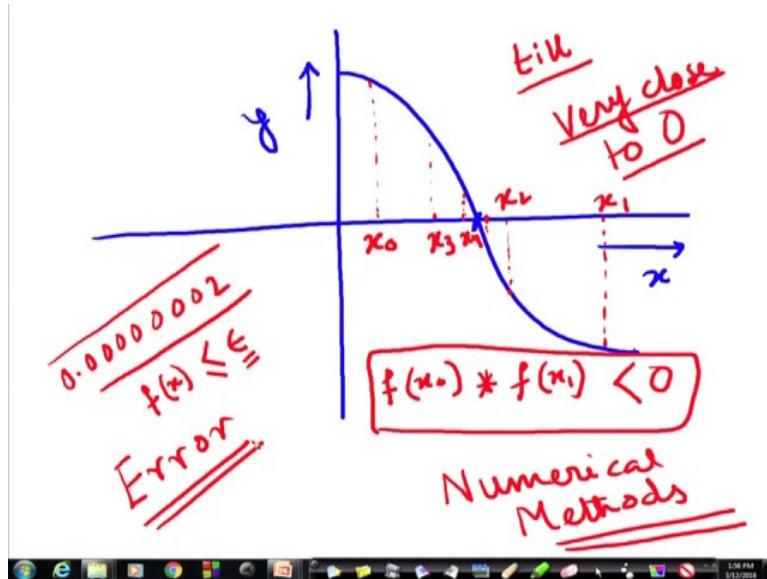
$$y = m \cdot x + c$$



I could have another  $f_2(x)$ , as  $4x$  this is a polynomial of the  $4x$  plus 3 maybe. Now anybody who remembers cool coordinate geometry this is a linear equation, because it is a polynomial of degree 1; that means, if we say this to be  $y$  then this is  $y$  equals  $m \cdot x + c$ . So,  $m$  this 4 is nothing, but the slope of this straight line.

So, there will be some slope of this straight line, this is linear. This one is not linear this quadratic. So, I can have the different functions which can be written as a polynomial and when the polynomial is equated to 0 it becomes an equation and we want to solve that equation. Problem number 2 is interpolating a function. I will describe what interpolation is a little later. But, before that let us try to see a very simple way of finding the root of a function; say I have got a function  $x$  and  $y$  alright and a function is something like this.

(Refer Slide Time: 17:48)



Now, I want to find out this point. What is the value of  $x$ ? This value, for which this  $y$  is 0. Now the method that we will be talking about is known as the bisection method. This is a very common and interesting approach, you have seen in binary search also we have partitioned the array into 2 halves and then went to one half. Very similar to that let us see how the bisection method tries to find out the root. First we start finding 2 arbitrary points say  $x_0$  and  $x_1$ , such that  $f(x_0)$  and  $f(x_1)$  are of different signs. So, here you can see this is plus, this is minus.  $f(x_1)$  is the  $y$  value corresponding to  $x_1$ . So, if the  $y$  value of  $x_0$  and  $x_1$  are opposite then it is immediately understood that the root must lie somewhere in between .

So, we will then try to find out the midpoint of  $x_0$  and  $x_1$ . Suppose the midpoint of  $x_0$  and  $x_1$  is  $x_2$ . So, if  $x_2$ ,  $f(x_2)$  and  $f(x_0)$  are out of opposite sign, then the root must be I have (I have reduced my space right) so, it is between  $x_2$  and  $x_0$ . So, I again find out the value the midpoint of  $x_2$  and  $x_0$ . Suppose that midpoint is here, suppose this is  $x_3$  and these 2  $f(x_3)$  and  $f(x_2)$  are of opposite signs. Therefore, I will find the mid midpoint of these 2. So, suppose the midpoint of these two is this  $x_4$  here.

Now still this one and this one are of opposite sides therefore, I will try to find out the midpoint of these two. So, I will come somewhere here. In that way we approach till what till we find that the value of  $y$  is very close to 0, very close to 0. Why am I saying very close to 0 not exactly 0 there are reasons for that, reasons we are coming to that. A computer works with finite representation of numbers.

So, we may not get exact 0, but suppose I get this and then I can assume that to be 0 because it is very small and often we call it that that the value of  $f_x$  at that point is less than equal to some very small value epsilon that will decide a prior.

So; however so, you have seen this approach. Now one thing that you can quickly think of that how would I know that these 2 are of opposite signs  $f_x 1$  and  $f_x 0$  are of opposite signs? If I take the product of this if I take the product of this then; obviously, if they are opposite signs the product will be negative that is less than 0.

So, at every point we check whether they are less than 0. If there is no root, suppose it goes like this then I will not find any point where they are of opposite signs ok. So, that is the basic idea of bisection method. Now, when I carry it out through a computer such attempts to solve such problems known as programming numerical methods.

We will start with some representative, relatively easy one examples of those. Some numerical methods, one of them is finding the root of a function, root of a polynomial. Now, while doing these numerical methods we always encounter errors and our algorithm will be better, if the error is less. Now what do I mean by error? Say, I am computing  $4$  by  $3$ . Now what is the correct result, the most accurate? Do you know that no it is  $1.3\ 3\ 3\ 3\ 3$  it goes on.

(Refer Slide Time: 24:12)

$$\frac{4}{3} = 1.333333 \dots$$



Now, you know our computers have got some storage locations of some 8 by 8 bits or 16 bits like that.

So, depending on that I have got a finite capacity to store the data. Therefore, I may represent this as say 1.3 3 3 3, I just start with say 6 digits all right, 6 7 8; I can store 8 digits maximum. 6 after decimal 1 decimal and this so, 6 places.

(Refer Slide Time: 12:51)

$$\begin{aligned} & \text{Exact value} \\ & 1.333333 \\ & \text{Error} = \frac{\text{Exact value}}{\sim \text{Computed value}} \\ & 1.523 \quad 1.522823 \\ & \text{Error} = \underline{0.000177} \\ & \text{Absolute error} \\ & = | \text{Error} | \\ & \text{Relative error} = \frac{\text{Abs. Error}}{|\text{Exact value}|} \end{aligned}$$



Therefore, the actual thing was much more. So, I am actually committing some error. So, the error in computation is the exact value, the difference of the exact value, by this m and the computed value, it can be positive or negative all right. Suppose the result is actually 1.5 2 3 that is the exact value suppose and during my computation I got 1.5 2 2 8 2 3 then the error is the difference between these 2 and the error is therefore, if I subtract this it will be 0.0 0.0 0 0 1 7 7 - that is the error. Now the smaller this error is more accurate my result is. I think it is very clear. Now there are 2 types of errors also, one is a round off error for example, here I could have rounded it off as 1.5 2 3.

So, suppose the actual thing was this and I rounded it to 1.5 2 by approximating if I go by 2 bits. That is one type of error, other type of error is truncation error -this was there I have just dropped this 1.5 2, because I could not store more.

So, there are two types of errors right. So, truncation error and round off error. Another term that you need to know is the absolute error -absolute error is nothing, but the absolute value of the error and relative error is absolute error divided by exact value, the absolute of the exact value. Absolute error is the error's absolute part by the exact value and percentage error will be relative error times 1000 relative error times 1000.

(Refer Slide Time: 28:21)

$$\% \text{ error} = \text{Rel. error} * 100$$



So, we can say that percentage error is relative error in to 100 .

So, now just look at this have a quick look at these definitions. Absolute error means the absolute value of the error, relative error is absolute error divided by the exact value and then we come to the percentage error, which is relative error times 100.

Now, another important thing that we have to consider is accumulated error, but that will consider later. In the next lecture we will move. So, whatever we do the way we do the

computation we must be very careful about the algorithm be such that it is it does not accommodate too much error.

However, discussing about error analysis in general is beyond the scope of this course right now. So, we will in the next lecture start with the bisection method as I have explained and then move to some other methods.

**Problem Solving through Programming in C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture-50**  
**Bisection Method**

We were talking about errors and round off errors and percentage errors. Now, I will briefly show you that if we commit an error how that error continuously gets accumulated and ultimately has a much larger effect.

(Refer Slide Time: 00:46)

<u>Exact value</u>	<u>Compute</u>	<u>Error</u>
$x_0 = 9.98$	10	0.02 ←
$x_1 = 10 \cdot x_0 = 99.8$	100	0.2
$x_2 = 10 \cdot x_1 = 998$	1000	2.0

*Relative acc of error*  
 $= \frac{\text{Accumulated error}}{\text{Exact value}}$   
 $= \frac{0.02}{9.98} \quad \boxed{0.002004}$   
 $= \frac{0.2}{99.8}$



For example, suppose the exact value is 9.98 and we compute 10 all right. So, the inherent error, first the error actually is 0.02 right.

Now, as we go on iterating suppose  $x_1$  is 10 times  $x_0$ , then the exact value should be 99.8, but here we will get 100. So, you can see that the error has increased to how much 0.2. Now if the next iteration  $x_2$  is again 10 times  $x_1$  then it will be 998 whereas, the computed value will be 1000. So, the error is becoming 2.

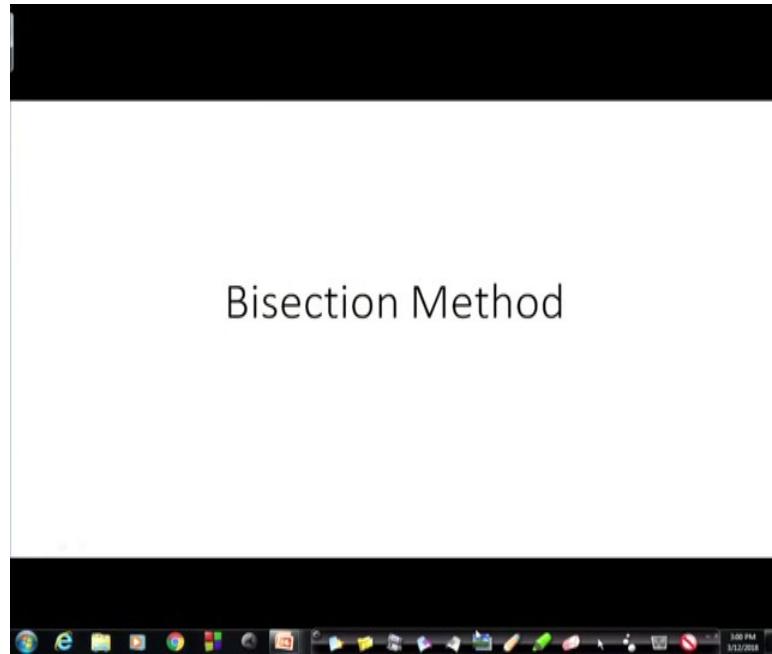
So, you see how with if we start with an inherent error how that error accumulates over time? Ok. So, we can say a very important term is relative accumulation of error, which is accumulated error divided by exact value for that iteration.

For example in the first iteration the accumulated error was 0.0 2 divided by the exact value which was 9.9 8, but later on it became 0.2 divided by 99.8. So, this is this was 0.002004, but whatever that is it is not that important I do not want to confuse with you with this.

Now there are some cases where this accumulation of error actually goes on increasing. If the rate of accumulation error decreases or if the rate of accumulated error increases, but the rate of relative error decreases, then we call it a stable algorithm.

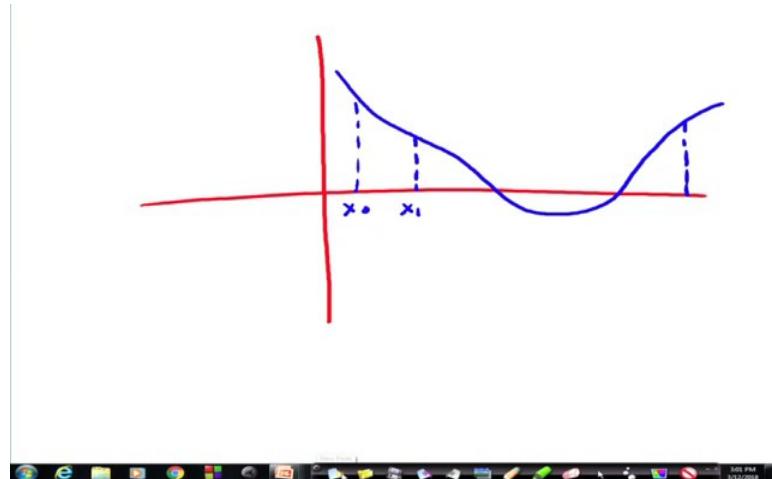
However, I am not going into the details and the intricacies of this, but this is just to give you an idea how the error propagates through iterations. So, we must be very conscious about the rate of increase of this error .

(Refer Slide Time: 04:50)



With these words we move to the algorithm which we are planning to discuss in this lecture that is a bisection method.

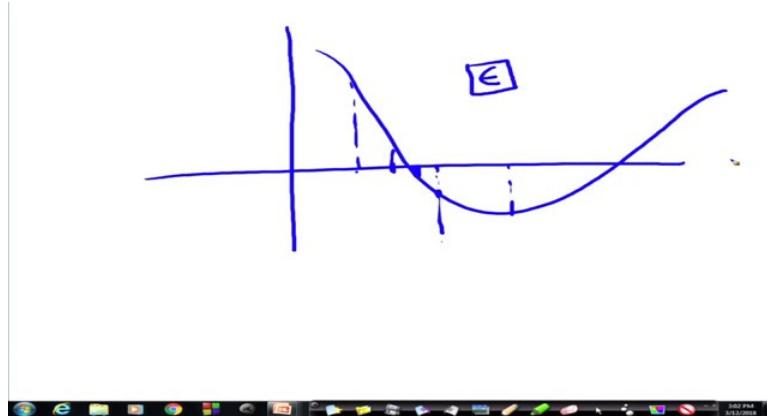
(Refer Slide Time: 05:10)



I have already told you that the bisection method is given a particular function, some function, on this x and y axis. If I have some function that moves in this way then we start with any two points any two points, arbitrary points here and maybe here, now these two points will not do.

Because if I select these two points then both of them are positive so, that if I had selected this point for example, these two points  $x_0$  and  $x_1$  that would not have served my purpose because I do not know where the root is there or not.

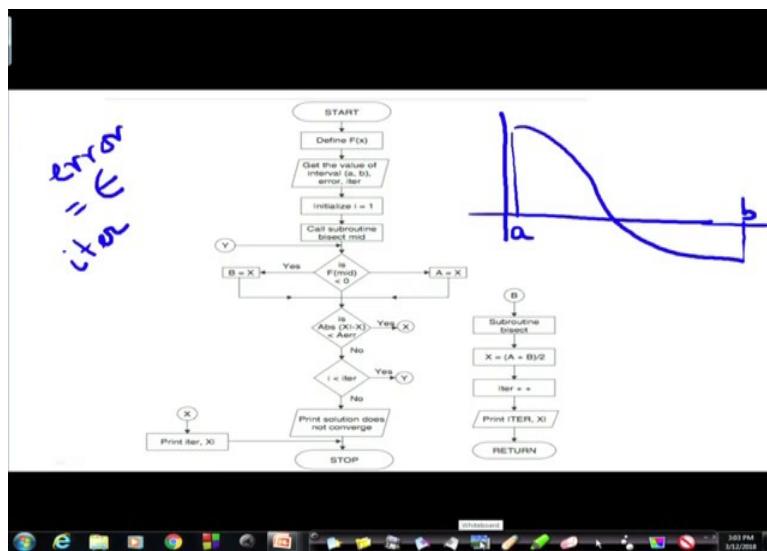
(Refer Slide Time: 05:52)



So, I would rather select two points, which are of opposite signs and therefore, I know that somewhere in between the root lies. So, I will take the midpoint of this, somehow here, if this point is negative then I will keep the positive fixed and I will find out the value of  $y$ . So, these two are opposite signs. So, the root must be somewhere here.

So, in that way I come to this one and find out the value of the root here. In that way I go on dividing it till I come very close to the root as it is being shown here. It is very close to the root. Now how close, that will depend on my decision that is the basic approach of bisection method.

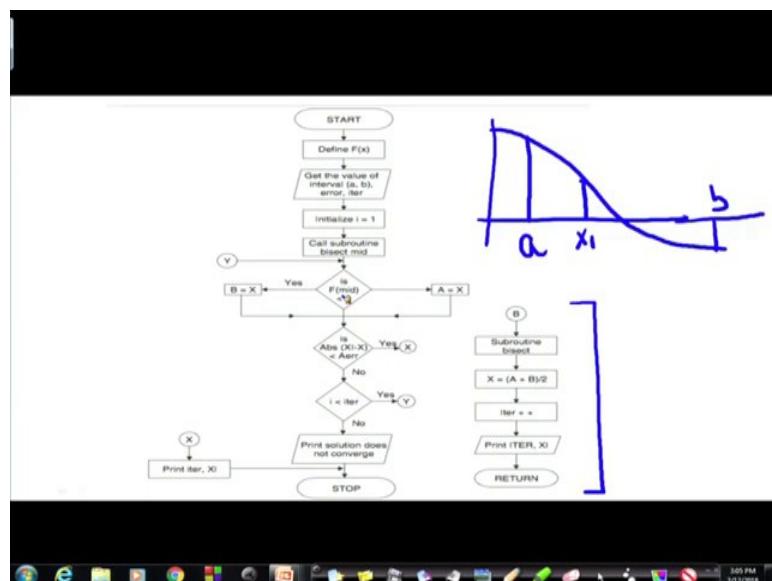
(Refer Slide Time: 06:51)



So, with that let us try to have a look at the algorithm, therefore, first we start, we define the function  $f(x)$  and get the value of the interval  $A$   $B$  that is there is a function. So, here there is a function and the function can be long enough. So, I take the limits, I have to find out the root within this interval  $a$  and  $b$  and I find out how much error is required. So, how much error is acceptable.

So, this allowed error is the epsilon that I was talking of and also the number of iterations, because it may be that in some case I am not finding the root, because I am going on looking at say for example, this sort of scenario and my  $a$  is here my  $b$  is here.

(Refer Slide Time: 08:00)

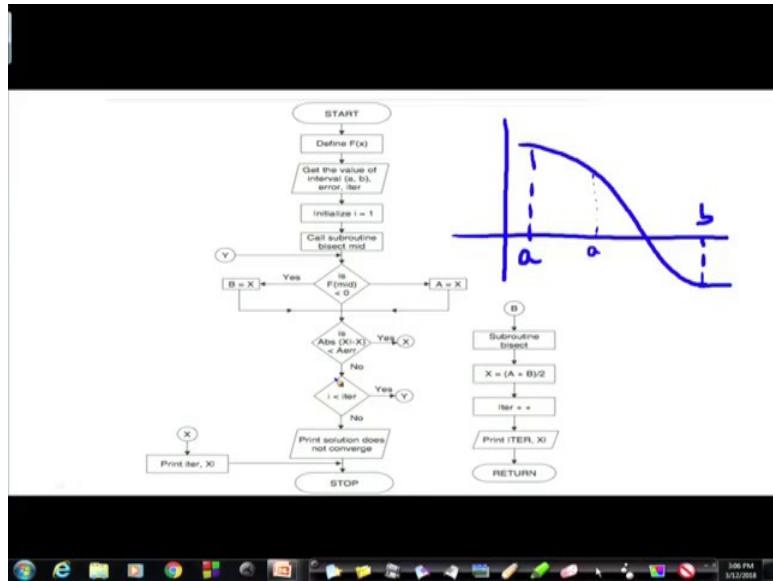


Then; obviously, the root does not lie between this. So, I will go on doing this bisection and again i am doing this bisection, for how long will I go on? But still it may be that I might not find the root therefore, there is a maximum limit that is kept -number of iterations.

Now I initialise I to be 1, just some index to be 1 and then I call a subroutine or a function, bisect the midpoint. So, here is a sub subroutine, you see what it does? Or a function, What does it do here? It finds out between A and B the midpoint and increments the iteration and prints the value of x 1. x 1 is a middle point.

So, if my function was like this and this was a this was b, then the root mass lies somewhere here. I find out the midpoint of this. So, this becomes x 1. There is a next one and what is the iteration? Next, is F; that means F mid less than 0?

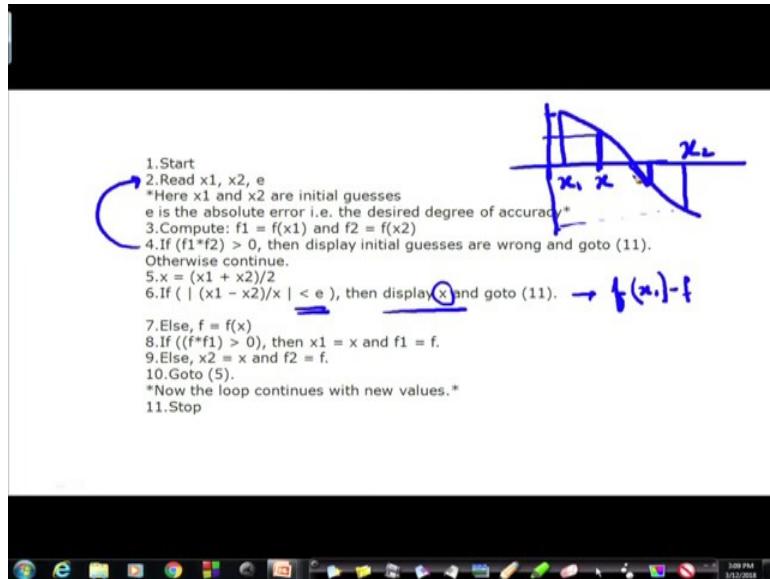
(Refer Slide Time: 09:46)



So, there can be 2 things. Here I select this to be my b and this to be my a. Now; obviously, if I take the midpoint then the midpoint will be somewhere here, a and b are of different signs. If  $F_{mid}$  is not less than 0, then a should be x; that means, I will now move it on this side and this should be the next a and between these two I will have to find out and every time I am trying to find out -whatever value of  $F_X$  that I compute, is it less than the absolute error ? If yes then - x , I am getting my solution otherwise I am going on doing this.

So, this is a flow chart of the whole thing, but I think you will be more interested in looking at the algorithm. Let us look at the algorithm for a second and the program. Here is algorithm,you see this is much more understandable to you.

(Refer Slide Time: 11:07)



I read  $x_1$ ,  $x_2$  and the error. Here  $x_1$  and  $x_2$  are the initial guesses all right. Here is my thing, I have taken this is  $x_2$ , this is  $x_1$ . ‘e’ is absolute error; that means, how much error is permissible, compute  $F_1$  that is  $FX_1$ , compute for this function this value and  $f_2$ , compute this value . If  $f_1$  and  $f_2$ , the product of these two is greater than 0 ; that means, my initial guesses are wrong, because both of them are positive, then I can do many things I will instead of going to 11( my initial guesses are wrong) I will again ask for new guess all right.

So, I take a new guess and I find that it is less than 0. Then I take in this step  $x_1$  plus  $x_2$ , mid of that. So, suppose mid of that is this one and that is becoming  $x$ , read this as  $fx_1$  minus  $fx_2$  divided by  $x$  is less than  $e$  then display  $x$ ; that means, if my error between these 2 points, the difference is less is 0.0002 and that I can assume as 0 then I will display this particular value of  $x$ .

So, right now it is not the case otherwise I will make this  $f$  to be  $fx$  I take this all right. Now between these two I again divide, I come here and in this way I go on alright. So, you will be able to write the program as the program runs. So, just to show you (I am sure you can write the program yourself )how can you translate this in the form of a code.

(Refer Slide Time: 14:16)

```
C Program for Bisection Method Source Code

#include<stdio.h>
#include<math.h>
float fun (float x)
{
    return (x*x*x - 4*x - 9);
}
void bisection (float *float a, float b, int *itr)
/* this function performs and prints the result
of one iteration */
{
    *x=(a+b)/2;
    ++(*itr);
    printf("Iteration no. %d X = %f\n", *itr,
    *x);
}

void main ()
{
    int itr = 0, maxmitr;
    float x, a, b, allerr, x1;
    printf("\nEnter the values of a, b, allowed error and maximum
iterations:\n");
    scanf("%f %f %f %d", &a, &b, &allerr, &maxmitr);
    bisection (&x, a, b, &itr);
    do
    {
        if((fun(a)*fun(x)) < 0)
            b=x;
        else
            a=x;
        bisection (&x1, a, b, &itr);
        if (fabs(x1-x) < allerr)
        {
            printf("After %d iterations, root = %f\n", itr, x1);
            return 0;
        }
        x=x1;
    }
    while (itr < maxmitr);
    printf("The solution does not converge or iterations not sufficient");
    return 1;
}
```

So, here you see let us try to understand this code it is a C program for the bisection method, I have included a st stdio dot h, math dot h and there is some function, because I have to find the root of a particular polynomial.

So, for example, here it is given, this is a polynomial. So, what is this polynomial it is  $x^3 - 4x - 9$ . So, that value of  $fx$  has to be computed. So, the function is the `fun`; `fun` is the name of the function, then there is another function `bisection`, this function performs and prints the result of one iteration.

So, it is  $a + b / 2$ ; now in an earlier lecture we had talked about this -what is this ?Because here when I am calling this function, I am calling by reference how? I am just passing the address `x` and whatever I do here once again is a part of revision you can see. If I come here - `float star x`; that means what? `x` is some variable and I have just passed the address of that.

(Refer Slide Time: 15:44)

```
C Program for Bisection Method Source Code

#include<stdio.h>
#include<math.h>
float fun (float x)
{
    return (x*x*x - 4*x - 9);
}
void bisection (float *x, float a, float b, int *itr)
/* this function performs and prints the result
of one iteration */
{
    *x=(a+b)/2;
    ++(*itr);
    printf("Iteration no. %3d X = %7.5f\n", *itr,
    *x);
}

void main ()
{
    int itr = 0, maxmitr;
    float x, a, b, allerr, x1;
    printf("\nEnter the values of a, b, allowed error and maximum
iterations:\n");
    scanf("%f %f %f %d", &a, &b, &allerr, &maxmitr);
    bisection (&x, a, b, &itr);
    do
    {
        if (fun(a)*fun(x) < 0)
            b=x;
        else
            a=x;
        bisection (&x1, a, b, &itr);
        if (fabs(x1-x) < allerr)
        {
            printf("After %d iterations, root = %6.4f\n", itr, x1);
            return 0;
        }
        x=x1;
    }
    while (itr < maxmitr);
    printf("The solution does not converge or iterations not sufficient");
    return 1;
}
```

So, star x is the content of this; say 50. So, I passed it on and in the main function I am passing on the address of x . This we have seen earlier. So, I am taking the midpoint and incrementing the iteration; iteration is also a call by reference and float a float b are two points in between which have been passed on.

Now, what is being done in the main function? In the main function I am setting the iteration to be 0 and here I am saying how many iterations are permitted?

(Refer Slide Time: 16:34)

```

C Program for Bisection Method Source Code

#include<stdio.h>
#include<math.h>
floatfun (float x)
{
    return (x*x*x - 4*x - 9);
}
void bisection (float *x, float a, float b, int *itr)
/* this function performs and prints the result
of one iteration */
{
    *x=(a+b)/2;
    ++(*itr);
    printf("Iteration no. %3d X = %7.5f\n", *itr,
*x);
}
void main ()
{
    int itr = 0, maxmitr;
    float x, a, b, allerr, x1;
    printf("\nEnter the values of a, b, allowed error and maximum
iterations:\n");
    scanf("%f %f %f %d", &a, &b, &allerr, &maxmitr);
    do
    {
        if (fun(a) * fun(x1) < 0)
            b=x1;
        else
            a=x1;
        bisection (&x, a, b, &itr);
        if (fabs(x1-x) < allerr)
        {
            printf("After %d iterations, root = %6.4f\n", itr, x1);
            return 0;
        }
        x=x1;
    }
    while (itr < maxmitr);
    printf("The solution does not converge or iterations not sufficient");
    return 1;
}

```

Maximum number of iterations;  $x$ ,  $a$ ,  $b$ , allowed error and some  $x_1$  value is given. Enter the values of  $a$ ,  $b$ , allowed error and maximum iterations. So, all these I read - the range  $a$  and  $b$  between, which points I have to do,  $a$  and  $b$  and how much is the allowed error and what is the maximum number of iterations? Then with this I call bisection. What do I do? Bisection and  $x$ ; that means, this will give me the midpoint. I will call bisection here -  $a$ ,  $b$  iterations.

So,  $a$  is being passed here,  $b$  is being passed here, and the number of iterations is being passed here. Now here I am finding the midpoint and that midpoint is being returned here is common and then here at this point I call the function; that is I am computing the polynomial.

(Refer Slide Time: 17:54)

```

C Program for Bisection Method Source Code

#include<stdio.h>
#include<math.h>
float fun (float x)
{
    return (x*x*x - 4*x - 9);
}
void bisection (float *x, float a, float b, int *itr)
/* this function performs and prints the result
of one iteration */
{
    *x=(a+b)/2;
    ++(*itr);
    printf("Iteration no. %d X = %7.5f\n", *itr,
*x);
}
void main ()
{
    int itr = 0, maxmitr;
    float x, a, b, allerr, x1;
    printf("\nEnter the values of a, b, allowed error and maximum
iterations:\n");
    scanf("%f %f %f %d", &a, &b, &allerr, &maxmitr);
    bisection (&x, a, b, &itr);
    do
    {
        if((fun(a)*fun(x)<0))
            b=x;
        else
            a=x;
        bisection (&x1, a, b, &itr);
        if(fabs(x1-x) < allerr)
        {
            printf("After %d iterations, root = %6.4f\n", itr, x1);
            return 0;
        }
        x=x1;
    }
    while (itr < maxmitr);
    printf("The solution does not converge or iterations not sufficient");
    return 1;
}

```

If the polynomial ie the value of the polynomial at a, and the value of the polynomial at x . there is a midpoint that is less than 0. So, what happened? This was my scenario, here was a and here was b, now let me draw it.

(Refer Slide Time: 18:30)

```

C Program for Bisection Method Source Code

#include<stdio.h>
#include<math.h>
float fun (float x)
{
    return (x*x*x - 4*x - 9);
}
void bisection (float *x, float a, float b, int *itr)
/* this function performs and prints the result
of one iteration */
{
    *x=(a+b)/2;
    ++(*itr);
    printf("Iteration no. %d X = %.5f\n", *itr,
    *x);
}
void main ()
{
    int itr = 0, maxitr;
    float x, a, b, allerr, x1;
    printf("\nEnter the values of a, b, allowed error and maximum
iterations:\n");
    scanf("%f %f %f %d", &a, &b, &allerr, &maxitr);
    bisection (&x, a, b, &itr);
    do
    {
        if (fun(a)*fun(x) < 0)
            b=x;
        else
            a=x;
        bisection (&x1, a, b, &itr);
        if (fabs(x1-x) < allerr)
        {
            printf("After %d iterations, root = %.6f\n", itr, x1);
            return 0;
        }
        x=x1;
    }
    while (itr < maxitr);
    printf("The solution does not converge or iterations not sufficient");
    return 1;
}

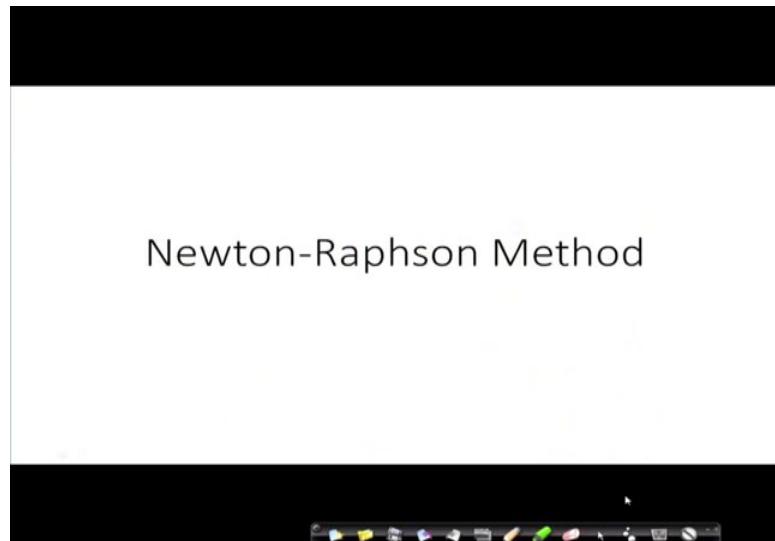
```

So, here was b and here was a. So, now, what I do? I got the meet point somewhere here, then the value of the function at this point and the value of the function at this point is a negative. Therefore, I move this b to x. x is becoming b and I do the same thing.

Otherwise if it was on the other side I would have made x to be a. Is this clear? Then again I call bisection. After calling bisection I find if the absolute error of x<sub>1</sub> minus x is less than the allowed error, then I will print the root.

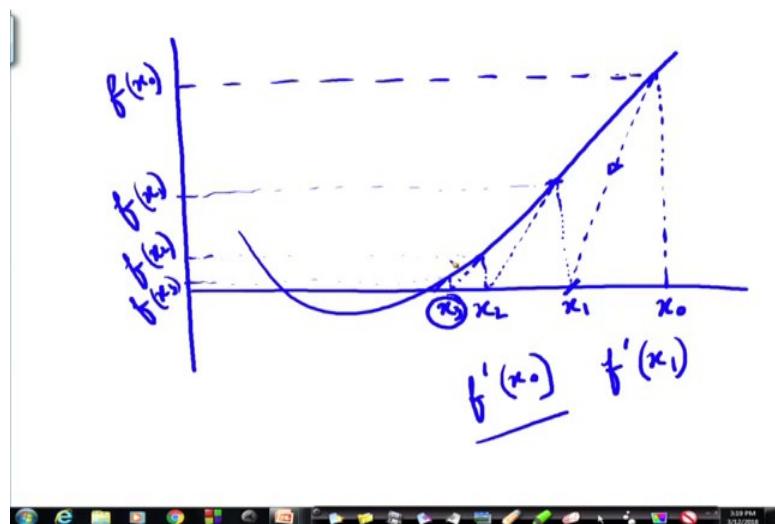
So, this will go on until I exceed the maximum iteration. So, this is how I code and all of you should be able to practice this yourself. Next we will move to another algorithm, which is that another method, which is known as the Newton-Raphson method.

(Refer Slide Time: 20:18)



This method adopts a different approach to find the root of the function let us try to understand this briefly.

(Refer Slide Time: 20:40)



So, I have got a function like this; now Newton-Raphson method, what it does is, it starts at some  $x_0$  and the corresponding value of the function at  $x_0$  is  $f(x_0)$ . Now what it does it finds out the tangent at this point. So, what would that tangent be? That tangent is nothing, but  $f'(x_0)$ , because we know prime means  $\frac{dy}{dx}$ .

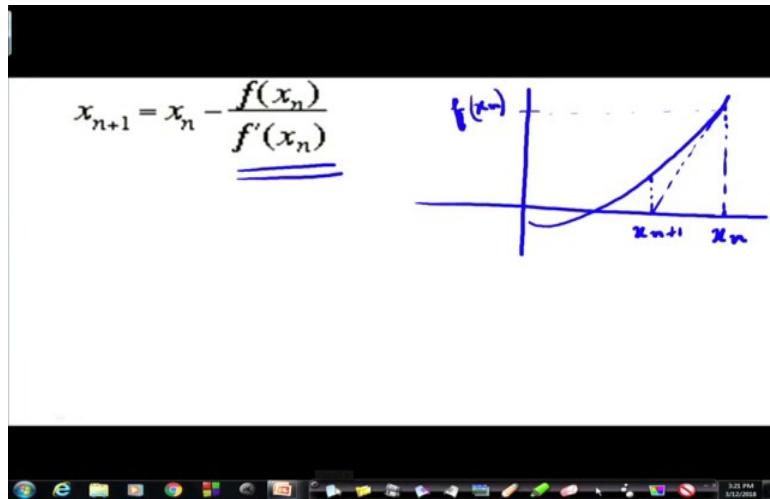
So, I draw the tangent here, the tangent intersects the x axis at some point. Let this be the value  $x_1$ , I drew the tangent and got the value  $x_1$ . I come here and from here and from here for  $f(x_1)$  at every stage have to check whether the value is close to 0 or not; obviously, this is not the case.

Therefore, I draw another tangent from here and what is this tangent? This tangent is  $f'(x_1)$ ; derivative at this point. So, this becomes  $x_2$ , now I find out  $f(x_2)$ . Now I again compare whether  $f(x_2)$  is very close to 0 or not. Still it is not the case. So, I draw a tangent at this point.

So, this is  $x_3$  and I find out if  $x_3$ . Suppose this value  $f(x_3)$  is very close to 0. Suppose this is within my allowed error, then  $x_3$  is the root. Otherwise if it was not there from here again I would have to draw a tangent in this way it goes all right.

So, this is the essence of Newton-Raphson's method. So, what are you doing here? We are taking a function starting with a point and finding a tangent to that curve to the function at that point and see where that tangent intersects the x axis, from there I find out  $f(x_2)$  and then I go on doing this. So, I think this geometrical exposition will be very helpful to you. So, next let us try to see how Newton-Raphson method works.

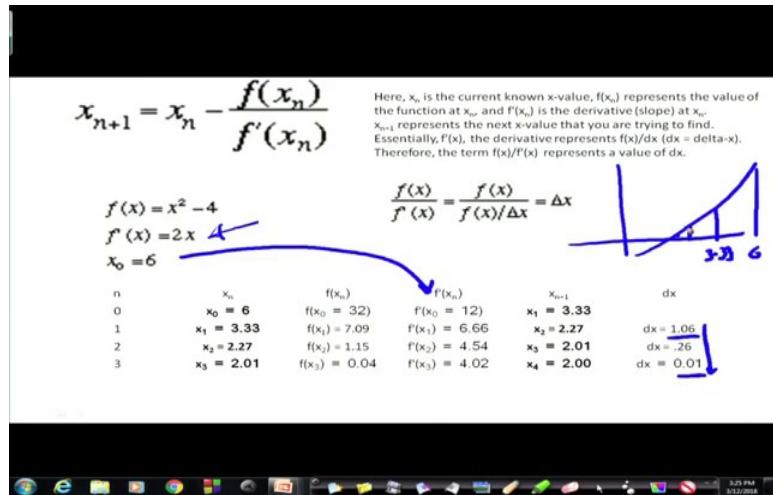
(Refer Slide Time: 24:46)



So, at every stage  $x_n + 1$  is  $x_n$  minus  $f(x_n)$  by  $f'(x_n)$ , why? Because of the simple reason that I had this curve and this was my  $x_n$ , from there I drew the tangent. So, this was  $f(x_n)$  and I drew the tangent here, then if I divide this and subtract from here I will get this  $x_n + 1$  here.

So, you see it is coming in the other way because at every stage I am computing this. Next, here  $x_n$  is the current known value of  $x$ ,  $f(x_n)$  represents the value of the function,  $f'(x_n)$  is the derivative of the slope at that point,  $x_n + 1$  represents the next  $x$  value that you are trying to find.

(Refer Slide Time: 25:53)



So, this expression is coming from the fact that  $f(x) \approx f(x_0) + f'(x_0)(x - x_0)$ , where  $d x$  is delta minus  $x$  therefore, the term  $f(x) - f(x_0)$  takes is actually the value of  $d x$  that means, by how much I should count down.

So, you can see from this expression  $f(x) \approx f(x_0) + f'(x_0)(x - x_0)$ ; that means, the delta  $x$ ; actually this is the delta  $x$  part and that means, how I am shifting this  $x$ . So, the  $x$  was here and I am shifting it by delta  $x$  and coming here, again shifting it by delta  $x$  and coming here like that I am going all right.

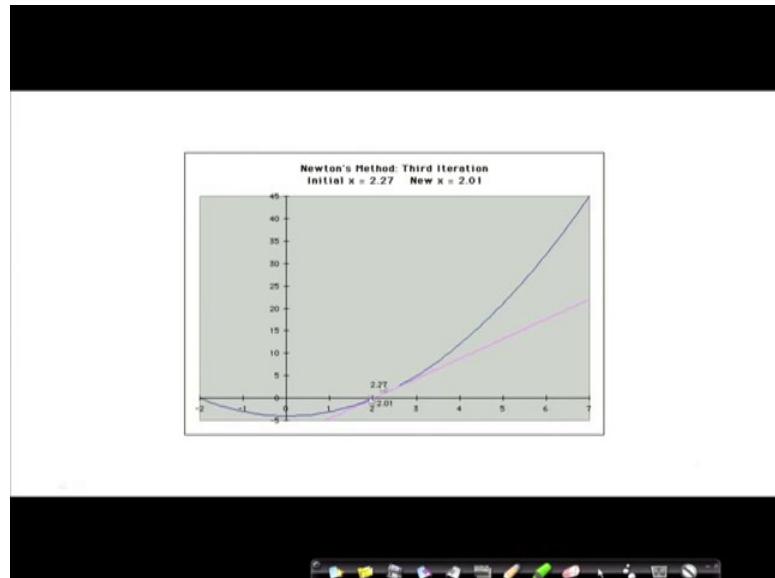
So, suppose  $f(x) = x^2 - 4$ , then  $f'(x) = 2x$  and  $x_0 = 6$ , suppose I assumed  $x_0$  to be 6. So, here is how it goes. First iteration  $x_0$  is 6,  $f(x_0)$  is supposed 32, 32 because 6 squared 36 minus 4.  $f'(x_0)$  is what?  $f'(x_0) = 2x_0$  that is 12. This is  $x_n$  and the next value will be  $x_{n+1}$ ; that means, 6 minus 32 by 12. So, whatever that is I subtract and I get the 3.33 next iteration I come to 3.33.

So, it was something like this that I started with 6 and then I moved to 3.33. So, you see it is converging very fast 3.33, then at 3.33 the value of  $f(x_1)$  is 7.09, here if you compute this, the derivative will be twice of this, that is 6.66, derivative is  $2x$ , 6.66. If I subtract this 6.66 divided by 7.09, subtract it from 3.33 is 2.27. So, my  $dx$  is 1.06. I go on like this and ultimately I come to a  $dx$  of 0.01, I assume.

So, next time it is 2.27. I start with that again find the next value to be 2.27 so it will be 2.01 and with 2.01 I compute and gradually you see the  $dx$  is coming down. As the  $dx$  is coming

down; that means, I am approaching the actual root. So, this is Newton-Raphson method and we can very easily code it.

(Refer Slide Time: 29:48)



So, here is an example. Now you can see this it starts with 6 goes to 3.3 3, then from 3.3 3; I am coming to 2.2.7 then from 2.2.7, I am coming here and gradually the error is not increasing. So, it is very much converging. So, I get the solution with 2.0 1.

