MODULE 1- What is a computer

A **computer** is a machine that is able to take in information (input), do some work on or make changes to the information, to make new information (output). Basically it is just an electronic device which executes certain instructions. These instructions are given in a certain format, and are called programs. Computers aren't just the PCs that you see in your school labs; alarm clocks, mobile phones, ATM machines and CD players are also types of computers, which do only one kind of a job.

HISTORY:
Nobody knows who built the first computer. This is because the word "computer" used to mean a person who did maths as their job (a human computer). Because of this, you can say that humans were the first computers. But human computers got bored doing the same maths over and over again, and so they made tools and devices to help them get the answers to their problems. An abacus is an example of an ancient computer which was used to speed up calculations. This part of computer history is called the "history of automated calculation," which is a fancy phrase for "the history of machines that make it easy for me to do this same maths problem over and over without making mistakes." However, even though he didn't actually make one, Charles Babbage is called the father of modern computers, as he first came up with the idea of a computer that could be programmed. He wanted to make a machine that could do all the boring parts of math, (like the automated calculators) and could be told to do them different ways (like the programmable machines.) He called it the "The Analytical Engine". Because Babbage did not have enough money and always changed his design when he had a better idea, he never built his Analytical Engine. Ada Lovelace is the world's first programmer who helped program the Analytical Engine. However, the beginning of the 20th century saw the emergence of vacuum tubes, and in 1941 came Konrad Zuse's electromechanical "Z machines". The Z3 (1941) was the first working machine that used binary arithmetic. Binary arithmetic means using "Yes" and "No." to add numbers together. You could also program it. Another milestone in computer history was the invention of the ENIAC, which could add numbers the way people do (using the numbers 0 through 9), and is sometimes called the first general purpose electronic computer.Before the mid-1970s, only big businesses and the government owned computers. The computers themselves were big—as big as a room, sometimes even as large as a house! Not only were they big, but they were so expensive that a household family wouldn't think of buying one. Computers cost millions of dollars and most people did not know what they would do with one if they had one. Now computers have changed our lives so much that we don't know what we would do without them.

ANALOG and DIGITAL COMPUTERS:

There are two types of computers: Analog and digital. Analog computers are used to process analog data, that is data of continuous nature such as includes temperature, pressure, speed, weight, voltage, depth etc. an example is the Speedometer of a car which measures speed. They were the first computers being developed and provided the basis for the development of the modern digital computers.They do not require any storage capability because they measure and compare quantities in a single operation. Then, in the 1930s, scientists invented digital computers, which made them easier to program. Digital computers are the computers that we see today, and are based on binary language. They are faster than analog computers and can store results and data too. ENIAC and Z3 were first generation digital computers and soon analog became obsolete as technology improved.

WHY DO WE NEED COMPUTERS

Firstly, computers are fast at what they do. A computer tends to be much faster than a human when it comes to doing tasks that involve working with a lot of information. Secondly, computers tend to have better memory. Computers remember specific facts and figures with high accuracy, for as long as the hard drive holds up. This is particularly good when you need to remember large amounts of highly detailed information, like numbers. And also, computers don't get tired. You may have left a computer on for days before, and you might have also noticed that the computer is not tired and doesn't slow down just because it has been on a while. In certain safety applications, computers do a great job by watching a process 24/7 in order to help humans be aware of any potential danger.

A WORLD WITHOUT COMPUTERS

A world without computers is impossible to imagine! Try living without any form of computing device for a day. You can't go anywhere in a car built after 1980 or so, it has a computer. You can't visit the Internet or go to the ATM. You wont be able to call anybody on the phone, or sms. You wont be able to watch TV, as all programs are manipulated in some form or fashion via computer. When you go to a supermarket, there won’t be cashier machines, or barcode scanners. On a higher level, there would be no textbooks to study from, because research, compilation of information, printing etc. are done using computers today. Even electricity and water grids are controlled by powerful computers in the midst. Hence, today life without computers is impossible, despite their many flaws and disadvantages caused.

Module 2 - How does computer work?

What is a computer?

A computer is an electronic machine that processes information—in other words, an information processor: it takes in raw information (or data) at one end, stores it until it's ready to work on it, chews and crunches it for a bit, then spits out the results at the other end. All these processes have a name. Taking in information is called input, storing information is better known as memory (or storage), chewing information is also known as processing, and spitting out results is called output.

Once you understand that computers are about input, memory, processing, and output, all the junk on your desk makes a lot more sense:

- Input: Your keyboard and mouse, for example, are just input units—ways of getting information into your computer that it can process. If you use a microphone and voice recognition software, that's another form of input.
- Memory/storage: Your computer probably stores all your documents and files on a hard-drive: a huge magnetic memory. But smaller, computer-based devices like digital cameras and cellphones use other kinds of storage such as flash memory cards.
- Processing: Your computer's processor (sometimes known as the central processing unit) is a microchip buried deep inside. It works amazingly hard and gets incredibly hot in the process. That's why your computer has a little fan blowing away—to stop its brain from overheating!
- Output: Your computer probably has an LCD screen capable of displaying high-resolution (very detailed)graphics, and probably also stereo loudspeakers. You may have an inkjet printer on your desk too to make a more permanent form of output.

What's the difference between hardware and software?

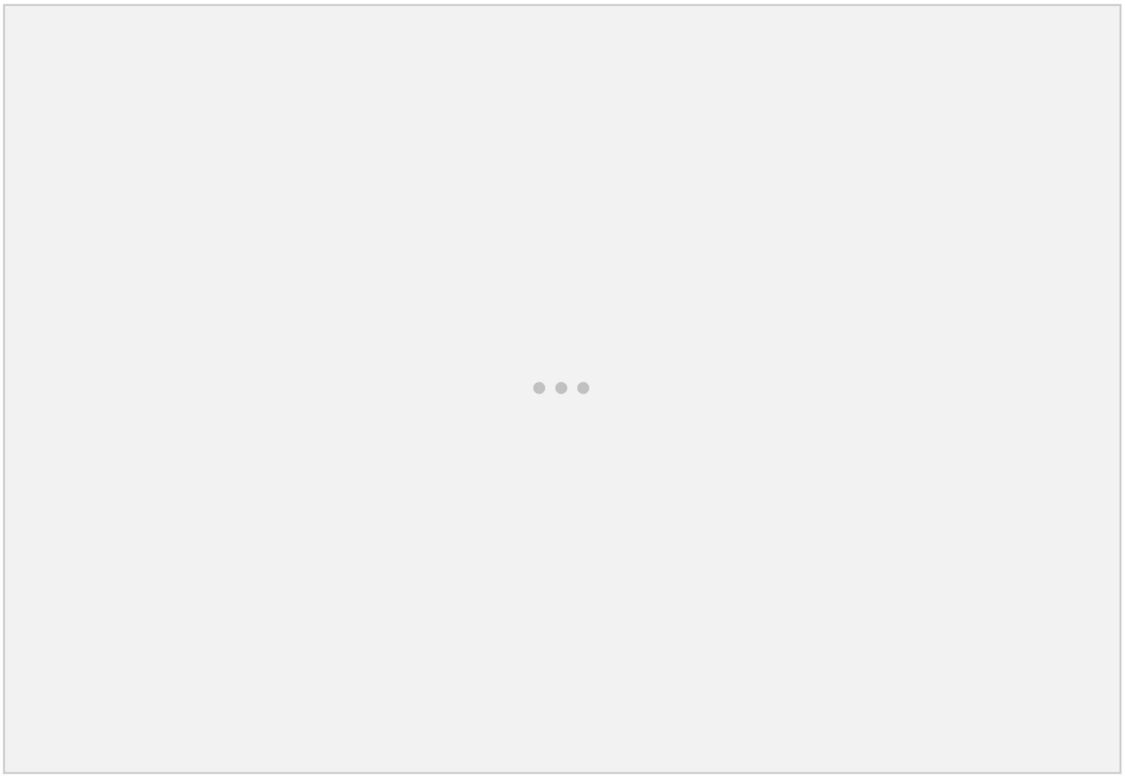
The beauty of a computer is that it can run a word-processing program one minute—and then a photo-editing program five seconds later. In other words, although we don't really think of it this way, the computer can be reprogrammed as many times as you like. This is why programs are also called software.

They're "soft" in the sense that they are not fixed: they can be changed easily. By contrast, a computer's hardware—the bits and pieces from which it is made (and the peripherals, like the mouse and printer, you plug into it)—is pretty much fixed when you buy it off the shelf. The hardware is what makes your computer powerful; the ability to run different software is what makes it flexible. That computers can do so many different jobs is what makes them so useful—and that's why millions of us can no longer live without them!

What is an operating system?

An *operating system* (OS) is software that manages computer hardware and software resources and provides common services forcomputer programs. The operating system is an essential component of the system software in a computer system. Application programs usually require an operating system to function.