(Refer Slide Time: 30:18)

**Newton Raphson Method Algorithm:**

```

1.Start
2.Read x, e, n, d
*x is the initial guess
e is the absolute error i.e the desired degree of accuracy
n is for operating loop
d is for checking slope*
3.Do for i =1 to n in step of 2
4.f = f(x)
5.f1 = f'(x)
6.If (|f1| < d), then display too small slope and goto 11.
7.x1 = x - f/f1
8.If ([(x1 - x)/x1] < e), the display the root as x1 and goto 11.
9.x = x1 and end loop
10.Display method does not converge due to oscillation.
11.Stop

```

So, quickly the algorithm will look like this - again I will read x, the a maximum error allowed number of iterations and d. d is for checking the slope, here are the comments - x is the initial guess, absolutely error is e, n is the number of iterations.

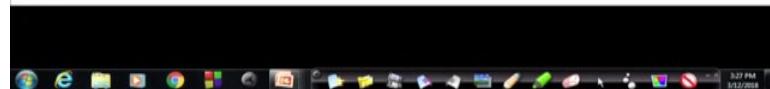
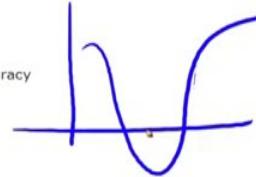
So, do in a loop, i to n in steps of 2,  $f(x)$  is equal to  $f(x)$  and then  $f_1$  is  $f'(x)$ . Now these are 2 functions, which you have to write and now why am I keeping this check? Why am I keeping this check?  $f_1$  which is a slope if  $f'$  dashed  $x$  is too small; that means, what that the slope is nearly horizontal; that means, I am not going to get something like this. If I come to this point and try to find a slope of this, the slope will be very horizontal.

So, this is not a good choice. In that case I have to reduce it and come to a point, where I can find a slope all right. Now, in that way I go on and find the whether it is coming to the close to the root and go on. Now, if it goes on the iteration goes on say for example, it is possible that I am missing the loop, coming close to that- say a curve like this.

(Refer Slide Time: 32:05)

#### Newton Raphson Method Algorithm:

```
1.Start  
2.Read x, e, n, d  
*x is the initial guess  
e is the absolute error i.e the desired degree of accuracy  
n is for operating loop  
d is for checking slope*  
3.Do for i =1 to n in step of 2  
4.f = f(x)  
5.f1 = f'(x)  
6.If ( |f1| < d ), then display too small slope and goto 11.  
7.x1 = x - f/f1  
8.If ( [(x1 - x)/x1] < e ), the display the root as x1 and goto 11.  
9.x = x1 and end loop  
10.Display method does not converge due to oscillation.  
11.Stop
```



The slope of the curve was such that I was trying to come here and somehow I miss the root, I go to another point. So, that is another special case I need not bother you with that right now. So, let us have a quick look at the program.

(Refer Slide Time: 32:28)

```

#include<stdio.h>
#include<math.h>
float f(float x)
{
    return x*log10(x) - 1.2;
}
float df (float x)
{
    return log10(x) + 0.43429;
}

void main()
{
    int itr, maxitr;
    float h, x0, x1, allerr;
    printf("\nEnter x0, allowed error and maximum
iterations\n");
    scanf("%f %f %d", &x0, &allerr, &maxitr);
    for (itr=1; itr<=maxitr; itr++)
    {
        h=f(x0)/df(x0);
        x1=x0-h;
        printf(" At iteration no. %3d, x = %9.6f\n", itr, x1);
        if (fabs(h) < allerr)
        {
            printf("After %3d iterations, root = %8.6f\n", itr, x1);
            return 0;
        }
        x0=x1;
    }
    printf(" The required solution does not converge or
iterations are insufficient\n");
    return 1;
}

```

So, here we are trying to find out the root of a function  $x \log x$  to the base 10 minus 1.2. So, that is the function that function is embodied in another C function. Now  $d f$  is nothing, but  $f$  dashed  $x$ .

(Refer Slide Time: 33:12)

```

#include<stdio.h>
#include<math.h>
float f(float x)
{
    return x*log10(x) - 1.2;
}
float df (float x)
{
    return log10(x) + 0.43429;
}

void main()
{
    int itr, maxitr;
    float h, x0, x1, allerr;
    printf("\nEnter x0, allowed error and maximum
iterations\n");
    scanf("%f %f %d", &x0, &allerr, &maxitr);
    for (itr=1; itr<=maxitr; itr++)
    {
        h=f(x0)/df(x0); ←
        x1=x0-h;
        printf(" At iteration no. %3d, x = %9.6f\n", itr, x1);
        if (fabs(h) < allerr)
        {
            printf("After %3d iterations, root = %8.6f\n", itr, x1);
            return 0;
        }
        x0=x1;
    }
    printf(" The required solution does not converge or
iterations are insufficient\n");
    return 1;
}

```

So, if this function is given, I also keep  $f$  dashed  $x$  written. So, this is  $f x$ , this is  $f$  dashed  $x$ . Now I know I have already pre coded them and that will return me the value for different values of  $x$ .

So, now again I read as scan f, i read the initial x 0, the allowed error, the maximum iteration. Now, then in this loop what I do I find fx by f dashed x and that is h -how much I should change the initial value. The initial value was x 0 with which I started, I subtract that and come to the next point.

If absolute value at that point is less than error then that is a solution. Otherwise, I will go up and repeat this. Now if I go on and ultimately if I overshoot the maximum iteration, then I can say that the required solution does not converge or the iterations are inefficient.

So, Newton-Raphson usually gives us a very fast way of finding the root, but sometimes it does not converge and that is one problem of that. However, there are many other sophisticated ways of finding roots, just to summarise I would like to say that what we have learnt in the past couple of lectures is that one of the major technological requirements, a computational requirements are finding roots of polynomials for many solutions for many engineering solutions. I have to solve equations.

For that there are many methods we have just gone through to simpler methods one is the bisection method and the other one that we saw just now is the Newton-Raphson method. Next we look at something else called interpolation and other things.

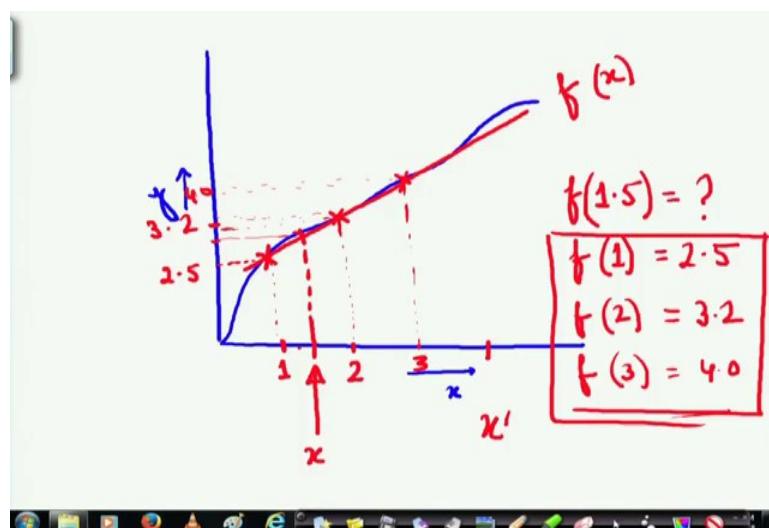
Thank you.

**Problem Solving Through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 51**  
**Interpolation**

Today we will first discuss a technique called interpolation. We have seen how a function can be represented in a computer in the form of a table or a 2 dimensional array. Now what is interpolation? Let us try to understand.

(Refer Slide Time: 00:52)



Say, I have got a function, where I know the, I have got a function which is something like this. But I do not know the function before that, all right? If I had known the function or the description of this function, in that case given any  $x$ , I could have found out the corresponding  $y$ , right.

But suppose the function is not given, instead what is given to us are for some specific  $x$ 's (say  $x = 1$ ) I have been given this value, all right? This value of  $y$ . For  $x$  equal to 2, I have been given this value of  $y$ , all right? This particular value of  $y$ . Similarly, for 3, I am given a particular value of

y, but I do not know what is the value of this function for 1.5. So, if I call this to be the function f of x, then f of 1.5 is not known. Although I know that f of 1 is say something 2.5, I know f of 2 to be maybe 3.2, f of 3 maybe something like 4.

But this information is known to me. Given this known part, can I find out what are the intermediate values for if 1.58, 1.3, 2.5, 2.6? That is the task of interpolation; that means, given some values known, I want to find out the value of the function for some independent value of the dependent variable. So, for a particular x, what is the part? What is the y? That's what I want to find. Now you see you can look at this blue line and say that it's so, simply you draw it like this and you will find the value of y.

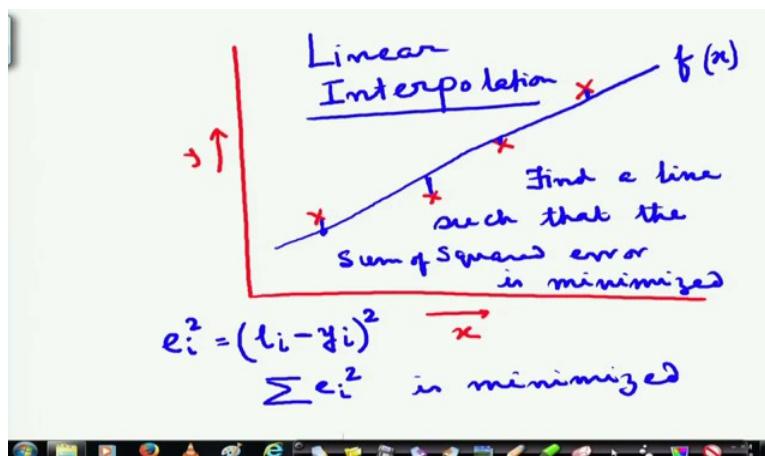
But unfortunately, what I have is not the blue line. What I have is only the rate crosses and this line is not there. I do not know these 3 points - whether the curve is something like this or the curve could be a straight line through these points, right? In that case you see, if it was the case, then the value of x, this particular x would be different from this. So, I have to actually see that what is the curve that best fits all these given points. This problem is known as interpolation.

Now, I can extend it suppose some table is given to you, where values for f 1, 2 and 3 are given and so, the range is from one to 3, but you are asked for some particular x prime, which is beyond this range. In that case it is also interpolation on the other side; that means, beyond the boundary. That is known as extrapolation, all right? Right now, so, basically the concept of extrapolation and interpolation are always the same.

So, here what I will try to see is given some values of fx, some known of fx for some particular x's, how do you find out the value of fx for some intermediate value of x? That is the task of interpolation. Now here I show I had shown a curve usually you can also start with a table like this, where some x's are given (Refer Slide Time: 06:03) say 0.1, 0.2, 0.3, 0.4, 0.5 and for each of them we have got some y value 2.7, 3.2, 1.6 comes down again, say, 0.6, and again it goes up say 3.2.

Now, if this be the table, then my question is for interpolation, what is the value? What is the value for  $x$  which is 0.27? What is the value of this? So, that is the task of interpretation. Now, obviously, a simplest possible thing is if I can fit in a line which is known as a linear interpolation.

(Refer Slide Time: 07:04)



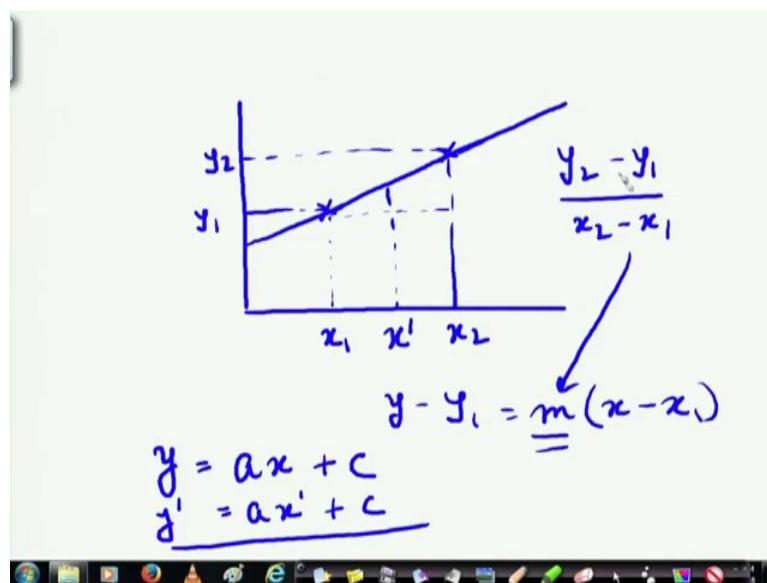
So, I have got some points here, all right? I try to fit in a line in between them or let me do it in a different way not exactly this line, because this line is matching 2 points, but not the others.

So, this is by  $y$  and this is  $x$ . So, there could be some points like this something here and something here like this equidistant points. Somehow I draw line like this. It is a straight line, although my drawing is a little curved, but it is a straight line. So, one way is that I try to find out a line such that I minimise the error. So, of course, if I assume that this line is representing my function, then I can see there is some error here, there is some error here and in other places.

I could have fitted another line also, I could have changed this line a little bit. One of the ways is to find the line that minimises the error. Now it depends on how you define the error,. Now here you can see it is a negative error, it is a positive error, like that it can go on. So, one way is to find the line such that the squared sum of squared error is minimised. That can be one way; that means, what is an error error? When  $e_i$  is (when I take whatever value my line is saying, let let me call it  $y_i$ ) whatever is the actual value minus  $y_i$ .

If I consider that to be the error or the other way, now you can see it is a positive or negative when I take the error of all these. So, that could be a sum of error. But here what I am saying is that, I can have the square of this error, and I can try to find out a line such that  $e_i^2$  is minimise so that I get a very good line. Now suppose I get a very good line like this, then I will have I have the equation of that line. Now given 2 points it is very simple, right, let us take the simplest case first.

(Refer Slide Time: 11:18)



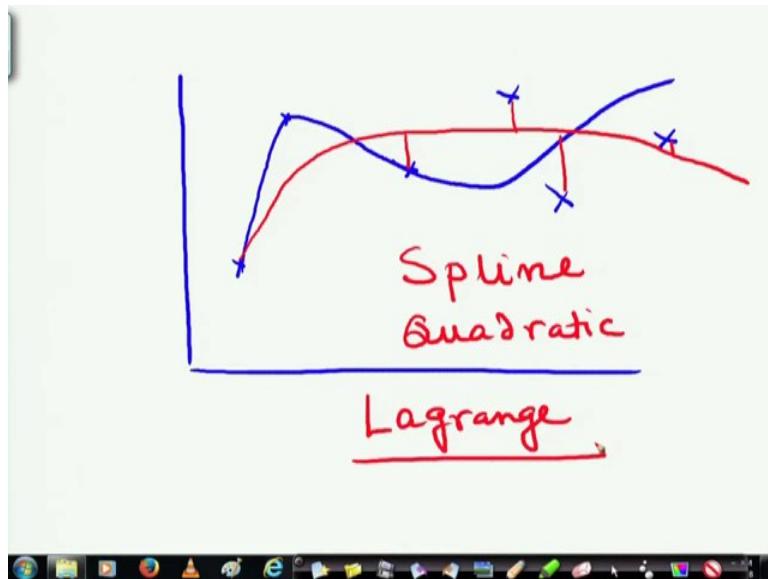
So, if there be only 2 points given, then I can certainly draw a unique line between these 2 points, right? So, I can find out the equation of these. So, this is x or school level co ordinate geometry  $x_1, y_1, x_2, y_2$ . So, you can find out the slope of this line. You can find the slope of this line, which will be nothing but  $y_2 - y_1$  divided by  $x_2 - x_1$ . So, once you get the slope, then you can find out the line. So, the school level equations like  $y - y_1$  is equal to  $m$  into  $x - x_1$ , where  $m$  is this  $m$  is a slope.

So, you can find the line so, when you find the line, then your problem is solved. Suppose you get a line of the equation  $a x + c$ ,  $c$  is this and you get the coefficients like this. Now given any  $x$ , I give  $x'$  then from that equation I can straight way find the value of  $y$   $a x' + c$ , right? That's straight forward, but the problem is that often it is not only 2 points, but the points are rather distributed as I was showing in the earlier one like this. Where I do not get an exact line; that is cutting through all the points. Here I will try to minimise the error and I will choose such a line.

Now, when I choose such a line; that means, I have chosen an equation. Therefore, again from the linear equation, I can find out the value of  $y$  for any given  $x$ . So, this approach is known as linear interpolation; however, the linear interpolation often as you can see the error will be there. So, people try to find out more accurate solutions. So, first of all we have seen what is linear interpolation, the basic idea is very simple we try to fit in the best possible straight line, through all the points. So, that the error is minimised, total error is minimised.

So, one way is to minimise the sum of the squares or there could be some other measures you can take for error, that should be minimised so, you get a line to your satisfaction. So, once you get a line to your satisfaction, you know the equation of the line. Therefore, given any  $x$  you can find the corresponding  $y$ . Now fitting this line always, the there may be situations where the points are distributed in such a way for example points are something like this, all right, is very difficult to find the line over here.

(Refer Slide Time: 14:46)



So, often I would like to find out a curve, or maybe another curve even better, that can be that goes through these points like this, where I have got errors, but I am trying to fit in a curve. So, that is not a line, this is not a linear interpolation, where are further extensions like quadratic interpolation, spline interpolation, cubic spline and all those. So, it is better to know the names, spline interpolation, quadratic interpolation, where I want to fit in a quadratic curve etc.

But today we will discuss another interpolation technique which is known as LaGrange interpolation, in the name of the mathematician who invented it. So, let us have a look at the lagrange interpolation.

(Refer Slide Time: 16:08)

Given a set of  $k+1$  data points  $(x_0, y_0), \dots, (x_j, y_j), \dots, (x_k, y_k)$

The Lagrange Polynomial is

$$\ell_j(x) := \prod_{\substack{0 \leq m \leq k \\ m \neq j}} \frac{x - x_m}{x_j - x_m} = \frac{(x - x_0)}{(x_j - x_0)} \cdots \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \cdots \frac{(x - x_k)}{(x_j - x_k)},$$

Interpolation as you have understood is given a set of  $k$  plus 1 data points -  $x_0, y_0, x_1, y_1, x_2, y_2$  like that up to  $x_k, y_k$ , so many data points are given. The LaGrange polynomial is this - I have to find out for a particular  $x$ ,  $x$  minus  $x_n$  divided by  $x_j$  minus  $x_m$ , where  $m$  is varying from 0 to  $k$ . For all these  $m$  is varying. So,  $x_0, x_1, x_2, x_3$  for all these I will do that and  $x_j$  minus  $x_m$ , where  $j$  is for a particular LaGrange interpolation.

So, you see it is for  $1 \ 1 \ x$  it will be  $x$  minus  $x_0$  by  $x_1$  minus  $x_0$ ,  $x$  minus  $x_0$  by  $x_1$  minus  $x_0$ , like that it will go on, all right, so, this is the LaGrange polynomial. So, using this polynomial let us see how it works, just to give you an example.

(Refer Slide Time: 17:40)

$\ell_j(x) := \prod_{\substack{0 \leq m \leq k \\ m \neq j}} \frac{x - x_m}{x_j - x_m} = \frac{(x - x_0)}{(x_j - x_0)} \cdots \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \cdots \frac{(x - x_k)}{(x_j - x_k)},$

$x_0 = 1 \quad f(x_0) = 1$   
 $x_1 = 2 \quad f(x_1) = 4$   
 $x_2 = 3 \quad f(x_2) = 9.$

$$L(x) = 1 \cdot \frac{x-2}{1-2} \cdot \frac{x-3}{1-3} + 4 \cdot \frac{x-1}{2-1} \cdot \frac{x-3}{2-3} + 9 \cdot \frac{x-1}{3-1} \cdot \frac{x-2}{3-2}$$

$$= x^2,$$

$x_0 = 1 \quad f(x_0) = 1$   
 $x_1 = 2. \quad f(x_1) = 8$   
 $x_2 = 3 \quad f(x_2) = 27$

$$L(x) = 1 \cdot \frac{x-2}{1-2} \cdot \frac{x-3}{1-3} + 8 \cdot \frac{x-1}{2-1} \cdot \frac{x-3}{2-3} + 27 \cdot \frac{x-1}{3-1} \cdot \frac{x-2}{3-2}$$

$$= 6x^2 - 11x + 6.$$

I have reproduced that polynomial, that I had shown in the earlier slide here again, you can say this. Now suppose I have got the points given - look here, the points at a given are for  $x$  value 1,  $f(x)$  is 1. For  $x$  value 2,  $f(x)$  is 4, 3 and this 9. So, I have to fit in a polynomial. So, my polynomial is  $L(x)$ , now each term of this is one  $L(x)$ . So, this is  $L_1(x)$ , this is  $L_2(x)$ , this is  $L_3(x)$ , like that ok. So, you see the value of  $x$  is 1, 1 times  $x$  minus  $x$  minus 1.  $M$  will be something that would be not the same as  $j$ .

So, it will be I have got the 3 points so, I will not take this 1,  $x$  minus 2 divided by 1 minus 2 times the second term  $x$  minus 3 divided by 1 minus 3 plus 4 now I am taking this one so, it is 4 and, in that case, I leave out 2, my  $x$   $j$  is now 2. So,  $x$  minus 1( I will not take  $x$  minus 2 here. Because look at this  $m$  not equal to  $j$ . So, for the other  $x$ 's, I have selected this row. So, I will only concentrate on the other rows) divided by 2 minus 1 into  $x$  minus 3 the other row divided by 2 minus 3. Here I take 9 - this value and I am taking this row.

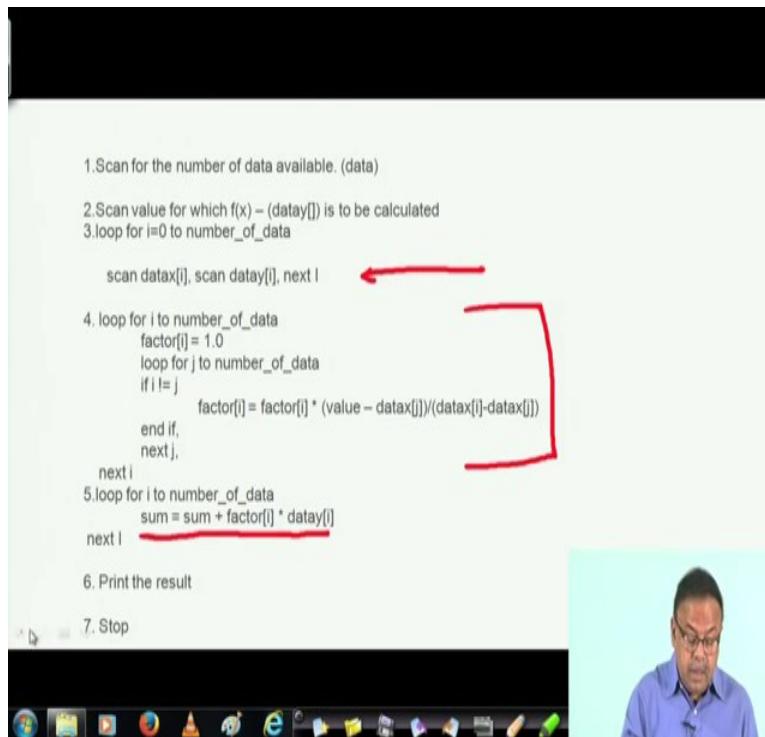
Now, so, now I am taking this row, right? I have taken 9. So,  $x$  minus 1 by 3 minus 1,  $x$  minus 2 by 3 minus 2. If I solve this I am getting a polynomial  $x$  square. Possibly you can see that is a perfect fit with the given data points -  $x$  1,  $f(x)$  is  $x$  square, that is 1, 2 it is  $x$  square, 4, 3 it is  $x$  squared 9. So, it is in this case, there is no it is absolutely a perfect fit . So, this is how I get the polynomial, all right? Let us try once again. For these data value, for  $x$  0 equal to 1, the  $f(x)$  is 1, for  $x$  equal to 2 it is 8, for  $x$  equal to 3 it is 27.

So, obviously, you can see it is no longer  $x$  square, all right? It is not also  $x$  cube, because here it is cube, it is matching the cube it is actually  $x$  cube, right? But let us try to fit in a polynomial here. So, what is that polynomial using the LaGrange method, I will first check one. So, I am taking first row, taking this value one here, then  $x$  minus 2, 1 minus 2. Times  $x$  minus 3, 1 minus 3 plus next time I take the second row. 8 times  $x$  minus 1 divided by 2 minus 1, because here it is the value is 2. For the third row it will be again  $x$  minus 1, 3 minus 1,  $x$  minus 2 , 6 minus 2.

By doing this I am not getting  $x$  cube which would be a perfect fit, but I am getting a close enough polynomial which is  $6x^2 - 11x + 6$ , which will be approximating the cube function very closely, you can see that. So, it is not that LaGrange polynomial will always give you the perfect fit, but it gives you a very close fit. So, this is what is known as LaGrange

interpolation. So, if I want to write a program and try to solve it, I left to implement a function, I have to write a function that will implement this or this, right?

(Refer Slide Time: 23:24)



The screenshot shows a computer screen with a slide containing an algorithm for linear interpolation. The algorithm steps are:

1. Scan for the number of data available. (data)
2. Scan value for which  $f(x) - (\text{datay}[i])$  is to be calculated
3. loop for  $i=0$  to  $\text{number\_of\_data}$ 
  - scan  $\text{datax}[i]$ , scan  $\text{datay}[i]$ , next i
4. loop for  $i$  to  $\text{number\_of\_data}$ 
  - $\text{factor}[i] = 1.0$
  - loop for  $j$  to  $\text{number\_of\_data}$
  - if  $i \neq j$ 
    - $\text{factor}[i] = \text{factor}[i] * (\text{value} - \text{datax}[j]) / (\text{datax}[i] - \text{datax}[j])$
  - end if,
  - next j,
- next i
5. loop for  $i$  to  $\text{number\_of\_data}$ 
  - $\text{sum} = \text{sum} + \text{factor}[i] * \text{datay}[i]$
- next i
6. Print the result
7. Stop

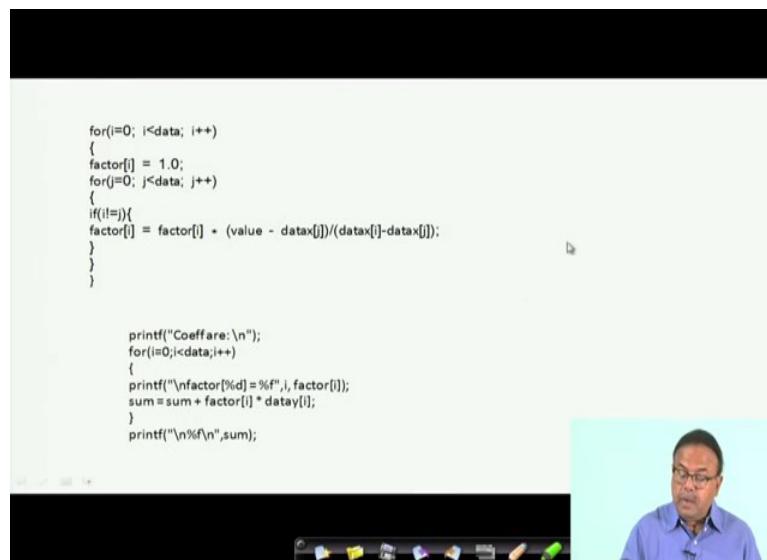
A red bracket and arrow are drawn on the slide to highlight the calculation of the factor in step 4. The bracket groups the assignment statement  $\text{factor}[i] = \text{factor}[i] * (\text{value} - \text{datax}[j]) / (\text{datax}[i] - \text{datax}[j])$ . An arrow points from the start of this group to the opening brace of the inner loop.



Let us see how we can do that. So, here is an algorithm, scan the number of data available for x and y values. Now here there is the how many data points? This is telling how many data points, you are finding the data. Now here loop for i equal to number of data factor, then here you are computing this. Here you see what we are trying to do? At this point we have just read the data, and here in a loop we are trying to compute, here we are finding the sum.

But here in the loop, I am finding the factor so, if you go to the earlier the slide, you will find that I have got these are my factors, right? I am finding each of these factors, I am multiplying them, this is a factor, this whole thing is a factor, and then I multiply that with the coefficient and add them up. So, I am doing the summation at the end. So, here you see that I am taking the factor then I am taking whatever is the value (this part you can yourself compute)

(Refer Slide Time: 24:51)



```

for(i=0; i<data; i++)
{
    factor[i] = 1.0;
    for(j=0; j<data; j++)
    {
        if(i!=j){
            factor[i] = factor[i] + (value - datax[j])/(datax[i]-datax[j]);
        }
    }
}

printf("Coeffare:\n");
for(i=0;i<data;i++)
{
    printf("\nfactor[%d] = %f",i,factor[i]);
    sum=sum + factor[i] * datay[i];
}
printf("\n%f\n",sum);

```

The key part is computation of the factor part, where I am showing this part, you can see that for I 0 to data, i plus plus I am initially a factor is one.

Because I am multiplying that, then if i is not equal to j, I will take factor times the value minus the data. So then ultimately, I am printing the coefficient. So, you can translate that into the form of a c function. That is the basic idea of LaGrange interpolation, you can similarly write programs for linear interpolation, and other interpretation techniques. So, the point of this course

is not because I assume that you now know how to do programming and these are simple loop type of programming.

So, you will be able to do that. So, whenever you want to solve apply some numerical methods, you will have to first look at the technique then try to find out the expression and you will have to write an algorithm so that you can solve it ok. In the next lecture we will discuss 2 other very interesting techniques for integration and differential equation solver.

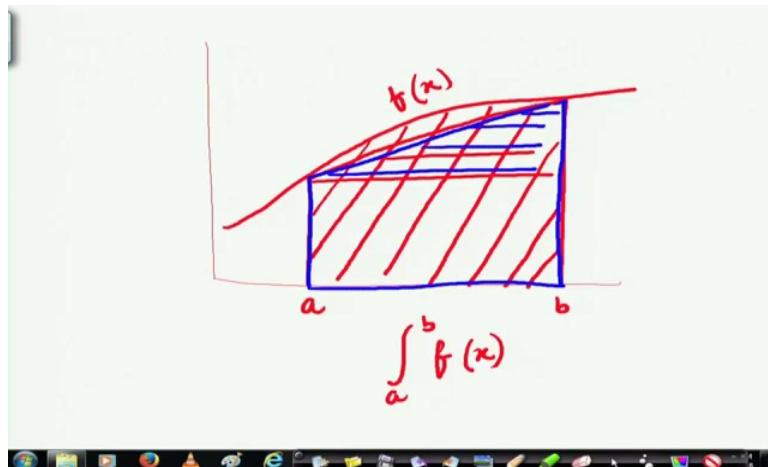
Thank you.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 52**  
**Trapezoidal Rule and Runge-Kutta Method**

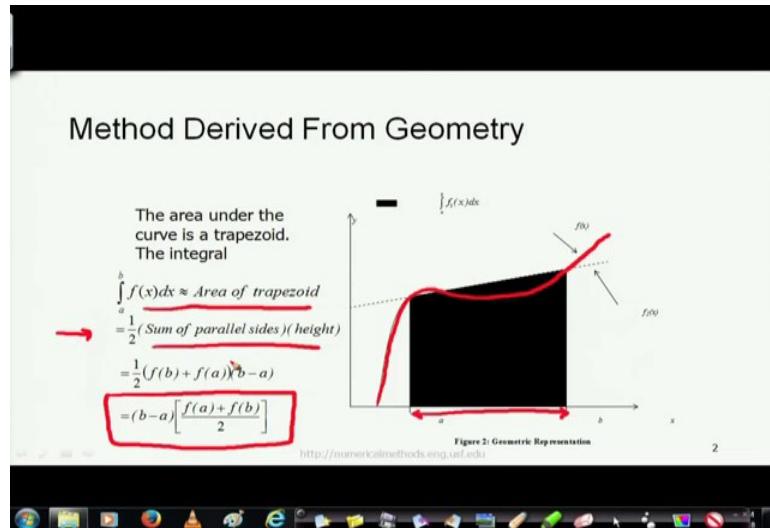
In this lecture, we will be looking at another numerical method technique; we will actually look at two. First, we look at integration, how we can integrate a function, there are several methods for doing that. We will look at only one method and you can after that; you can look up at for other methods. Next will proceed to see, how ordinary differential equation can be solved using numerical techniques using program. So, first of all let us start with integration.

(Refer Slide Time: 00:52)



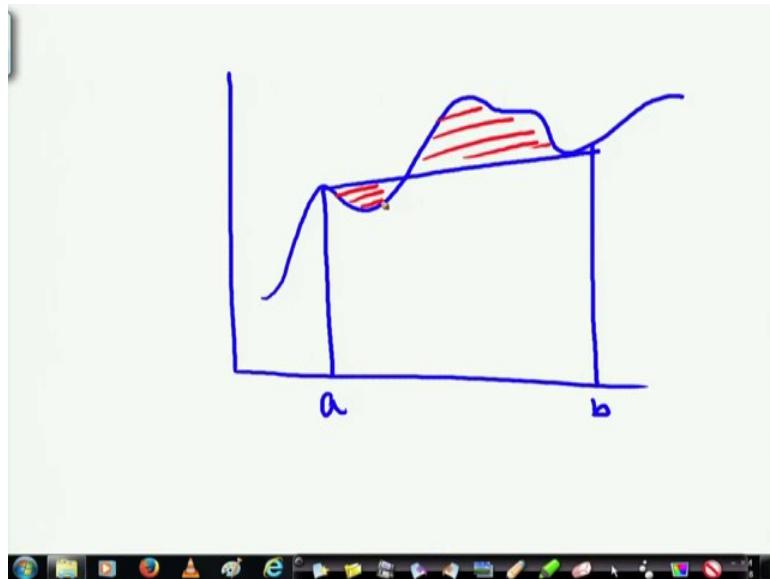
Now, all of you know; what integration means given a particular function integration is say I have got a function like this and I want to integrate. Suppose this function is  $f(x)$  and I want to integrate it within the range  $a$  and  $b$ . So, we write that as  $\int_a^b f(x) dx$  and you also know that this integration actually means the area that is under this curve. This value is the integration. So, we will look at how we can solve this problem. The simplest and very simple method is a trapezoidal method which we will discuss here.

(Refer Slide Time: 01:48)



Let me go back to this case. Now, you can see that I can approximate this curve in a very simplistic case using a trapezoid. So, I have got a trapezoid here, let me draw in blue. So, the area under this trapezoid is approximating the area under the curve, but that is becoming possible here (except for this error) because the curve was very much not very skewed in that sense, but if the curve were something like this.

(Refer Slide Time: 02:41)



If it was something like this and I was trying to integrate it between these ranges then if I had fitted a trapezoid here, then the amount of error would be much more because I will be committing errors at this points. I am not considering this, I am over considering this points, etc.

So, the error will be more. So, it is not always the case that a simple trapezoid, one single trapezoid will solve, but let us start with that and try to understand how we can go ahead with the problem. So, the area under the curve in this particular curve as you can see here, this particular curve that has gone through this is a trapezoid, under the curve is a trapezoid. This part if I assumed to be a trapezoid in that case the integral of  $f(x) dx$  is the area of the trapezoid and we know that the area of a trapezoid is not nothing but half the sum of the parallel sides; that means,  $f(a) + f(b)$  divided by the height.

Now, if I look at it these are the parallel sides then the height is this  $-b - a$ , this amount, right. So, this is a known result therefore, I can see if I can approximate a curve by a single trapezoid in this way,  $b - a$  times  $f(a) + f(b)$  plus  $f(a) + f(b)$  by 2 or  $f(a) + f(b)$  by 2, but of course, we will see that there can be errors due to this approximation, but this is a simple formula which we can quickly compute. It is very easy to write a program for that, you have got a function that will

compute the curve. So, you call it for a and call it for b and compute this expression, you will get the integral all right.

(Refer Slide Time: 04:56)

The vertical distance covered by a rocket from  $t=8$  to  $t=30$  seconds is given by:

$$x = \int_8^{30} \left( 2000 \ln \left[ \frac{140000}{140000 - 2100t} \right] - 9.8t \right) dt$$

a) Use single segment Trapezoidal rule to find the distance covered.  
b) Find the true error,  $E$ , for part (a).  
c) Find the absolute relative true error,  $|e_r|$ , for part (a).

So, here is an example why is it important suppose a vertical distance is covered by a rocket from time 8 to 30. So, my timeline is from 8 seconds to 30 seconds is given by this formula. So, the vertical distance overall, the total vertical distances is this is - a complicated formula. Now using single segment trapezoidal rule, let us try to find the distance covered.

(Refer Slide Time: 05:39)

**Solution**

a)  $I \approx (b - a) \left[ \frac{f(a) + f(b)}{2} \right]$

$$a = 8 \quad b = 30$$

$$f(t) = 2000 \ln \left[ \frac{140000}{140000 - 2100t} \right] - 9.8t$$

$$f(8) = 2000 \ln \left[ \frac{140000}{140000 - 2100(8)} \right] - 9.8(8) = 177.27 \text{ m/s}$$

$$f(30) = 2000 \ln \left[ \frac{140000}{140000 - 2100(30)} \right] - 9.8(30) = 901.67 \text{ m/s}$$

4

<http://numericalmethods.eng.usf.edu>

So, we know for 8 and for 30, if I compute this function (this was my function).

So, for  $f(8)$ , my function is yielding this value and for  $f(30)$ , the function is yielding this value. We can compute, using your calculator you can find it out but you need not do right now. So, you can compute the values at these 2 points because this is the overall function. So, this is the  $f(8)$  and  $f(30)$ . So, what will be my integral? My integral will therefore, be  $30 - 8$ ,  $b - a$ ,  $f(a) + f(b)$  plus  $ab$  by 2.

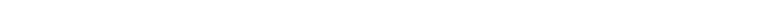
(Refer Slide Time: 06:22)

Solution (cont)

b)  $E_t = \text{True Value} - \text{Approximate Value}$   
 $= 11061 - 11868$   
 $= -807 \text{ m}$

c) The absolute relative true error,  $|\epsilon_t|$ , would be  
 $|\epsilon_t| = \left| \frac{11061 - 11868}{11061} \right| \times 100 = 7.2959\%$

<http://numericalmethods.eng.usf.edu> 6



So, that is coming to 11868 meter, the distance covered is this, but however, the exact value; if you computed the exact value if you do a detailed computation will be 11061 meter. So, there is an error; obviously, there is an error because I have approximated the curve using a trapezoid. Now, let us see how great is the error .So, we can see that the true error is minus 807 meter, right; that is quite significant 800 meters and the absolute relative error which is the actual error ie true value and the computed value and divided by the true value you find that I am getting more than 7 percent error, how can you better it ?

(Refer Slide Time: 07:30)

**Multiple Segment Trapezoidal Rule**

In Example 1, the true error using single segment trapezoidal rule was large. We can divide the interval  $[8,30]$  into  $[8,19]$  and  $[19,30]$  intervals and apply Trapezoidal rule over each segment.

$$f(t) = 2000 \ln\left(\frac{140000}{140000 - 2100t}\right) - 9.8t$$

$$\int_8^{30} f(t) dt = \int_8^{19} f(t) dt + \int_{19}^{30} f(t) dt$$

$$= (19 - 8) \left[ \frac{f(8) + f(19)}{2} \right] + (30 - 19) \left[ \frac{f(19) + f(30)}{2} \right]$$

7

<http://numericalmethods.eng.usf.edu>

So, our answer will be instead of fitting in a single trapezoid, I can try to fit in more than one trapezoid here like something like this - I fit in one trapezoid here, I fit in another trapezoid here and hereby I can approximate, I can minimise my error to some extent.

So, here; what we are trying to do instead of taking 8 and 30 and fitting in one trapezoid for the whole thing what we are trying to do is we are fitting in one trapezoid for 8 to 19 and other for 19 to 30. So, now, again using the same formula we find then here is one 19 to 8 to 19, this is the integral plus the other trapezoid is giving me this. So, I am fitting in 2 trapezoids now.

(Refer Slide Time: 08:45)

The slide title is "Multiple Segment Trapezoidal Rule". Below it, the text "With" is followed by three equations:  
 $f(8) = 177.27 \text{ m/s}$   
 $f(30) = 901.67 \text{ m/s}$   
 $f(19) = 484.75 \text{ m/s}$

Below these, the text "Hence:" is followed by a mathematical derivation:  
$$\int_8^{30} f(t) dt = (19 - 8) \left[ \frac{177.27 + 484.75}{2} \right] + (30 - 19) \left[ \frac{484.75 + 901.67}{2} \right]$$
  
The terms  $\frac{177.27 + 484.75}{2}$  and  $\frac{484.75 + 901.67}{2}$  are highlighted with red brackets. The result is:  
 $= 11266 \text{ m}$

At the bottom of the slide, there is a URL: <http://numericalmethods.eng.usf.edu>. The slide number is 8.

So, if I do that; then let us see whether the result is being bettered? So, I compute  $f(8)$  at  $f(30)$  and  $f(19)$  and compute both of these values, the areas under the curve here and here and the result is 11266 meters. Now how far is it from the actual?

(Refer Slide Time: 09:10)

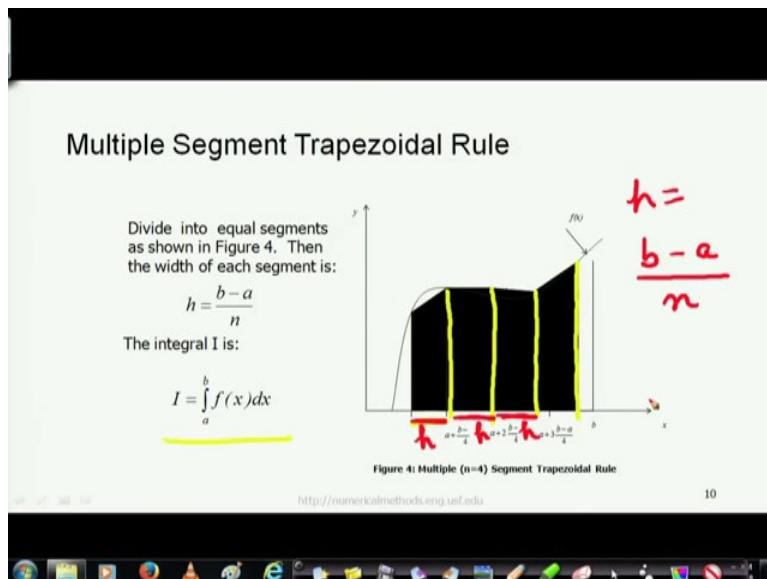
The slide title is "Multiple Segment Trapezoidal Rule". Below it, the text "The true error is:" is followed by:  
 $E_t = 11061 - 11266$   
 $= -205 \text{ m}$

Below this, the text "The true error now is reduced from -807 m to -205 m." is displayed.