So here is the basic map of how computer work.

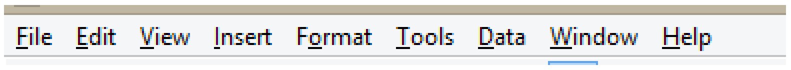


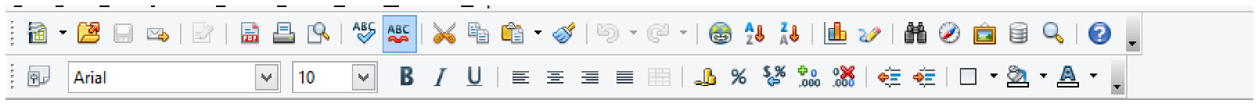
Module - 3: Applications of Computer

- Computer technology has made several important impacts on our society. Today computer is playing very important role in every field of life. Many activities in daily life can be performed very easily and quickly. A lot of time is saved and overall cost is reduced to solve a particular problem.
- Many business activities are performed very quickly and efficiently by using computers. The administrative paperwork is also reduced by using computers. Many business use websites to sell their products and contact their customers.
- Computers are also used by students as well as teachers to collect information on different topics.
- Computers are used in banks for record keeping and maintaining accounts of customers.
- Nowadays, computers can be used to watch television programs on the Internet. People can also watch movies, listen music, and play games on the computer.
- At home, computers are used to maintain personal records and to access information on the Internet. People can also use computers at home for making home budgets etc.
- Nearly every area of the medical field uses computers. For example, computers are used for maintaining patient history & other records. They are also used for patient monitoring and diagnosis of diseases etc.
- These are but a few uses of computers. In fact computers have become so useful now that it is impossible to imagine a life without them.

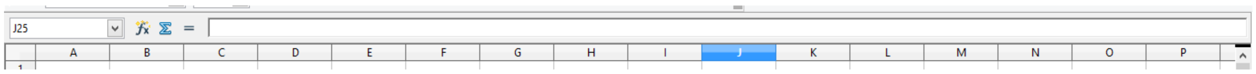
Openoffice Calc

- Openoffice Calc is just an open source version equivalent of Microsoft Office Excel. By open source we mean that the software is free, and can be redistributed.
- In this session, we will get familiar with Openoffice Calc and make it useful to us. So, now lets get started.
- First, the interface.
- The toolbar. Here we have the options to edit, insert etc