Extending this procedure to divide the interval into equal segments to apply the Trapezoidal rule; the sum of the results obtained for each segment is the approximate value of the integral.

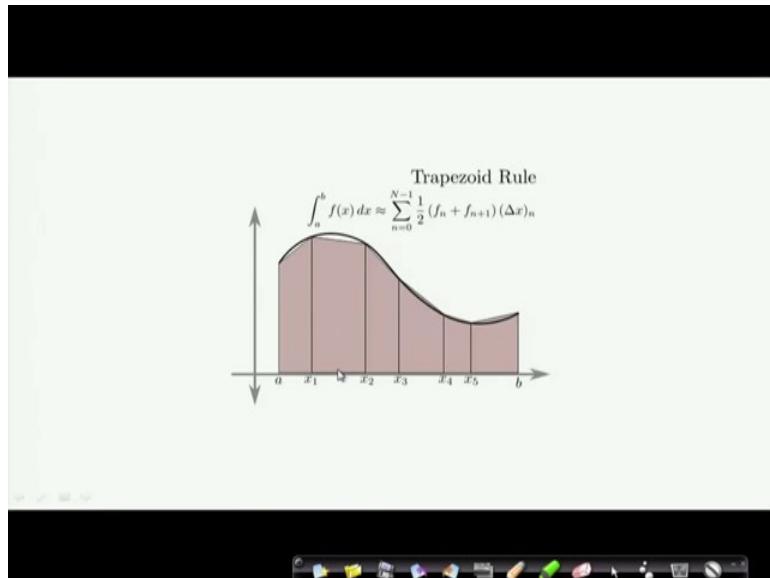
At the bottom of the slide, there is a URL: <http://numericalmethods.eng.usf.edu>. The slide number is 9.

Now you can see that the true error has reduced from 807 meters to 205 meters. So, that tells us that if we can extend this procedure and fitting more and smaller but more number of trapezoids my error will come down further still. (Refer Slide Time: 09:34)



The multiple segment trapezoidal rule; therefore, is that we divide into equal segments. So here is one trapezoid here is another trapezoid here is another trapezoid and here is another trapezoid. I am getting 4 trapezoids here and trying to formulate this. I can do that in this way. So, the integral is this whole thing which will be a sum of these trapezoids. Remember these lines, these distance is the same. So, I am dividing  $b$  minus  $a$  by some particular value  $n$  and that is my  $h$ . So, this is  $h$ , this is  $h$ , this is  $h$  like that I am going for equidistant points and drawing the trapezoid from there.

(Refer Slide Time: 10:41)



So, therefore, if I follow this, as is shown in this way, the trapezoidal rule can be written as integral from a to b is sum of half  $f_n$  plus  $f_n$  plus 1; that is  $f(a) + f(b)$  by 2 times the particular distance that is there,  $x_0 - x_1$  minus  $x_1 - x_2$  minus  $x_2 - x_3$  and so on. Usually do it in the equidistant way.

(Refer Slide Time: 11:15)

## Multiple Segment Trapezoidal Rule

The integral  $I$  can be broken into  $n$  integrals as:

$$\int_a^b f(x)dx = \int_a^{a+h} f(x)dx + \int_{a+h}^{a+2h} f(x)dx + \dots + \int_{a+(n-2)h}^{a+(n-1)h} f(x)dx + \int_{a+(n-1)h}^b f(x)dx$$

Applying Trapezoidal rule on each segment gives:

$$\int_a^b f(x)dx = \frac{b-a}{2n} \left[ f(a) + 2 \left( \sum_{i=1}^{n-1} f(a+ih) \right) + f(b) \right]$$

http://numericalmethods.eng.usf.edu

12



Ok. So, now, multi segment trapezoidal rule is therefore, an integral  $I$  which can be broken down, this is simple. From  $a$  to  $a + h$ , I can have  $f(x)dx$ ,  $a + h$  to  $a + 2h$   $f(x)dx$  plus  $a + n - 2h$  to  $a + n - 1h$   $f(x)dx$  plus  $a + n - 1h$  to  $b$ .

So, all the segments serving as it together. Applying this I get this formula  $b - a$  divided by  $2n$  because  $2$  is coming  $n$  times. So,  $f(a) + f(b)$  plus  $2$  into  $f(a + ih)$  because that is coming twice here once and here once, you see here if  $a + h$  will come then here  $a + h$  will come right.

(Refer Slide Time: 12:13)

## Example 2

The vertical distance covered by a rocket from  $t_0$  to  $t_1$  seconds is given by:

$$x = \int_{t_0}^{t_1} \left( 2000 \ln \left[ \frac{140000}{140000 - 2100t} \right] - 9.8t \right) dt$$

- a) Use two-segment Trapezoidal rule to find the distance covered.
- b) Find the true error,  $E_t$ , for part (a).
- c) Find the absolute relative true error,  $|e_r|$ , for part (a).



So, using this let us do the example again the same thing using 2 segment Trapezoidal rule, we could find that the error is coming down.

(Refer Slide Time: 12:19)

## Solution

a) The solution using 2-segment Trapezoidal rule is

$$I = \frac{b-a}{2n} \left[ f(a) + 2 \left\{ \sum_{i=1}^{n-1} f(a+ih) \right\} + f(b) \right]$$

$$n = 2 \quad a = 8 \quad b = 30$$

$$h = \frac{b-a}{n} = \frac{30-8}{2} = 11$$

<http://numericalmethods.eng.usf.edu>



(Refer Slide Time: 12:23)

## Solution (cont)

Then:

$$I = \frac{30-8}{2(2)} \left[ f(8) + 2 \left\{ \sum_{i=1}^{1-1} f(a+ih) \right\} + f(30) \right]$$

$$= \frac{22}{4} [f(8) + 2f(19) + f(30)]$$

$$= \frac{22}{4} [177.27 + 2(484.75) + 901.67]$$

$$= 11266 \text{ m}$$

<http://numericalmethods.eng.usf.edu>



(Refer Slide Time: 12:24)

## Solution (cont)

b) The exact value of the above integral is

$$x = \int_0^{30} \left( 2000 \ln \left[ \frac{140000}{140000 - 2100t} \right] - 9.8t \right) dt = 11061 \text{ m}$$

so the true error is

$$\begin{aligned} E_t &= \text{True Value} - \text{Approximate Value} \\ &= 11061 - 11266 \end{aligned}$$

<http://numericalmethods.eng.usf.edu>



We have already seen that the true value of error is coming down.

(Refer Slide Time: 12:28)

## Solution (cont)

c) The absolute relative true error,  $|e_t|$ , would be

$$|e_t| = \left| \frac{\text{True Error}}{\text{True Value}} \right| \times 100$$

$$= \left| \frac{|11061 - 11266|}{11061} \right| \times 100$$

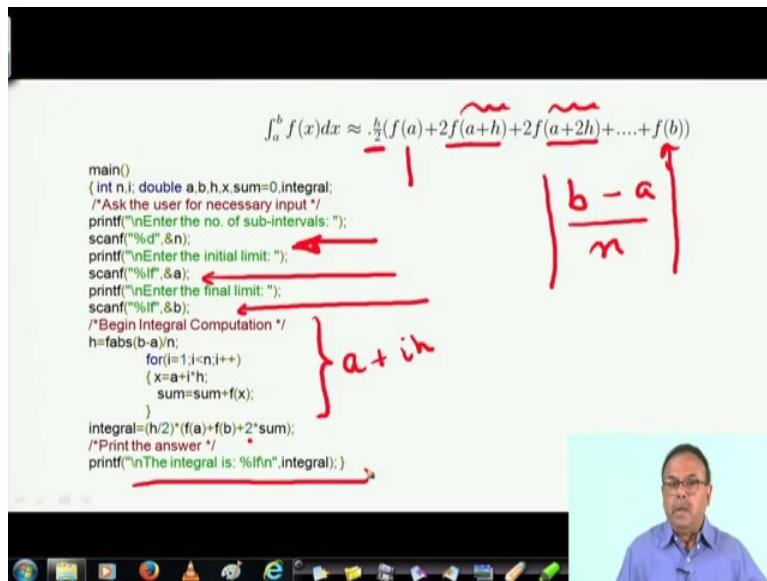
$$= 1.853\%$$

<http://numericalmethods.eng.usf.edu>



And the absolute relative error has come to 1.853.

(Refer Slide Time: 12:23)



So, now let us come to c programming, straightway; how can we encode it using a c program, you see here. So, I am reproducing the result here again.

So, integral of  $f(x) dx$  between  $a$  and  $b$  will be  $h$  by  $2 f(a) + 2 f(a+h) + \dots + 2 f(a+(n-1)h) + f(b)$ , why this 2 is coming because in the first zone  $f(a)$  and  $f(a+h)$  second zone  $f(a+h)$  plus  $f(a+2h)$ . So, each of these intermediate points are coming twice, that is why this is 2 and I have got this formula. Now, as a c programmer, your task is very simple you see here that I have defined  $n, i$  whatever. I am asking the user for the necessary inputs. how many number of subintervals that you want to have? The initial limit  $a$ , you are reading the initial limit  $b$ , all those things you are reading.

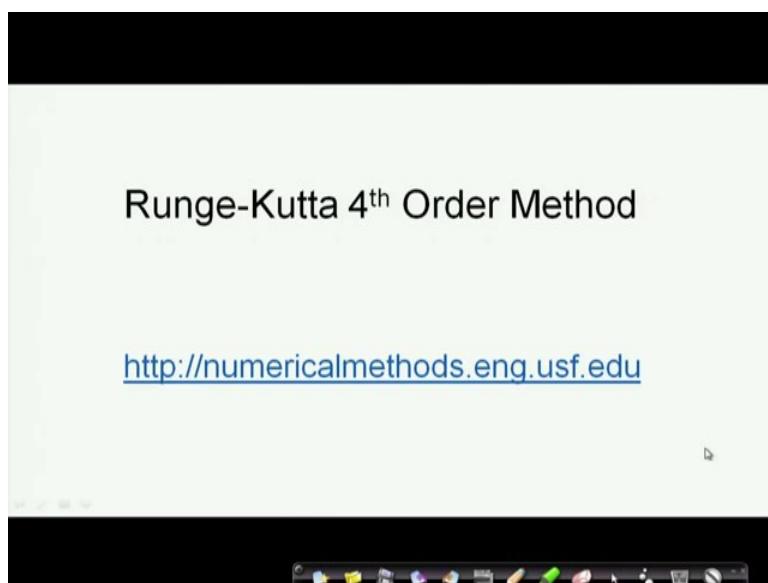
Now, the integral computation is done here. What is being done? I am finding out the absolute value of  $b$  ( $b$  minus  $a$  divided by  $n$  and I am taking the absolute value of that) because it could have gone on the other side also. So, now, here I am just computing the sum, what sum? This part. Initially  $x$  is  $a + ih$ ,  $sum$  is  $sum + f(x)$ . So, next time, it is becoming  $2, I$  am adding all

those things here. So, here I am in a loop, I am doing a plus i h initially i is 1. So, 1 h, 2 h like that I am going on.

Ultimately and I am completing the sum here, some plus f x and note that f x is a separate function that is being kept somewhere here and then ultimately I find the sum; sum is these things - a plus h these points. Now integral is h by 2- this part, f a plus f b plus f a plus f b plus twice the sum a plus a h, a plus 2 h, a plus k h like that t a plus i h has been completed inside this loop and that I had with 2 here and here is my integral. So, that is the trapezoidal rule.

The program is so simple, if you understand the concep. Next we will move to another very important engineering computation that is needed in solving ordinary differential equations. Quickly lets all of you know what a differential equation is.

(Refer Slide Time: 15:42).



We will in particular look at Runge Kutta fourth order method and here I will show examples that you can also find in this site of University of South Florida, numerical methods and I have

taken the slides from them with the permission. Now you see how to write an ordinary differential equation. Now an ordinary differential equation you know -it is  $\frac{dy}{dx} = f(x, y)$ .

(Refer Slide Time: 16:02)

How to write Ordinary Differential Equation

How does one write a first order differential equation in the form of

$$\frac{dy}{dx} = f(x, y)$$

**Example**

$$\frac{dy}{dx} + 2y = 1.3e^{-x}, y(0) = 5$$

is rewritten as

$$\frac{dy}{dx} = 1.3e^{-x} - 2y, y(0) = 5$$

In this case

$$f(x, y) = 1.3e^{-x} - 2y$$

21

So how do I write it? I write it as suppose this is something that is given,  $\frac{dy}{dx} + 2y$  is  $1.3e^{-x}$  and what is this part? This is the initial condition. Now this can be rewritten as (just by changing the directions because I have to bring it to this form)  $\frac{dy}{dx}$  is  $1.3e^{-x} - 2y$ . So, in this case we will assume that our  $f(x, y)$ , there is a  $\frac{dy}{dx}$  which is  $1.3e^{-x} - 2y$  this is our differential equation that we will have to solve. (Refer Slide Time: 17:18)

**Runge-Kutta 4<sup>th</sup> Order Method**

For  $\frac{dy}{dx} = f(x, y), y(0) = y_0$   
 Runge Kutta 4<sup>th</sup> order method is given by  
 $y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h$   
 where  
 $k_1 = f(x_i, y_i)$   
 $k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right)$   
 $k_3 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h\right)$   
 $k_4 = f(x_i + h, y_i + k_3 h)$

http://numericalmethodsganguli.net/

22

So, for  $\frac{dy}{dx}$ , the Runge Kutta fourth order method, (I am not going into the derivation of it for positive of time and you can always look at look into this at any website or you can look at any numerical method textbook) takes 4 terms and if this is the expression, given this  $\frac{dy}{dx}$  my task of solving a differential equation is to find a particular  $y$ , right? So, what we are trying to do  $y_i$  plus 1 is  $y_i$ ; some particular  $y_i$  plus 1 by 6 followed by a term and what is that term  $k_1$  plus 2  $k_2$  plus 2  $k_3$  plus  $k_4$  - this whole thing multiplied by  $h$ .  $H$  is the again the sampling that is the distance between the individual points that we looked at; now what is  $k_1$ ?

When I am taking for  $y_i$ ,  $f(x_i, y_i)$  - that particular function is  $k_1$ . What is  $k_2$ ? Suppose there is a curve. Now I have been given the slope, I do not know the curve. If know the curve then I can find out the value of any particular  $y_i$  plus 1 given any  $y_i$ . Now given any  $y_i$ , I am trying to guess the curve, right? I am trying to solve the curve. So,  $x_i$  was here,  $x_i$  plus half  $h$ , I am taking whatever was my  $h$ , I am taking half of  $h$  and what is the  $y$  part of this function  $y_i$  plus half of  $k_1 h$ .

So, because  $k_1$  was  $f_i$ ,  $k_1$  was the function that was giving  $y$  given an  $x$ . So, I am taking this. What is  $k_3$ ? This part is same,  $x_i$  plus half  $h$ , but this part is now becoming much more

predictive. So, it is half of k 2 h ; that means, whatever has been completed here times h and k 4 is x i plus h, the last one is x i plus h because I am trying to solve the equation within this zone h. So, x i plus h, i start with x i and this is x i plus h times k 3 h. Here there is no half. Now this derivation you can look at, but ultimately I multiply it with 1 by 6.

(Refer Slide Time: 20:39)

## Runge-Kutta 4<sup>th</sup> Order Method

For  $\frac{dy}{dx} = f(x, y), y(0) = y_0$

Runge Kutta 4<sup>th</sup> order method is given by

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h$$

where

$$k_1 = f(x_i, y_i)$$

$$k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1h\right)$$

$$k_3 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2h\right)$$

$$k_4 = f(x_i + h, y_i + k_3h)$$

<http://numericalmethods.eng.usf.edu> 22

So, given this Runge Kutta formula let us quickly look at an example, that we have a nice thing to look at.

(Refer Slide Time: 20:43)

**Example**

A ball at 1200K is allowed to cool down in air at an ambient temperature of 300K. Assuming heat is lost only due to radiation, the differential equation for the temperature of the ball is given by

$$\frac{d\theta}{dt} = -2.2067 \times 10^{-12} (\theta^4 - 81 \times 10^4), \theta(0) = 1200K$$

Find the temperature at  $t = 480$  seconds using Runge-Kutta 4<sup>th</sup> order method.

Assume a step size of  $h = 240$  seconds.

$$\frac{d\theta}{dt} = -2.2067 \times 10^{-12} (\theta^4 - 81 \times 10^4)$$

$$f(t, \theta) = -2.2067 \times 10^{-12} (\theta^4 - 81 \times 10^4)$$

$$\theta_{i+1} = \theta_i + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4)h$$

RK

23

Suppose a ball is at 1200 Kelvin and is allowed to cool down in air in an ambient temperature of 300 Kelvin. So, here the ball was heated and the air is cooling it down. Assuming that the heat is lost only due to radiation the differential equation for the temperature is given here. This one, where theta 0; we know the initial condition is 1200 Kelvin. Find the temperature at t 480. So, what is my x i? So, now, if I assume a step size of 240, I want to find out the temperature at 480. So, suppose it was at a particular temperature after 480 seconds.

So, here is the time, it was at 1200 and I have got some radiation formula using which it is coming down. So, I want to find out what would the temperature be at 480 seconds from the starting point where it was at 1200 degree Kelvin. I want to find out this temperature. That is the y i I want to find out ,given the slope of this differential equation.

So, assuming a step size of each to be 240, if I take half of this then 240 then d theta d t, you can compute that, here. So, my formula will be theta i plus k 1 plus 2 k 2 plus 2 k 3 plus k 4 divided by 6 times h; that is the Runge Kutta formula. So, that is what I want to find out .

(Refer Slide Time: 22:59)

So step 1 - you see, I am not going into all these calculations, but I am finding out the value of k 1. I am finding out the value of k 2 using this value of k 1, k 1 is being used here and I find out the value of k 2. Again the value of k 2 is being used here, I find out the value of k 3 and k 3 is being used here I find out the value of k 4, I find out those values manually

.(Refer Slide Time: 23:36)

**Solution Cont**

$$\begin{aligned}
 \theta_1 &= \theta_0 + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4)h \\
 &= 1200 + \frac{1}{6} (-4.5579 + 2(-0.38347) + 2(-3.8954) + (0.069750))240 \\
 &= 1200 + \frac{1}{6} (-2.1848)240 \\
 &= 675.65K
 \end{aligned}$$

$\theta_1$  is the approximate temperature at  
 $t = t_1 = t_0 + h = 0 + 240 = 240$

$\theta(240) \approx \theta_1 = 675.65K$

<http://numericalmethodsguy.uic.edu>



So, the solution therefore, the theta 0 was 1200; initial condition and here I put in the values of this times h; h was 240. So, I have taken it at the midpoint. So, I find that the temperature that would be would be 675.65; that is the approximate temperature at 240. So, this will be the value at 240. Next I have to find out at 480. So, what would I do?

(Refer Slide Time: 24:18)

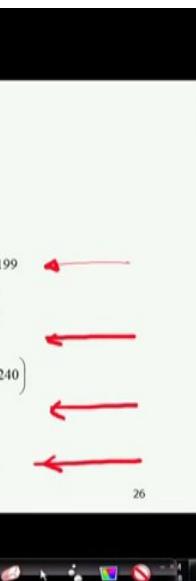
**Solution Cont**

**Step 2:**  $i = 1, t_i = 240, \theta_i = 675.65K$

$$\begin{aligned}
 k_1 &= f(t_1, \theta_1) = f(240, 675.65) = -2.2067 \times 10^{-12} (675.65^4 - 81 \times 10^8) = -0.44199 \\
 k_2 &= f\left(t_1 + \frac{1}{2}h, \theta_1 + \frac{1}{2}k_1h\right) = f\left(240 + \frac{1}{2}(240), 675.65 + \frac{1}{2}(-0.44199)240\right) \\
 &= f(360, 622.61) = -2.2067 \times 10^{-12} (622.61^4 - 81 \times 10^8) = -0.31372 \\
 k_3 &= f\left(t_1 + \frac{1}{2}h, \theta_1 + \frac{1}{2}k_2h\right) = f\left(240 + \frac{1}{2}(240), 675.65 + \frac{1}{2}(-0.31372)240\right) \\
 &= f(360, 638.00) = 2.2067 \times 10^{-12} (638.00^4 - 81 \times 10^8) = -0.34775 \\
 k_4 &= f(t_1 + h, \theta_1 + k_3h) = f(240 + (240), 675.65 + (-0.34775)240) \\
 &= f(480, 592.19) = 2.2067 \times 10^{-12} (592.19^4 - 81 \times 10^8) = -0.25351
 \end{aligned}$$

<http://numericalmethodsguy.uic.edu>

26



Step 2; I have taken had half point, I found out now my initial value theta 0 is changing.

Now, again do the same thing - find out k 1, find out k 2, find out k 3, find out k 4. Now with this initial condition (using the same function) what will be h; h will be again 240 because I have to find it out at 480 degree temperature.

(Refer Slide Time: 24:55)

**Solution Cont**

$$\begin{aligned}
 \theta_2 &= \theta_1 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h \\
 &= 675.65 + \frac{1}{6}(-0.44199 + 2(-0.31372) + 2(-0.34775) + (-0.25351))240 \\
 &= 675.65 + \frac{1}{6}(-2.0184)240 \\
 &= 594.91K \quad \text{←}
 \end{aligned}$$

$\theta_2$  is the approximate temperature at

$$\begin{aligned}
 t_2 &= t_1 + h = 240 + 240 = 480 \\
 \theta(480) &\approx \theta_2 = 594.91K
 \end{aligned}$$

<http://numericalmethodsganguly.ust.edu>

The video player shows a man in a blue shirt speaking. The toolbar below the video includes icons for back, forward, and search.

So, now, I find out that using theta on is now 675.65 that is approximate value that I got earlier and I compute this. I find out using the same Runge Kutta method that this is the approximate temperature at at 480 seconds which is 594.91 degree Kelvin . This is how we apply Runge Kutta method and it's very useful.

(Refer Slide Time: 25:30)

## Solution Cont

The exact solution of the ordinary differential equation is given by the solution of a non-linear equation as

$$0.92593 \ln \frac{\theta - 300}{\theta + 300} - 1.8519 \tan^{-1}(0.00333 \theta) = -0.22067 \times 10^{-3} t - 2.9282$$

The solution to this nonlinear equation at  $t=480$  seconds is

$$\theta(480) = 647.57 K$$

<http://numericalmethods.eng.usf.edu>

28

So, the exact solution of the differential equation is 647.57, if I solve it now, we got it, it was how much 594.91 and 594.91.

So, it was not very far. It around 50 degree Kelvin, that is certainly an approximation.

(Refer Slide Time: 25:55)

```
#include<stdio.h>
#include<math.h>
float f(float x, float y);
int main()
{
    float x0, y0, k1, k2, k3, k4, k, y, h, xn;
    printf("Enter x0, y0, xn, h:");
    scanf("%f %f %f %f", &x0, &y0, &xn, &h);
    x=x0;
    y=y0;
    printf("\n\nX(t)\tY(t)\n");
    while(x<xn)
    {
        k1=f(x,y);
        k2=f((x+h/2.0),(y+k1*h/2.0));
        k3=f((x+h/2.0),(y+k2*h/2.0));
        k4=f((x+h),(y+k3*h));
        k=((m1+2*m2+2*m3+4*m4)/6);
        y=y+k*h;
        x=x+h;
        printf("%f\t%f\n", x, y);
    }
}
```

$$y_{i+1} = y_i + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4)h$$

$$k_1 = f(x_i, y_i)$$

$$k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right)$$

$$k_3 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h\right)$$

$$k_4 = f(x_i + h, y_i + k_3 h)$$

$$y = y + k * h$$

So, now we have understood; what is Runge Kutta method.

Now, I want to write a c program for it. Again you might find the approach to be mathematically very novel maybe intricate, you may find some initial difficulty in understanding, but I am sure you will understand it very fast, but you will see that encoding it as a c program is or as a program is really simple. Now you are learning C, in future will be using MATLAB and other things, you will be able to solve it very easily, I am showing you the c solution here. On this side, I have kept what we learnt till now,  $y_i + 1$ ; that means, in our case the temperature at 480 degree, it is temperature at 240 degree plus this and  $k_1$  is  $f(x, y)$ ,  $k_2$  is  $f(x + h, y + h)$ , this one  $f(x + h/2, y + h/2)$  etc.

So, now let us look at the program we have got the math function and everything ready. Now somehow this function has to be written. Now this function, here its being shown as a very simple function, it can be any function, the differential equation function, the earlier functions that we have shown or the thelog function which you will have to write. So, here is an example of a simple function;  $x - y$  by  $x + y$ . With the  $dy/dx$  given, that will come in this function. So, now, you see here I did how many times? And  $h$  value,  $x_0$   $x_1$  value, all these things I read. Now the key things comes here, this is the implementation of the Runge Kutta method. So, very simple, you see, I am computing  $k_1$ ,  $k_1$  is  $f(x_0, y_0)$ .

So, I am coming to this function computing  $x_0$   $y_0$ , then I am going back computing  $k_2$  for  $x_0 + h$  by 2.  $h$  has been read . This  $h$  has been read, then  $y_0 + m_1$ , I am sorry, here it should be  $k_1$ , this would be  $k_2$ , all these  $m$ 's you should read it as  $k$ . This is  $k_3$ , this is  $k_2$  times  $h$ . So, actually we are computing this thing straight way and then  $y$  is assigned  $y + k_3$ .

So, it will be this statement will be  $y + k_3$  and  $x + h$ . I am incrementing  $x$  and going on. I am doing it for 2 intervals till I come over here. So, I ultimately come to this print f and I print the value of  $y$  for a particular  $x$ ; that is the Runge Kutta. So, this is in a straight way amenable to some c program and for each of these  $f$ 's, you are calling the function every time.,

So, this is the Runge Kutta method for solving a differential equation. So, I will encourage you to look at other methods of integration like Simpsons one third rule which is a very popular method. This is known as Runge Kutta fourth order method because we are taking 4 terms, here, but this works very well for most of the engineering problems. So, I will encourage you to write programs on this and later on hence forth, we will move to another interesting aspect that is known as recursion a new style of programming which will take up in the next lecture.

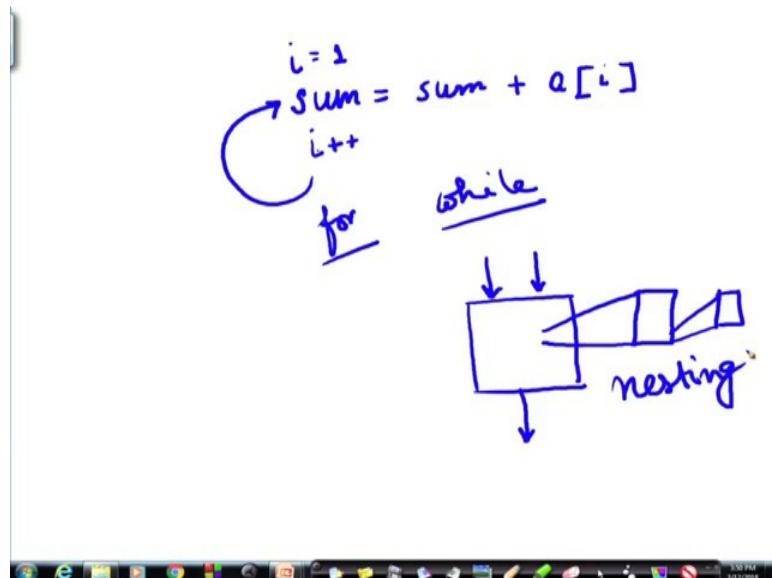
Thank you.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 53**  
**Recursion**

Today, we will be discussing on a new concept of programming which is very interesting, but possibly not very much familiar to you. That is known as Recursion. You know repetitions; how they are implemented in C programming. Whenever I want to do a particular work.

(Refer Slide Time: 00:55)

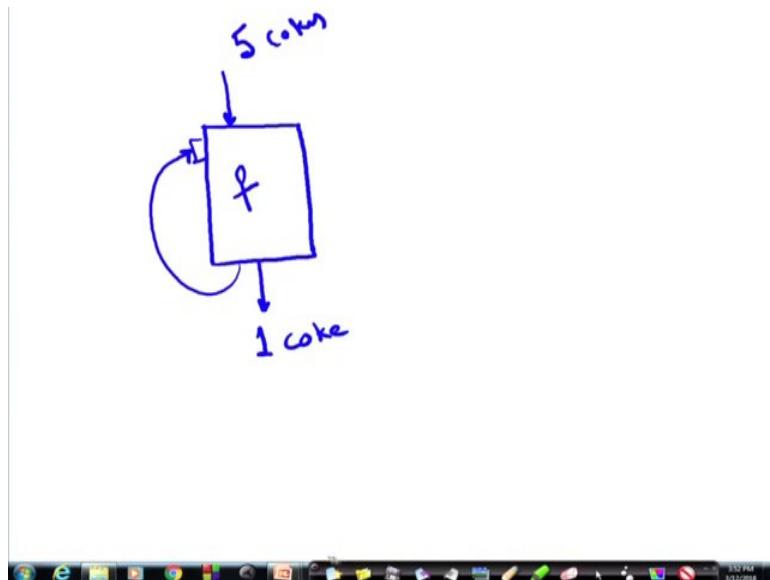


For example I want to add 10 numbers, then I add the sum plus sum plus sum a i with i equal to 1 and then I repeat it i plus plus and I go on repeating it and this repetition is done in the form of a for loop or while loop.

Now, recursion is a different way of doing this. When a function calls itself; that means, a function; a particular function will have some inputs and we will deliver one output. Now you know nesting of functions that in order to achieve this objective of taking these inputs and

delivering these outputs, a function could call other functions from another function and then ultimately, return to this function which could be done and this was known as nesting.

(Refer Slide Time: 02:24)

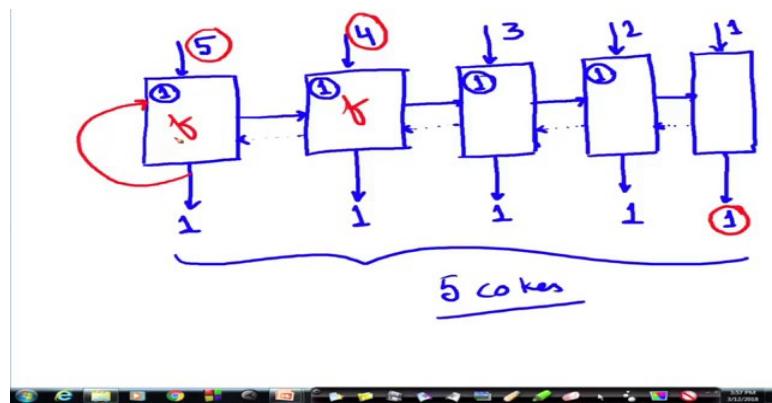


But recursion is a little different, here there is a function which is supposed to be to deliver some output. Now in order to do this, it is actually repeating itself; that means, let me give you an example. Suppose, I have a machine which can generate 1 apple, it can generate 1 apple, produce 1 apple or maybe let us say, it produces a vending machine which can deliver 1 coke, alright, 1 piece of coke or 1 can of coke.

So, you put in some input some commands and it gives you 1 coke. Now, it has been asked to deliver 5 cokes. So, how can you deliver 5 cokes? It can deliver only 1 coke at a time. So, as we know in our knowledge of iteration, this is some function; function is delivering coke. So, it can be repeated, this switch can be pressed, there is a switch, it can be pressed 5 times. So, first, 1 coke comes out, second coke comes out so; that means, this is being repeated 5 times.

So, that is the conventional way of iterating, repeating the same thing in order to get the 5 cokes. Let us try to do it in another way, let us assume that this machine can deliver 1 coke at a time, but it can also clone itself. So, let us see how it looks like.

(Refer Slide Time: 04:33)



So, I need 5 cokes but I can deliver only 1. So, I can deliver only 1 coke but I will not deliver it till I am ensured that all the 5 cokes are deliverable.

So, what it does? It clones itself makes another one copy of itself say and activates - this using which I can deliver one coke. So, I asked him to give a 4 cokes. Now this one can also deliver only 1 coke and finds that ok, I cannot deliver everything. So, I keep 1 coke ready, but I cloned myself and another copy, I activate him - this machine and say please deliver 3 cokes, but it cannot also deliver 3 cokes, it can deliver only 1. So, nothing is being delivered, but only kept noted. So, it puts somebody another clone please deliver 2 cokes, I have got 1.

So, it is also not delivering because it is not being able to satisfy the request of 2 cokes which has been opposed by this machine. So, it now comes to another clone. So, asks this clone please

deliver 1 coke. Now as you know this machine can deliver 1 coke. So, now, it delivers 1 coke and tells ok, I have delivered. Now this one, then since it has delivered, it delivers the other coke which it could do , it also tells its requestor, its earlier version that I have done my thing, you do yours. So, what will it do?Remember it can only deliver one coke at a time.

So, it will deliver another coke, it passes on to its caller or its generator that I have delivered. So, it delivers the other coke and ultimately this one also knows that it's child has delivered the coke. So, ultimately we get 5 cokes. This was one this is one way in which if each of this function, each of these blocks, although they are the same function is being activated with different requests and they are waiting till the request can be fulfilled.

But in the meanwhile it is passing on the request to another one and as soon as this could fulfil it, it passes it on and then it goes back. In the case of iteration what would have happened? 5 cokes would be generated by calling the same function. So, if I call each of these as functions, then there is no difference between these function, the only difference is the value with which it is being called and this process is successful because ultimately there will be a situation when this function of this machine will be able to deliver what it has been asked to- deliver 1 coke. So, that will be done.

Therefore, we can go back and have everybody else deliver the same thing, this is the principle of recursion. I just used it as a fun example, but let us now come to a little more serious look at this.

(Refer Slide Time: 09:26)

## Recursion

- A process by which a function calls itself repeatedly.

- Either directly.

- X calls X.

- Or cyclically in a chain.

- X calls Y, and Y calls X.

$$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$$

4!

$$5! = 5 \cdot 4! = 5 \cdot 4 \cdot 3!$$

= 5 · 4 · 3 · 2!

- Used for repetitive computations in which each action is stated in terms of a previous result.

- $\text{fact}(n) = n * \text{fact}(n-1)$

                 =                 

$$= 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$$

55



So, it is a process by which a function calls itself repeatedly, either directly like X is calling X or cyclically in a chain like X is calling Y, Y is calling X basically used for repetitive computations. So, the best thing is you look at this example. All of us know that factorial 5 is nothing, but 5, 4 times 4, times 3, times 2, times 1.

Now, you see the same thing, I can say; what is this part? 4 multiplied by 3 multiplied by 2 multiplied by 1; this is factorial 4. So, you see I am expressing factorial 5, factorial function in terms of itself. 5 times factorial 4 and factorial 4 can again be expressed as 4 times factorial 3 and factorial 3 can be expressed as 3 times factorial 2 and factorial 2 and be expressed in terms of factorial 1. So, 5, 4, 3, 2 and factorial 1 is the end. So, that is 1. So, I am certainly getting the factorial there without any further expansion.

So, we can write in general factorial of n is n times factorial of n minus 1 and factorial of n minus 1 will be n minus 1 into factorial of n minus 2, factorial of n minus 2 will be n minus 2 into factorial of n minus 3. In that way it will go on, but it will be successful only if there is a terminating point where which we often call the basis condition. I hope you have understood this.

(Refer Slide Time: 12:00)

## Contd.

- For a problem to be written in recursive form, two conditions are to be satisfied:
  - It should be possible to express the problem in recursive form.
  - The problem statement must include a stopping condition

$$\text{fact}(n) = n * \text{fact}(n-1)$$

56

So, two conditions are to be satisfied , in order for us to write a recursive formula. One is - it should be possible to express the problem in the recursive form just like factorial n is n times factorial n minus 1 and also there is another point that is the problem statement must include a stopping condition, what was my stopping condition in the case of factorial? Factorial 1 is 1.

So, that was the stopping condition; what was the stopping condition in the example of delivering the coke? When the machine was asked to deliver only 1 coke, then it can complete. So, that is a stopping condition and then we go back.

(Refer Slide Time: 13:10)

## Contd.

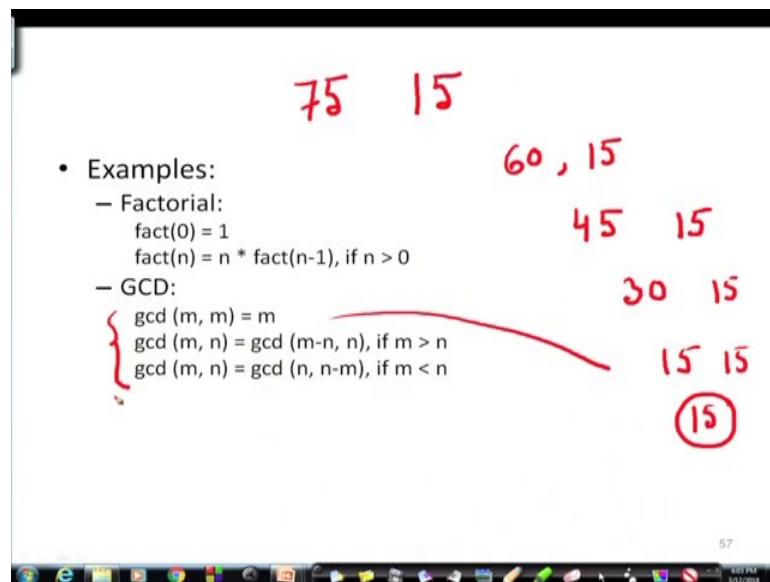
- For a problem to be written in recursive form, two conditions are to be satisfied:
  - It should be possible to express the problem in recursive form.
  - The problem statement must include a stopping condition

$$\begin{aligned} \text{fact}(n) &= 1, && \text{if } n = 0 \\ &= n * \text{fact}(n-1), && \text{if } n > 0 \end{aligned}$$

56

So, the stopping factorial n is 1 if n is equal to 0 or n equal to 1, otherwise, it is n into factorial n minus 1 if n is greater than 0. So, ultimately, it will go on and ultimately, it will conclude.

(Refer Slide Time: 13:42)



Another example; greatest common divisor, we can express that in a recursive form, it is very interesting, you can look at it that the greatest common divisor of the same number is itself that is the key to the logic. GCD of m and m is m and GCD of m and n is GCD of m minus n and n or the other way. So, for example, GCD of 15 and 5 also 75 and 15 will be GCD of 60 and 15, GCD of 60 and 15 would be for GCD of 45 and 15, GCD of 45, 15 would be GCD of 30 and 15, GCD of 30 and 15 will be GCD of 15 and 15. Now I have got the stopping condition that GCD of the same number will be 15.

So, my result will be 15 is it clear? So, that is a recursive definition of GCD. So, most of the interesting problems can be expressed in the form of in the recursive form and that helps in writing a very second and tight code.

(Refer Slide Time: 15:26)

- Examples:
- Factorial:  
 $\text{fact}(0) = 1$   
 $\text{fact}(n) = n * \text{fact}(n-1)$ , if  $n > 0$
- GCD:  
 $\text{gcd}(m, m) = m$   
 $\text{gcd}(m, n) = \text{gcd}(m-n, n)$ , if  $m > n$   
 $\text{gcd}(m, n) = \text{gcd}(n, n-m)$ , if  $m < n$
- Fibonacci series (1,1,2,3,5,8,13,21,...)

$f(n) = f(n-1) + f(n-2)$

$f(n-2) = f(n-3) + f(n-4)$

57

Here is another example of recursion Fibonacci series - a series like this one, then one, then 2, 2 is what the sum of the previous 2 elements 1 plus 1 and 3, what is 3? 3 is the sum of the previous two ; 2 and 1, then 5, what is 5? 5 is the sum of the previous two.

Then 8; what is 8? 8 is the sum of the previous two. 5 and 3, then 13, what is 13? 13 is the sum of the previous two, 8 and 5, then 21; what is 21? 21 is the sum of the previous two, 13 and 8, you can see how nice this pattern is. Can you think of how we can write it in a recursive form, how we can express it in a recursive form? If you think a little bit, it will be very easy, it will be something like Fibonacci number of n is Fibonacci of n minus 1 plus Fibonacci n minus 2.

So,  $f(n)$  - this one, is the sum of the earlier 2 Fibonacci sequences. Now  $f(n)$  minus 2 will be what will be  $f(n)$  minus 3 plus  $f(n)$  minus 4. Similarly,  $f(n)$  minus 1 will be  $f(n)$  minus 2 plus  $f(n)$  minus 3 in that it will go on, but when will it stop? The stopping condition is that Fibonacci of 1 is one. We will see that.

So, if we try to express it in the recursive form, it turns out to be Fibonacci of 0 is 1.

(Refer Slide Time: 17:48)

The slide contains the following text:

- Examples:
  - Factorial:
$$\text{fact}(0) = 1$$
$$\text{fact}(n) = n * \text{fact}(n-1), \text{ if } n > 0$$
  - GCD:
$$\text{gcd}(m, m) = m$$
$$\text{gcd}(m, n) = \text{gcd}(m-n, n), \text{ if } m > n$$
$$\text{gcd}(m, n) = \text{gcd}(n, n-m), \text{ if } m < n$$
  - Fibonacci series (1,1,2,3,5,8,13,21,...)
$$\text{fib}(0) = 1$$
$$\text{fib}(1) = 1$$
$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2), \text{ if } n > 1$$

Handwritten red annotations on the right side of the slide show the recursive expansion of  $\text{fib}(5)$ :

$$\begin{aligned} \text{fib}(5) &= \text{fib}(4) + \text{fib}(3) \\ &= (\text{fib}(3) + \text{fib}(2)) + (\text{fib}(2) + \text{fib}(1)) \\ &= (\text{fib}(2) + \text{fib}(1)) + (\text{fib}(1) + \text{fib}(0)) \\ &= 1 + 1 \\ &= 2 \end{aligned}$$

That means the first element is one, Fibonacci of 1 is 1. There is a second element, then Fibonacci of  $n$  is Fibonacci of  $n$  minus 1 plus Fibonacci of  $n$  minus 2. So, now, suppose I give you Fibonacci of 5, how do you write that? It will be Fibonacci of 5, it will be Fibonacci of 4 plus Fibonacci of 3 and Fibonacci of 3 will be Fibonacci of 2 plus Fibonacci of 1 and Fibonacci of 1 we know is 1.