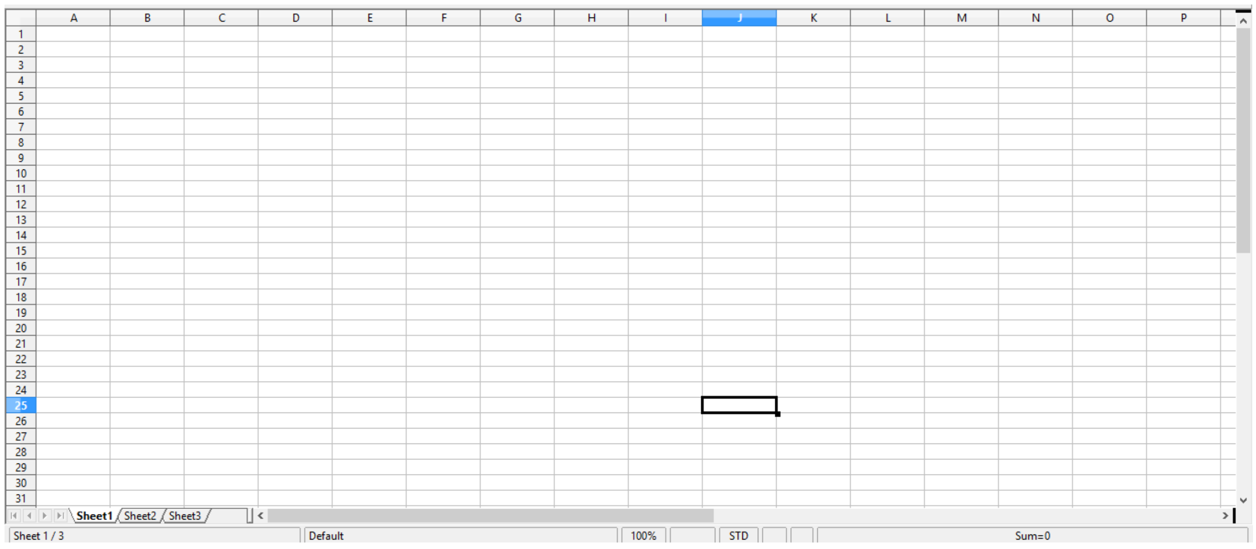




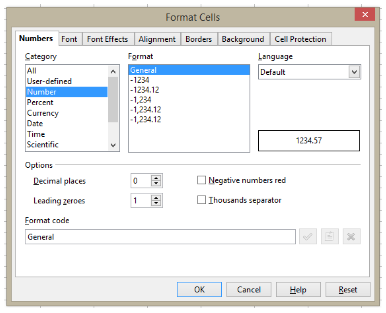
- The function and formatting toolbar. The function toolbar include the options to create a new sheet, open a file, print, cut, paste etc. The format toolbar includes the options to format the inputs given into the sheet.



- The formula toolbar. Here we enter the expression so that the calc will evaluate them and give the output we need.



- The Spreadsheet. This is where we would be working on mostly. The datas are written in this.
- Each and every rectangle in this spreadsheet is called as cell. We can enter anything - a character or a number or a combination of both by selecting the cell.
- Now, we will learn the functions that Calc provides us.
- The first is Format cells. This tells the calc what the column corresponding to that cell should only contain. We can do it by right clicking the cell and selecting the data type from the choices provided in the left column



- The next one is find and replace. In the function toolbar, there's a binocular icon. That corresponds to "find and replace" function. It will find the cell that has the value given by you and replaces it with another value provided by you. By default it is not case sensitive in case if you are finding a word and replacing it.
- Now we come to data validation. Suppose we want to restrict the values that can be given in a particular column or a set of column. For that we will use Data Validation. It can be accessed from **Data->Validity**. There we can set the type of data. And the range of the cells in which these properties are applicable. Moreover, we can tell the Calc to display the error message whenever invalid values are entered.
- Now, the functions like count and sum. The count function allows us to calculate the number of input data given in a particular column. The sum function calculates the sum of all the values given in a column.

- We can also sort the datas in a column. It can be done easily by selecting a particular and then clicking on either sort in ascending order or descending order button in the function toolbar.
- So we have entered the datas so far. In the Calc we see columns and rows drawn in it. However that’s not how it is going to appear in the paper when we print. It will only show the values we have entered in it on a plain sheet. To draw the tables around the variable we have draw borders function in the format toolbar. First select the cells around which the border has to be drawn and then select the draw Border function. It will show list of how the borders can be drawn. Choose the required kind and it will be drawn around the selected cells.
- Now, the Charts. Charts are what make calc more useful. It allows us to compare the datas of different column easily by making use of charts. To draw a chart out of the datas of a single column select that particular column and then go to **Insert->Chart** in the toolbar. The chart selection window appears from which we can select the type of chart we want. We can combine multiple columns and compare their graphs. To do it just select multiple columns and go to **Insert->Chart** in the toolbar. If we want to plot an X-Y graph, ie graph of the values in one column against the values of other column. We can select those two columns and then open the chart menu from the toolbar and then select XY graph from the choices given. Note that the first column will be taken as X values when plotting the graph.

Module - 4: Introduction to Programming

- After learning about the basics of using a computer, we come to the next part, a very important and useful part called programming.
- On its own, a computer isn't very smart. A computer is essentially just a big bunch of tiny electronic switches that are either on or off. By setting different combinations of these switches, you can make the computer do something, for example, display something on the screen or make a sound. That's what programming is at its most basic—telling a computer what to do. But to tell the computer to do something, you need to know its language. For example, we understand malayalam, english, some of us knows hindi, tamil etc. But you can't go to a Bengali and speak to him in Malayalam! Similarly, to speak to computers, you need to speak in their language. There are many such languages, called programming languages, and today we are going to learn about a very easy and useful one called C language.
- C was developed in 1970s, and many operating systems, including Linux and UNIX, are made using this language. It is procedural, which means the programs are written as step by step instructions. To create a C program, you need two things, a text editor program and a compiler. A text editor like MS Word or Notepad is used to type your program in and save it. A compiler is used to convert the code we wrote into machine language, which the computer can understand and execute. C is very easy to learn, and quite fun too, so lets get started!

- **BASIC RULES**

  1. In a C program, each statement should be written in each line in order

  2. All statements should written in small case letters.

  3. Every statement must end with ; (semicolon).

DATA VARIABLES

Most C programs includes the programmer giving the computer data or information, and the computer works with it and does what the programmer tells it to do. So, the basic unit of C are data variables. A data variable is like a bucket or a box which is named and is using for storing things. You can put data and values into these 'buckets' and when the program needs these data, they obtain it from the variables or 'buckets'. For example, consider a C program where you want to add two numbers. Here, you create 2 variables, **a** and **b**, and a third variable **c** for the answer. Now suppose you want to add 4 and 5. On entering the numbers, the program will put 4 in the variable/'bucket' **a** and 5 in **b**. After adding up, the program will put the answer in **c**. Now if you had put 6 and 7, then 6 would be put in a and 7 in b. So, the values or data in a variable may change, but the name of the variable doesn't, just like you can put different things in a buckets, but the bucket won't change at all!

- **DATA TYPES:**
  The type and amount of data you can store in a variable depends on its type. Some of the most commonly used variable types are as follows:
  1. Integer - an integer variable used to store numbers, i.e. whole numbers (despite the name). An integer variable is denoted by **int.** Int can only store data of size 2 bytes in it.
  2. Floating point - **float** data type is used to store decimals and values of size 4 bytes maximum.
  3. Char - **char** is used to store characters, such as letters, numbers and symbols of size limit 1 byte.
  C also includes many other data types such as double, long int, signed and unsigned char etc..
- **CREATING AND DECLARING A VARIABLE**
  Now we have studied about what variables are and the different types of variables, let us take a look at how to create a variable, how to declare it and use it in a C program.
  Firstly, you should name a variable. A variable name can be a combination of letter, numbers and symbols, but the first character should be a letter or the underscore (_) symbol. For eg, **age** and **_yz** are valid variable names but **2er** or **%og** isn't.
  Next, we declare or create the variable in the program, along with it's type. For example,
  ```
  int x;
  ```
  Here, with this statement, we create a variable called x, which can be used to store integers. Now to assign a value or put data into our 'bucket' x,
  ```
  x = 5;
  ```
  We can combine these two statements to create something like this:
  ```
  int x = 5;
  ```
  Like in maths, we can also assign the value of one variable to another.
  ```
  int x,y;
  x = 4;
  y = x;
  ```
  Here, two integer variables are declared i.e. x and y. X is assigned the value 4. After that the value in x is assigned to y.
- Using variables, one can do many arithmetic operations such as addition, subtraction, increment, decrement etc. We will look at these in the next parts.

Module - 5: Input from the User

- A computer would be pretty useless without some way to talk to the people who use it. The computer can’t think on it’s own. So, we have to make use of programming language to help the computers interact with the user.
- Now, let us compare ourselves with the computers. If we want to do any tasks we need some information or raw data about the tasks assigned to us. Similarly, the computers also need data to accomplish tasks. The computer thus gives back the information based on the data given to it just like how we come out with result after we do the task.
- Now, let’s get familiar with the keywords which we will be using in computer classes. **'Input'** is information supplied to the computer. **'Output'** is information provided by the computer.
- We will be dealing with inputs that are to be taken from the user and modifying it depending on the required output by using programs.
- In the programs, we take inputs using the **scanf()** function.
- The header file **<stdio.h>** is included in the program for the functions scanf() and printf() to work. That is, in order to enable the input and output commands to work. Header file is what’s included within the angular bracket of **#include** at the beginning of the code.
- There are certain syntax to be followed when writing scanf(). In the scanf(), we use the input data type specifiers to let the computer know what kind of data the user is going to give. Some of them are identified as“**%d**” (for **integer** data type), ”**%f**” (for **float** datatype), “**%c**” (for **char** data type), “**%s**“ (for **strings**). Also, we use ampersand symbol(**&**) before the variable name after the comma followed by the data type specifier. For example, in order to take the integer input, we write it as **scanf(“%d”,&integervariable);** Similarly for taking character input, we use **scanf(“%c”,&charactervariable);**
- Now lets start using the code to manipulate the input given by the users. It’s simple!
- **Manipulating Input Datas:**

  Sample programs like: changing the sign of a number, incrementing/decrementing a number, printing the next character of a given input character, printing the product/sum/difference of 2 numbers.