So, I know this and then Fibonacci of 2 will be Fibonacci of 1 plus Fibonacci of 0, I know that this is 1. So, I get this; this is done. Similarly, once this is done I got this number. So, what would that be? This will be 2 and so third one will be 2, then Fibonacci of 4 will be 2 plus 3, 5; in that way it will go on. Now we see; how we can write a function, how we can express this Fibonacci or this factorial, the recursive expression in the form of a function.

(Refer Slide Time: 19:36)

Example 1 :: Factorial

```
long int fact(n)
int n;
{
    if (n == 0)
        return (1);
    else
        return (n * fact(n-1));
}
```

Let us see here, I am writing this, you had seen earlier functions written for factorial. So, you can see factorial  $n$  is if  $n$  equal to 0, return 1, otherwise return  $n$  times factorial  $n$  minus 1. So, what will happen? How will this be executed? What will it return? While returning, it will again call this function, again, this function will start in the same way just by replacing  $n$  with  $n$  minus 1 and here, it will come out with  $n$  minus 2. So, it will again be called with  $n$  minus 2 and so on and so forth. It will go on. Now how is that executed?

(Refer Slide Time: 20:35)

## Mechanism of Execution

- When a recursive program is executed, the recursive function calls are not executed immediately.
  - They are kept aside (on a stack) until the stopping condition is encountered.
  - The function calls are then executed in reverse order.

The function as I said is not executed immediately, when I asked that coke machine to deliver 5 cokes, it did not deliver immediately, it could deliver 1 coke, but held it back, but created another machine to deliver n minus 1 cokes and that machine held it back and generated another machine to deliver n minus 2 cokes. In this way, it went on. They are kept aside on a stack one after another until the stopping condition is encountered. So, it remembered that I have to deliver one.

So, if you look at this, I do not know whether that will visible or not. Here here you see everybody remembered that I have to deliver, but they did not deliver, when the stopping condition was met after that this back chain started. So, they are kept aside, but not delivered immediately. Then the function calls are executed in reverse order. Again you can see that they are executed in reverse order in order to get the solution.

(Refer Slide Time: 22:14)

### Example :: Calculating fact(4)

— First, the function calls will be processed:

```
fact(4) = 4 * fact(3)
fact(3) = 3 * fact(2)
fact(2) = 2 * fact(1)
fact(1) = 1 * fact(0)
```

— The actual values return in the reverse order:

```
fact(0) = 1
fact(1) = 1 * 1 = 1
fact(2) = 2 * 1 = 2
fact(3) = 3 * 2 = 6
fact(4) = 4 * 6 = 24
```

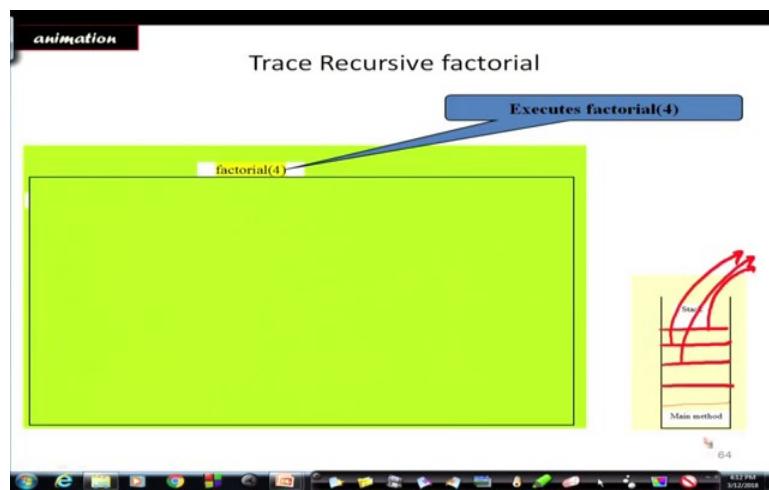


Say I am calculating factorial 4, first the function calls will be processed. Factorial 4 is factorial 4 times factorial 3, then factorial 3 is 3 times factorial 2, factorial 2 is 2 times factorial 1; factorial 1 is 1 times factorial 0 and factorial 0 is 1.

Now, the actual values will return in the reverse order; 1 into 1 is 1. So, factorial 1 is complete. So, 1 into 2 is 2 that goes here, fact 2 is 2, in that way, it goes on. So, it goes back in this

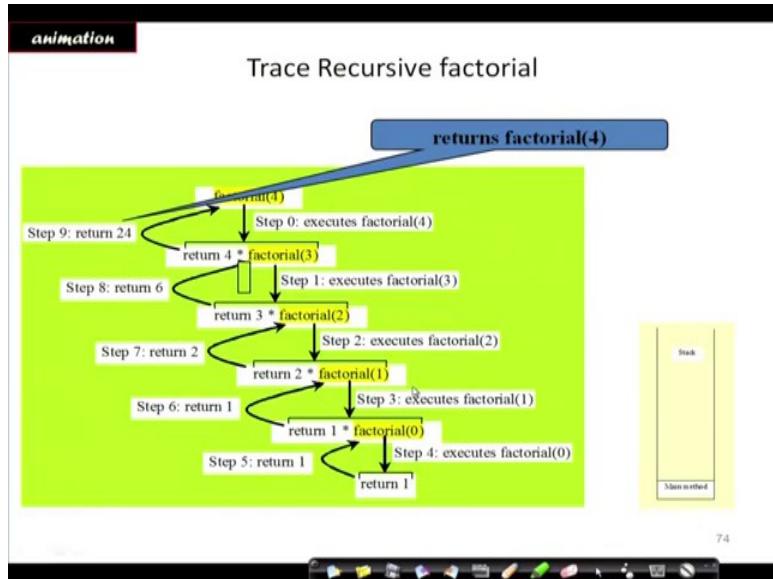
direction ultimately, we get the result from here. So, the actual values return in reverse order. So, factorial 0 is 1, factorial 1 is 1 times 1 is 1, now the reverse order is being done 2 into 1 is 2, 3 into 2 is 6 and 4 into 6, 6; 24 in that way, it is being computed.

(Refer Slide Time: 23:36)



So, here is a stack. Stack is a data structure, stack is a way of storing data where we stored the data just where whatever comes in first; obviously, goes out last because if I put something here and above that I put something just like a stack of books, you will have to take it out in the other way, this one will come out first then this one will come out then this one will come out. So, let us see how it works.

(Refer Slide Time: 24:14)



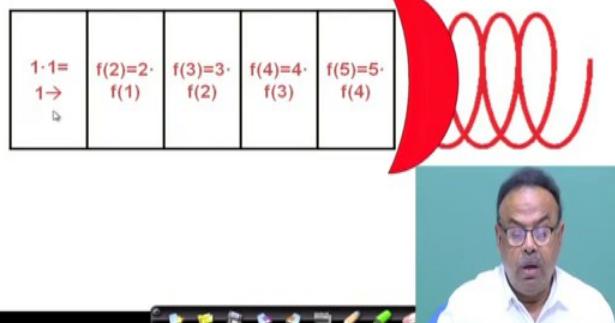
So, factorial 4, factor step 0 executes factorial 4. Now return 4 times factorial 3, you see here is the recursive call, it is calling itself return 4 times factorial 3, what is factorial 3, oh you do not know; what is factorial 3? Return 3 times factorial true 2. Oh you do not know; what is factorial clue 2 ? Then return 2 into factorial 1 or you still do not know; what is factorial 1? Return 1 into factorial 0 and so, now, you know factorial 0 is 1. So, you get 1 and now you could not answer these questions earlier so I have broken it down and given you an easier solution.

And so, now, you go back here and you return one into one then you go back here, return 1 into 2, return 2 and you go back here . This is what is meant by recursion if we redo it if we.

(Refer Slide Time: 25:46)

## Recursive Algorithms Computer Implementation

```
long factorial(int n){  
    if (n<=0) return 1;  
    return n*factorial(n-1);  
}
```



So, here is the computer implementation of that factorial if  $n$  is less than 0, return 1, return  $n$  times factorial  $n$  minus 1.

So, computes 5. So, it is again recursively expressing itself  $f(4)$  is being expressed in terms of  $f(3)$ , everything is being expressed,  $f$  is being expressed in terms of  $f$ ,  $f(3)$  is being expressed in terms of  $f(2)$ ,  $f(2)$  is being expressed in terms of  $f(1)$ ,  $f(1)$  is being expressed in terms of  $f(0)$  and I know what  $f(0)$  is. So,  $f(0)$  is 1. So, I go back and I know that now  $f(0)$  is 1.

Therefore  $f(1)$  is 1 into 1 and in this way, I go back. Now, you see it had expanded in this way. Now it is shrinking 1 into 1 is 1. So, now, I know factorial 1. So, this will shrink; now I know what is factorial 2 that is 2. So, this will shrink 3 into 2; 6, it is shrinking coming here - 24 and then here it is 120; that is how as if in a spring it got expanded and then it contracted back, this is recursion return it ultimately returns factorial 5 to be 120.

(Refer Slide Time: 27:10)

## Recursive Algorithms Computer Implementation

```
long factorial(int n){  
    if (n<=0) return 1;  
    return n*factorial(n-1);  
}
```

87

(Refer Slide Time: 27:10)

### Another Example :: Fibonacci number

- Fibonacci number  $f(n)$  can be defined as:  
$$f(0) = 0$$
  
$$f(1) = 1$$
  
$$f(n) = f(n-1) + f(n-2), \text{ if } n > 1$$
- The successive Fibonacci numbers are:  
0, 1, 1, 2, 3, 5, 8, 13, 21, ....
- Function definition:

88

Another example will be Fibonacci number. We will explain it in the next lecture.

**Problem Solving Through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 54**  
**Recursion (Contd.)**

So, we are looking at recursion and that is a new style of programming where we can express a particular function in terms of itself like I can express factorial n in terms of factorial n minus 1 ok. Another very common example and easy example of recursion is Fibonacci numbers, we have already told you; what Fibonacci sequence is, the Fibonacci sequence can be expressed as f 0, the 0th Fibonacci number is 0. The next one is also 1.

(Refer Slide Time: 00:57)

Another Example :: Fibonacci number

- Fibonacci number  $f(n)$  can be defined as:  
$$f(0) = 0 \leftarrow$$
$$f(1) = 1 \leftarrow$$
$$f(n) = f(n-1) + f(n-2), \text{ if } n > 1$$
- The successive Fibonacci numbers are:  
0, 1, 1, 2, 3, 5, 8, 13, 21, .....
- Function definition  

```
int f(int n)
{
    if (n < 2) return (n);
    else return (f(n-1) + f(n-2));
}
```



And henceforth, all other ones are sum of the previous two; so, 0, 1, 1, 2, 3, 5, 8, 13, 21, so and so forth. Therefore, we should be able to define it in terms of a recursive function because you can see this function f and this function f are the same, only variations are in these parameters. I am expressing the same function in terms of these parameters. So, the function definition will be is very simple f; int end some integer, if n is less than 2, then return n if n is less than 2 if it is 0 then 0.

If it is 1, then it is 1, otherwise what did you return? Return  $f_n$  minus 1 plus  $f_n$  minus 2, sum of the previous 2 Fibonacci numbers. Now this is interesting because again if you see how this will be computed, it will be first expanding what are the things I have to compute and when it meets stopping condition then it starts collecting back and comes back.

(Refer Slide Time: 02:31)

## Tracing Execution

- How many times the function is called when evaluating  $f(4)$  ?

```

graph TD
    f4[f(4)] --> f3[f(3)]
    f4 --> f2[f(2)]
    f3 --> f2_1[f(2)]
    f3 --> f1_1[f(1)]
    f2 --> f1_2[f(1)]
    f2 --> f0_1[f(0)]
    f1_1 --> f0_2[f(0)]
    f1_2 --> f0_3[f(0)]
  
```

**9 times**

If I want to compute Fibonacci of 4, how many times will that function be called? Let us look at the expansion of this. So, how will it happen? I want to compute Fibonacci of 4 which is Fibonacci of 3, Fibonacci of 2, these 2 should be added. So, I have not yet have found out anything, I am just decomposing the problem .

There is a very very important concept that in order to solve the problem, I want to decompose it into smaller sub problems. For  $f_4$ , I have to solve it by solving  $f_3$  and  $f_2$ . Now for solving  $f_3$ , I have to solve  $f_2$  and  $f_1$ , I further decomposed it. Now for solving  $f_2$  I have to solve  $f_1$  and  $f_0$ . Even now the entire thing has not been broken down,  $f_2$ , for that I have to solve  $f_1$  and  $f_0$ . Now I have expanded the whole thing.

Now, I know that  $f_1$  is 1,  $f_0$  is 1. So,  $f_2$  will be 1 plus 1; 2, this is known, all these endpoints of this structure are known - 1 0, 1, 1, 0, 1. So, I go on adding them and ultimately, I will get  $f_4$ . Now it is in a way inefficient, because the same thing is being computed repeatedly, but to a

practiced programmer, it will make your programming much more easier, I mean less line lines of codes if you can express it in a much better way.

So, you can see here; how many times the function was called? 1, 2, 3, 4, 5, 6, 7, 8, 9 times; 9 times the same function was called in order to compute f 4.

(Refer Slide Time: 05:02)

The screenshot shows a presentation slide with a black header and footer bar. The main content area is white. At the top, it says "Example Codes: fibonacci()". Below that, there is a bullet point "- Code for the **fibonacci** function". Underneath the bullet point is the C-like pseudocode for the Fibonacci function:

```
long fibonacci( long n )
{
    if (n == 0 || n == 1) // base case
        return n;
    else
        return fibonacci(n - 1) +
               fibonacci( n - 2 );
}
```

In the bottom right corner of the slide area, there is a small number "90".

Now the stopping condition of the base condition is very very important. So, if n is 0 or n is 1, then I return 1, this is the base case, unless I reach at this point, I will not be able to compute the entire solution.

Return Fibonacci of n minus 1 plus Fibonacci of n minus 2 - that is the code for the Fibonacci number. Now whenever I will have too many calls in that case, I should avoid them as much as possible and. So, what is the difference?

(Refer Slide Time: 06:00)

## Recursion vs. Iteration

- Repetition
  - Iteration: explicit loop
  - Recursion: repeated function calls
- Termination
  - Iteration: loop condition fails
  - Recursion: base case recognized
- Both can have infinite loops
- Balance
  - Choice between performance (iteration) and good software engineering (recursion)

101

Between recursion and iteration? In iteration there is an explicit loop for  $i = 0, i < n$ . In the case of recursion, the base condition must be recognised whenever we are getting the base condition; factorial one or factorial 0 Fibonacci of 0 or Fibonacci of 1.

So, there is an explicit loop whereas, in case of recursion, it is repeated function calls. Termination iteration if the loop condition is no longer satisfied while this condition do, if that condition fails, then we come out of the loop. In the case of recursion, the base condition must be recognised whenever we are getting the base condition; factorial one or factorial 0 Fibonacci of 0 or Fibonacci of 1.

So, those are the base conditions, if wrongly programmed both can have infinite loop. So, the performance wise iteration often gives faster result, but it is a good software engineering practice to gradually get accustomed to recursion as you do more and more programming, you will see that you will be able to express the things in a much subtler way.

(Refer Slide Time: 07:29)

## Performance Tip

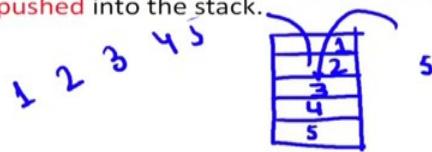
- Avoid using recursion in performance situations. Recursive calls take time and consume additional memory.

103

(Refer Slide Time: 07:37)

## How are function calls implemented?

- In general, during program execution
  - The system maintains a **stack** in memory.
    - **Stack** is a **last-in first-out** structure.
    - Two operations on stack, **push** and **pop**.
  - Whenever there is a function call, the **activation record** gets **pushed** into the stack.



104

So, whenever there is a performance issue try to avoid recursion, it will require additional memory also. There is a particular type of storage that is required in recursion that is known as stack. Stack is a last in first out type of structure.

Briefly, let me tell you; how this thing is done because stack is nothing, but a structure where we can push in data from one side - say I put say 5 first, I push. So, 5 comes here, then I push 4, then I do n minus 1, 3 is pushed, then 2 is pushed then n minus 1 again and 1 is pushed. Now when I take out the data, the data will come out as 1, then 2, then 3, then 4, then 5 in the reverse order . So, the 2 operations are push and pop.

Popping out from the stack and pushing inside the stack.

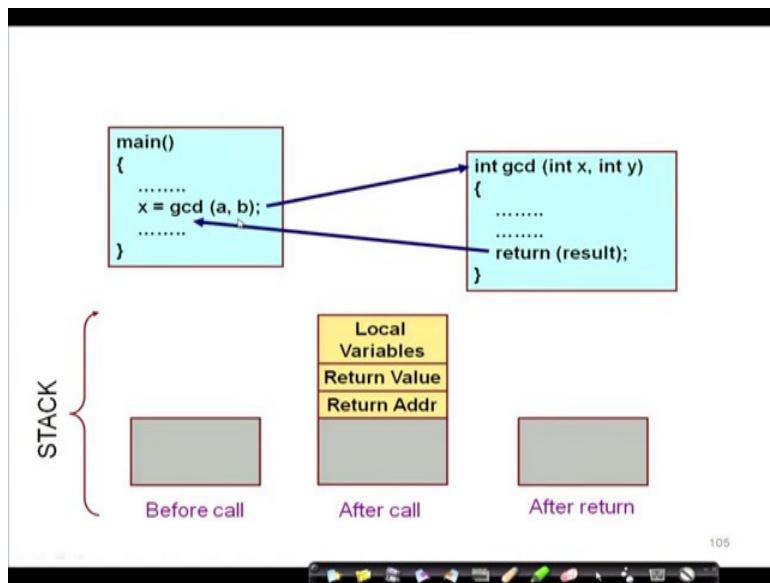
(Refer Slide Time: 08:52)

How are function calls implemented?

- In general, during program execution
  - The system maintains a **stack** in memory.
    - **Stack** is a **last-in first-out** structure.
    - Two operations on stack, **push** and **pop**.
  - Whenever there is a function call, the **activation record** gets **pushed** into the stack.
    - Activation record consists of the **return address** in the calling program, the **return value** from the function, and the **local variables** inside the function.
    - At the end of function call, the corresponding **activation record** gets **popped** out of the stack.

So, this stack data structure becomes very handy for implementation of function, recursive functions, we will show some examples.

(Refer Slide Time: 08:59)

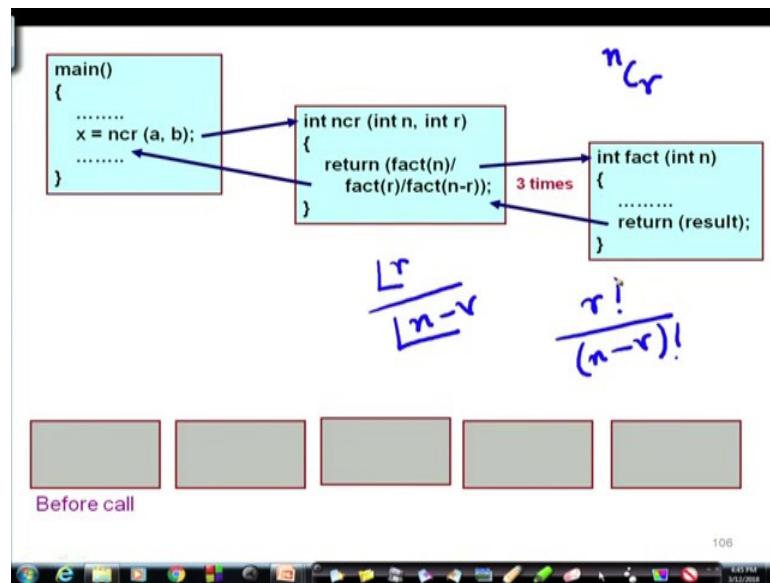


Like here for example, I want to compute the gcd of a b. Now typically what happens is I call this, gcd is computed, I return.

Now, here when I call something say; I was here, my program flow was here I went in. So when I went in all the local variables and everything here were stored and I had to remember where I will be returning back. So, all those things are stored in the stack and without a stack data structure, it is very difficult to implement recursion and for that matter any function call.

So, you see here is a function and so all those return addresses are stored, before call this stack was empty, now after its returning it is taking out from the top of the stack and again I come to know where I was.

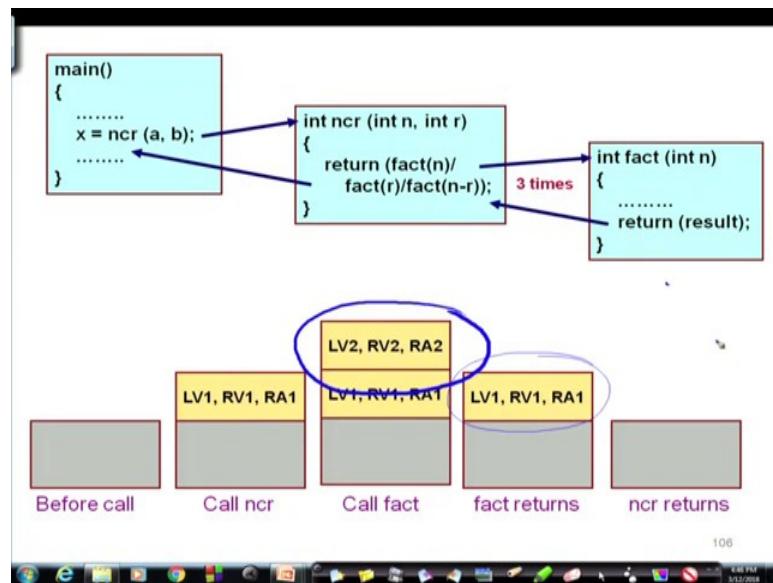
(Refer Slide Time: 10:09)



So, that is so, similarly you see here. N choose r a b; that means, n c r . n choose r, if I compute, then n choose r is factorial r divided by factorial of n minus r or some people write it in this way - factorial of n minus r. So, how did I do that? So, how can I implement it?

So, here n c r has been called from here factorial has been called and then where do I return? I have to come here, ultimately I have to return here. So, I should not lose the path.

(Refer Slide Time: 11:10)



So, what the stack does is when I make the call, the first call the local variables here will be stored here on the stack and again I make another call from here. So, local variables are there, I am calling fact, as I go in here, the local variables here has stacked up and then when I return this part will be taken out and passed on to this.

So as it returns this part is taken out and I am here, it can again continue and then it returns, when it returns here this part will be taken out .

(Refer Slide Time: 12:13)

## What happens for recursive calls?

- What we have seen ....
  - Activation record gets pushed into the stack when a function call is made.
  - Activation record is popped off the stack when the function returns.
- In recursion, a function calls itself.
  - Several function calls going on, with none of the function calls returning back.
    - Activation records are pushed onto the stack continuously.
    - Large stack space required.
    - Activation records keep popping off, when the termination condition of recursion is reached.

107

So, that is for normal function call, how the stack remembers where I should go back. In the case of recursive calls what happens what we have seen is activation record gets pushed into the stack when a function I will call is made. In recursion a function calls itself.

So, several function calls are going on with none of the calls getting back. So, all the activation records are collected. So, you need not delve into that too much.

(Refer Slide Time: 12:54)

- We shall illustrate the process by an example of computing factorial.
- Activation record looks like:



108

I will show it by an example of a computing factorial. So, an activation record is the local variables and the return value, what the function should return and where it should return.

(Refer Slide Time: 13:11)

### Example:: main() calls fact(3)

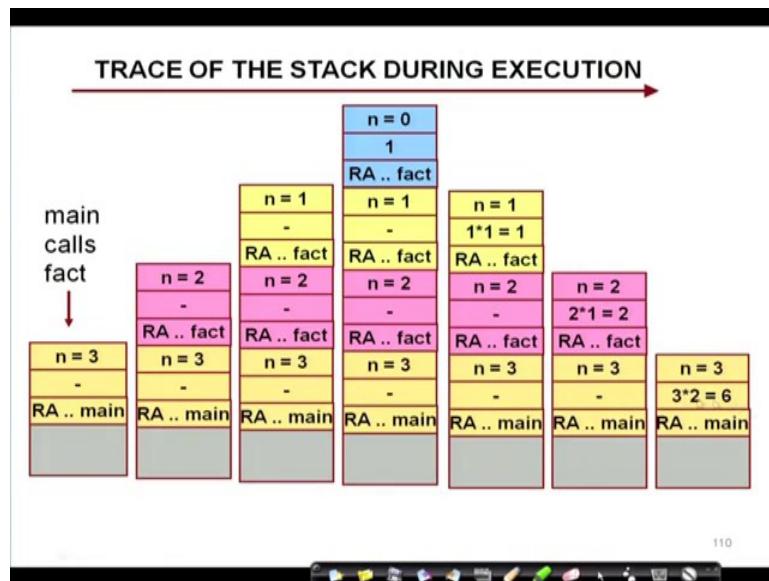
```
main()
{
    int n;
    n = 4;
    printf ("%d \n", fact(n) );
}
```

```
int fact (int n)
{
    if (n == 0)
        return (1);
    else
        return (n * fact(n-1));
}
```

109

So, that with that say the main function is calling fact 3 and here is a fact n, if n equal to 0 return one otherwise n times fact n minus 1.

(Refer Slide Time: 13:28)



So, main calls fact. So, when it calls the value n is equal to 3, so there is no return value, the return address is in the main I am remembering that. Next again fact is calling itself; so, now, fact is calling fact 3 is calling fact 2 and my return address is fact. So, you see it has been stacked up. Next fact 2 will call fact 1, so what is this? The stack is growing and its return is in fact.

Now, next time, it will be fact 0. Till now, there was no return value. Now the return value is 1 and return addresses fact. So, as I do that I return, then I have got a return value because now I have come to this point. So, 1 into 1 will be 1 and I am returning to fact, as I return the stack will contract and what is the return result? 2 times 1 that is 2, that is a factorial 2 and I am returning to fact. So, I return again. Now I am coming for last time in the fact with n equal to 3 that started here.

So, and the result is 6. Now I return to main. So, at every stage look at this, I know I remember from where I started and from where I am returning back, nothing is lost using this stack. So, stack is a very interesting data structure, that helps us in many ways, especially in implementing things like recursion.

(Refer Slide Time: 15:26)

The slide has a black header bar. Below it, the title "Do Yourself" is centered. To the left of the title is a bulleted list: "• Trace the activation records for the following version of Fibonacci sequence." To the right of the list is a code block in a light blue box:

```
#include <stdio.h>
int f (int n)
{
    int a, b;
    if (n < 2)      return (n);
    else {
        a = f(n-1);  X
        b = f(n-2);  Y
        return (a+b); }
}
main()
{
    printf("Fib(4) is: %d \n", f(4));
}
```

Below the code block, the word "main" is underlined with a red arrow pointing to the "main" function definition. To the right of the code block is a yellow box containing the following table:

Local Variables (n, a, b)
Return Value
Return Addr (either main, or X, or Y)

Handwritten annotations in blue and red are present: "X" and "Y" are written near the first two recursive calls to f(n-1) and f(n-2). Arrows point from these labels to the respective lines of code. A large blue arrow points from the "Return Value" row in the table to the "Return (a+b);" line in the code.

So, one assignment that I am leaving to you do it yourself, trace the activation record for the following version of Fibonacci. Please note down the code include stdio dot h, int f, f is the Fibonacci function, a and b if n is less than 2 return n ,if it is 0 then return 0.

If it is 1 return 1, otherwise a is fn minus 1, b is fn minus 2, I have done it in a different way, fn minus 1 and fn minus 2. So, if n minus 1 has to be solved separately and fn minus 2 should be solved separately, then we will return a plus b, then the main will print. So, just for fun you try to draw the activation record of this version of the function, please note it down, take some time and note down this function and you see on this side I have shown.

How the activation record will look like local variables, you can see n, a and b. Return value you have to keep whether it is in Fibonacci or in main, either in main or in x or y, . This is x and this is y, alright not these 2 ,these are not aligned properly.

Here is the return to the main, either return to main or to x or to y and what is the return value? Draw the activation record of this and then, we will see how much you could do it I am sure you will be able to do it. So, today we have learnt a new style of programming that is recursion and

also we discussed in the last class. So, recursion is a type of writing functions where the function calls itself and that makes many functions to be written much more subtle and that is a very good software engineering practice.

Although as a beginner, If you find difficulty in that; you need not bother too much about it, you have got iteration at your disposal and you can solve most of the problems with iteration, practically all the problems, you can do. May be in some cases, it may be a little more difficult to write, but ultimately it is will be possible. So, if you find difficulty with recursion, you can set it aside for the time being, but we have to discuss it because that is a very nice way of writing functions, we will continue with the concept of structures in the next lecture, a new thing will be introduced that is called structure.

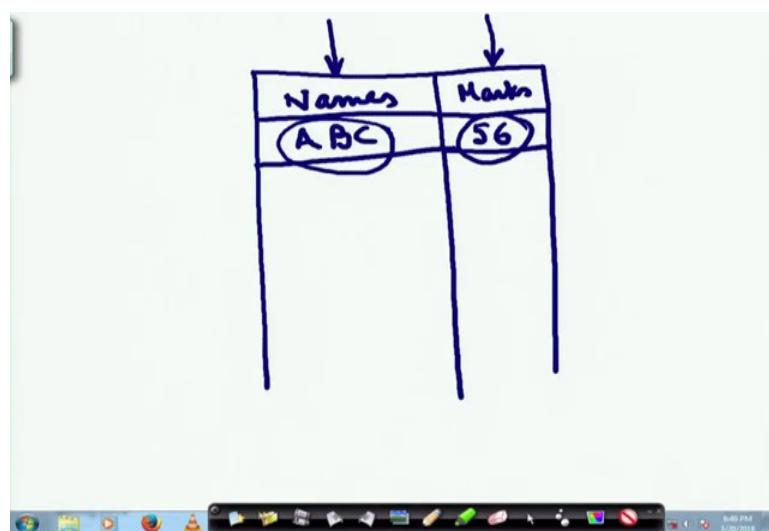
Thank you.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 55**  
**Structure**

Earlier, we had talked about storage of data in arrays. Now, what we found; what we mentioned in the case of arrays is that in an array we can store data only of a particular type; say an array of integers or an array of floats or an array of characters, but we could not mix different types in the same array. For example, we had faced the problem of representing the student's database where we will have the student's name as well as the student's marks.

(Refer Slide Time: 00:57)

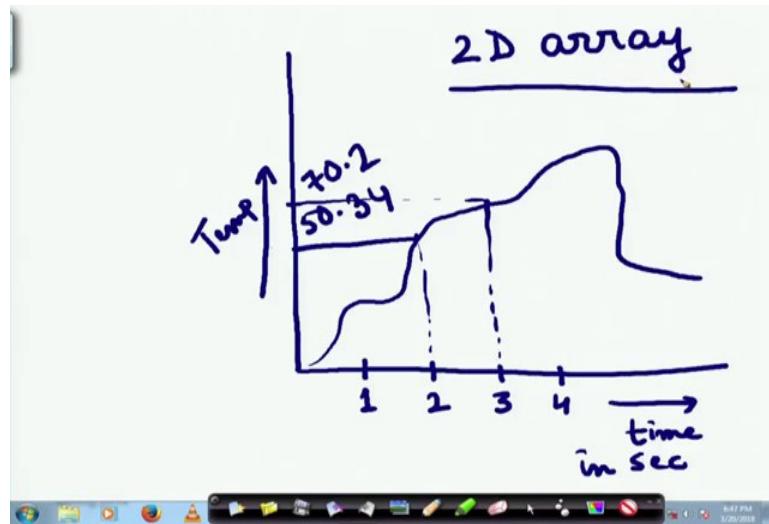


So, we needed something like this; actually that is what was desired; that will have the names here and the names will be nothing but an array of characters. So, here will be some names ABC and here will be the marks 56 A. Now this representation requires two arrays; one is the representing the names and the other is representing marks .

Now; however, as the diagram is showing here, we are trying to represent them in an unified structure in the same structure, but that is not possible in the case of an array, here we have got 2

different data types. This is an array of characters and this is an integer. For example, we can see that if I had stored something lets say .

(Refer Slide Time: 02:05)



For example, I want to store a graph where at every point; 1, 2, 3 seconds for example, this time in seconds and here I am measuring the temperature and if I have a graph which is something like this, I cannot say that at every point, the temperature will also be an integer therefore.

So, suppose here at point number 2, the temperature is 50.34, at point number 3, the temperature can be 70.2. So, if I had tried to represent that in the form of a 2D array, in that case, it wouldn't have been possible because a 2D array is also an array and therefore is of only one type. We have to declare a 2D array as an array of integer or as an array of characters or array of float.

(Refer Slide Time: 03:38)

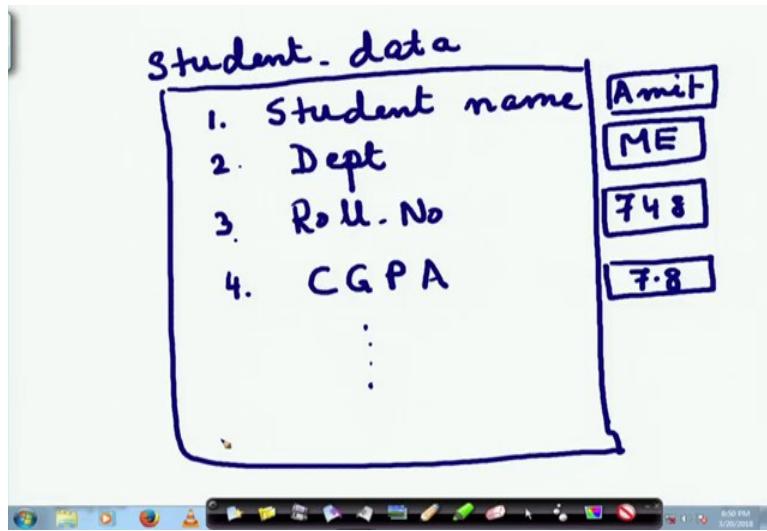
Time	Temp
1	50.6
2	70.3
3	

array  
X

So, I cannot represent that in a 2D array where one side say for example one column will be temperature and another column will be the time; its time and temperature. So, that wont possibly be executed in a 2 dimensional array because while time will be an integer (1, 2, 3) temperatures can be something different 50.6, 70.3, etc. So, an array is not possible.

So, our question is then what is the type of data structure; what is the type of arrangement by which I can represent data of different types together ? The answer to that is what we will be looking at today that is structure. Structures in C allows us to represent a combination of different data types.

(Refer Slide Time: 04:53)

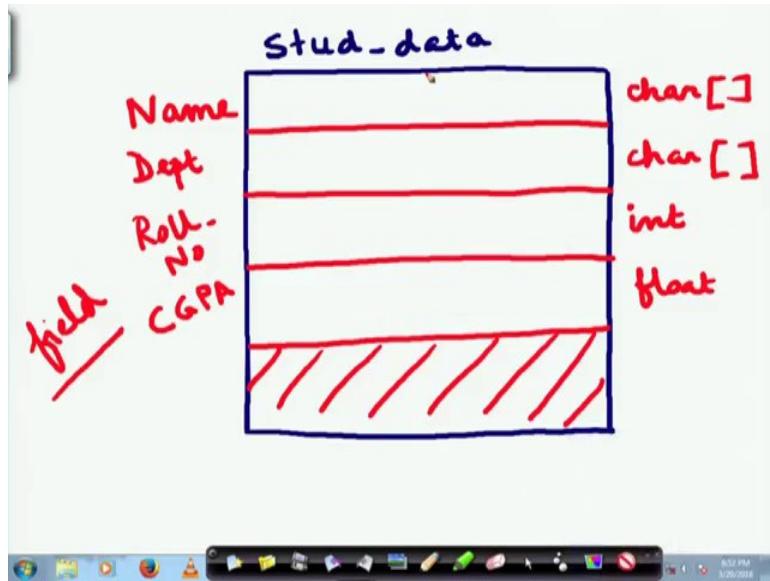


For example, let me give an example of a structure, say I want to define student data and student data will consist of the student name, lets say the department of the student, roll number of the student, may be the CGPA of the student, grade, point, average of the student, etc.

If I want to store them together as a common piece, the student data, then this entire thing has got components of different data types. For example, what will be student name say for example, Amith will be either a string or an array of character, department, somebody can say mechanical engineering that can also be a character string, whereas, roll number 748 will be an integer and CGPA say 7.8 will be a floating point number.

So, what we can see that in order to store the student data we have to have a mixture of different data types. So, a structure allows us to do exactly this where I can consider this student data as a structure. So, let me redraw it in a different way.

(Refer Slide Time: 06:44)



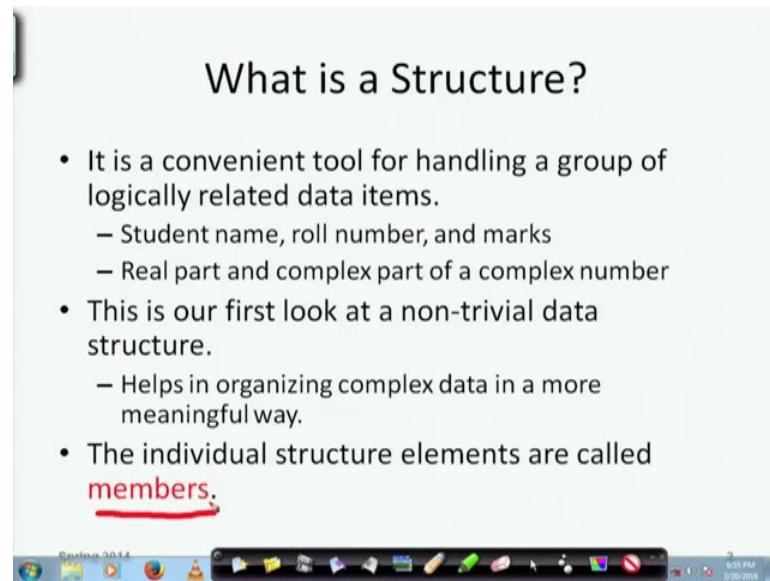
So, I draw a box which is representing the student data. I name it stud data and I have got different fields in this. Each field has gotten some identification. So, this field is storing the name and this field is basically of a type character array. So, another field is name is department and its type is also character array, another field ; remember that I every time I am using that term field of student data.

So name, department, what else do we have - role number which will be of type integer and CGPA which will be float. Suppose there is no other field. So, we call this entire box to be a structure - this is a structure and what is contained in this structure? There are 4 fields that define the structure - the 4 fields are name, department, roll number and CGPA, these 4 fields are defining a structure. Now C allows us to define structures of this type and we learn how we can define such things and let us go ahead a little bit, what is a structure? It is a convenient tool for handling a group of logically related data items here.

(Refer Slide Time: 08:55)

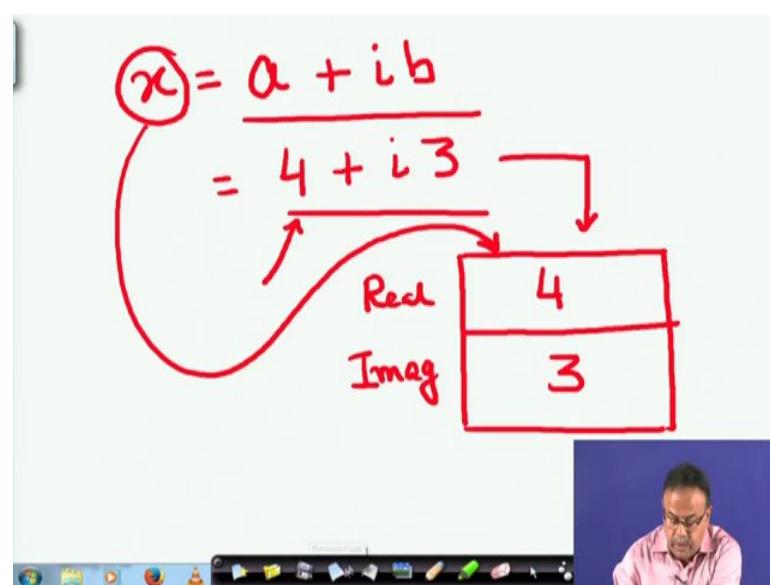
## What is a Structure?

- It is a convenient tool for handling a group of logically related data items.
  - Student name, roll number, and marks
  - Real part and complex part of a complex number
- This is our first look at a non-trivial data structure.
  - Helps in organizing complex data in a more meaningful way.
- The individual structure elements are called members.



Of course, the logically related data items were the different information fields or information component that are related to a particular student. So, till now, we saw very simple data structures like array. Now here for example, I have got related components - student name, roll number and marks or CGPA or for example a real part and imaginary part of a complex number.

(Refer Slide Time: 09:43)



$x = a + ib$   
 $= 4 + i3$

Real      Imag

4
3

For example, if I had tried this, a complex number (all of you know that a complex number is expressed as a plus ib) where x is a complex number a plus i b. So, maybe x is equal to 4 plus i 3. All of you are aware of that. Now how do I represent I know how could I can represent an integer, I know, how I can represent a floating point number, but now the question is; how I can represent such a complex number? Now well let me have a structure like this which will have 2 fields, one is the real part, there is a real part. So, I call it the real part another part is the imaginary part and I can store this number simply as 4, 3.

So, the imaginary part is 3 and the real part is 4. So, the complex number instead of being an integer or a float is essentially a structure. This is another very common and easy use of structures. So, whenever I refer to this variable x which is of type complex, I will be referring to this structure and neither at 4 or nor at 3. I will be looking at this whole thing.

So, so, it helps us in organising the complex data in a much more meaningful way. The individual structure elements that I was talking of which i was mentioning as fields is also known as members. Now you will also find other names of structures . So, structures are sometimes called records and each of these horizontal boxes that we are drawing are called fields of that record or when we referred to it as structures we call each of those smaller boxes as members.

(Refer Slide Time: 12:12)

## Defining a Structure

- The composition of a structure may be defined as:

```
struct tag {  
    member 1;  
    member 2;  
    :  
    member m;  
};
```

— struct is the required keyword.  
— tag is the name of the structure.

Struct stud-data  
{ name  
dept  
roll  
cgpa } :

So the composition of a structure can be defined as follows. See in a structure I can put a tag, now this is very important; what is the tag that I am putting over here?