(Codes in the next page)

Code for the Sample Programs:

- **Changing the sign of the number:**

```
#include<stdio.h>
int main()
{
        int number,negative;
        printf(“Enter the number: ”);
        scanf(“%d”,&number);
        negative = number*-1;
        printf(“The number with changed sign is: %d“,number);
```

-

```
        return 0;
}
```

- **Incrementing (Or decrementing) a number:**

```
#include<stdio.h>
int main()
{
        int input,output;
        printf(“Enter the number: ”);
        scanf(“%d”,&input);
        output= input+1;              //change the operator ‘+’ to ‘-’ for decrementing.
        printf(“The new number is: %d“,output);
        return 0;
}
```

- **Printing the next character of a given input character:**

```
#include<7>
int main()
{
        int input,output;
        printf(“Enter the character: ”);
        scanf(“%d”,&input);
        output= input+1;            //change the operator ‘+’ to ‘-’ for previous character
        printf(“The new character is: %d“,output);
        return 0;
}
```

- **Printing the Sum of two numbers:**

```
#include<stdio.h>
int main()
{
        int number1, number2, sum;
        printf(“Enter the two numbers: ”);
        scanf(“%d”,&number1);
        scanf(“%d”,&number2);
        sum = number1+number2;
        printf(“The sum of given two numbers is: %d“,sum);
        return 0;
}
```

- **Printing the Difference of two numbers:**

```
#include<stdio.h>
int main()
{
        int number1, number2, difference;
        printf(“Enter the two numbers: ”);
        scanf(“%d”,&number1);
        scanf(“%d”,&number2);
        sum = number1-number2;
        printf(“The difference of given two numbers is: %d“,difference);
        return 0;
}
```

- **Printing the Product of two numbers:**

```
#include<stdio.h>
int main()
{
        int number1, number2, product;
        printf(“Enter the two numbers: ”);
        scanf(“%d”,&number1);
        scanf(“%d”,&number2);
        product = number1*number2;
        printf(“The product of given two numbers is: %d“,product);
        return 0;
}
```

Module - 6: Conditional Statements

- In certain situations a programmer requires set of statements to be executed if certain conditions are satisfied and certain other statements to be executed if these conditions are not satisfied. Decision control statements such as if, if –else, conditional statement help a programmer in doing exactly this.
- **If statement**-General syntax

```
if (this condition is true)
  {execute these statements;}
```

If the condition inside the brackets is true then the statements given below it are executed,

otherwise these statements are skipped. Conditions are expressed using relation statements (==,!=,<,>,<=,>=).Explain each relationship statement in brief. Multiple statements can be executed after an 'If' statement by giving them in braces, single statements don't require braces. Example-

```
main()
{int num;
printf("Enter a number");
scanf("%d",&num);
if (num<=10)
    printf("The number entered is less than 10");
}
```

- **If –else statement**- In the above example, as we can see there is no way to print a statement saying the number is greater than 10 if it is .To do this we need an else statement after the if. The ‘if-else’ statement executes a set of statements given after the ‘if’ if the conditions following it are satisfied, or else it executes the statements given after ‘else’, unlike the ‘if’ statement which does nothing if the conditions are not satisfied.

General syntax-

```
If (this condition is true)
  {execute statements;}
Else
  {execute these statements;}
```

Statements following if are called the if block and the statements following the else are called the else block. Conditions and statements following ‘if’ or ‘else are given in a manner similar to that of the ‘if’ statement. Example-

```
main()
{int a, b;
printf(“Enter three numbers”);
scanf(“%d%d”,&a, &b);
if (a>b)
    printf(“%d”, a, ” is greater than ”,”%d”, b);
else
    printf(“%d”, b, ” is greater than ”,”%d”, a);
}
```

· **Nested if-else statements**-It is possible to include if-else statements inside the body of the if or else part of another if-else statement.example-

```
main()
{int a, b, c;
printf("Enter three numbers");
scanf("%d%d%d",&a, &b, &c);
if (a >b)
{if (a>c)
 printf("%d", a, " is the greatest number");
else
 printf("%d", c, " is the greatest number");
}
else
{if(b>c)
 printf("%d", b, " is the greatest number");
else
 printf("%d", c, " is the greatest number");
}}
```

· **Use of logical operators**-the three logical operators are

1.**&&**-AND

2.||-OR

3!-NOT

An if statement using the AND operator to combine two conditions is executed only if both the conditions evaluate to true. An if statement using the OR operator to combine two conditions is executed if either of the two conditions evaluate to true.An if statement uses the NOT operator before a condition then it is executed if the condition evaluates to false The AND and OR operators are binary operators while the NOT operator is a unary operator. These can be used in combinations to combine two or more conditions and can help in reducing the number of if-else statements. Example-insurance of driver using logical operators

```
main()
{char ms,sex;
int age;
printf (“Enter the age,aex and marital status”);
scanf (“%d%c%c”&age,&sex,&ms);
if((ms==’M’)||(ms==’U’&&sex==’M’&&age>30)||(ms==’U’&&sex==’F’&&age>2
5))
  printf(“Driver is insured”);
else
  printf(“Driver is uninsured”);}
```

· **Hierarchy of logical operators-**

```
!
*  /  %
+  -
<  >  <= >=
```

== !=

&&

||

=

The logical operators are evaluated in order of their hierarchy with the not statement being of the highest priority. Example-

!a||b>c&&b>a

will be evaluated in the following order

((!a)||((b>c)&&(b>a)))

· **Conditional operators**-Syntax

Expression 1 ? expression 2: expression 3

If expression one is true then expression two is executed or else expression two is executed.