Say for example, earlier I was talking of student data. So, my tag could be stud data as I had written. So, I have to write struct stud data and inside that I have got several members and what where my members? My member was name, my member was roll number, department and CGPA. So, I had 4 members. Now each of these were of different types that is something which is very important to understand. So, struct is the required keyword I must write, this word - struct. This 'tag' is the name of the structure like stud data. So, that is a structure. Just as we had done for other variables we had done say int x float y, etcetera.

(Refer Slide Time: 13:46)

## Defining a Structure

- The composition of a structure may be defined as:

```
struct tag {  
    member 1;  
    member 2;  
    :  
    member m;  
};
```

*int x;  
float y;*

*struct book*

*book { } =*

Spring 2014

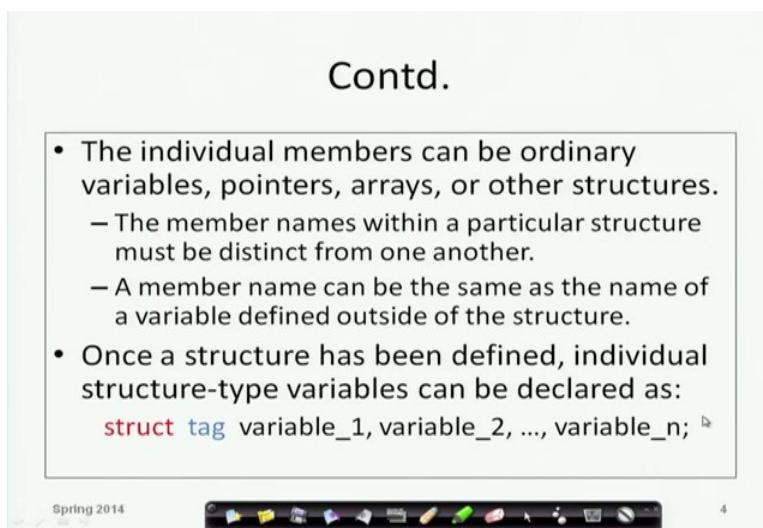
So, here also instead of int or float I have to say struct and what is struct? That is defined. So, first of all I have to define the structure. Int and float are defined by default any c compiler will understand what int or float is, but if I write struct, then by default the compiler will not understand what the structure is because there can be different structure, one structure can be for student names, one structure can be for book details - if I had thought of a book detail and wanted to store that in a structure, what will be the components? The members will be of course the title of the book, the author of the book the publisher year of publication, the ISBN number, etc.

So, therefore, each of these members will again be of different types. So, it will be a heterogeneous organisation just as it was with the student data. If I want to define books for example, I would have to write struct book followed by the definition of the members. So, this is equivalent to as if I am drawing a structure and its name is becoming - book and I am defining what are the members of this structure. Let us move a little ahead.

(Refer Slide Time: 15:48)

**Contd.**

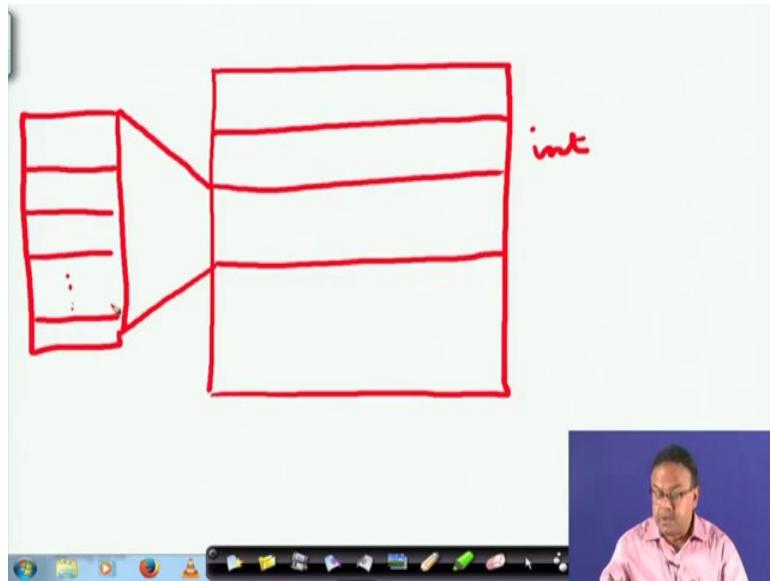
- The individual members can be ordinary variables, pointers, arrays, or other structures.
  - The member names within a particular structure must be distinct from one another.
  - A member name can be the same as the name of a variable defined outside of the structure.
- Once a structure has been defined, individual structure-type variables can be declared as:  
`struct tag variable_1, variable_2, ..., variable_n;`



The screenshot shows a presentation slide with a light blue header containing the text "Contd.". Below the header is a white rectangular box with a thin black border, containing a bulleted list of points about structure members. The list includes information about member names being distinct and being able to be the same as variables outside the structure, and how to declare individual structure-type variables. At the bottom of the slide is a standard presentation navigation bar with icons for back, forward, search, and other controls. The overall background of the slide is white.

So, member 1, member 2, etcetera are individual member declarations. We will see more examples of this. The individual members can be of different types and that is the beauty of structure, the individual members can be ordinarily variables, pointers, arrays or other structures, it can even be other structure that is very important it can be some variable, just as name or it can be an array - say name was a character array or it can even be a structure. That means, I can have a structure within a structure. So, it is possible that I will have something like this.

(Refer Slide Time: 16:29)



One field of the structure is a character array, another field of the structure can be an integer variable or another field of the structure can be a structure itself, it can be another structure where there are more members inside, that is also possible .

So, that is the beauty and the flexibility that we get from structures. So, the member names within a particular structure must be distinct from one another, we cannot put the same name to 2 different structures. Once a structure has been defined individual structure type variables can be declared as (we will see) struct tag variable one variable 2 variable n, I think, it will be much more clearer, if we go through an example, let us look at this example A.

(Refer Slide Time: 17:44)

**Example**

- A structure definition:  
`struct student {`  
char name[30];  
 int roll\_number;  
 int total\_marks;  
 char dob[10];  
 };
- Defining structure variables:  
`struct student a1, a2, a3;`

First I start with a structure definition - struct student and then I put just 2 parentheses to show that there will be some members defined inside.

Now, how do I define the members? I say that one member is name which is an array of character. The next one is roll number which is an integer. So, roll number is a variable. I said that a structure can field can be variable or can be array or can be another structure. So, here is an example where you have an array and a variable of type int again, total marks can be integer, data of birth can be character and sizes. So, data birth is again of type array character array because the date of birth can be say 10 Jan, 2010. So, like that it can be a character array.

So, this is how a structure is defined. Now, I am now defining the variables, I have defined the structures here. So, I know once the compiler reads this and finds that well I some definition of some variable called student I know that that is a structure and what is the constituent of that structure? That will be these fields.

Now this is just a type - just as we had int float, etcetera or char those where data types. Now, I am saying that I have got a structure of type this where there are 4 fields, but the data has not yet been put. This is very important though, there are 4 fields, one field is name and I say that is an array of thirty characters and there is a role number roll which is an int, total marks is an integer and date of birth is an array, but you see that there are no data already put inside this array.

Now, we have name variables - a 1, a 2, a 3 are variables of type student structure and what is that student structure this is a structure. So, a 1 will just be a copy of this structure and a variable, a 1 may have (if I take this) may have some name like amith here, roll number may be 560 and total maybe 700 and date of birth can be something. That is one particular instance of this particular structure that has been defined. This part must be very clear to all of you.

So, I have defined 3 variables a 1, a 2 and a 3 each of type; what kind of type? type structure student.

(Refer Slide Time: 21:35)

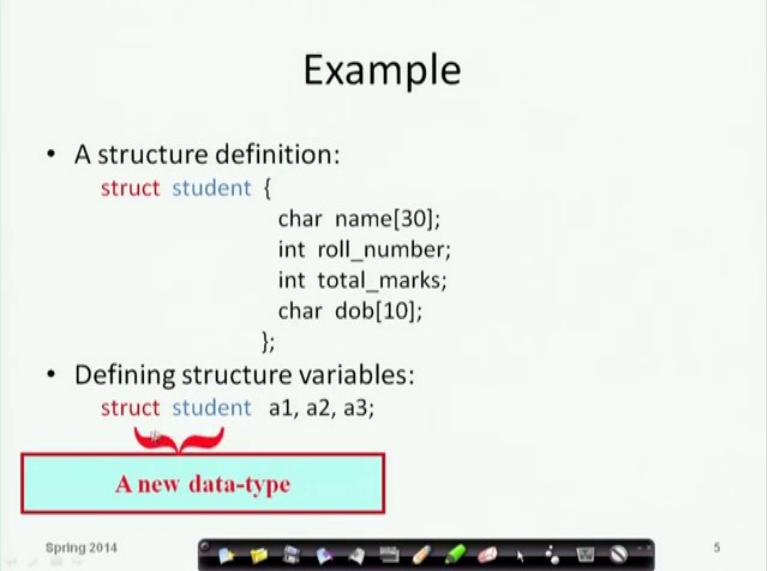
## Example

- A structure definition:

```
struct student {  
    char name[30];  
    int roll_number;  
    int total_marks;  
    char dob[10];  
};
```
- Defining structure variables:

```
struct student a1, a2, a3;
```

**A new data-type**



Spring 2014

5

So, type is struct student. We have got these fields as its constituents; so struct student tells us a new data type and these are the variables of the data type.

(Refer Slide Time: 21:38)

A Compact Form

- It is possible to combine the declaration of the structure with that of the structure variables:

```
struct tag {  
    member 1;  
    member 2;  
    :  
    member m;  
} variable_1, variable_2,..., variable_n;
```

- In this form, "tag" is optional.

Spring 2014

6

So, it is also possible to write it in a much more compact form like. So, struct tag and then i write struct student and I declare char name 40, int roll no, int total marks,char date of birth 10 and then I could have put the variables a 1, a 2, a 3 inside this note here, the semicolon is coming here that is that is the end of the definition.So, here I have done it in a much more compact form compared to the earlier one. In this form of course, this tag is optional. (Refer Slide Time: 22:40)

4.5 + i 7.2

struct complex  
{ float real;  
float imag;  
};

struct complex x1, x2;

Spring 2014

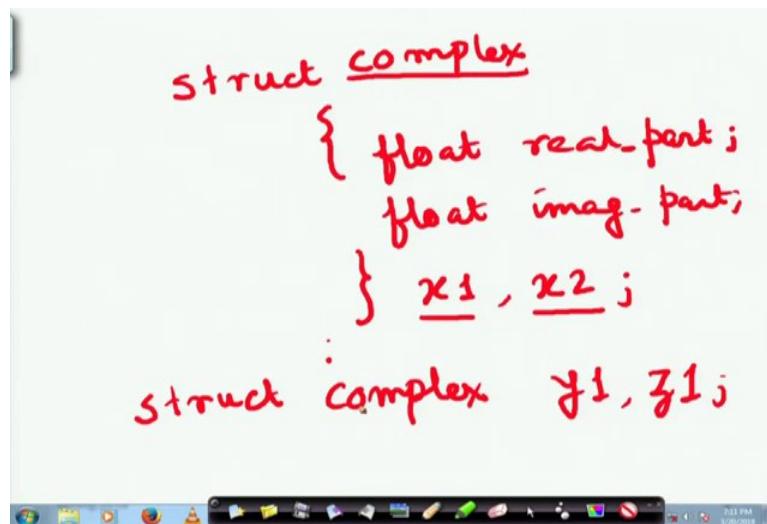
7:06 PM  
3/26/2018

So, what I mean is say I define once again let me give an example to you say a complex number I can write that as struct complex and members are int(need not be always int). So instead if I want to store data of type say 4.5 plus i 7.2, then I would need float real; float imaginary and

that is the end of the structure and I can say struct complex x 1, x 2, this is one way of representing structure.

What does it mean? It means that I have first defined the structure called complex which will have these 2 components and then I defined that x 1 and x 2 are 2 variables of type this complex. If I do it in this way I have got an advantage that later on I can define some other variables also y 1 y 2 over here or I can also define it as struct float real part.

(Refer Slide Time: 24:36)



So that you do not face confusion, let us float real part; float imaginary part and then I can say x 1, x 2. Note that here, I have not written the thing complex, I have not written it. I could have written, it would not have affected me, in any adverse way, but it is not necessary to write it over here. So, if I write this how will the compiler interpret it? Always you should think when compiler looks at it what will it think what would it possibly do?

So, the compiler is trying to understand ok now there is no ambiguity because it knows that x 1 and x 2 are 2 variables of type structure and the structure is defined here itself. I need not give any name, but the advantage of giving a name is say I give the name complex here. Here I have defined x 1 and x 2, if I had done it in this way I could have later on, down the line in this program, have written struct complex say y 1 z 1( its possible, because I have already defined struct complex ).

So, there is a choice over here which you can think of let us go ahead a little bit. So, in this case the tag is optional.

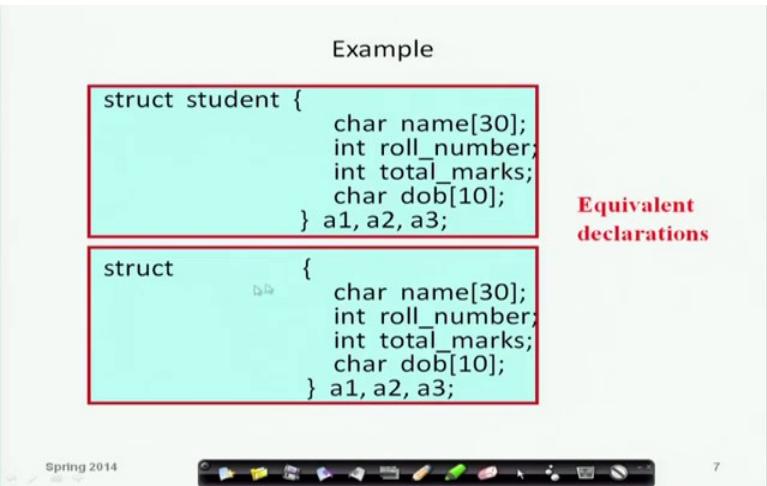
(Refer Slide Time: 26:40)

Example

```
struct student {  
    char name[30];  
    int roll_number;  
    int total_marks;  
    char dob[10];  
} a1, a2, a3;
```

```
struct {  
    char name[30];  
    int roll_number;  
    int total_marks;  
    char dob[10];  
} a1, a2, a3;
```

Equivalent declarations

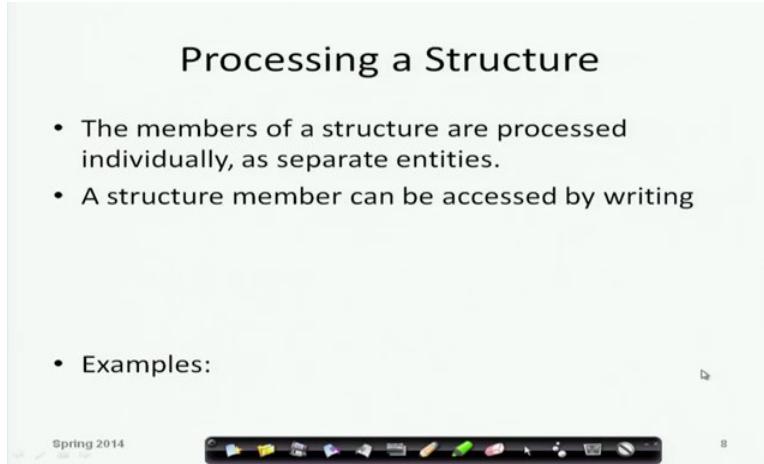


So, here is an example we will conclude this lecture with this example. So, struct student character name thirty; int roll number; int total marks; character data birth 10, a 1, a 2, a 3; that means, a 1, a 2 and a 3 are 3 students, 3 pieces of student information or I could have done it in the way I have just now shown, the same thing without giving any tag over here, these are equivalent.

(Refer Slide Time: 27:09)

## Processing a Structure

- The members of a structure are processed individually, as separate entities.
  - A structure member can be accessed by writing
- 
- Examples:



Spring 2014

8

Now, next lecture, we will talk about how do you process a structure that will come in the next lecture.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 56**  
**Structure (Contd.)**

So, in the last lecture we had seen how a structure is defined. Now we will see more examples of that in this course. So, today we will be concentrating on how a particular structure can be processed. In order to process a structure, we can we have to process every field of the structure. We can operate on every field of the structure.

(Refer Slide Time: 00:42)

**Processing a Structure**

- The members of a structure are processed individually, as separate entities.
- A structure member can be accessed by writing variable.member

Hand-drawn diagram:

```
graph TD; stud[stud] --- name[name]; stud --- rollno[roll-no]; a1[a1, a2, a3]
```

The diagram shows a structure variable named "stud" at the top. Below it, two fields are listed: "name" and "roll-no". At the bottom, three instances of the structure are shown as "a1, a2, a3".

So, we can see that a structure how can we access the member of a structure. Say for example, there is a variable like name. So, we can say overall name of the structure is say student all right; and in the inside student we have got name, roll number etcetera now I can. So, my variable is say all right let me say I have defined like this struct student, then all these then the variables are a 1, a 2, a 3 I am sure you are getting confused a little bit.

(Refer Slide Time: 01:40)

## Processing a Structure

- The members of a structure are processed individually, as separate entities.
- A structure member can be accessed by writing variable.member

```
struct student
    { char name[10];
      int roll-no;
    };
x1. roll-no = 720;           { char name[10];
x2. name [3] = 'j';         int roll-no;
x2. name [ ] = "....."; } x1, x2;
```

- Examples:

So, let us define it in the proper way as we are done in the last class. Suppose I define struct student, char name 10, int roll number all right suppose only two fields are there and I also say that the variable is variables are x 1 comma x 2 these are 2 variables of the type of this structure whose name is student. So, what is my variable named? x 1. So, here I can say x 1 x 1 dot this is the dot operator which tells about the member, now which member say I want to get the roll number. x 1 dot roll number assign 720 all right. So, this is how I can get access to a particular member of a particular structure. So, I can also similarly write x 2 dot name.

Because that members name is name right name 3 is j all right because name has got a 10 field character ok. I could have written x 2 dot name and I could have assigned a string to this all these are possible.

(Refer Slide Time: 03:46)

The screenshot shows a presentation slide with a light gray background. At the top center, the title 'Processing a Structure' is displayed in a bold, black, sans-serif font. Below the title, there is a bulleted list of four items. The first two items are standard bullet points. The third item contains a sentence with two underlined words: 'variable' and 'member'. The fourth item is another bullet point followed by a dash and some code examples. At the bottom of the slide, there is a decorative footer bar featuring various icons, and the text 'Spring 2014' is visible on the left side.

- The members of a structure are processed individually, as separate entities.
- A structure member can be accessed by writing variable.member  
where **variable** refers to the name of a structure-type variable, and **member** refers to the name of a member within the structure.
- Examples:
  - a1.name, a2.name, a1.roll\_number, a3.dob;

So, if this is clear then we go ahead a little bit were variable refers this variable is referring to is referring to the name of a structure type variable and member is the referring to the field or the particular member element ok. The examples are a 1 dot name, a 1 dot roll number, a 3 dot date of birth etcetera.

(Refer Slide Time: 04:12)

Example: Complex number addition

```
#include <stdio.h>
main()
{
    struct complex
    {
        float real; ←
        float complex; ←
    } a, b, c;

    scanf ("%f %f", &a.real, &a.complex);
    scanf ("%f %f", &b.real, &b.complex);

    c.real = a.real + b.real;
    c.complex = a.complex + b.complex;
}
```

```
printf ("\n %f + %f j", c.real,
       c.complex);
```

$$x \quad y$$

$$x + y$$

$$x = (a) + i(b)$$

$$y = (m) + i(n)$$

$$x + y = (a+m) + i(b+n)$$

So, now here let us take the old example that we had seen that is a complex number and how we can perform complex number addition. Now, just to refresh your mind say I have I have got two complex numbers  $x$  and  $y$  and I want to perform  $x$  plus  $y$ . Suppose  $x$  is  $a$  plus  $i b$  and  $y$  is  $m$  plus  $i n$  then  $x$  plus  $y$  is the real parts are added separately. So, it is  $a$  plus  $m$  plus  $i$  then the imaginary parts are added separately,  $b$  plus  $n$  that we know from our school level knowledge.

Now how can I perform this addition using the concept of a structure. So, here you see in the program, we define a structure called complex and there are three variables  $a$ ,  $b$  and  $c$  of this structure type of type complex all right. Now, what are? There that components are one is float sorry I am sorry one is real part, another is I should rename this is actually the imaginary part its renamed in this way here in this program it would have been better, if we I had written float real part float imaginary part however.

Now, scan f I am reading the two numbers  $a$  is a variable,  $a$  is real part is being read and  $a$  is complex part is being read note the format statements because both of them are float we are putting percentage f all right. And so, I read  $a$ . So, you see normally I would have just done scan

f amperes and a, but here I have to do separately I have to read the real part and the imaginary part, then the sum is being stored in c.

So, c is real will be c is the sum. So, it is real will be sorry it is a real part will be the real part of a plus the real part of b and the imaginary part or the as it is written here complex part is the complex part of imaginary part of a plus the imaginary part of b. Then I am printing look at how I am printing it, printing percentage f the real part of c and plus then I put j and then the complex part of c.

(Refer Slide Time: 07:19)

Example: Complex number addition

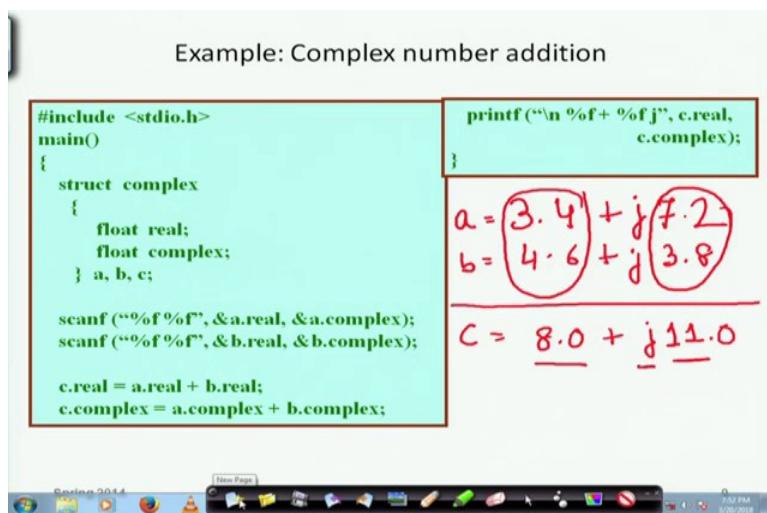
```
#include <stdio.h>
main()
{
    struct complex
    {
        float real;
        float complex;
    } a, b, c;

    scanf ("%f %f", &a.real, &a.complex);
    scanf ("%f %f", &b.real, &b.complex);

    c.real = a.real + b.real;
    c.complex = a.complex + b.complex;

    printf ("\n %f+ %fj", c.real,
           c.complex);
}
```

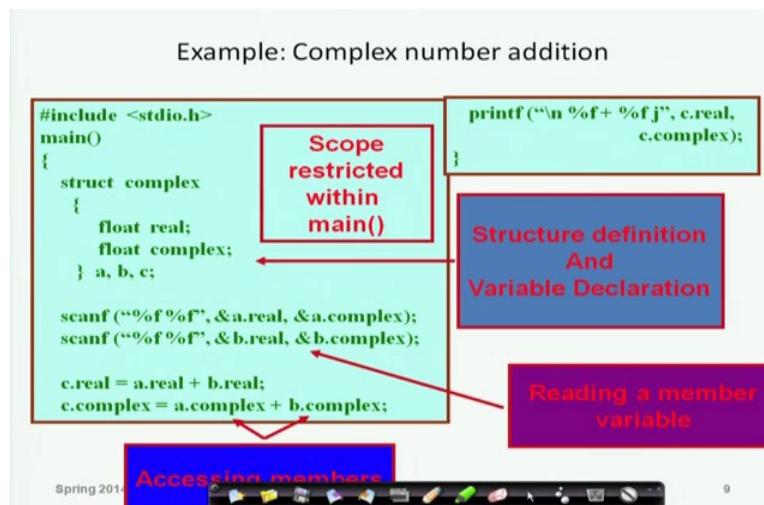
$$a = (3.4) + j(7.2)$$
  
$$b = (4.6) + j(3.8)$$
  
$$C = \underline{8.0} + j\underline{11.0}$$



So, what is being done is suppose I have got 2 numbers, 3.4 plus here I am using j some people I usually use i or j whatever you do? So, you write 7.2 that is a is real part is this, and b is real part is say 4.6 plus j 3.8 now if I add them c will be these real parts are being added. So, it is 8.0 plus the imaginary part is added that is 11 j 11 all right. So, what will be printed is 8 plus j then the complex part 7.0.

So, this is a I think this is clear to you this not very difficult to understand let us move ahead.

(Refer Slide Time: 08:22)



So, you have seen here, if you note here you have got the structure definition specified and here are the very variable definitions. And here we are doing the reading and adding them part by part how am I accessing the members. I am accessing the members using this dot operator and whenever I am reading the member I am actually reading the members, I am not reading the structure as a whole.

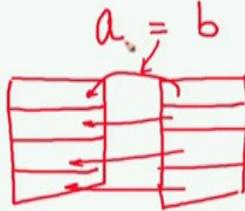
Now, one thing to note is when I am declaring this structure inside the main, then the scope of these variables real and complex etcetera or this structure as such is its scope is within the main.

So, if a structure is defined within a function, then as we exit that function the life of that particular structure will end at that point.

(Refer Slide Time: 09:24)

## Comparison of Structure Variables

- Unlike arrays, group operations can be performed with structure variables.
  - A structure variable can be directly assigned to another structure variable of the same type.



The slide shows a comparison between arrays and structures. It highlights that unlike arrays, where element-wise comparison is required, structures can be directly assigned to each other. A diagram illustrates this by showing two structures, 'a' and 'b', as tables. Red arrows show the mapping of elements from structure 'a' to structure 'b', demonstrating that the entire structure is copied.

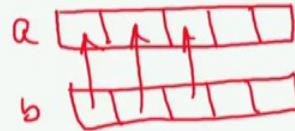
Now, there are some things which we can do in a much more simpler way, you remember that in the case of arrays we could not compare arrays together. That array a 1 is equal to a 2 that was not possible, if we had to compare the array I had to check them element by element. On the other hand in the case of a structure, a structure variable can be directly assigned to another structure variable of the same type. For example, here if a is a structure, then I can assign another structure b to it directly.

So, all the elements if they are of course, of the same type; So, here is one structure, here is another structure now if I do this operation ; that means, assignment then this will be copied here, this will be copied here, this will be copied here, this will be copied here all right.

(Refer Slide Time: 10:36)

## Comparison of Structure Variables

- Unlike arrays, group operations can be performed with structure variables.
  - A structure variable can be directly assigned to another structure variable of the same type.



But if a and b were arrays in that case, that would not have been possible that a is an array and b is an array. So, the elements just by assigning a assign b, I could not assign these b elements to a that is not allowed in the case of an array; however, that is possible in the case of a structure.

Secondly now here one important thing is that they must be of the same type all right otherwise of course, the members and the fields will not match.

(Refer Slide Time: 11:11)

## Comparison of Structure Variables

- Unlike arrays, group operations can be performed with structure variables.
  - A structure variable can be directly assigned to another structure variable of the same type.  
 $a1 = a2;$ 
    - All the individual members get assigned.
  - Two structure variables can not be compared for equality or inequality.  
if ( $a1 == a2$ ) .....
  - Compare all members and return 1 if they are equal; 0 otherwise.

Spring 2014

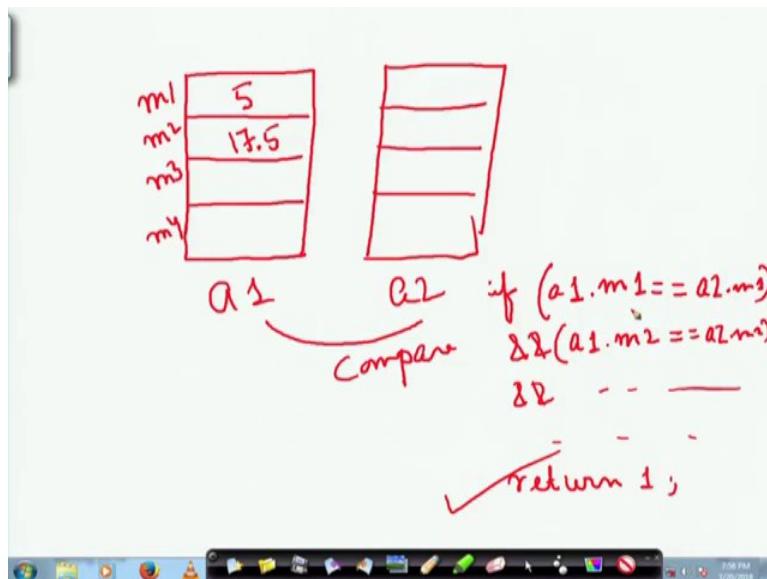


10

So, a 1 assigned a 2 I can do, all the individual members get assigned. Two structure variables cannot be compared for equality or inequality. Now this is important that two structure variables I cannot compare them, that structure a 1 whether that is equal to a 2 that will have to do compare I mean, what I said just a couple of minutes back was inadvertent that is wrong.

Structures we cannot compare we can assign in a short, but comparison of structures we cannot do as a whole all right. Two structure variables cannot be compared for equality or inequality now for that, what we have to do we can write a simple function say I can write and you can take it as an assignment, that we can write struct comp, struct c m p where you will take 2 structures as input and we will compare them and how do you compare them? We will do say variable a 1 dot member 1 equals to is whether its if it is equal to variable a 2 dot member 1, in that way if you do then it is possible.

(Refer Slide Time: 12:32)



So, what we can do here is, I have got two structures and there are 2 variables of the same type of the same structure type one is a 1 and one is a 2 and I have got some values here all right. And say this is member 1, member 2, member 3 member 4 like that now in order to compare if I want to compare them what I have to do?.

I will have to see if this is a 1, a 1 dot m 1 is this also m 1 is equal to a 2 dot m 1 and a 1 dot m 2 is equal to a 2 dot m 2 and so and so forth. If I do then return 1 that could be my function ; that means, if all these are matching, then only I will say that they have been compared to be the same otherwise it will return false. So, here other else return false. So, you have to write it nicely you can take it up take it up as an assignment arrays of structures.

(Refer Slide Time: 14:07)

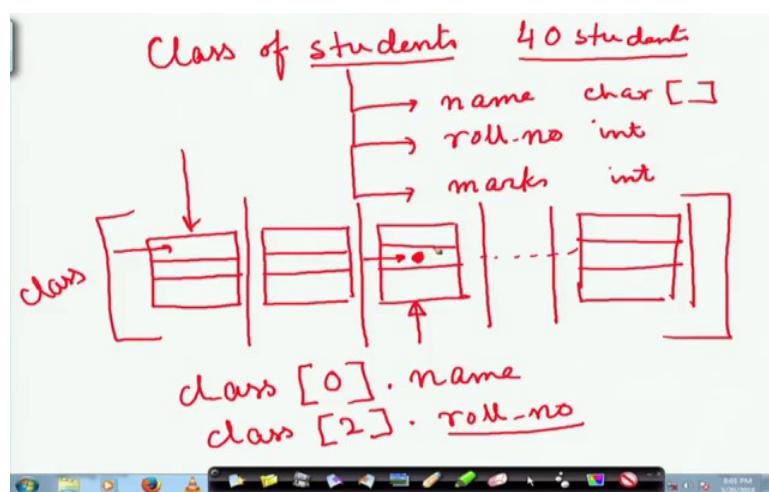
## Arrays of Structures

- Once a structure has been defined, we can declare an array of structures.  
`struct student class[50];`
- The individual members can be accessed as:
  - `class[i].name`
  - `class[5].roll_number`



Now, once we know this, we can now make an array of structure why is that useful let us take one problem. Let us say that I am trying to store the information of a class of students.

(Refer Slide Time: 14:24)



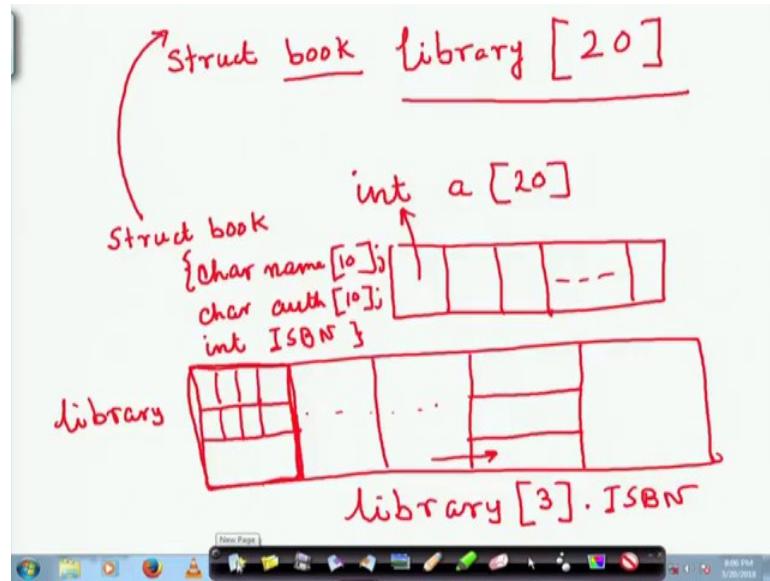
Where maybe there are say 40 students and their information about each student has got different components; name in a particular class, roll number and say marks total marks. So, marks is an integer let me say, roll number is also an integer and name is a character array something all right. Now that so, this box this can be also depicted as a box having three fields name, roll number and marks, but I want to have that for all the 40 students.

So, I will have 40 such small boxes, each having 3 fields one after another. So, that can be represented as an array you know that in an array we can store data of only the same type. So, that is not causing any problem here, because each of these boxes are of the type struct students therefore, I can very well consider them to be an array of such structures.

So, if I call this class is this array, then each of these boxes are an element of that array. So, if I say class 0, which element am I looking at? This element right; if I take class 0 dot name, then which field am I looking at? This particular field; if I write class 2 dot roll number, which field am I looking at? Class 2. So, it is an array 0 1 2 you know that just in c any array starts with 0. So, 0 1 2 I come over here and which field am I looking at roll number roll number is the second field. In that way I can access each element of this array, either as a structure or as an individual member of that structure all right.

So, in such cases or the same thing could be applicable, if I want to represent a library. So, in that case what will happen? A library will consist of books right. So, suppose it is a small library, where I have got a library of 20 books all right 20 books.

(Refer Slide Time: 17:50)



So, books is an array of type what or let me let me name it different way. So, I am say in my library is consisting of 20 books at most, and what is the type of library when we write in a array? We write int a 20 float a 20 like that. So, here what is the type of library? Type of library will be struct book. So, because every element of this library, when I write int a 20; that means, I am talking of an array where there are 20 elements and each element is of type int.

So, when I am talking of this struct book library 20 what will that mean? That will mean that here is an array library having 20 such field 20 such positions elements and each element is a structure called book. I have not defined here the structure of book. So, I should have done it earlier.

Suppose I do here struct book, char name 10 semicolon that is the name of the book. Char author 10 the author of the book int ISBN and I close that. Suppose this is a structure of the book having 3 fields only I have it could have been more fields number of pages year of publication etcetera I am not showing that. So, each of these elements have got 3 fields here name which is itself an array of 10 characters, author which is itself an array of 10 characters and an integer ISBN number.

So, this is the whole element, the first or 1 element of the array library. And library is an array which houses 20 at the most such individual structures. So, if I say library 3 dot ISBN where am I going which one I am referring to? 0, 1, 2, 3 library 3 and the last element of that this is the ISBN number because this is a last one all right.

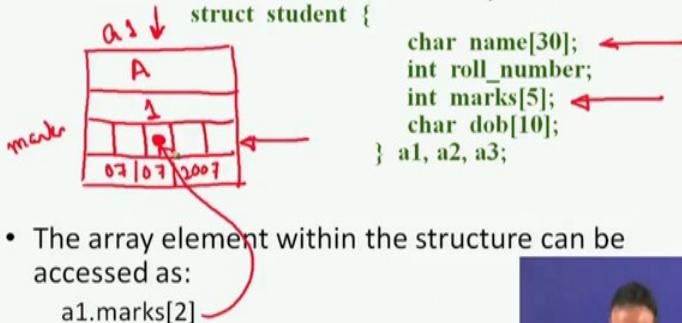
So, therefore, using structure allows us to store different types of data together, and again array of structure is a very powerful facility that has been given to us, using which we can store many more things and we get much more flexibility. Let us go here, once a structure has been defined like the structure book that was defined we can declared an array of structures, for example, struct student class 50.

So, note here that class is an array having 50 elements. So, it is something like this here is class with 50 elements 0 to 49, and what is each element? Each of these elements is a structure of type student must have been defined somewhere earlier.

(Refer Slide Time: 22:47)

## Arrays within Structures

- A structure member can be an array:



- The array element within the structure can be accessed as:

a1.marks[2]



The individual members can be accessed by class whichever element are you want to have say for example, struct student, one field is int roll number, there is a name, there is marks and character. So, this is the student and I have got 3 variables a 1, a 2, a 3. The array element within the structure and I have defined an array earlier that is a class of 50 right. So, now, I have got an array of which every element is a structure like this, and the structures are variables a 1, these are also the student variables also my class is consisting of the same structure.

The array element within the structure can be accessed as now you can very easily guess, if a 1 dot marks 2. Now, here this example is telling us something more, here this example is showing that any element of have and a member of this structure student can also be an array, that we had defined earlier this is also an array.

Now, suppose I have got this student a 1 his name is a, roll number is say 1, and marks is itself an array where the marks of literature, history geography, maths, science everything is stored and the date of birth is a character array all right. Now so, it is say 07 07 2007. So, that is a character array.

Now, when I am saying a 1 marks 2; that means, I am going to this field is the marks field, I am going to the a 1 structure a 1 variable, and I am taking its structure and coming to the marks field or membership member and I am coming to the element 0 1 2 this particular element this particular element I am coming to by this, this you must understand.

So, I can have an array within a structure, also I can have an array of structure. Till now what we are discussing was and here what we discussed was an array of structure right. And here what we have just now shown you is what we have just now shown you is an example of that an array can also be a part of a structure.

(Refer Slide Time: 26:02)

Defining data type: using *typedef*

- One may define a structure data-type with a single name.
- General syntax:

Spring 2014

13

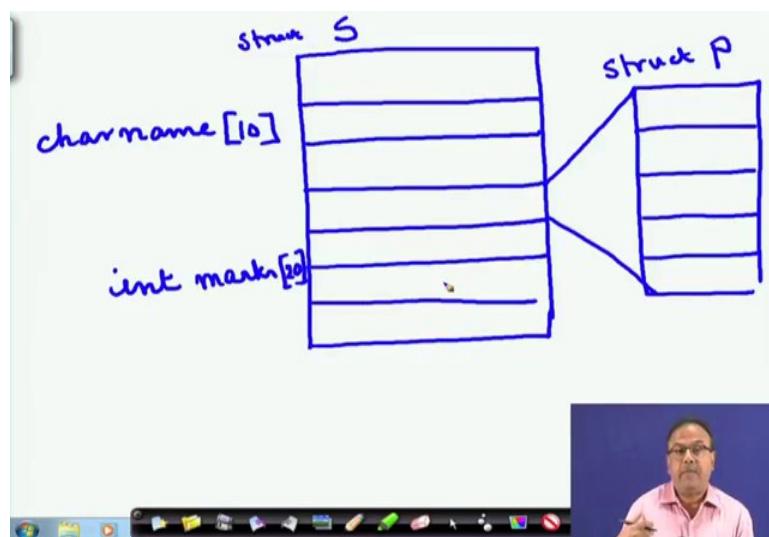
Next lecture we will be talking about a new thing that is that is also very important type def. That is very much useful for that facilitates us to create new type definitions and we can write much simpler programs with this we will take it up in the next lecture.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 57**  
**Structure with typedef**

We were discussing about structures and in particular we have discussed about the way the structure is formed and in that context we have seen that a structure can contain within itself some members which can also be a structure themselves.

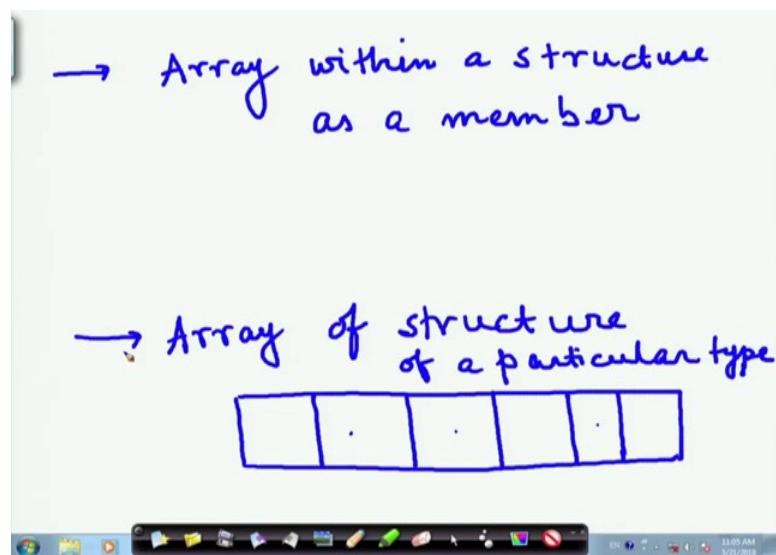
(Refer Slide Time: 00:32)



For example, this one member which can itself be a structure consisting of other members. So, this is a structure struct S in which there is another struct say P which is a member of this, also some member can be an array. This can be say name 10 char; that means, it is a character array of name 10, similarly it could be some member like int marks 20.

Now, we have seen that we can have arrays within structure also in the last lecture we have talked about arrays of structure. So, there is a scope of confusion between these two that is why I want to make it clearer that there can be an array within a structure as a member.

(Refer Slide Time: 02:08)



So the other thing is we can have an array of structure. So, in this case we have got an array, every element of that array is a structure of a particular type. We know that an array can hold data elements of the same type. So, if I have one particular structure defined like say student, it can be an array of students. So, each of them is of the type student. So, these two must be differentiated very clearly all right. (Refer Slide Time: 04:03)

## Arrays within Structures

- A structure member can be an array:

```
struct student {  
    char name[30];  
    int roll_number;  
    int marks[5];  
    char dob[10];  
} a1, a2, a3;
```

- The array element within the structure can be accessed as:

a1.marks[2]



So, as you can see that here a structure member can be an array, we saw it in the last class last lecture that say for example, a structure student has got an array char, character array name thirty, all right, this is a part of an array. Now we know arrays within structure and we have also seen arrays of structure, where we can have a number of elements of that particular structure.