Example

int x, y;

scanf("%d",&x);

y=(x> 5 ? 3 : 4);

This statement will store 3in y if x is greater than 5, otherwise it will store 4 in y.

· **Switch statement-**The control statement which allows us to choose from a number of choices is called a switch statement.Syntax

Switch (integer expression)

{case constant 1:

do this;

```
    break;
 case constant 2:
    do this;
    break;
 case constant 3:
    do this;
    break;
default:
    do this;
    break;
}
```

The integer expression is evaluated and compared with each of the constants. If it is equal to any of the constants then the statements following that constant are executed. Or else the default statements are executed. If the break statement is absent then all the cases following it are executed until the next break statement is encountered or the switch block comes to an end. Char or integer values can be given to the switch statement or the cases since the characters are stored as integer values in the memory. Float values cannot be given to a switch statement.

Example-

```
main()
{int i;
printf("enter a number");
scanf(“%d”, i);
{case 1:
    printf(“the number entered was 1”);
    break;
case 3:
    printf(“the number entered was 3”);
    break;
case 4:
    printf(“the number entered was 4”);
    break;
default:
    printf(“the number entered was not 1, 3 or 4”);
    break;
}}
```

Module - 7: Loop Statements

- There may be a situation, when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.
- · Programming languages provide various control structures that allow for more complicated execution paths.
- · C programming language provides the following types of loop to handle looping requirements.
    - While loop
    - For loop
    - Do..While loop

WHILE LOOP

- · A **while** loop statement in C programming language repeatedly executes a target statement as long as a given condition is true.
- · **Syntax:**
- The syntax of a **while** loop in C programming language is:

```
while(condition)
{
   statement(s);
}
```

- Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any nonzero value. The loop iterates while the condition is true.
- When the condition becomes false, program control passes to the line immediately following the loop.
- Here, key point of the *while* loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

- Example:

```
#include <stdio.h>
int main ()
{
   /* local variable definition */
   int a = 10;
   /* while loop execution */
   while( a < 13 )
   {
            printf("value of a: %d\n", a);
            a++;
   }
   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
value of a: 10
value of a: 11
value of a: 12
```

FOR LOOP

- A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.
- Syntax:

The syntax of a **for** loop in C programming language is:

```
for ( init; condition; increment )
{
  statement(s);
}
```

Here is the flow of control in a for loop:

1. The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
2. Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.
3. After the body of the for loop executes, the flow of control jumps back up to the **increment**statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
4. The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the for loop terminates.

DO WHILE

Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop in C programming language checks its condition at the bottom of the loop.

A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax:

- The syntax of a **do...while** loop in C programming language is:

```
do
{
  statement(s);
}while( condition );
```

1. Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested.

2. If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop execute again. This process repeats until the given condition becomes false.

Module-8: Arrays

- Suppose we have many datas of the same type to be stored. Like marks of students, for example. It'd be a burden to create variables for each and every student who is present in the class.
- So, in order to resolve this problem of not being able to store multiple datas of same type together. We have arrays. Arrays are just modification of a normal variable. Here we have a chain of memory spaces to store it continuously under one name rather than just a single memory space as it is for a variable. It's easy to use. Now lets get familiar with it.
- To create it we have to follow a syntax for creating the array. The syntax is as follows:

  datatype variable_name[array size];

  For example,

  ```
  int marks[6];
  float percentage[10];
  ```

  Here, we have created an array named marks of size 6. In this we can store 6 elements of integer data type. Similarly we have created array percentage, but the data type is float and can store 10 elements.

  Note that all the elements in the array are of same type. It can be int, float, char. Array size mentions the number elements in the array, ie, the number of elements to included under a single name.
- So, we have created the array. Now, we need to access each elements in the array. Each element in the array are referred by using index that refers to it's position. We call the number that corresponds to it's position as **subscript** of the array. In arrays, the subscript starts from **0** till (**array_size - 1**). So in order to access the first element, the subscript to be used is 0. And the nth element by using subscript as n-1.