(Refer Slide Time: 04:54)

## Arrays of Structures

- Once a structure has been defined, we can declare an array of structures.

struct student class[50];



- The individual members can be accessed as:

class [i]. name  
class [2]. roll.no  
...



So, here we have seen that we have got a an array class, the array is class which can hold up to 50 elements and each element of this class is a structure of type student.

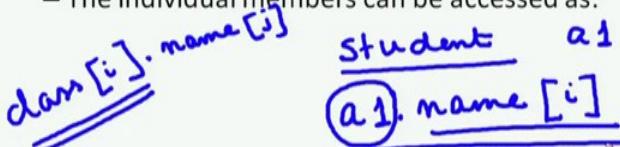
When I come to say class i, then I am actually accessing a structure a particular structure in that array. Now that structure has got number of fields therefore, if I have class i dot name, then I will get a particular name (if that be the field) or I can say class 2 dot roll number.

So here there is an array of structures and we can handle them just as in this case where the index is pointing to the particular element, and then by dot operator we are going inside that element and looking at that array.

(Refer Slide Time: 06:32)

## Arrays of Structures

- Once a structure has been defined, we can declare an array of structures.

```
struct student class[50];
```
- The individual members can be accessed as:  


Whereas in the case of an array within a structure we look at that in a different way, suppose if I come to the structure; student and suppose there is a variable of variable of name say a 1 as of type student, then I can say a 1 dot name and I can take a particular character of that.

So, look at the difference of this, this is one and the other one was class i dot name j. So, here I am accessing the particular element from that structure, and here I am coming to the structure and going to that array element within that structure. So, this is an usual point of confusion among many students that is why I was repeating this.

Next we will start looking at an important style of writing programs which is `typedef`.

(Refer Slide Time: 07:46)

## Defining data type: using typedef

- One may define a structure data-type with a single name.
- General syntax:

int  
float  
char



That facilitates our programming with structure. As the name `typedef` implies, we are defining some type. We know that `int` is a type, `float` is a type, data type `char` is a type etcetera. Now, I can also define my own type which I will be using in my program using this `typedef` command.

So, we may define a structured data type using a `typedef` command like this, let us see the syntax.(Refer Slide Time: 08:36)

## Defining data type: using typedef

- One may define a structure data-type with a single name.
- General syntax:

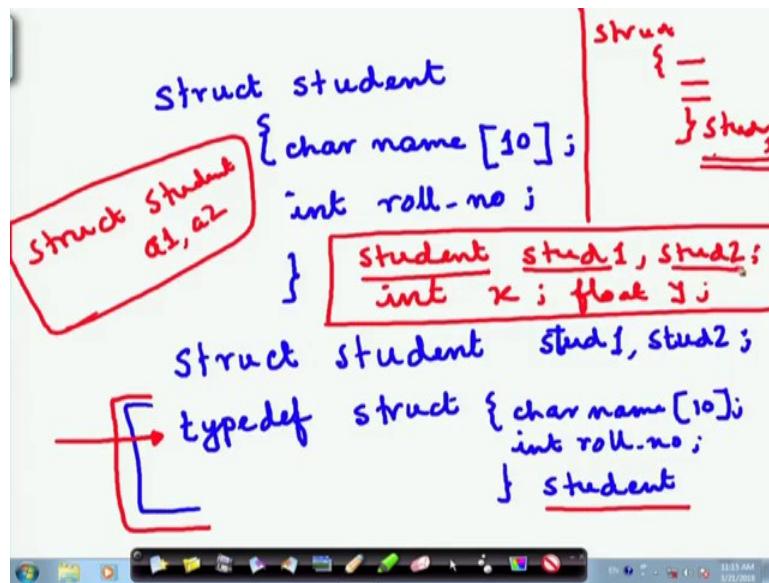
```
typedef struct {  
    member-variable1; char name[ ];  
    member-variable2; int roll ;  
    .  
    member-variableN;  
} tag; student;
```

- `tag` is the name of the new data-type.



The syntax is I say typedef struct member variable 1 member variable 2 member variable n and then some tag. Tag is the name of the new data type. Let me illustrate this by differentiating this with the earlier thing. Earlier we had written something like this struct student.

(Refer Slide Time: 09:07)



Then char name 10 int roll number. So, I was defining a structure in this way, and I was later on referring to the structure as struct student say student 1, student 2, in this way. Now, what I am trying to do here is, I am trying to define a type, suppose I do not write this, what I am doing instead I am saying typedef struct same thing char name 10, int roll number and naming this type as student. If I do this then next time when I declare the student 1 and student 2, I will do it in a different way I can do it much simply. See how I am writing now, in the red box.

If I just simply write student, student 1 comma student 2 that is sufficient. Compare this with the way we had declared int x float y. So, here int and float were some types, now here student 1 student 2 are two variables of type student. Now, this type is not a default type it is not pre-

defined in c but I can use it because I have been already using this typedef statement has defined the structure as student.

But please differentiate between this and the normal definition of the student as we do normally. Say struct something and then I say student. I say struct (we have seen this) and I write the members here and then define that structure as the type student. Now this is a definition for a particular structure as student. Now this, if I have to use it then I have to say struct student followed by a 1 a 2, student 1 student 2 whatever. But in this case if I do this typedef, I am saved from this problem. So, let us have a little look at how we will go about it.

So, that is a tag. So, here typedef a particular structure, just like here it could be char name or int, roll; note the semicolon at the end of if all this declarations, ultimately I name this tag semi say for example, here as student. So, tag is the name of the new data type.

(Refer Slide Time: 13:59)

### typedef : An example

```
typedef struct{
    float real;
    float imag;
} _COMPLEX;

_COMPLEX a,b,c;
```



So, given that here is an example of complex number. So, we are defining a type called complex, ignore this.

So, typedef something having real part and an imaginary part is known as a data type complex and then I am saying complex a b c just like int a b c, float a b c, I can write complex a b c because complex has already been defined here.

(Refer Slide Time: 14:33)

The slide has a light green header bar with the title "Structure Initialization". Below the title is a bulleted list:

- Structure variables may be initialized following similar rules of an array. The values are provided within the second braces separated by commas.
- An example:

Handwritten red annotations are present on the right side of the slide:

- A bracket under "char name[ ];" is labeled with an arrow pointing to it.
- A bracket under "int roll;" is labeled with an arrow pointing to it.

A red arrow points from the bottom of the slide towards a video frame of a person speaking.

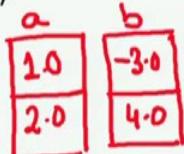
Because complex has already been defined; so, I think this is clear right ?

Next we move to, how do we initialise a structure ? We saw that we can initialise an array similarly we can initialise a structure as well. Say a structure variables may be initialised following similar rules like an array. The values are provided in a within the second braces separated by commas for example, here complex a.

(Refer Slide Time: 15:14)

## Structure Initialization

- Structure variables may be initialized following similar rules of an array. The values are provided within the second braces separated by commas.
- An example:  
\_COMPLEX a={1.0,2.0}, b={-3.0,4.0};



11:39 AM 5/2/2018

Complex a, is a complex variable 1.0 comma 2.0 what does it imply? It implies that 'a' is a structure having two fields and one field is 1.0 another is 2.0. Similarly b is another field, which is initialised to minus 3.0 and 4.0. Just as we did it in the case of arrays, we do it for individual structural variables here, but you have to put all the values initial values for all the fields of the array.(Refer Slide Time: 16:12)

## Structure Initialization

- Structure variables may be initialized following similar rules of an array. The values are provided within the second braces separated by commas.
- An example:  
\_COMPLEX a={1.0,2.0}, b={-3.0,4.0};

a.real=1.0; a.imag=2.0;  
b.real=-3.0; b.imag=4.0;

15

So, this is what happens through this initialisation.

(Refer Slide Time: 16:17)

## Parameter Passing in a Function

- Structure variables could be passed as parameters like any other variable. Only the values will be copied during function invocation.

```
void swap(_COMPLEX a, _COMPLEX b)
{
    _COMPLEX tmp;    void swap(int a,
                                int b);
    tmp=a;
    a=b;
    b=tmp;
}
```

The last thing that we will be talking about is parameter passing in a function. How do we pass a structure as a parameter to a function? We have seen how arrays can be passed similarly how can we pass structures to a function like any other variable? Just like any other variable we can pass it like for example, here you can see that there is a swap between complex variable a and complex variable b. Now, a and b are both structures. So, I am saying just as we use to write void swap int a int b just like that, here I write complex a complex b. It is also a call by value. Now so, here is again the typical assignment.

(Refer Slide Time: 17:37)

An example program

```
#include <stdio.h>

typedef struct{
    float real;
    float imag;
} _COMPLEX;

void swap(_COMPLEX a, _COMPLEX b)
{
    _COMPLEX tmp;
    tmp.real = a.real;
    tmp.imag = a.imag;
    a.real = b.real;
    a.imag = b.imag;
}
```

So, here is an example program using `typedef`. What is being done here is I am defining a type complex, `typedef struct` - they are two parts real and imaginary and this data type is known as complex. Now I am writing a function `swap`. So, I am taking another variable which is of type complex, `tmp` is of type complex. So, when I do assign `a` to `tmp`, then this `tmp` variable which is of type complex will copy the variables (say here it was 3.0 and minus 2) so, this will come here it will become 3.0 and this will become minus 2.

So, that is `tmp`. Now I copy `b` to `a`. I am sorry not this one, there was some `b` and the values of `b` are copied to `a`, not here, these are copied here and here. So, it might be that this is changed to 5, this is changed to minus 2.5 and then I copy `tmp` to `b`. So, this is again copied back. So, member by member the copy is done. So, here we illustrate what you mean by `typedef` and a function that is using it.

(Refer Slide Time: 19:55)

Example program: contd.

```

void print(_COMPLEX a)
{
    printf("(%f , %f) \n", a.real, a.imag);
}

main()
{
    _COMPLEX x={4.0,5.0},y={10.0,15.0};
    print(x); print(y);
    swap(x,y);
    print(x); print(y);
}

```

So, now suppose how can we print a structure? Say here is a function, the main function is calling print x and print y. So, say for example, my main function is initialising (now you understand all this things) x to 4, x real part to 4, imaginary part to 5 right and y's real part to 10 and imaginary part to 15. So, now, I am calling print x. So, it is calling print x. So, x is being copied call by value to another local variable a, which is local to this function print. So, a is holding the complex variable x and print f a dot real a dot imaginary.

So, field by field I print them. Then I come back I print y. Y is copied to a and the same thing happens, then I call swap. Now, swap as a function that we just now saw where we take tmp and we copy it to tmp, and then I copy b to a, and then tmp to a in that way we carry out the swapping.

So, it is purely call by value and then I print x and print y. After I swap, what will happen in this case? If I go back to this case, where I am passing among this a and b, the x and y are being swapped inside that function, but will that be swapped in actuality? You check that yourself.

(Refer Slide Time: 22:10)

x, y

## Returning structures

y = add(x, y)

- It is also possible to return structure values from a function. The return data type of the function should be as same as the data type of the structure itself.

```
COMPLEX add (COMPLEX a, COMPLEX b)
```

```
{ COMPLEX tmp;
```

tmp.real = a.real+b.real;  
tmp.imag = a.imag+b.imag;

```
} return (tmp);
```

this tmp will be assigned to this complex variable z, that is how we can return a structure from a function.

(Refer Slide Time: 24:52)

## Returning structures

- It is also possible to return structure values from a function. The return data type of the function should be as same as the data type of the structure itself.

```
COMPLEX add (COMPLEX a, COMPLEX b)
{
    COMPLEX tmp;
    tmp.real = a.real+b.real;
    tmp.imag = a.imag+b.imag;
    return(tmp);
}
```

Direct arithmetic operations are not possible with Structure variables.



19

So, structure facilitates us in many ways but direct arithmetic operations are not possible with structure variables; that means, I cannot just add a and b when both of them are structures. I have to do the arithmetic operations over its members. So, with that we conclude our discussion on structures and you will be given assignments on structures, and this will enable you to write different types of handle different types of data types together and using the typedef, you can design your own complex data type which you can utilise further.

So, please practice using structures it is not at all difficult, only a little practice and a little understanding is required.

Thank you.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

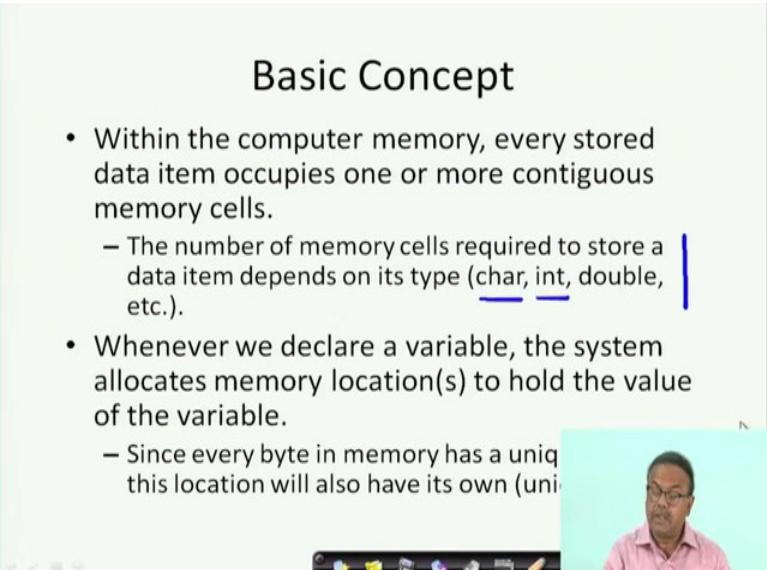
**Lecture – 58**  
**Pointer**

In today's lecture we will look at very important concept of programming, it is required to conceptualise and understand this thing very well so, that you can have more flexibility with programming. We have visited this idea the concept of pointers, earlier in the context of our discussions of call by value and call by reference.

(Refer Slide Time: 00:48)

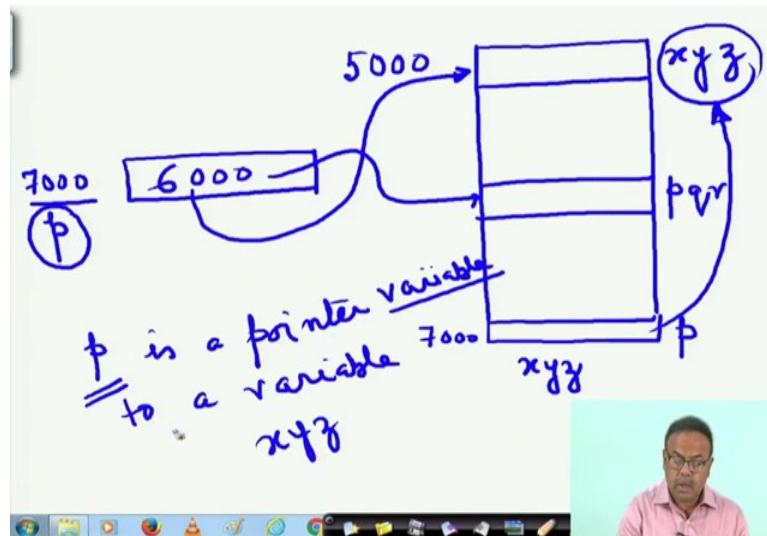
**Basic Concept**

- Within the computer memory, every stored data item occupies one or more contiguous memory cells.
  - The number of memory cells required to store a data item depends on its type (`char`, `int`, `double`, etc.).
- Whenever we declare a variable, the system allocates memory location(s) to hold the value of the variable.
  - Since every byte in memory has a unique address, this location will also have its own (unique) address.



So, if you recall at that time, we had talked about the variables, which are in the memory right ?

(Refer Slide Time: 00:59)



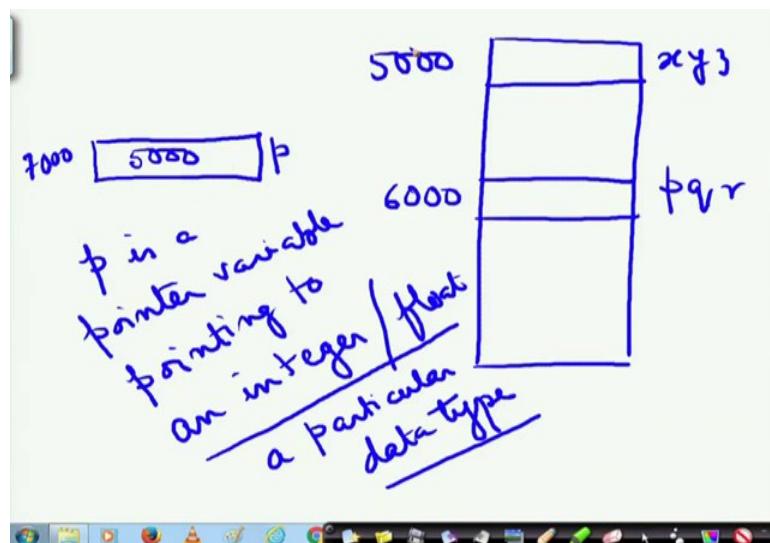
So, suppose I am talking of a variable - x y z is the name of a variable. So, that variable has got some address, that address maybe say 5000 in the memory and who has allocated this address? That address has been allocated by the compiler.

Now, if I have another memory location, say a part of this which I am just drawing separately whose address is say 7000? Say this one - 7000 and inside this location 7000 as it's content, I write 5000 and I say that whatever is the content of this location 7000 that is the variable I am interested in.

So, can I say that if I now give a symbolic name p to this 7000 or say this is p, that this p is pointing to x y z because p is containing the address of x y z. Therefore, p is a pointer to a variable x y z. Right now p is pointing to x y z, but if I just change the value of this location 7000 and make it say 6000 then it will probably point to some other location here.

So, p is therefore, a pointer variable, but when I say that it is a pointer variable, then it is not pointing to any particular data, it can point to a type of data. So, instead of making this statement that p is a pointer variable to a variable x y z. Say this one is p q r now as I change this, now p is a pointer variable to a variable p q r.

(Refer Slide Time: 04:02)



So, if I generalise it lets say that in this situation where I have got x y z at 5000 and p q r at 6000 and 7000 is a pointer p then I can say p is a pointer variable pointing (say x y z and p q r who are both integers) to an integer or it could be a float or it could be a some other data type. So, pointed to a particular data type. Why I specify the data type will be clear in some time from now. The most important concept here is that p is a pointer, p is a variable and variable is holding some value and that value is nothing, but an address of another variable all right address of another variable.

So, now let us look at this the basic concept, every stored data items occupies one or more memory cells, whenever we declare a variable the system allocates memory locations to that variable, we know that very well; so, need not spend more time on that. Now this is important, you already know that the number of memory cells required to store a data item depends on its type, typically for char we need one byte, for int we need two bytes, for float we need four bytes etcetera.

So, since every byte in memory has an unique address, this location will also have its unique address. Every element will have a unique address.

(Refer Slide Time: 06:38)

## Contd.

- Consider the statement  
`int xyz = 50;`
  - This statement instructs the compiler to allocate a location for the integer variable `xyz`, and put the value `50` in that location.
  - Suppose that the address location chosen is `1380`.

<code>xyz</code>	→	variable
<code>50</code>	→	value
<code>1380</code>	→	address



So, let us see here the same example, consider the statement `int x y z` assigned `50`. This statement means that the compiler will allocate for this `x y z` some location and put the value `50` in that

location. Suppose the address of x y z is 1380. Here x y z is a variable and 50 is the value and 1380 is the address of that variable we know that, we have discussed it earlier.

(Refer Slide Time: 07:18)

### Contd.

- During execution of the program, the system always associates the name xyz with the address 1380.
  - The value 50 can be accessed by using either the name xyz or the address 1380.
- Since memory addresses are simply numbers, they can be assigned to some variables which can be stored in memory.
  - Such variables that hold memory addresses are called pointers.
  - Since a pointer is a variable, its value is in some memory location.



Now, during execution of the program, when the program is being executed the system always associates the name x y z with the address 1380. So, whenever in the program we find x y z, the variable x y z being referred, it will go to the memory location thirteen eighty and fetch that. So, the value 50 can therefore, be accessed by going to the location 1380 and accessing it. Now, the variables which are holding these addresses are known as the pointers. Now, memory address is just numbers.

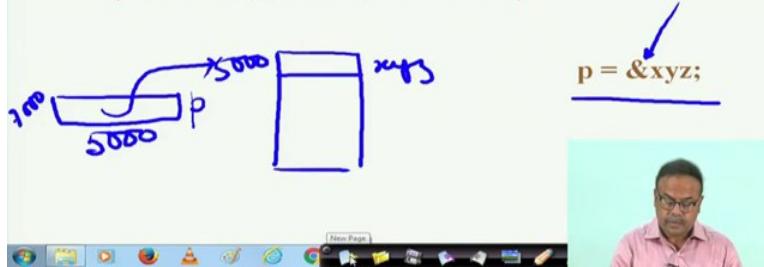
So, I can also store them in some variables and these variables which are for example, the address of the variable that will hold the address of variable x y z is the pointer to x y z. So, it is also naturally stored in some memory location.

(Refer Slide Time: 08:25)

## Contd.

- Suppose we assign the **address of xyz** to a variable **p**.

— p is said to point to the variable **xyz**.



So, this is just the example that I was discussing right now. Suppose we assign the address of x y z to a variable p, then p is said to point to the variable x y z.

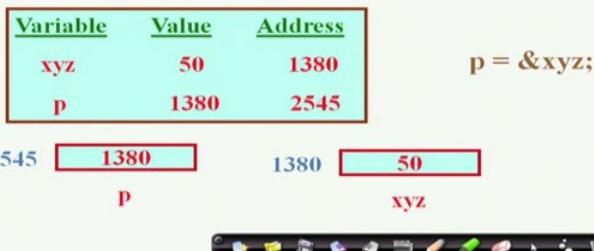
I was comfortably drawing this that there is 5000 here, x y z and 5000 here and I was simply saying that this variable which was in location 7000 was pointing to this ; that means, this was being loaded with 5000. How is that being done? How is 5000 being written inside this location? The statement is just like any other assignment. It will be that these variables name is p, p is assigned the address of x y z. So, p assigned and x y z.

(Refer Slide Time: 09:32)

## Contd.

- Suppose we assign the **address of xyz** to a variable **p**.

— p is said to point to the variable **xyz**.



Here we can see that the variable x y z is the address is 1380 value is 50. P is a pointer variable whose value is 1380 because it is pointing to x y z.

In our diagram that I was drawing by hand, this p was holding the value 5000 and its address here is say 2545, in my diagram it was 7000. So, this is a picture 1380 is the address of 50 and p which is located as 2545 is holding the address 1380. Whenever you find difficulty in dealing with pointers my suggestion to the student is to draw a piece of diagram and then make the whole picture clear in front of you.

(Refer Slide Time: 10:33)

## Accessing the Address of a Variable

- The address of a variable can be determined using the '**&**' operator.
  - The operator '**&**' immediately preceding a variable returns the **address** of the variable.
- Example:  
`p = &xyz;`
  - The **address** of xyz (1380) is assigned to p.
- The '**&**' operator can be used only with a **simple variable** or an **array element**.

`&distance  
&x[0]  
&x[i-2]`



Now you know that if I put the operator and, ampersand immediately before the variable it will return the address of the variable and x y z will give me the say the value 1380, which is the address of x y z.

The address of x y z is assigned. This operator ‘and’ can be used only with a simple variable or with an array element. For example ‘and’ distance, ‘and’ x 0; that means, the address of the first location of the array x and of x i minus 2.

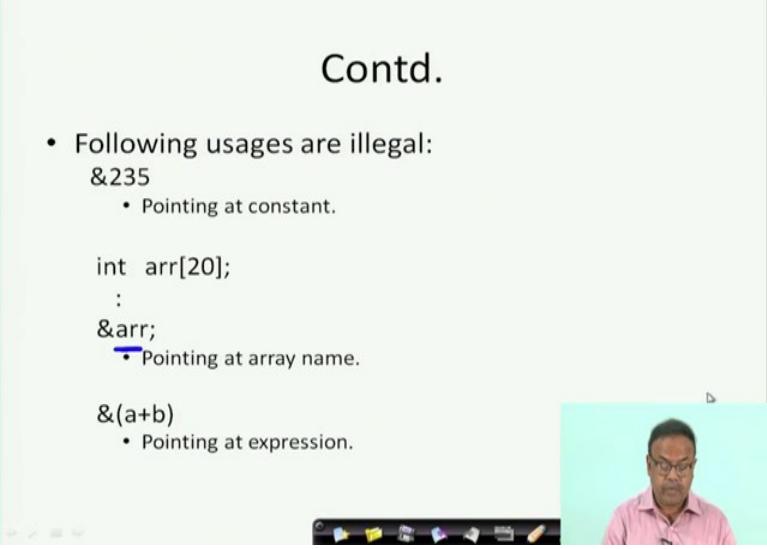
(Refer Slide Time: 11:23)

### Contd.

- Following usages are illegal:  
`&235`
  - Pointing at constant.

```
int arr[20];
:
&arr;
• Pointing at array name.
```

- Pointing at expression.  
`&(a+b)`



Now what is illegal? This is illegal; the reason is obvious. Two thirty 5 is not a variable it is a constant.

So, it does not have any fixed position in the memory. So, it is not a memory location therefore, it does not have address so its meaningless. Pointing to a constant is not possible, I cannot say and arr, because this shows that arr is a particular variable, but it is an array therefore I cannot show it like this, it is just pointing at an array name, it is not pointing at the array.

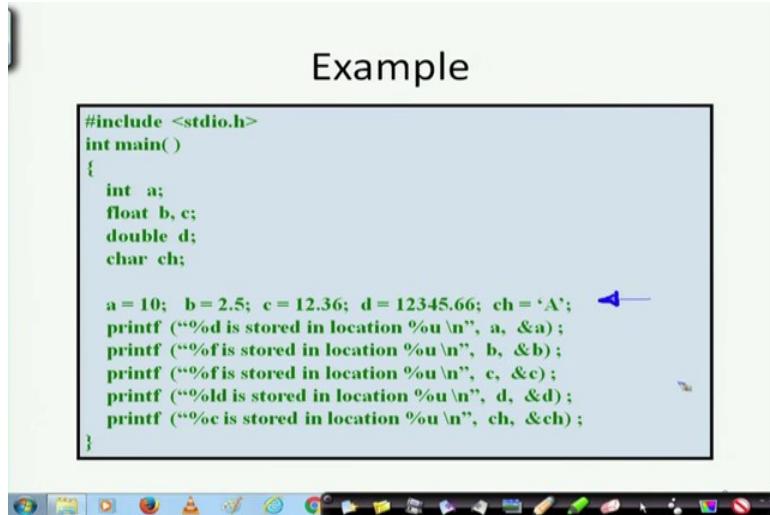
We cannot also do this and a plus b - that is also not possible because(Refer Time: 12:19) if I have a as some value and b has some value then I add them and it will be a value and a value does not have any address, that is pointing at an expression.

(Refer Slide Time: 12:26)

### Example

```
#include <stdio.h>
int main()
{
    int a;
    float b, c;
    double d;
    char ch;

    a = 10;  b = 2.5;  c = 12.36;  d = 12345.66;  ch = 'A';
    printf ("%d is stored in location %u\n", a, &a);
    printf ("%f is stored in location %u\n", b, &b);
    printf ("%f is stored in location %u\n", c, &c);
    printf ("%ld is stored in location %u\n", d, &d);
    printf ("%c is stored in location %u\n", ch, &ch);
}
```

A screenshot of a Windows operating system desktop. In the center is a terminal window titled "Terminal". Inside the terminal, there is C code enclosed in a black-bordered box. Below the code, the terminal shows the output of the program's execution. The desktop taskbar at the bottom features icons for various applications like File Explorer, Task Manager, and Control Panel.

So, here is a quick example I think I had shown this to you earlier say I have got a number of variables - one character, one double, floats. I am assigning some values to these, here I am putting some values, and print f a particular variable 'a' is stored in the address and a b is stored in address and b, c is stored in address and c so and so forth therefore, if we run it, we will get the addresses.

(Refer Slide Time: 13:12)

**Output:**

```
10 is stored in location 3221224908 [a]
2.500000 is stored in location 3221224904 [b]
12.360000 is stored in location 3221224900 [c]
12345.660000 is stored in location 3221224892 [d]
A is stored in location 3221224891 [ch]
```

Incidentally variables a,b,c,d and ch are allocated to contiguous memory locations.

So, 10 is stored in location so and so, 2.5 is stored in location so and so, all right?

Now, here incidentally all these are contiguous locations, but they may not be contiguous locations also. So, 10 is stored in location, ‘a’ having the value 10 is stored in location so, and so like that it goes on.

(Refer Slide Time: 13:46)

## Pointer Declarations

- Pointer variables must be declared before we use them.
- General form:  
`data_type *pointer_name;`

Three things are specified in the above declaration:

`int *x;`



The diagram shows a variable 'x' represented by a rectangle containing an 'x'. An arrow points from this variable to a stack of four rectangular boxes, representing memory cells. This illustrates that the pointer 'x' contains the memory address of the first cell in the stack.

So, pointer declarations, when we declare some variable as a pointer, the typical, the standard form is this. Data type shown in red (that is very important) and this star - this is something new that you also saw when we were discussing call by reference, for example, int is a type star x y z x, what does it mean? It means that x is a pointer, which is enabled or which is allowed to point to variables of type integer. Only integers can be pointed to by x.

(Refer Slide Time: 15:02)

## Pointer Declarations

- Pointer variables must be declared before we use them.
- General form:  
`data_type *pointer_name;`  
Three things are specified in the above declaration:  


So, similarly I could have said float star p ; that means, p will be a pointer, that can only be pointing to floating point numbers. So, three things are specified in this disk in this declaration.

(Refer Slide Time: 15:33)

The screenshot shows a presentation slide with a light blue background. At the top center, the title 'Pointer Declarations' is displayed in a bold, black font. Below the title, there is a bulleted list of two items:

- Pointer variables must be declared before we use them.
- General form:

Under the 'General form:' bullet point, a code snippet is shown in red text:

```
data_type *pointer_name;
```

Below the code snippet, the text 'Three things are specified in the above declaration:' is written in black. Following this, a numbered list of three items is provided:

1. The asterisk (\*) tells that the variable `pointer_name` is a pointer variable.
2. `pointer_name` needs a memory location.
3. `pointer_name` points to a variable of type `data_type`.

At the bottom of the slide, there is a standard Windows-style taskbar with icons for various applications like Internet Explorer, File Explorer, and Google Chrome. The date and time '2:00 PM 3/21/2013' are also visible at the bottom right of the slide area.

One is that this star tells that the variable, pointer name or p whatever you call it, is a pointer variable, not a normal variable and asterisk is telling that it is a pointer variable. Pointer name is a variable therefore, it needs a memory location and pointer name points to a variable of the specified data type. So, these three things you must remember when you are handling with the pointer.(Refer Slide Time: 16:08)

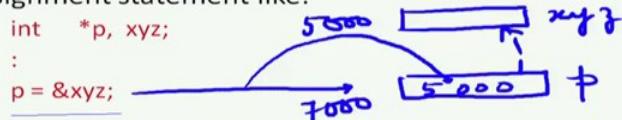
## Contd.

- Example:

```
int *count;  
float *speed;
```

- Once a pointer variable has been declared, it can be made to point to a variable using an assignment statement like:

```
int *p, xyz;  
:  
p = &xyz;
```



So, here is an example int star count what does it mean? That means, count is a pointer variable. Why a pointer variable? Because this is preceded with the star and where can this pointer variable point to? To all data type data variables of type integer. Speed is again a pointer because it has got this asterisk and where can speed point to? Speed can point to variables of type float. Once a pointer variable has been declared, it can be made to point to a variable using an assignment statement like this - int star p x y z.

So, int star p is a pointer variable, x y z is an integer. So, you see by the same declaration I have declared two things - one is a pointer to an integer and an integer. So, if I make such an assignment like p is assigned and x y z. So, x y z is here and its location can be 5000 and p is a pointer variable and so when I do this assignment, this might be in the location 7000, but when I do this what it does is it loads this 5000 here. So, now, p is pointing to x y z. I hope it is clear now.

(Refer Slide Time: 17:50)

## Contd.

- Example:

```
int *count;  
float *speed;
```

- Once a pointer variable has been declared, it can be made to point to a variable using an assignment statement like:

```
int *p, xyz;
```

```
:
```

```
p = &xyz;
```

– This is called **pointer initialization**.

So, this is called pointer initialisation.

(Refer Slide Time: 17:51)

## Things to Remember

- Pointer variables must always point to a data item of the *same type*.

```
float x; ·  
int *p;  
: → will result in erroneous output  
p = &x; ←
```

- Assigning an absolute address to a pointer variable is prohibited.

```
int *count;  
:  
count = 1268;
```

The things to remember is the pointer variables must always point to an item of the same type. One pointer variable, either it points to an integer or it points to a float or it points to an array of characters whatever it is.

Now so, here for example, this is an error, why it is this is an error? X is of type floating point, x is a variable and p is a pointer which is allowed to point to only integer, but here I have made an assignment where I am assigning to p, the address of the floating point number - that is not allowed.

So, therefore, I am forcing p to point to x, but the type of p and the type of x are different. So, assigning an absolute address to a pointer is prohibited. You cannot do this, you cannot force a pointer to a constant value, that you must keep in mind.

(Refer Slide Time: 19:00)

### Accessing a Variable Through its Pointer

- Once a pointer has been assigned the address of a variable, the value of the variable can be accessed using the indirection operator (\*).

int a, b;  
int \*p;  
:  
✓ p = &a; →  
b = \*p; ⇒ b = a  
↓ b is assigned the val of a  
b is assigned the val of the var pointed by p

Equivalent to

The diagram illustrates the memory layout. At address 100, there is a box labeled 'a' containing the value '50'. Below it, another box contains the value '50' with an arrow pointing to it from the text 'b is assigned the val of a'. To the left, there is a pointer variable 'p' containing the address '100'. A red arrow labeled 'Equivalent to' points from the code 'b = \*p;' to this diagram. Handwritten notes explain that 'b' is assigned the value of 'a' and then the value of the variable pointed to by 'p'.

So, how do I access a variable through a pointer? Once a pointer has been assigned the address of a variable, the value of the variable can be accessed through the indirection operation. So, let us give an example first.

For example, a b are two integers. Let us draw this I am sorry, suppose a is a location, b is another location, both of them are integers and p is a variable, which is allowed to point to

integers and what I do? I assign to p the address of a. So, the address of 'a' let us say is 1000 or 100. So, 100 is written here. So, it comes here.

So, what did this statement do? 100 was the address of a, that has been assigned to p. Now what is being done by this? B is being assigned star p; b is getting the content of wherever p is pointing. Suppose this was 50, then b is getting 50. So, little bit of confusion can occur because of these two star ps. Please understand that this star is just telling you that p is a pointer because it is coming in a declaration statement. On the other hand here it's is not a declaration statement it is an assignment statement. I have already declared p to be a pointer. So, p is a pointer.

So, once a pointer has been assigned the address of a variable, that is done in this step the value of the variable can be accessed using the indirection operation; that means, which variable say I could have done p assigned b assigned a. Instead of doing that what I am saying is (just note my two English statements) one is the value of a is being assigned to b or b is being assigned the value of a this is statement number 1; statement number 2 is b is being assigned the value of the variable that is being pointed at by p .

So, here one is b is assigned the value of a; one statement. Here that is this statement. And this statement is b is assigned the value (instead 'of' a, I am saying is being assigned the value) of the variable pointed by p.

So, this is an indirect way of saying that. So, this is something that you must very clearly understand, look at this once again.

(Refer Slide Time: 23:21)

## Accessing a Variable Through its Pointer

- Once a pointer has been assigned the **address** of a variable, the **value** of the variable can be accessed using the **indirection operator** (\*).

```
int a, b;  
int *p;           Equivalent to      b = a  
:  
p = &a;  
b = *p;
```

So, I can now say that this thing is equivalent to b assigned a, but I have done that in a indirect way.

(Refer Slide Time: 23:21)

### Example 1

```
#include <stdio.h>  
main()  
{  
    int a, b;  
    int c = 5;  
    int *p;  
  
    a = 4 * (c + 5);  
  
    p = &c;  
    b = 4 * (*p + 5);  
    printf ("a=%d b=%d\n", a, b);  
}
```

*Equivalent*

*10  
\*4  
40 → a*

*100 → c  
100 → p  
b*

*4 \* (5+5)*

If you have understood this then pointer should be clear to you, here lets look at one example integer a b c assign 5 and p is a pointer to integers. Now, in this statement what has been done?

'a' has been assigned 4 times c plus 5. So, c is being added to 5. So, it is becoming 10 and 4 times 10 is 40, 40 is being assigned to a. Here what is being done? P is being assigned, c is address. So, wherever c was let's say address 1000. So, that is being assigned to a variable p. So, p is becoming 1000.

Now, b what is b being assigned? b is being assigned 4 times star p plus 5. What is star p? Star p is nothing, but the variable c, it's pointing to that and whatever was its value suppose it was 10, it was sorry it was 5. So, that value is being taken and 5 is being added to that. So, it becomes 10 times this. So, these two are essentially equivalent.

(Refer Slide Time: 25:17)

Example 2

```
#include <stdio.h>
main()
{
    int x,y;
    int *ptr;
    x = 10;
    ptr = &x;
    y = *ptr;
    printf ("%d is stored in location %u \n", x, &x);
    printf ("%d is stored in location %u \n", *&x, &x);
    printf ("%d is stored in location %u \n", *ptr, ptr);
    printf ("%d is stored in location %u \n", y, &*ptr);
    printf ("%u is stored in location %u \n", ptr, &ptr);
    printf ("%d is stored in location %u \n", y, &y);

    *ptr = 25;
    printf ("\nNow x = %d \n", x);
}
```

So, here is another example, you have seen this example very similar to this that I am defining a pointer, p t r is a pointer, x is 10, you just think of what it should print? X is 10, p t r is the address of x and y is being assigned star p t r, what does this mean? P t r is pointing to 10 and y is being assigned to star p t r; that means, y is being assigned to 10 or rather I should say y is being assigned x and x was 10. So, y is become becoming 10.

Now, if I print, instead of x what will be printed? Instead of x the, value of x will be printed. 10 is stored in the address this. Similarly now what is this? ‘star and x’ - what does this mean? ‘and x’ is the address of x and where that is pointing to - these two should be same.

So, if you look at if you look at the result - this is equivalent why? ‘And x’ is the address of x and star just like star p; that means, the content of where the pointer is pointing to. So, these two are same right.

(Refer Slide Time: 27:18)

### Example 2

```
#include <stdio.h>
main()
{
    int x,y;
    int *ptr;

    x = 10 ;
    ptr = &x ;
    y = *ptr;
    printf ("%d is stored in location %u \n", x, &x);
    printf ("%d is stored in location %u \n", *x, &x);
    printf ("%d is stored in location %u \n", *ptr, ptr);
    printf ("%d is stored in location %u \n", y, &ptr);
    printf ("%u is stored in location %u \n", ptr, &ptr);
    printf ("%d is stored in location %u \n", y, &y);

    *ptr = 25;
    printf ("\nNow x = %d \n", x);
}
```



So, ptr is ‘and x’. So, these two are all equivalent ok. If I say ‘and x’ ; that means, I am taking the address of star ptr.

(Refer Slide Time: 27:37)

#### Output:

```
10 is stored in location 3221224908
3221224908 is stored in location 3221224900
10 is stored in location 3221224904
```

```
Now x = 25
```

Address of x: 3221224908

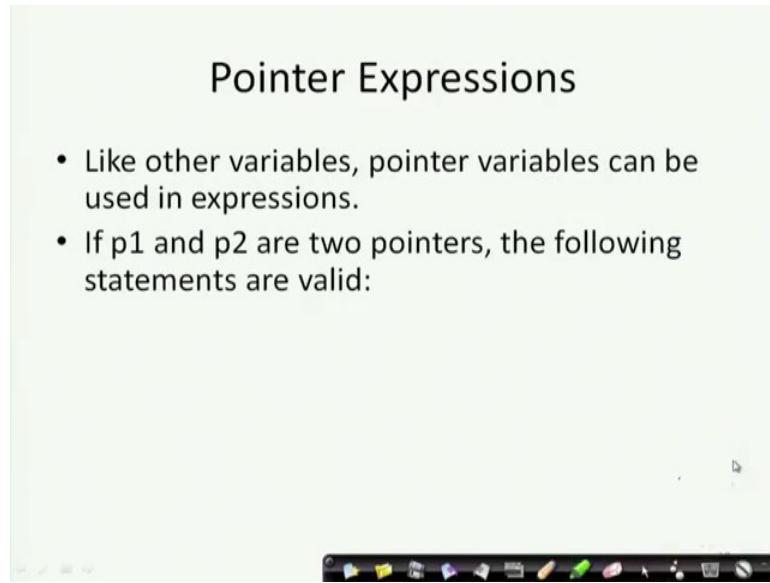
Address of y: 3221224904

Address of ptr: 3221224900



So, this is something you can play with yourself and you will get a printout like this. So, these two are the same.

(Refer Slide Time: 27:46)



Now, here we are coming to something more which is known as pointer expressions and we will deal with that in the next lecture.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 59**  
**Pointer (contd.)**

So, we were discussing about pointers and we have seen that pointer is a variable. So, it is a variable, but a variable that points to some other variable, but naturally the question that can arise is since pointers are variables so, you should be able to do some sort of operations like arithmetic operations on them. So, the answer is yes we can do that and in today's lecture we will look at exactly that - pointer expressions. (Refer Slide Time: 00:46)

The slide has a light gray background. At the top center, the title 'Pointer Expressions' is written in a large, black, sans-serif font. Below the title, there are two bulleted points in a smaller black font:

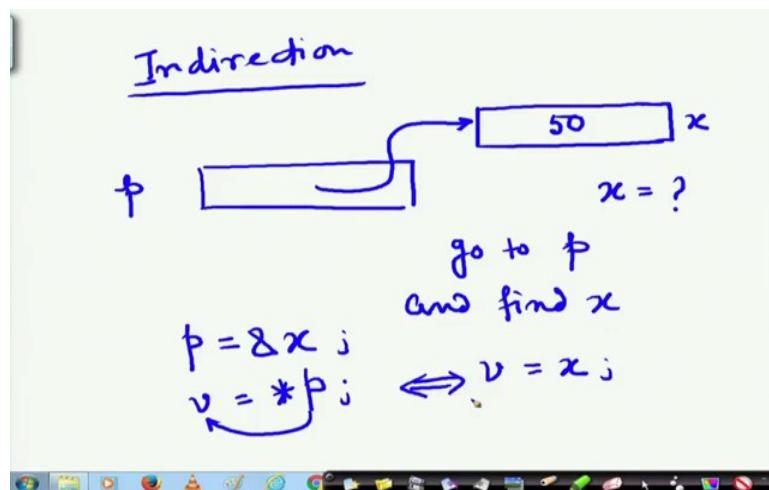
- Like other variables, pointer variables can be used in expressions.
- If p1 and p2 are two pointers, the following statements are valid:

Below the second bullet point, there is a handwritten-style diagram. It shows a horizontal line with two arrows pointing to boxes labeled '50' and '60'. Above this line, the text 'sum = \*p1 + \*p2;' is written. To the left of the first arrow, there is a handwritten '110' with an upward-pointing arrow. To the right of the line, there is a small video player window showing a man in a pink shirt. At the bottom of the slide, there is a standard Windows-style taskbar with various icons.

Now, this concept of pointers is a very strong component of the c language and it is not the case that in all languages this pointer is there, but we are discussing pointers specifically, because it will give you a very good idea about what indirection is. We had mentioned about indirection right.

So, for example, I am just before moving into the actual discussion, let me come to this that - what is an indirection ?

(Refer Slide Time: 01:20)



Somebody asks you the address of toms house, you can do two things - you can either give him the address of toms house or you can give him the address of johns house. So, that he goes to johns house and asks john to get the address of toms house. So, I do not know toms address, but I know john who knows toms address. This is indirection - one level of indirection.

Another level of indirection could be that (second level of indirection). You can go to ram and ram will know the address of john who knows the address of tom. So, that is second level of indirection. So, in our case we have got a particular variable  $p$ , let me just call this variable  $x$ , and that  $x$  has got some value. I am not saying that  $x$  has also got an address.

But I am not saying what is the value of  $x$ ? What is  $x$ ? That is my question and instead of answering that, I am giving you a pointer to  $x$  and I am asking that ok I tell you I will give you the answer, go to  $p$  and find  $x$  right. So, then  $p$  must be assigned the address of  $x$ . So, I go to this, then I come to  $p$  and get the value of  $x$  as I want to have that in some other variable  $v$ , where I want to have  $*p$ .

So, indirectly I could have done simply - v assigned x. These two are equivalent, that is what we discussed in the last class right. That is why it is an indirection that often comes in very handy, very useful, when we carry out many computations.

So, like other variables pointer variables can also be used in expressions. If p 1 and p 2 are two pointers the following statements are valid - star p 1 plus star p 2. So, what is being meant by that? Suppose p 1 is pointing to something, where there is 50 and p 2 is pointing to something which is stored as 60 - then what is sum? Sum becomes 50 plus 60, star p 1 plus star p 2, 50 plus 60 so, that should be 110.

(Refer Slide Time: 04:55)

## Pointer Expressions

- Like other variables, pointer variables can be used in expressions.
- If p1 and p2 are two pointers, the following statements are valid:  
`sum = *p1 + *p2;  
prod = *p1 * *p2;  
prod = (*p1) * (*p2);  
*p1 = *p1 + 2;`

The diagram shows a pointer variable `p1` pointing to a memory location containing the value `50`. Another pointer variable `p2` points to a memory location containing the value `60`. The expression `*p1 + 2` is evaluated as `50 + 2`, resulting in `52`.

Next similarly, I can have this. But here it would be easier nicer to read, if I had put parentheses. So, that I had not confused about this Asterix and this Asterix. These have got completely different significances, this is a multiplication and this asterix is saying it is a content of a particular pointer.

So, similarly the these two are equivalent of course, I have already shown that. Now this is also possible star p 1. So, in my earlier drawing p 1 was 50, 50 plus 2 - that is coming over here. So, whenever you are finding difficulty as I suggested just draw a simple diagram; p 1 is pointing to some other variable which has got the value 40.

So, star p 1 is what? Star p 1 is that particular - variable x and plus 2 that is equivalent to x plus 2 which is 42. So, this one is being modified to 42.

(Refer Slide Time: 06:29)

## Pointer Expressions

- Like other variables, pointer variables can be used in expressions.
- If p1 and p2 are two pointers, the following statements are valid:

```
sum = *p1 + *p2;
prod = *p1 * *p2;
prod = (*p1) * (*p2);
*p1 = *p1 + 2;
x = *p1 / *p2 + 5;
```

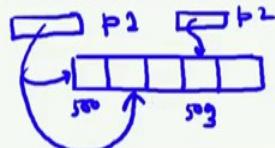


Similarly, I can do other operations like this where you must understand that this is actually just like another variable - an integer variable or whatever type of variable p1 is and it is simple. No other complications in that.

(Refer Slide Time: 06:48)

## Contd.

- What are allowed in C?
  - Add an integer to a pointer.
  - Subtract an integer from a pointer.
  - Subtract one pointer from another (related).
    - If  $p_1$  and  $p_2$  are both pointers to the same array, then  $p_2 - p_1$  gives the number of elements between  $p_1$  and  $p_2$ .
- What are not allowed?



Now, what are allowed in C? There are certain things are allowed in C and some things are not allowed in C, I can add an integer to a pointer, I can subtract an integer from a pointer, I can subtract one pointer from another and say if  $p_1$  and  $p_2$  are both pointers to the same array, then  $p_2$  minus  $p_1$  gives the number of elements between  $p_1$  and  $p_2$ . For example, suppose here there is an array. So,  $p_1$  is pointing here and  $p_2$  is pointing here. Then the number of elements  $p_2$  minus  $p_1$  will be just the subtraction of these addresses.

Suppose this was 500, 501 (if it is a character) 502, 503. So, I have got three elements in between right. So, these are all allowed. I can subtract an integer from a pointer. I can add an integer to a pointer. So, if I add an integer to a pointer that is  $p_1$ ,  $p_1$  plus 1. So, that means it will point to this point, this element. What are not allowed? The things that are not allowed are you cannot add two pointers.

(Refer Slide Time: 08:22)

## Contd.

- What are allowed in C?
  - Add an integer to a pointer.
  - Subtract an integer from a pointer.
  - Subtract one pointer from another (related).
    - If **p1** and **p2** are both pointers to the same array, then **p2 - p1** gives the number of elements between **p1** and **p2**.
- What are not allowed?
  - Add two pointers.  
 $p1 = p1 + p2;$
  - Multiply / divide a pointer in an expression.  
 $p1 = p2 / 5;$   
 $p1 = p1 - p2 * 10;$



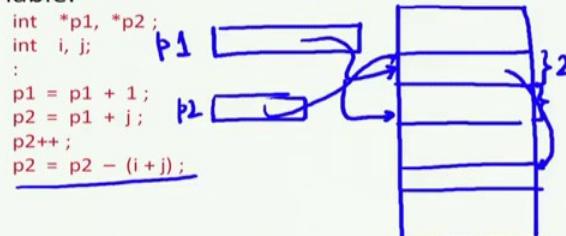
The reason is obvious - **p 1** is a pointer and **p 2** is a pointer. Now these two are two different locations. So, say **p 1** is in location 7000 pointing to some variable, and **p 2** is in location 10,000 pointing to some other variable.

So, what does **p 1** plus **p 2** mean? 7,000 plus 10,000 - 17,000 that does not mean anything. That can be a point to some garbage value or something else; So, that is not allowed. Compiler will hold you for that, multiply or divide a pointer in an expression, that is also not allowed. You cannot multiply, you can just add an integer, subtract an integer or subtract one pointer from another, these three you can do.

(Refer Slide Time: 09:14)

## Scale Factor

- We have seen that an integer value can be added to or subtracted from a pointer variable.



Here is something which is known as a scale factor, let us see whether you understand this or not. We have seen that an integer value can be added or subtracted.

So, here let us look at this, p 1 and p 2 are two pointers of type integer. I mean when p 1 is pointing to an integer, p 2 is also pointing to an integer all right. Now i,j are two integer variables. p 1 is p 1 plus 1 so that means what? p 1 was pointing somewhere, but I am just adding some constant to that. So, here is a memory location and say p 1 is pointing to this and suppose it is an integer so, this is pointing to an integer. Now p 1 plus one means this will point to the next integer.

Now, I am not saying that if an integer takes two bytes each of them are of two bytes. So, it just comes to the next integer. That is why the type is important. Depending on the type it is updating either by 2 or by 1, but p 1 plus 1 means I am going to the next. p 2 is assigned; p 1 plus j where j can be something.

So, p 2 was pointing somewhere here, I am upgrading that with the j value, going to the j th next. Similarly p 2 plus plus, p 2 assigned any arithmetic operation I can do all right.

(Refer Slide Time: 11:15)

## Scale Factor

- We have seen that an integer value can be added to or subtracted from a pointer variable.

```
int *p1, *p2 ;  
int i, j;  
:  
p1 = p1 + 1;  
p2 = p1 + j;  
p2++;  
p2 = p2 - (i + j);
```
- In reality, it is not the integer value which is added/subtracted, but rather the **scale factor** times **the value**.

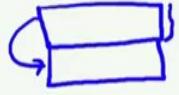


In reality it is not the integer value which is added or subtracted, but rather the scale factor times the value - that is one times the size of the integer if it were. So, it is  $j$  times the size of an integer, 2 bytes, 4 bytes - that is what that is why it is called the scale factor. So, this is not this 1, but one next, two next, here  $j$  th next, it is like that.

(Refer Slide Time: 11:50)

Contd.

Data Type	Scale Factor
char	1
int	4
float	4
double	8



– If  $p1$  is an integer pointer, then  
 $p1++$   
will increment the value of  $p1$  by 4.



So, for character the scale factor is 1, integer is 4, if 4 bytes take one integer, float 4. Now this could be 2 - that depends on what the scale factor is. So, if I write here for an integer pointer, assuming the my computer is actually doing this - that each of them is 4, then  $p1$  plus plus is adding  $p1$  by 4, so it is going to the next integer.

Now, in between there are 4 bytes. So, I am going to the next integer.

(Refer Slide Time: 12:31)

## Example: to find the scale factors

```
#include <stdio.h>
main()
{
    printf ("Number of bytes occupied by int is %d \n", sizeof(int));
    printf ("Number of bytes occupied by float is %d \n", sizeof(float));
    printf ("Number of bytes occupied by double is %d \n", sizeof(double));
    printf ("Number of bytes occupied by char is %d \n", sizeof(char));
}
```

sizeof (int)  
sizeof (float)



So, there is one quick way of finding out, how we can find out what is the representation in my system? There is a nice inbuilt function called size of. So, if I give size of int, the system returns me the value 4 or 2 depending on how much, how many bytes does int consume.

Similarly, I could have given size of float - that will tell me how many bytes has a float consumed so and so forth. So, that is one way to find the scale factors. So, number of bytes occupied by float is size of float. If you give that the system will give you the number of bytes required for that representation.

(Refer Slide Time: 13:28)

Returns no. of bytes required for data type representation

```
#include <stdio.h>
main()
{
    printf ("Number of bytes occupied by int is %d \n", sizeof(int));
    printf ("Number of bytes occupied by float is %d \n", sizeof(float));
    printf ("Number of bytes occupied by double is %d \n", sizeof(double));
    printf ("Number of bytes occupied by char is %d \n", sizeof(char));
}
```

Output:

```
Number of bytes occupied by int is 4
Number of bytes occupied by float is 4
Number of bytes occupied by double is 8
Number of bytes occupied by char is 1
```



So, if in a system you would run this and you find that it is float is 4, int is 4 then you know what my scale factor is.

(Refer Slide Time: 13:40)

### Passing Pointers to a Function

- Pointers are often passed to a function as arguments.
  - Allows data items within the calling program to be accessed by the function, altered, and then returned to the calling program in altered form.
  - Called **call-by-reference** (or by **address** or by **location**).
- Normally, arguments are passed to a function **by value**.
  - The data items are copied to the function.
  - Changes are not reflected in the calling program.



Now, just like and for every case we are thinking of how do we pass an array to a function, how do we pass a structure to a function, here again we look at how do we pass a pointer to a function. Pointers are often passed as parameters to a function and if you have thought about it you must have already discovered. Now, always it allows the data items, within the calling programs to be accessed by the function, altered and then returned to the calling function in the altered form, this says the calling by reference.

Normally arguments are passed to a function by value, we had discussed this. Now this is called as call by reference or call by address, now you can see this - how this is done. Because in call by value we have seen that in the swap function for example, it was swapped within the function, but that x y and the main functions x y were two different entities, therefore whatever change was there that was lost.

But if I had just passed on the pointer then whatever change I do in the pointer in that particular location, I simply pass on the address and make a change over there, then the change is reflected because it is the same location that is known as call by reference.

(Refer Slide Time: 15:15)

## Example: passing arguments by value

```
#include <stdio.h>
main()
{
    int a, b;
    a = 5; b = 20;
    swap(a, b);
    printf ("\n a = %d, b = %d", a, b);
}

void swap (int x, int y)
{
    int t;
    t = x;
    x = y;
    y = t;
}
```

a and b  
do not  
swap

x and y swap

### Output

a = 5, b = 20



Now, here is an example, you have seen this passing by value, I am repeating this - a was 5 and b was 20, I call swap a b. Here, what happened ? x was 5, b was 20. I changed it. Now x was 20 and b was 5, but when I returned and printed this a and b they were completely different. So, swapping was not reflected. Therefore, here x and y was swapped but a and b do not swap. So, the output will be the same 5 and 20.

(Refer Slide Time: 15:57)

Example: passing arguments by reference

```
#include <stdio.h>
main()
{
    int a, b;
    a = 5; b = 20;
    swap (&a, &b);
    printf ("\n a = %d, b = %d", a, b);
}

void swap (int *x, int *y)
{
    int t;
    t = *x;
    *x = *y;
    *y = t;
}
```

But if I had done through reference, here you see how I pass it on. Here what I am passing you have must have thought about it. Swap instead of sending that value, what I am sending is the address of that, address of a and address of b and here inside the function, what I am accepting? I am accepting the pointer.

So, here (x and y) I know that what is coming to me is a pointer. We discussed it earlier also, but let me repeat because there is a very fundamental idea, a is 5, b is 20 and now I swap. So, here do I have x? No. I have got this x which is nothing, but 'and a' and I have got this y which is nothing, but 'and b'. So, therefore, they are pointing to these points.

Now, when I swap, I am actually swapping the content of x. So, the content of x ; that means, here that is going to t and then content of y, indirection, I go from here, follow my cursor. I go here and that one is going as the content of a, content of x. So, here it is becoming 20 and then t is coming as a content of y. What is y? y is here. So, this is coming as 5.

So, when I come out and print here - a and b as you can see has changed. So, you see how did I pass on the parameter? Look at this - I have passed on the address and I have accepted them in my function as the pointer.

(Refer Slide Time: 18:12)

### Example: passing arguments by reference

```
#include <stdio.h>
main()
{
    int a, b;
    a = 5 ;  b = 20 ;
    swap (&a, &b);
    printf ("\\n a = %d, b = %d", a, b);
}

void swap (int *x, int *y)
{
    int t ;
    t = *x ;
    *x = *y ;
    *y = t ;
}
```

**\*(&a) and \*(&b)**  
**swap**

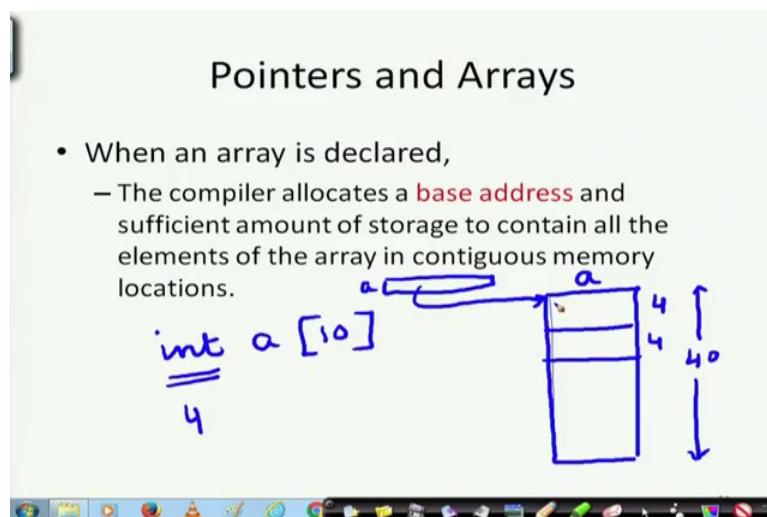
**\*x and \*y**  
**swap**

**Output**

a = 20, b = 5

So, with this we will get the correct answer x and y swap a and b also swap. So, the answer is - that is correct one as we expected. So, now let us skip this a little bit and let us go to something else as pointers and arrays.

(Refer Slide Time: 18:36)



When an array is declared, the compiler allocates a base address and sufficient amount of storage. You know that when I declare something as an array, int a[10] and if I say that int, I have discovered using size of fact that int takes 4 bytes, then for every element 4 bytes are kept and 40 such locations are allocated for the array a.

Now when I refer to this array a, because we saw that we pass on an array by reference to a function the reason behind that is that this name array a is the same as a pointer to the first location of the array. So, it is as if equivalent to a is a pointer that is pointing to the first location of this array a. They are equivalent.

(Refer Slide Time: 20:04)

The screenshot shows a presentation slide with a light gray background. At the top center, the title 'Pointers and Arrays' is displayed in a bold, black font. Below the title, there is a bulleted list of four items, each preceded by a black bullet point. The text is color-coded: some words like 'base address' and 'constant pointer' are highlighted in red. The slide has a standard Windows-style taskbar at the bottom with icons for various applications like File Explorer, Internet Explorer, and others.

- When an array is declared,
  - The compiler allocates a **base address** and sufficient amount of storage to contain all the elements of the array in contiguous memory locations.
  - The **base address** is the location of the first element (index 0) of the array.
  - The compiler also defines the array name as a **constant pointer** to the first element.

So, the base address is a location of the first element of the array, the compiler also defines the array name as a constant pointer to the first. When I declared it with the compiler it also keeps a constant pointer, that pointer you cannot change ok.

(Refer Slide Time: 20:25)

## Example

- Consider the declaration:  
`int x[5] = {1, 2, 3, 4, 5};`  
— Suppose that the base address of x is 2500,  
and each integer requires 4 bytes.

Element	Value	Address
x[0]	1	2500
x[1]	2	2504
x[2]	3	2508
x[3]	4	2512
x[4]	5	2516



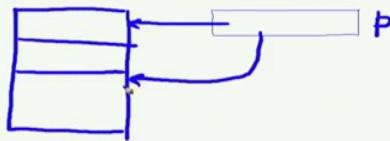
For example, when I say `x[5] = {1, 2, 3, 4, 5}` then suppose the base address is 2500, each integer requires 4 bytes, then the elements will be `x[0]` will be 2500, `x[1]` will be 2504 so and so forth and the pointer will be 2500.

(Refer Slide Time: 20:43)

## Contd.

`x ⇔ &x[0] ⇔ 2500;`

- `p = x;` and `p = &x[0];` are equivalent.
- We can access successive values of x by using `p++` or `p-` to move from one element to another.



If we go up 2500. So, there is a pointer x - which means ‘and address of x 0’, which is 2500 .

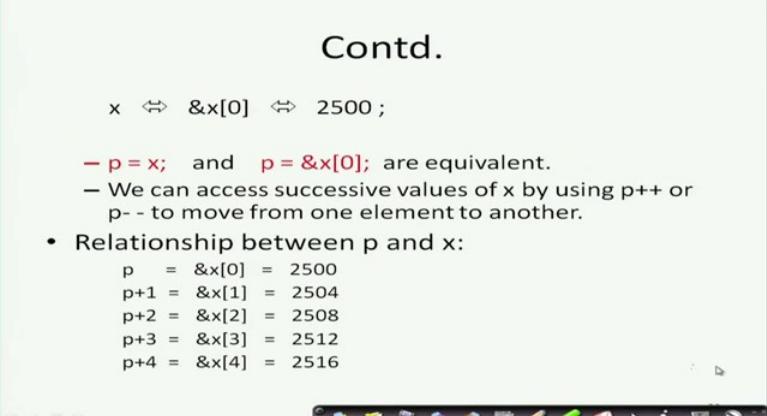
So, `p` assigned `x` and `p` assigned ‘and of `x[0]`; are equivalent. So, we can access successive values of `x` by using `p plus plus` or `p minus minus` to move from one element to another.

So, I have got the p, I have got this pointer which is pointing to the array. I cannot change that pointer, but if I do p plus plus - this pointing to the first element, then I go to the next element of the array record by the scale factor. So, this is if I do p plus plus I am actually doing p plus 4. In that way I can move across.

(Refer Slide Time: 21:56)

**Contd.**

```
x  ⇌  &x[0]  ⇌  2500 ;  
  
— p = x; and p = &x[0]; are equivalent.  
— We can access successive values of x by using p++ or  
p- - to move from one element to another.  
• Relationship between p and x:  
    p    =  &x[0]  =  2500  
    p+1  =  &x[1]  =  2504  
    p+2  =  &x[2]  =  2508  
    p+3  =  &x[3]  =  2512  
    p+4  =  &x[4]  =  2516
```



So, the relationships should be clear here. So, p plus 1 is the next one, p plus 2 is the next one - that we have already explained.

(Refer Slide Time: 22:02)

Example: function to find average

```

#include <stdio.h>
main()
{
    int x[100], k, n;
    scanf ("%d", &n);
    for (k=0; k<n; k++)
        scanf ("%d", &x[k]);
    printf ("\nAverage is %f", avg (x, n));
}

float avg (int array[], int size)
{
    int *p, i, sum = 0;
    p = array;
    for (i=0; i<size; i++)
        sum = sum + *(p+i);
    return ((float)sum / size);
}

```

The diagram illustrates the execution flow between the main program and the avg function. A blue arrow points from the line 'avg (x, n);' in the main program to the avg function definition. Inside the avg function, another blue arrow points from the variable 'p' to the first element of an array shown below, with a note '(float)' next to it. A third blue arrow points from the expression '\*(p+i)' to the same array element. The array is depicted as a horizontal row of boxes, each containing a value.

So, here is a function to find average, here you see we have got a main program, where I have got an array, 100 elements, for  $k$  0 to  $n$ , I am reading  $k$  and then I am calling this average  $x$ . I am calling average  $x$   $n$ . Now, here what goes in average  $x$   $n$ , you have seen this, now I am passing the pointer. When I am saying I am just passing the array actually I have passed on the pointer. So, whatever I did I am doing here. I am taking another star  $p$ , which is local which is pointing to the array.

So, my array was here and I am putting another pointer  $p$  which is pointing to this array and array means what? Array means the first location of the array. Then I carry on the sum here. I carry on with  $p$ . Here you see what I do. I take  $p$  and then I change  $p$ . Sum assigns star  $p$  plus  $i$ . So,  $p$  plus 1,  $p$  plus 2,  $p$  plus 3 and star  $p$  plus 1,  $p$  plus 2 means the content of this; these contents.

So, here in this way I am getting the sum and what do I return? Return float sum by size. So, I get sum; obviously, the array was integer, but now this is something called typecasting- although it was float, I put it in a bracket. That means, whatever is coming here I am converting that to float. Sum will be an array, sum of all integers will be an integer, but when I divide by that, although I did not declare 'sum' to be a floating point number, just by this sort of type typecasting.

I can make it a floating point number and then how do I return it? Yes you might have guessed correctly, that I do not need to return it because whatever I have done, here when I passed an

array, whatever was done that is being done here and what I am returning? I am returning this to the average. So, I will get this value all right. So, this should be clear.

(Refer Slide Time: 25:15)

### Example: function to find average

```
#include <stdio.h>
main()
{
    int x[100], k, n ;
    scanf ("%d", &n) ;
    for (k=0; k<n; k++)
        scanf ("%d", &x[k]) ;
    printf ("\nAverage is %f",
           avg (x, n)) ;
}

float avg (int array[], int size)
{
    int *p, i, sum = 0;
    p = array ;
    for (i=0; i<size; i++)
        sum = sum + *(p+i);
    return ((float)sum / size);
}
```

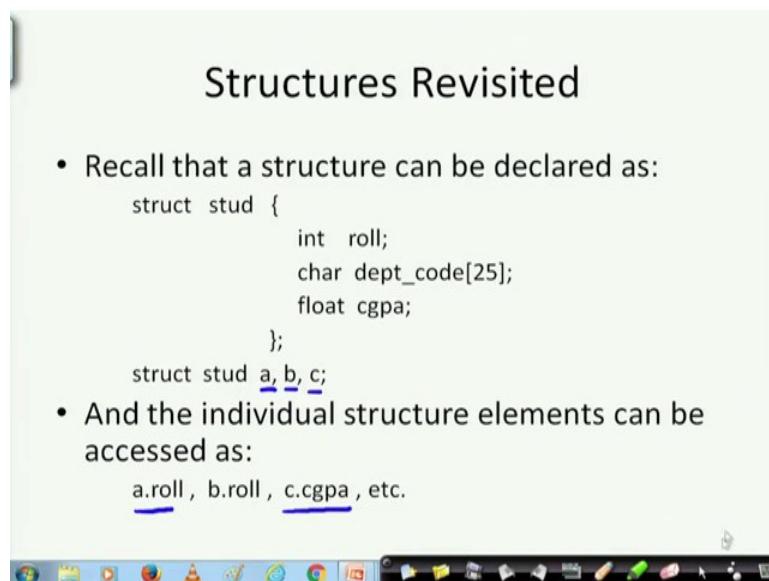
So, its clarified. Now,next thing that I will be discussing a little bit is dynamic memory allocation, I will take little time to explain that and that is a very important concept and after that we will move to a some discussions, basic discussions on file.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 60**  
**Pointer in Structures**

We have seen how structures are represented, we have also learnt about pointers.

(Refer Slide Time: 00:25)



The screenshot shows a presentation slide with a light green background. The title 'Structures Revisited' is centered at the top in a bold, black font. Below the title, there are two bullet points. The first bullet point states: 'Recall that a structure can be declared as:' followed by a code snippet. The second bullet point states: 'And the individual structure elements can be accessed as:' followed by another code snippet. At the bottom of the slide, there is a decorative footer bar featuring various icons, including a search icon, a file icon, and a power icon. The overall layout is clean and professional, typical of a university lecture slide.

- Recall that a structure can be declared as:

```
struct stud {  
    int roll;  
    char dept_code[25];  
    float cgpa;  
};  
struct stud a, b, c;
```
- And the individual structure elements can be accessed as:  
a.roll , b.roll , c.cgpa , etc.

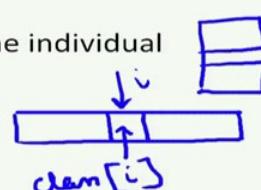
So, today we will be looking at the structures once again in a different light. As you can see here, a structure can be declared as is shown here the student structure struct stud is consisting of three members or three fields.

So, they are - the role, the department code which is a character array and cgpa which is a floating point number and a b c are three variables of the type s t u d, stud. The individual structure elements you now can be accessed by a dot role. So, a is this field or b dot role c dot c g p a. So, with this dot operator we can access them. This was known to us right.

(Refer Slide Time: 01:19)

## Arrays of Structures

- We can define an array of structure records as  
`struct stud class[100];`
- The structure elements of the individual records can be accessed as:  
`class[i].roll`  
`class[20].dept_code`  
`class[k++].cgpa`



The diagram illustrates the memory layout of an array of structures. It shows a horizontal bar labeled 'class[i]' with indices 'i-1' and 'i'. Above the bar, there are three separate boxes representing the fields of a structure: 'roll', 'dept\_code', and 'cgpa'. A blue arrow points from the text 'class[i].roll' to the 'roll' box.

Now, we can also see that, we can define an array of structures where class is an array of students, class size is 100 and each element is a structure of type stud. The structure elements of the individual records can be accessed with this dot operator like class i, any particular element of that, say for example, here. So, this is class i where i is this index, I come over here, and get a particular field of this. So, this has got a number of fields. So, I am coming to the role field or the department code field dot the cgpa field. This was also known to us.

(Refer Slide Time: 02:18)

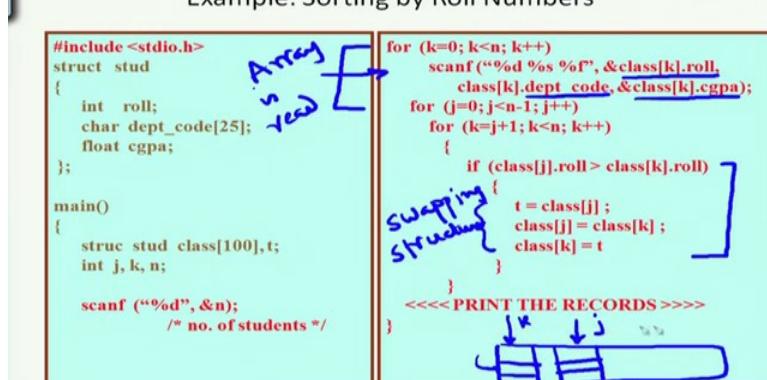
### Example: Sorting by Roll Numbers

```
#include <stdio.h>
struct stud
{
    int roll;
    char dept_code[25];
    float cgpa;
};

main()
{
    struct stud class[100], t;
    int j, k, n;

    scanf ("%d", &n);
    /* no. of students */

    for (k=0; k<n; k++)
        scanf ("%d %s %f", &class[k].roll,
               class[k].dept_code, &class[k].cgpa);
    for (j=0; j<n-1; j++)
        for (k=j+1; k<n; k++)
            {
                if (class[j].roll > class[k].roll)
                {
                    t = class[j];
                    class[j] = class[k];
                    class[k] = t;
                }
            }
    <<<< PRINT THE RECORDS >>>>
}
```



The diagram shows two states of an array 'class' of 100 students. On the left, it's labeled 'in' and shows the initial state with handwritten notes: 'Array' above the first row, 'in' next to 'roll', and 'field' next to 'dept\_code'. On the right, it's labeled 'Swapping Structure' and shows the state after a swap, with handwritten notes: 't' with arrows pointing to the temporary variable and swapped values. A large bracket at the bottom indicates the range of the print operation.

So, here we are trying to apply that to an example sorting by roll numbers. So, here we have got a set of students, look at the declaration here. Struct stud is the structure, having role department code and cgpa. Now, in the main function what are you doing? We are defining struct student class 100. Now, struct start has been declared, it has been declared globally, even before main that is possible.

Here I am saying that class 100 is one array and t and there are some integers. 't' is again a structure, t is also a structure of type stud. Now, I am reading the number of students, how many students are there in the class. The array can accommodate at most 100 but I am reading the value n. Now, for each of the elements, (look at this is our familiar for loop here) for k 0 to less n, I am scanning the class, the roll number of that particular student, the department code and the c g p.

So, in that way I read here. I am reading the array all right. The array is being read here at this point right. Now what I am doing now? Next let us look at this loop. What are you doing here? For some value of j, if the role of that student; so, here I have got my array, I am coming to j, a particular element and looking at the role number field, if that is greater than some other value k, if it is greater than the roll value of k, then I am exchanging them, just as we sort.

Now, what are we sorting here in that example earlier that we had done, we are sorting integers or sorting real numbers. Here what we are doing we are sorting the entire structure, as you can see the j is coming to t.

So, t is again a structure. k is coming to j and so, this is a swap operation, swapping structures. It is called swapping the structures and this I am doing for k which is a internal variable and j is the external variable. So, my diagram should be a little different.

(Refer Slide Time: 05:28)

Example: Sorting by Roll Numbers

```
#include <stdio.h>
struct stud
{
    int roll;
    char dept_code[25];
    float cgpa;
};

main()
{
    struc stud class[100],t;
    int j, k, n;

    scanf ("%d", &n);
    /* no. of students */
}

```

```
for (k=0; k<n; k++)
    scanf ("%d %s %f", &class[k].roll,
           class[k].dept_code, &class[k].cgpa);
for (j=0; j<n-1; j++)
    for (k=j+1; k<n; k++)
    {
        if (class[j].roll > class[k].roll)
        {
            t = class[j];
            class[j] = class[k];
            class[k] = t
        }
    }
    <<<< PRINT THE RECORDS >>>>
}

```

So, for every  $j$  I am looking at one  $j$  and I am varying  $k$  from here. This is  $k$ .  $k$  is varying from this point to this point to the end of the structure and comparing with this  $j$  value. If this is any element that is less than that is coming over here and it is being swapped. So, you have seen this swap algorithm earlier.

So, that is being swapped here and then I update  $j$  and I go on doing this. So, this is how I can apply the sorting algorithm that we had learnt, for sorting student records all right.

(Refer Slide Time: 06:20)

## Pointers and Structures

- You may recall that the name of an array stands for the address of its zero-th element.
  - Also true for the names of arrays of structure variables.
- Consider the declaration:
 

```
struct stud {  
    int roll;  
    char dept_code[25];  
    float cgpa;  
} class[100], *ptr;
```

Next the other thing that we have to look at is pointers and structures. How they can be intermingled? You may recall that the name of an array stands for the zeroth element of the array. So, if this is an array of the name a, then the name a and a 0 are synonymous and so, a can be considered to be a pointer that is pointing to a 0 all right.

Now, consider the declaration. This is also true for the names of arrays of structures. So, if I have a declaration like this, you look at the declaration yourself. So, you can see student is a structure and I have declared of the type student and array class 100 and star p t r.

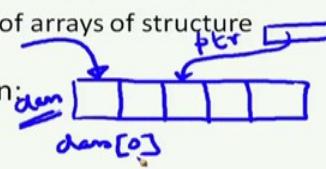
Now, what is star ptr? What does it mean? You know by now. Just as we had done this (keep it side by side) int star p what does it mean? It means that p is a pointer that points to integer data type. Here it means that ptr is a pointer that points to stud type of structures. So, now, what do I have?

(Refer Slide Time: 07:59)

## Pointers and Structures

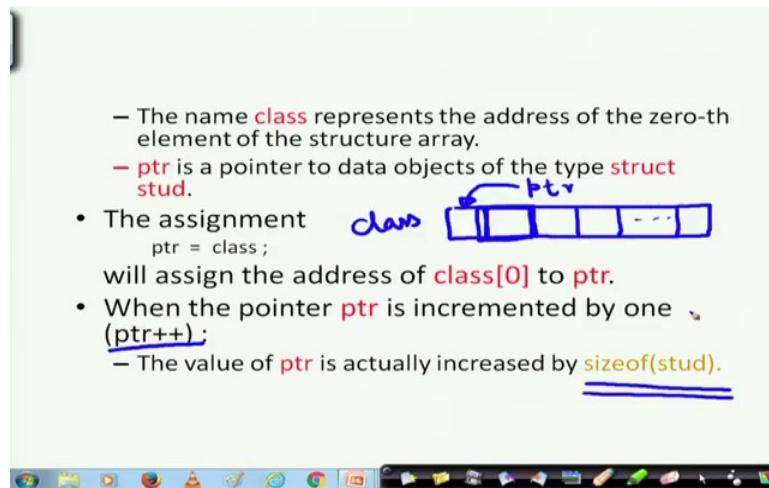
- You may recall that the name of an array stands for the address of its zero-th element.
  - Also true for the names of arrays of structure variables.
- Consider the declaration:

```
struct stud {  
    int roll;  
    char dept_code[25];  
    float cgpa;  
} class[100], *ptr;
```



I have got an array called class, each element of which is of type student structure and I have got a pointer ptr, which can point to such structures. It can point to such structures -any of these I have not yet initialised this.

(Refer Slide Time: 08:29)



The name class therefore, represents the address of the zeroth element of the structure array. So, if I have this then this class means it is a pointer that is pointing to this element plus 0. Just like an array and ptr is a pointer to the data objects of type struct, that we have seen.

The assignment - class assigned to ptr what will it make? It will make ptr to point to the first element of the array. Here is class and ptr is now pointing, (when I do this assignment note that they are of the same type) to this element.

So, it will assign ptr to class 0, understood? When the pointer ptr is incremented by 1, that is ptr plus plus you should be able to tell me what will happen. The ptr will be incremented to the next element of the array. So, the actual increment will be by a scale factor and what is that scale factor? That scale factor is size of stud, size of the structure stud and you know that it has got a roll number, it has got c g p a.

So, depending on the different data types that are housed is inside that structure it will vary. So, the value of ptr will point to the next element which will be incremented by size of stud.

(Refer Slide Time: 10:49)

The slide contains the following text:

- The name **class** represents the address of the zero-th element of the structure array.
- **ptr** is a pointer to data objects of the type **struct stud**.
- The assignment  
`ptr = class;`  
will assign the address of **class[0]** to **ptr**.
- When the pointer **ptr** is incremented by one (**ptr++**):
  - The value of **ptr** is actually increased by **sizeof(stud)**.
  - It is made to point to the next record.

(Refer Slide Time: 10:51).

The slide contains the following text:

- Once **ptr** points to a structure variable, the members can be accessed as:  
`ptr -> roll;`

Hand-drawn annotations explain the diagram:

- A structure array `class` is shown as a box divided into four horizontal sections. The top section is labeled `class[0]`, the second `class[1]`, the third `class[2]`, and the bottom section is empty.
- An arrow labeled `ptr` points to the first section `class[0]`.
- Another arrow labeled `ptr -> cgpa` points to the `cgpa` field within the `class[0]` section.
- A red bracket on the left side of the array is labeled `class[1]. cgpa`.
- Red arrows labeled `ptr = class[0]. cgpa` and `ptr + 1 -> cgpa` indicate the movement of the pointer from the first element to the second element of the array.

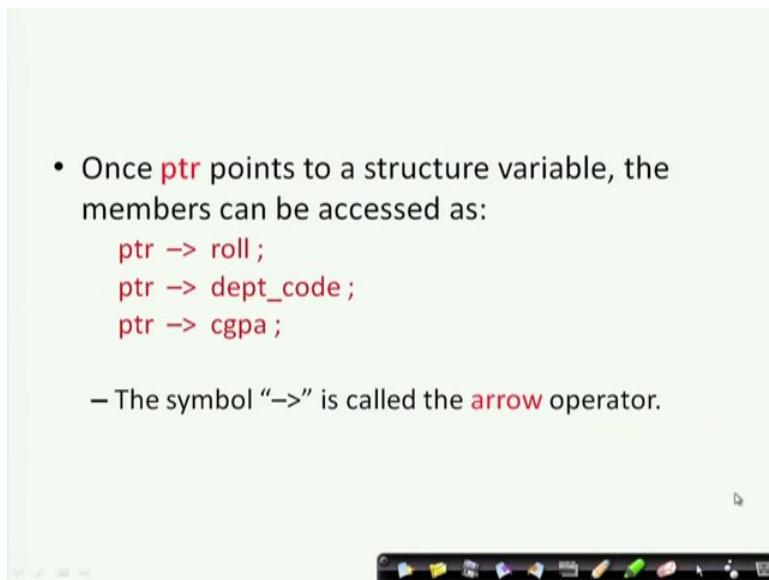
So, it's permit to point to the next record. Once **ptr** points to a structure variable, now this is something new, the pointers can be accessed as **ptr role**, that is possible. So, let us see what is happening you are being introduced to this operator, the arrow operator.

So, here or this only comes if the left side of this operator is a pointer. So, if suppose I have got some structure and pointer **p** or **ptr** is pointing to this structure and it has got different fields say **cgpa** is a field. If I write **ptr arrow c g p a**; that means that I am now pointing to this, I am actually accessing this element of the structure.

Just as suppose this structure is class and this element is class 1 say. I could have written class 1 dot cgpa and here I have already done this , the pointer ptr was assigned to class 0 and then I did ptr plus plus; that means, now where is ptr pointing to? ptr is pointing to class 1. So, I can now also do ptr arrow cgpa. These two are equivalent, this and this are equivalent.

So, here also you can understand this. So, let us move ahead.

(Refer Slide Time: 13:11)



Similarly, I can go to another field by ptr slash department code, I can go ptr cgpa, ptr roll ,this symbol is naturally called the arrow operator.

(Refer Slide Time: 13:31)

**Example**

```

#include <stdio.h>

typedef struct {
    float real;
    float imag;
} COMPLEX;

swap_ref(COMPLEX *a, COMPLEX *b)
{
    COMPLEX tmp;
    tmp=*a;
    *a=*b;
    *b=tmp;
}

print(COMPLEX *a)
{
    printf("(%.f,%.f)\n",a->real,a->imag);
}

```

You can read this type def, now here I am defining a type, the name of the type is complex and what is this type complex. It is a structure with a real and imaginary part, this is what we saw earlier.

Next I have got a function - print complex star , what does it mean? A is a pointer to type complex. So, print f, whatever there are two placeholders - a real, a imaginary. So, why is it possible? ‘a’ is a pointer. So, it is pointing to some complex number, whatever that is. ‘a’ is a pointer pointing to that and I am going to the real part, I am printing a real and this part is a imaginary. So, this is what is being done by this piece of function code.

So, now swap reference, I have got complex a and complex b. I am sorry it is not visible here. a and b there are two pointers in complex. So, there is t m p. I take another structure named t m p of type complex, and I swap the pointers here.

That is why you can work it out that; that means, that ‘a’ was pointing to some structure and that ‘a’ is pointing to this structure and this particular structure is copied to another structure temp, then ‘b’ was pointing to another structure b and a are swapped. So, now, ‘b’ is pointing to this and ‘a’ is pointing to this next I am making b point to this. So, that is how a swap operation can be done and how we passed on the parameter to this.

(Refer Slide Time: 16:01)

The screenshot shows a C program titled "Example". The code defines a complex number structure, swaps two complex numbers using pointers, and prints them. The output shows the swap was successful.

```
#include <stdio.h>

typedef struct {
    float real;
    float imag;
} COMPLEX;

swap_ref(COMPLEX *a, COMPLEX *b)
{
    COMPLEX tmp;
    tmp=*a;
    *a=*b;
    *b=tmp;
}

print(COMPLEX *a)
{
    printf("(%f,%f)\n",a->real,a->imag);
}

main()
{
    COMPLEX x={10.0,3.0}, y={-20.0,-4.0};

    print(&x); print(&y);
    swap_ref(&x,&y);
    print(&x); print(&y);
}
```

Output window:

```
(10.000000,3.000000)
(-20.000000,4.000000)
(-20.000000,4.000000)
(10.000000,3.000000)
```

So, the main program can be can define complex x, 10 and 3, y is minus 20 minus 10 because of the resolution it is not visible. So, print x and y then swap reference print x and y. Now, can you guess and tell me whether the swapping will work in this case or not ? The answer is yes it will work because it is a call by reference. So, whatever I am swapping here, that will be reflected in the main program also. So, here is an example. So, let us move ahead.

(Refer Slide Time: 17:04)

## A Warning

- When using structure pointers, we should take care of operator precedence.
  - Member operator “.” has higher precedence than “\*”.
    - $\text{ptr} \rightarrow \text{roll}$  and  $(\ast \text{ptr}).\text{roll}$  mean the same thing
    - $\ast \text{ptr.roll}$  will lead to error.
  - The operator “ $\rightarrow$ ” enjoys the highest priority among operators.



Now, one warning is there that, when using a structure pointers, we should take care of the operator precedence. Because here we have seen the star symbol as well as dot symbol, now this dot has a higher precedence than star, what does it mean?

It means that if I have got something like  $\text{ptr roll}$  and  $\ast \text{ptr dot roll}$  the dot will have a higher precedence. So, these two are the same thing -  $\text{ptr roll}$  ( $\text{ptr}$  is a pointer and the  $\text{roll}$  is that particular field) and  $\ast \text{ptr}$  is what?  $\ast \text{ptr}$  is the particular structure, and I am going to the  $\text{roll}$  field of that structure.

But if I had done this that would be an error why? That would be an error because this dot operator has got higher precedence than this star operator. So, basically I will try to get the role filled from the pointer  $\text{ptr}$ , but that is not the case, there is no  $\text{roll}$  field inside the pointer  $\text{ptr}$ . The  $\text{roll}$  field is there in the content of the pointer  $\text{ptr}$  that is  $\ast \text{ptr}$ .

So, I must make first  $\ast \text{ptr}$  within bracket then dot  $\text{roll}$ . So, this is correct, but this is wrong the and the operator arrow enjoys the highest priority among operators among all these operators it has got the highest priority.

(Refer Slide Time: 19:12)

## A Warning

- When using structure pointers, we should take care of operator precedence.
  - Member operator “.” has higher precedence than “\*”.
    - `ptr->roll` and `(*ptr).roll` mean the same thing.
    - `*ptr.roll` will lead to error.
  - The operator “->” enjoys the highest priority among operators.
    - `++ptr->roll` will increment roll, not `ptr`.
    - `(++ptr)->roll` will do the intended thing.



So, here in this case what will happen? plus plus ptr arrow roll - as we have said that this has got the highest priority, we will go to where the ptr is and go to the roll field of that and after that we will increment this plus plus, although it is shown as a pre increment. So, that will not have higher precedence than the arrow operator.

So, if I want to first increment the ptr and then get the roll, then I should do plus plus ptr arrow roll which will do the intended thing, I hope this is clear this part.

(Refer Slide Time: 20:00)

## Structures and Functions

- A structure can be passed as argument to a function.
- A function can also return a structure.
- The process shall be illustrated with the help of an example.
  - A function to add two complex numbers.



Now, structures and functions. Structures can be passed as argument to a function and a function that we have already seen. A function can also return a structure. So, we have seen that using two complex numbers how we can do that.

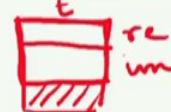
(Refer Slide Time: 20:23)

### Example: complex number addition

```
#include <stdio.h>
struct complex {
    float re;
    float im;
};

main()
{
    struct complex a, b, c; ✓
    scanf ("%f %f", &a.re, &a.im);
    scanf ("%f %f", &b.re, &b.im);
    c = add (a, b); ✓
    printf ("\n %f %f", c.re, c.im);
}
```

```
struct complex add (x, y)
struct complex x, y;
{
    struct complex t;
    t.re = x.re + y.re ;
    t.im = x.im + y.im ;
    return (t);
}
```



So, here again we look at that example that we have got a complex structure with two fields. I have now changed the names - float r e and float i m, and in the main function I am scanning

and a dot r e, real part of a, imaginary part of a, then real part of b then imaginary part of b and then I call the function add a b. What is happening here when I call this add a b? What am I passing? I am passing a b and here x y.

So, it is a call by value. So, here this is copied; now i have struct complex x y (there is no problem here it is not like swap I just want to add). So, I have got two internal variables x and y and t, that new structure t is taking x's real part and y's real part, then I am getting the real part of t and in the imaginary part of t there is nothing here. There is an excess space. In the imaginary part of t, I take the imaginary part of x and the imaginary part of y, then I am returning this t to c, where c is also of type complex since the types are matching, I can pass it on over there.

Now, this is one way of adding the complex numbers. Now if this is clear, now let us look at an alternative way of using the pointers.

(Refer Slide Time: 22:17)

### Example: Alternative way using pointers

```
#include <stdio.h>
struct complex {
    float re;
    float im;
};

main()
{
    struct complex a, b, c;
    scanf ("%f %f", &a.re, &a.im);
    scanf ("%f %f", &b.re, &b.im);
    add (&a, &b, &c);
    printf ("\n %f %f", c.re, c.im);
}
```

```
void add (x, y, t)
struct complex *x, *y, *t;
{
    t->re = x->re + y->re ;
    t->im = x->im + y->im ;
}
```

Here I just changed this, other things are the same. What I am doing here is I have called them, passing the address, now I am passing the pointers. In the earlier case I was passing the value here and now I am passing the pointers. I am passing address of a, address of b and address of c.

Now, inside this add function, what am I doing? I am taking  $x$  and  $y$ . So, these are pointers. I am taking them and  $t$ ,  $t$  is real.

So,  $t$  is a pointer that is pointing to another structure of the type complex and  $t$  is  $\text{r e}$ ,  $t \rightarrow \text{r e}$  means a real field of this  $t$ , will be taking the real part of  $x$  and real part of  $y$ . So, here  $x$  plus  $y$ , the sum will come, this the real part of  $x$  and real part of  $y$  will come here, and here the imaginary part of  $x$  and  $y$  will be added.

Now, when here I am not returning any  $t$ , because this  $t$  is nothing, but this  $c$ . So, this  $t$  and  $c$  are mapped, it is a call by reference. So,  $c$  is also actually pointing over here. So, I get the result here. So, this is another way in which in structures, we can use pointers also. So, these are relatively more advanced aspects and you will gradually practice this and we will get familiar.

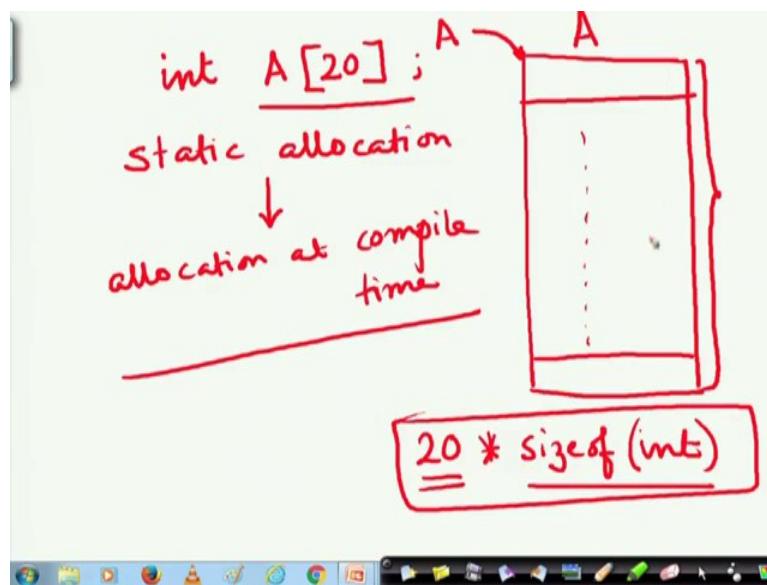
Thank you.

**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 61**  
**Dynamic Allocation and File**

We have looked at pointers and structures in detail and we have also seen how structures can also utilise pointers or in other words how pointers can be used in conjunction with structures. Now, we look at another very interesting use of pointers, but in general let me say that it is a very fundamental concept, from the memory allocation point of view.

Dynamic memory allocation is what we will be discussing now. Now, what if there is a dynamic memory allocation then; obviously, there must be something called the static memory allocation now what is static memory allocation? (Refer Slide Time: 01:07)

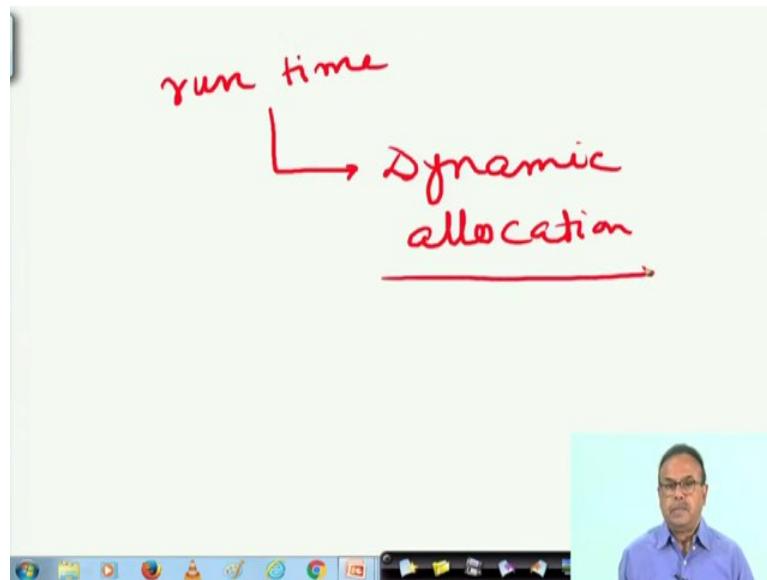


When we declare an array say int A 20, you know that the compiler will allocate 20 locations, 20 locations to house 20 integers to you and that will be named as A or you can also consider that there is a pointer A, which is pointing to the first element of the array. But you have got space for 20 locations, 20 integers allocated to you.

So, now, you know size of. So, how many bytes will be required? You can say 20 times size of int. So, size of int will return you how many bytes your particular system allocates for an integer and 20 such allocation. So, so many bytes will be allocated to you. Now when that is statically allocated that is allocated at compile time; so, when we say static allocation that means, allocation at compile time right.

Now, if for some reason you need more than 20 integers to be stored in this array A , you will need to redefine this whole thing or in some cases we do not know, we do not have an idea of how many date data items will come for example, you was you are actually storing student records in an array class, and you do not know how many students will join that class beforehand.

If you know that beforehand its fine or if you have an idea that what is the maximum number of students that can come, then its fine you can allocate it in the form of static allocation as we do in an array. (Refer Slide Time: 03:57)



However, when we do not know and the information comes at a runtime; that means, when is being executed, that will lead to what we call dynamic allocation of memory. So, let us look at how we can handle it.

(Refer Slide Time: 04:23)

The slide has a light green background. At the top center, the title "Basic Idea" is written in a bold black font. Below the title, there are two bullet points. The first bullet point contains three sub-points: "Many a time we face situations where data is dynamic in nature.", "– Amount of data cannot be predicted beforehand.", and "– Number of data item keeps changing during program execution.". The second bullet point contains two sub-points: "Such situations can be handled more easily and effectively using **dynamic memory management** techniques.".

So, the basic idea is I have already explained that we have an amount of data we cannot predict beforehand. So, we will use effectively whose dynamic memory allocation technique to do that.

(Refer Slide Time: 04:36).

## Contd.

- C language requires the number of elements in an array to be specified at compile time.
  - Often leads to wastage or memory space or program failure.
- Dynamic Memory Allocation
  - Memory space required can be specified at the time of execution.
  - C supports allocating and freeing memory dynamically using library routines.



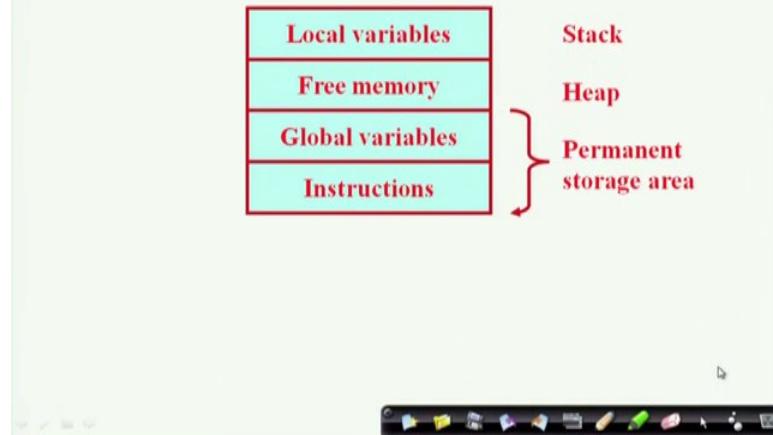
Now, C language requires a number of elements to be specified in compiled time, when we define in an array we need to specify that in compile time. Now, often that leads to wastage of memory or program failure. Why a program failure? Program failure will occur because if we exceed the amount of space that has been allocated, there will be a failure the program will give an error or it will exit abnormally. However, if we take recourse to dynamic memory allocation, we can solve this problem how? Memory space required can be specified at the time of execution how can we do that, how can I specify the amount of memory required at the time of execution?

If while running the program, just like the instructions and operators, if we had some special means, some special command, some special operator by which we can grab memory then it's possible. Now here you should understand that who allocates memory to us? It is the operating system which allocates the memory to us.

So, this like the print f, scan f all those things are system calls. We are making calls to the operating system, which is doing the required thing for us. Similarly there is a function called Malloc - Memory Allocated, malloc using which we can grab memory from the operating system how let us look at this.

(Refer Slide Time: 06:25)

## Memory Allocation Process in C



In memory in C, I have got different types of variables that is not so, much relevant right now. What is needed is this part. These global variables and instructions are there always specified.

(Refer Slide Time: 06:37)

## Contd.

- The program instructions and the global variables are stored in a region known as permanent storage area.
- The local variables are stored in another area called stack.
- The memory space between these two areas is available for dynamic allocation during execution of the program.
  - This free region is called the heap.
  - The size of the heap keeps changing

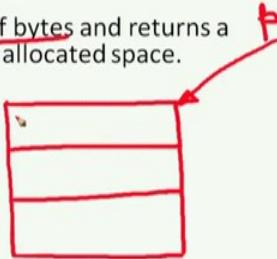


The local variables are there and there is some free memory. We can take from this free memory and use them as our local variables. The free region is has got a name heap.

(Refer Slide Time: 07:09)

## Memory Allocation Functions

- malloc ( )
  - Allocates requested number of bytes and returns a pointer to the first byte of the allocated space.
- calloc
- free
- realloc



A diagram illustrating memory allocation. It shows a stack of three rectangular boxes of different heights, representing memory blocks. A red arrow points from the text 'malloc' to the top edge of the top box, indicating that malloc allocates memory from the top of the stack.

Now, the most important thing is that we need some functions, which will give my program some memory addresses or memory blocks from the operating system storage of memory, which is known as heap from there it will be allocated to my program all right.

So, for that we have got four different functions - one is malloc, what does malloc do? Malloc allocates the requested number of bytes and returns a pointer to the first byte of the allocated space. So, what it does is something like this, what is happening? (Refer Time: 08:31). Malloc allocates a requested number of bytes and returns a pointer to the first byte of the allocated space. So, let us try to explain this.

So, when I do malloc, it is just like a function and it will return some memory bytes, some memory bytes. How many memory bytes? It will depend on, what I am requesting for. It is a requested number of bytes. So, malloc will have some parameters which will show later.

So, it will give me some amount of memory. Now, the operating system has got some free memory spread here and there. So, from there it is giving me some piece of memory, but

how do I know where is that piece of memory? For that, it is returning me a pointer say pointer p which is telling me that if you follow this pointer, you will get this piece of memory location. So, let us proceed a little bit.

(Refer Slide Time: 09:46)

## Memory Allocation Functions

- malloc
  - Allocates requested number of bytes and returns a pointer to the first byte of the allocated space.
- calloc
  - Allocates space for an array of elements, initializes them to zero and then returns a pointer to the memory.
- free
  - Releases previously allocated space.
- realloc
  - Modifies the size of previously allocated space.



Now, similarly when this memory block is given in response to malloc request, the actual memory is not initialised to some value. It can have any garbage value. But if I apply calloc; then it allocates space for the array of elements and initialises them to zero and returns a pointer. So, in this case if I want to have a chunk of memory where everything has been initialised to zero then I should use calloc. On the other hand the free function call will return this amount of memory, that was given to me in request to malloc, will be returned back to the heap, returned back to the operating system so, that it can be utilised by somebody else in future all right.

So realloc modifies the size of the previously allocated space. So, I have got some allocation and then I think that allocation is not enough. I want to change it. I can use realloc. However, we will be mostly concerned with malloc and free in our discussion.

(Refer Slide Time: 11:14)

## Allocating a Block of Memory

- A block of memory can be allocated using the function **malloc**.
  - Reserves a block of memory of specified size and returns a pointer of type **void**.
  - The return pointer can be assigned to any pointer type.
- General format:

$\text{ptr} = (\text{type } *) \text{malloc}(\text{byte\_size})$

$\underline{\underline{=}}$   $\underline{\underline{(\text{int } *)}}$

*pointer of type void*

*int*

So, a block of memory can be allocated using the function malloc and it reserves a block of memory and returns a pointer of type void. You know every pointer has got some type, but in this case with malloc returns some memory block, the pointer that it has returned is of type void, but then we have to do something. What we need to do? I know why I needed this memory. So, accordingly I will have to do that typecasting. So, once it returns me of type void, but that return pointer can be assigned to any pointer type. So, here you see.

So, here please note, malloc has got a parameter of byte size - how much memory I want, the amount of bytes which is required. Now this malloc has returned me up to this, it has returned me a pointer and pointer is of type void, but suppose if this amount of memory I want for the purpose of storing integer array then this type will be int star; that means, I am casting this. What is malloc returning? Malloc is returning a pointer, some pointer, but that pointer was of type void. Now when I am typecasting it to int star, then this void is no longer its type, it is becoming of type int and then I am assigning it to another variable p t r.

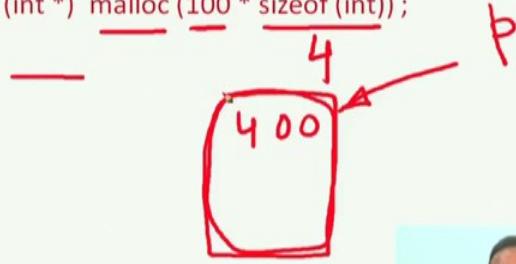
So, think of two things, first of all you have to decide on how many bytes you want. Accordingly you do malloc and then what type of data you want to store there? So, accordingly you do this typecasting like int star, float star, char star whatever you do and then you assign it to a particular pointer. Let us see how it will work. So, let us look an example.

(Refer Slide Time: 13:55)

## Contd.

- Examples

```
p = (int *) malloc (100 * sizeof (int));
```



Here, how do I know how many bytes I need? Suppose I need to store an element an array of 100 integers. So, what I do here is I ask for malloc 100 times size of int, size of int if it is 4, if int is 4 then I am getting 400 bytes.

Now, these 400 bytes that have been given to me is being pointed by some pointer of type void. So, next I make it int star and put it to p. So, p is now an integer pointer that is pointing to this entire block of 400 integers.

(Refer Slide Time: 14:49)

## Contd.

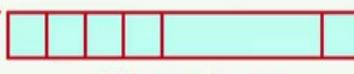
- Examples

```
p = (int *) malloc (100 * sizeof (int));
```

- A memory space equivalent to “100 times the size of an int” bytes is reserved.

- The address of the first byte of the allocated memory is assigned to the pointer p of type int.

p



400 bytes of space

So, a memory space equivalent to 100 times the size, we have got. So, here 400 bytes of space and p is a pointer pointing to the beginning of this. So, I have got this just using malloc, it was not declared beforehand.

Now so, this 100 can also be a variable n, if I read a particular variable n now how many students are there? scan f m and n. So, I read the number of students, then I can multiply that with n as well.

(Refer Slide Time: 15:35)

## Contd.

```
cptr = (char *) malloc (20);
```

- Allocates 10 bytes of space for the pointer cptr of type char.

```
sprt = (struct stud *) malloc (10 *  
sizeof (struct stud));
```

40



Next, you see here I am seeking memory for 20 characters, I do malloc 20 because I know a character takes one byte and then the pointer is of type void I am typecasting it to type character char star and assigning it to c p t r. Now it is actually wrong, it is allocating 20 bytes of space for the pointer.

So, integer character was simple, now I want to have for space for the entire structure students, now that size is larger. So, I do not know. So, I just have employ this function size of struct stud. So, I gets how many bytes? Say it requires say 40 bytes and say 10 such for 10 such students or n such students. I multiply with that I get so, much memory, now I have to typecast that to struct stud star and that goes as a structure pointer.

(Refer Slide Time: 17:00)

## Points to Note

- **malloc** always allocates a block of contiguous bytes.
  - The allocation can fail if sufficient contiguous memory space is not available.
  - If it fails, **malloc** returns **NULL**.



Now, malloc always allocates a block of contiguous bytes. Now suppose you are asking for 100 bytes and 100 bytes are not available then malloc will not be able to allocate you the space, in that case malloc will return in null. That is a null pointer, that is a special character special value it will return, that shows that I could not allocate a space.

So, I could not allocate it to you a valid pointer. So, it is a null pointer meaning thereby I failed in allocating your memory.

(Refer Slide Time: 17:44)

## Example

```
#include <stdio.h>
main()
{
    int i,N;
    float *height;
    float sum=0,
    printf("Input heights for %d
           students \n",N);
    for(i=0;i<N;i++)
        scanf("%f",&height[i]);
    for(i=0;i<N;i++)
        sum+=height[i];
    printf("Input the number of students. \n");
    scanf("%d",&N);
    height=(float *) malloc(N * sizeof(float));
    printf("Average height= %f\n");
    avg=sum/(float) N;
}
```



So, here is an example. Here let us look at it from one side - I n , float is a pointer, height is a pointer of type float, sum is 0 and average. So, what am I trying to do? Probably I am trying to find the average height of the class. So, input the number of students and I am reading ampersand. So, this one is ampersand n. So, this is n number of students.

Now, see I did not know how many students are there. So, I am getting this number of students here, I am getting n number of students here. Now I want to have an amount of spaces for the height. So, what I am doing? I am allocating n number of spaces, n is a variable here and size of float, whatever size of float is 4 bytes. So, n times 4 bytes, so much space is being allocated, and the pointer is height is a pointer of type float.

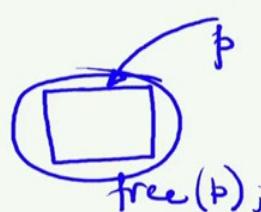
So, this pointer is being type casted to float. Then I get the input scan f in a loop, I am getting the heights one after another in an array and I am finding the sum of the heights, finding the average of the heights, where I am dividing sum which is a floating point number with float n. Here is another example of typecasting. So, you see I am dividing by n, but this is a floating point real number, n was an integer. So, I convert it to float and convert it divided.

So, this is how malloc works now. So, we have explained that.

(Refer Slide Time: 19:43)

## Releasing the Used Space

- When we no longer need the data stored in a block of memory, we may release the block for future use.
- How?
  - By using the `free` function.
- General format:  
`free (ptr);`



The slide is titled "Releasing the Used Space". It contains a bulleted list explaining the process of deallocation. The first two points are standard, but the third point includes a general format for the `free` function call, which is `free (ptr);`. To the right of the text, there is a hand-drawn diagram. It consists of a simple rectangle with a blue outline. Above the rectangle, the letter 'p' is written in blue. Below the rectangle, the expression `free(p);` is written in blue. A blue arrow originates from the letter 'p' and points to the top edge of the rectangle. Another blue arrow originates from the expression `free(p);` and points to the bottom-right corner of the rectangle. The entire slide is set against a background that looks like a computer desktop with various icons visible along the bottom edge.

Similarly the general format for freeing space is using by using the free function. So, general now suppose I have got a space allocated to me and that space, the only handle to that space is to the pointer p.

So, I free that pointer, I free p. So, the pointer is freed; that means, this pointer is freed means this location, this amount of memory goes back to the storage of the operating system and that is the heap.

(Refer Slide Time: 20:30)

## Releasing the Used Space

- When we no longer need the data stored in a block of memory, we may release the block for future use.
- How?
  - By using the **free** function.
- General format:

```
free (ptr);
```

where ptr is a pointer to a memory block which has been already created using **malloc**.

(Refer Slide Time: 20:32)

## Altering the Size of a Block

- Sometimes we need to alter the size of some previously allocated memory block.
  - More memory needed.
  - Memory allocated is larger than necessary.
- How?
  - By using the `realloc` function.
- If the original allocation is done by the statement

```
ptr = malloc (size);  
then reallocation of space may be done as  
ptr = realloc (ptr, newsize);
```



So, whatever we got in malloc that gives us some idea about how we can get space and reallocate space.

(Refer Slide Time: 20:36)

## Contd.

- The new memory block may or may not begin at the same place as the old one.
  - If it does not find space, it will create it in an entirely different region and move the contents of the old block into the new block.
- The function guarantees that the old data remains intact.
- If it is unable to allocate, it returns `NULL` and frees<sup>b</sup> the original block.

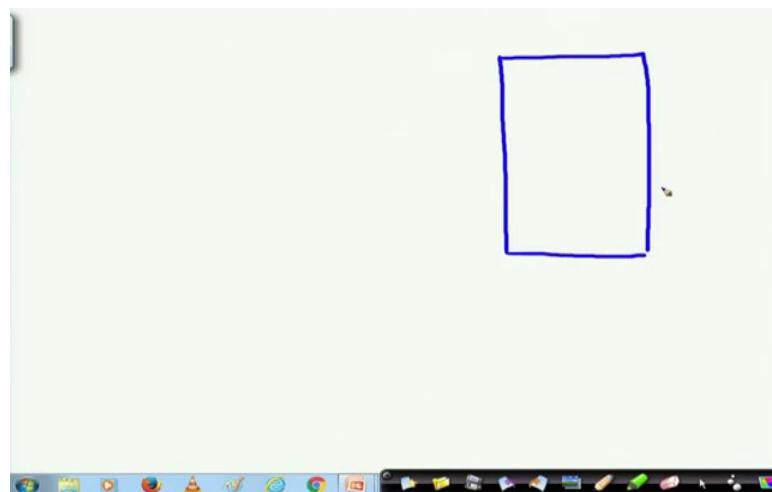


Now, briefly we I will be talking for the next 5 or 10 minutes on file handling; there is not much to understand about file handling thus you will learn as you do. Now, what is a file ? That is

something you have to understand. File is something where I want to write something write or read from.

So, I want to store something. I will take a particular file all right; and I will open that file and then I will write in the into that file then close that file and then in that way I may have 10 files.

Now, at a particular point of time I want to read a particular thing. So, I choose the particular file what do I do next? Open the file and read the file. Now, some files may be allowed to be read by others, some files can only be written in to and not read from, some files can have the option of read or write both. (Refer Slide Time: 21:55)



So, file is some space where I will be writing or reading from some storage.

So, this is there in the secondary memory and till now whatever variables we are talking about, those who are all in the primary memory. So, if I store it in a file it goes into the secondary memory. So, let us have a little idea of how files are handled.

(Refer Slide Time: 22:18)

## File handling in C

- In C we use FILE \* to represent a pointer to a file.
- fopen is used to open a file. It returns the special value NULL to indicate that it couldn't open the file.

```
FILE *fptr;
char filename[] = "file2.dat";
fptr = fopen (filename, "w");
if (fptr == NULL) {
    printf ("ERROR IN FILE CREATION");
    /* DO SOMETHING */
}
```



So, now again now we have learnt pointers. So, any file can be accessed using a pointer. Just as if I have the file of income tax. So, I will have a pointer that there is the file of income tax, here there is some file of road tax it will be here some file of your salary, it will be somewhere here, some file of your expenses it will be somewhere else.

So, there will be pointers. So, we use in c, the file star. We use file star to represent pointed to a file and if open is the command for opening a file. If a file cannot be opened then it will return a null just as in the case of malloc, we saw if nothing could be returned it was returning a null.

So, here for example, you see f p t r is a pointer of type file. That means, f p t r will be pointing to file. Now, I have got a character file name which is an array, file 2 dot dat it is a data file. So, fptr is f open file name and here when I do f open, I give the file name as well as the mode in which it can be opened and this w means it is in the right mode. So, what have I done here, I have called f open.

So, I am trying to open the file. So, this f open will return up file pointer. Now if this file pointer is null; that means, there was some error in file creation otherwise it will go on doing something.

So, quickly let us look at this, when I do f open it will open a file and will open it in a particular mode, read or write whatever I specify and it will return me a pointer. If a file is created successfully, it will return me a non null pointer.

(Refer Slide Time: 24:38)

## Modes for opening files

- The second argument of `fopen` is the *mode* in which we open the file. There are **three**
- "r" opens a file for reading
- "w" creates a file for writing - and writes over all previous contents (deletes the file so be careful!)
- "a" opens a file for appending - writing on the end of the file
- "rb" read binary file (raw bytes)
- "wb" write binary file

The second argument of `fopen` is the mode and there are three modes. R means the file is opened for reading, w means it creates a file for writing and writes over all the previous contents. So, if I open it in the write mode, whatever content was in that file is erased. And a opens a file for appending; that means, whatever is there after that it will be added.

So, if you have got something already stored and you do not want to destroy that and you want to add something more to that you will open it in mode a and rb reads a binary file, raw bytes we need not bother about that.

(Refer Slide Time: 25:25)

## The exit() function

- Sometimes error checking means we want an "emergency exit" from a program. We want it to stop dead.
  - In main we can use "return" to stop.
  - In functions we can use exit to do this.
  - Exit is part of the stdlib.h library
- ```
exit(-1);  
in a function is exactly the same as  
return -1;  
in the main routine
```

There is a function called exit which you have seen, sometimes in the emergency we can put exit minus one; that means, it tells that I have exited the function without success.

(Refer Slide Time: 25:38)

## Usage of exit( )

```
FILE *fptr;  
char filename[] = "file2.dat";  
fptr = fopen (filename,"w");  
if (fptr == NULL) {  
    printf ("ERROR IN FILE CREATION");  
    /* Do something */  
    exit(-1);  
}
```

Now, here you see use of exit. file f pointer, character file name is an array, file 2 dot dat. I tried to do something. So, the file pointer was null.

So, if it is null then what can I do? I will have to exit because of some reason the file could not be created.

(Refer Slide Time: 26:02)

## Writing to a file using fprintf( )

- **fprintf( )** works just like printf and sprintf except that its first argument is a file pointer.

```
FILE *fptr;
fptr= fopen ("file.dat","w");
/* Check it's open */
fprintf (fptr,"Hello World!\n");
```

The diagram illustrates the process of writing to a file. On the left, a blue box contains C code: 

```
FILE *fptr;
fptr= fopen ("file.dat","w");
/* Check it's open */
fprintf (fptr,"Hello World!\n");
```

. A blue arrow labeled "fptr" points from the variable declaration to the first argument of the `fopen` function. To the right, a small window titled "file.dat" shows the text "Hello World" inside. The window has a blue border and a blue title bar. Below the window is a taskbar with several icons.

So, f open we have seen, f print is a very important command. f print works just like print f and s print f, except that the first argument is a file pointer. So, we will see how it works. So, fptr is again the file pointer and I have opened the file dot dat in the right mode. Now, print f means now I am printing. Where am I printing? A file called file dot dat has been opened.

The name of the file is file dot dat and how do I identify it? I identify it with the fptr, the file pointer ok. So, it opened in the write mode. So, everything, whatever was there has been erased. So, I am writing, just as you have done print f then automatically by default it goes to the screen, here it is not default here I have said fptr.

So, whatever I write hello world, it will be written in this file not in the screen.

(Refer Slide Time: 27:27)

## Reading Data Using fscanf( )

• We also read data from a file using fscanf().

```
FILE *fptr;  
fptr=fopen ("input.dat","r");  
/* Check it's open */  
if (fptr==NULL)  
{  
    printf("Error in opening file \n");  
}  
fscanf(fptr,"%d%d",&x,&y);
```

input.dat  
↓  
20 30  
x=20  
y=30

Screen is another file, but that is a default file. Reading a data similarly, we printed using f print f, reading we can do using f scan f, forget about that part look at this. fptr - I am reading from not from the keyboard now. I am reading x and y two integers from a file which is pointed out by f p t r and what is that file f p t r? I have opened the file input dot dat.

So, you see in that file input dot dat 20 and 30 was written and so, f scan f I have read that, that was input dot dat, I have opened that in the read mode and I am reading from there. So, I am getting x to be 20 and y to be 30, not from the keyboard, but from the file.

(Refer Slide Time: 28:21)

## Reading lines from a file using fgets( )

We can read a string using fgets( ).

```
FILE *fptr;
char line [1000];
/* Open file and check it is open */
while (fgets(line,1000,fptr) != NULL) {
    printf ("Read line %s\n",line);
}
```

fgets( ) takes 3 arguments, a string, a maximum number of characters to read and a file pointer. It returns NULL if there is an error (such as EOF).



So, now here are some powerful commands. We can read a string using fgets, from a file I can read a string. So, here you see a file is f p t r and a line is of size 1000, while fgets line is not null; that means, I am getting from f p t r, a value and if it is not null; that means, it is not the end of the line, the file is open, I am reading the line using fgets. I am getting the line. fgets takes three arguments, what are the three arguments its taking? A string, a maximum number of characters, 1000 and a pointer from it returns. If there is an error at the end of file it shows EOF which is the end of file.

(Refer Slide Time: 29:22)

## Closing a file

- We can close a file simply using fclose( ) and the file pointer.

```
FILE *fptr;
char filename[]= "myfile.dat";
fptr= fopen (filename,"w");
if (fptr == NULL) {
    printf ("Cannot open file to write!\n");
    exit(-1);
}
fprintf (fptr,"Hello World of filing!\n");
fclose (fptr);
```

Opening

Access

closing



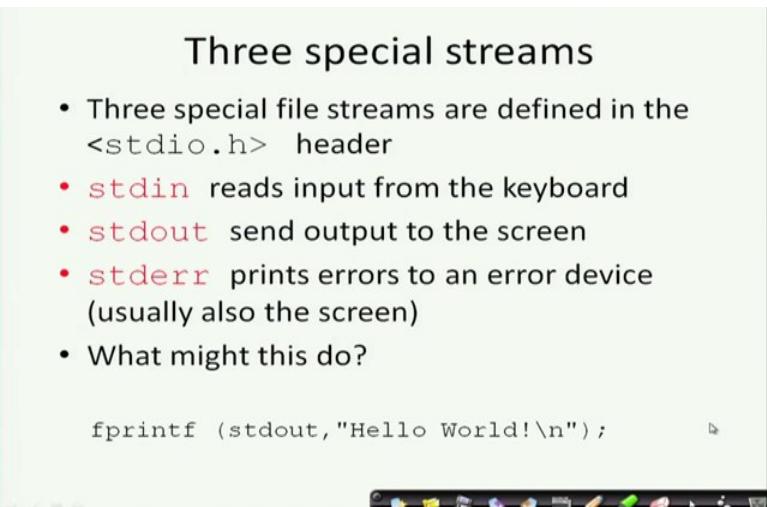
Now, this I think you can understand much better when you use yourself, now when we open a file after that we must close that file we can simply use a command f close and the file pointer. So, here you see f p t r, I opened the file in the right mode, I have written hello world over here, f print f means saying I am printing in the file, and then f close, I am closing the file by f close f p t r all right. So, here it is opening and here is access and here it's closing.

(Refer Slide Time: 30:01)

### Three special streams

- Three special file streams are defined in the `<stdio.h>` header
- `stdin` reads input from the keyboard
- `stdout` send output to the screen
- `stderr` prints errors to an error device (usually also the screen)
- What might this do?

```
fprintf (stdout, "Hello World!\n");
```

A screenshot of a terminal window titled "Terminal". It contains the command "fprintf (stdout, "Hello World!\n");" and its output, "Hello World!". The window has a standard OS X-style title bar and menu bar at the top.

You have seen that `std::in`, `std::out`, were two special cases of files, which are default files and `std::error` which was the printing of the error.

(Refer Slide Time: 30:16)

## An example program

```
#include <stdio.h>
main()
{
    int i;
    printf("Give value of i\n");
    scanf("%d",&i);
    printf("Value of i=%d\n",i);
    printf(stderr,"No error: But an example to show error message.\n");
}
```

Give value of i  
15  
Value of i=15  
No error: But an example to show error message.  
Display on The screen

So, here is an example program you can see that. Main, f print f s t d out give value of i; that means, where am I printing this? Here I am printing it to the standard output. I am reading I from the standard input. Now, f print f I am writing that the value of i is whatever value of i read. So there is no error . So, give value of i, it will first give value of i, you give 15, then f print f that i, then it will say value of i is equal to 15 and there are no error, but an example to show error message.

If there is an error for example I am being returned a null pointer in that case I can use output s t d error and say the file failed to open the file that sort of message.

(Refer Slide Time: 31:31)

## Input File & Output File redirection

- One may redirect the input and output files to other files (other than `stdin` and `stdout`).
- Usage: Suppose the executable file is `a.out`



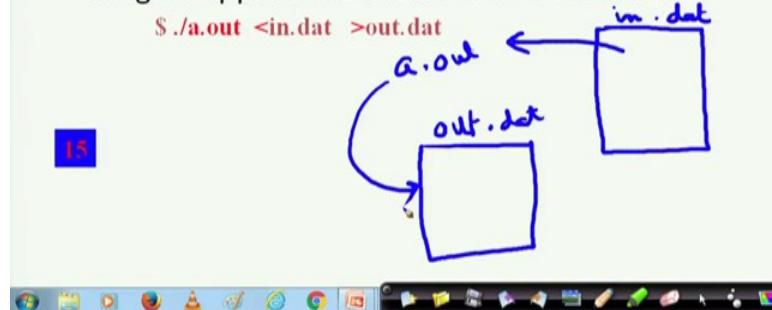
So, now another thing I will just talk about here, that will come in very handy to you that is say for example, you must be running the programs and you after you compile the program and link them you are creating an executable file, which is a dot out ( dot slash a dot out). Now usually what do you do? You have got the dollar, those of you using Linux shell, a dot out right and return.

Now, this dot slash a dot out what is it expecting? The input from the keyboard s t d in and the output is going to std out, but I do not want that. I want that I have got a file where my input data is there.

(Refer Slide Time: 32:28)

## Input File & Output File redirection

- One may redirect the input and output files to other files (other than `stdin` and `stdout`).
- Usage: Suppose the executable file is `a.out`



I call that in dot dat and I have got another file which is known as out dot dat. I want that the input to be taken from this.

So, I want a dot out to read the data from here and the result should be written here. I can do that in the unix environment very simply by this redirection operation. You see, a dot out will run taking data from in dot dat and sending the output data to out dot dat.

(Refer Slide Time: 33:10)

## Input File & Output File redirection

- One may redirect the input and output files to other files (other than `stdin` and `stdout`).
- Usage: Suppose the executable file is `a.out`



So, say for example, in dot dat has got 15. So, I do that and the program runs and says give the value of i (think of the earlier example, earlier program) and it reads from here and it prints the value of i to be 15. So, this whole thing is coming in out dot i.

So, let us once again look at this thing. Here I am asking, them to give the value of i which is being given and that is being scanned from the input file and this is being written on the output file. So, in the output file both these things are being written; consequently, you see, what I am getting is this output - out dot dat and in dot dat, there are two files.

So, this in this way you can use files for storing data you have to open the file, let me summarise a little bit, you will have to open the file, in the read mode, read the data from that file, do the operation, open another file in the write mode and write the data into that file. Thereby, whenever you require some file operations, you can easily do that and this is one example that we have shown which is very common during running your programs. If you store some data in a particular file and read from there and write into another file you can utilise this sort of structures this sort of commands.

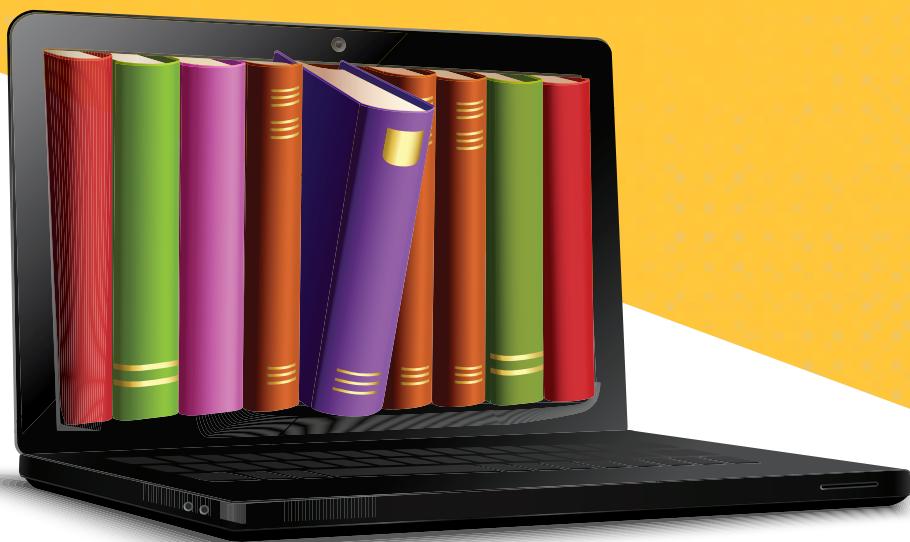
So, thank you very much; I think you have got an overall idea of how to write c programs and solve problems using programs because our the essence of our course was to solve problems through c programming.

So, you should choose different problems and you should try to write programs for solving those problems. So, first you have to find out the proper algorithm, and then write the C code for that, you have learnt everything about basic things about the C programming, I have not touched upon some special features, which you can also learn from the book like static variables and all those I have left out intentionally.

So, that you are not overloaded, you can solve it, the more you run the programs using the basic concepts that has been taught, you will be a good programmer and most importantly you will be able to think logically like a programmer, you will be able to think of an algorithm, and you will be able to translate that into a program.

Thank you very much.

**THIS BOOK IS  
NOT FOR SALE  
NOR COMMERCIAL USE**



(044) 2257 5905/08



nptel.ac.in



swayam.gov.in