  For example, to print the 3rd element of the array marks we use:

  ```
  printf("%d",marks[2]);
  ```
- So far we have seen the arrays using numbers. Now lets come to array of characters. We call them as **strings**. The only difference between strings and an array of numbers is the data type and it's size limit. Look at the example below to get an idea of it:

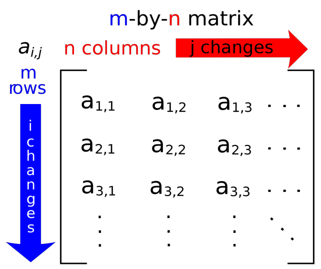
char name[6];

Here, we can store a word of 5 characters maximum. Here the size is one less than the size mentioned because the position after the last character is always reserved for null character. Null character is written by “\0”. This only marks the end of the word.

So, if we want to create a variable that store the name “Madhavankutty” we need to create a character array of size 14 so that the 13 characters of that name can be stored.

- The kind of array that we have seen are 1D array. That is they are linear chains of single data types. There’s another kind of array. It’s 2D array.
- 2D array is just the same as matrix. A matrix is just a rectangular arrangement of numbers in which each number’s position is represented using 2 indices. Hence we will make use of 2 subscripts in the array. The syntax to create such an array is:

$$\begin{bmatrix} 1 & 2 & 8 \\ 10 & 3 & 9 \\ 7 & 4 & 0 \end{bmatrix}$$


m-by-n matrix
$a_{i,j}$
n columns
j changes
m rows
i changes
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,1}$ $a_{3,2}$ $a_{3,3}$

int matrix[m][n];

Here we created a matrix having m rows and n columns which can store m. The way how the indices for each elements in the matrix is as shown in the figure (right figure). However in the arrays, we have to subtract both indices by 1 for it’s position just as we had to do in 1D array. Similarly, we can can create an array of float and double data types.

- However, when it comes to 2D array of characters. We are making a list for the strings to be stored in it. We can make use of this to store names in a list under a single variable name.

For example,

char names[15][31];

Here, we can store 15 names having maximum of 30 characters. The first subscript corresponds to size of the list and the second subscript corresponds to the string size which is always (**max. word size + 1**). However, the indexing of each character here is the same as that of the 2D integer arrays. For example, we can access the second character or the 3rd word in the list by using **names[2][1]**.

- Now lets start using codes.
    - Program to take input of marks of a students and calculate the total marks:

```
#include<stdio.h>
int main()
{
        int marks[5],total=0;
        printf("Enter the marks scored in 5 subjects: ");
        for(int i=0; i<5; i++)
                scanf("%d",&marks[i]);
        for(i=0;i<5;i++)
                total=total+marks[i];
        printf("The total marks obtained by the student is: %d",total);
        return 0;
}
```

    - Program to find the mean of the given discrete datas:

```
#include<stdio.h>
int main()
{
        int data[100],sum=0,number;
        float mean;
        printf("Enter the number of datas: ");
        scanf("%d",&number);
        printf("Enter the datas: ");
        for(int i=0;i<number;i++)
        {
                scanf("%d",&data[i]);
                sum=sum+data[i];
        }
        mean=(float)sum/(float)number;  //do give small explanation of type conversion
        printf("The mean of the given datas is: %f",mean);
        return 0;
}
```

- <u>Program to print the name to replace the vowels with space in the given string:</u>

```
#include<stdio.h>
int main()
{
        char string[101];
        printf(“Enter the string: “);
        scanf(“%s”,&string);
        for(int i=0; string[i]!=’\0’;i++)
                switch(string[i])
                {
                        case ‘A’:
                        case ‘a’:
                        case ‘E’:
                        case ‘e’:
                        case ‘I’:
                        case ‘i’:
                        case ‘O’:
                        case ‘o’:
                        case ‘U’:
                        case ‘u’:
                                string[i]=’ ‘;
                                break;
                }
        printf(“The new string is: %s” , string);
}
```

- <u>Calculating the sum of 2 matrices (fixed size)</u>

```
#include <stdio.h>
int main()
{
  int a[2][2], b[2][2], c[2][2];
  int i,j;
  printf("Enter the elements of 1st matrix\n");
  for(i=0;i<2;++i)
        for(j=0;j<2;++j)
        {
                scanf("%d",&a[i][j]);
        }
printf("Enter the elements of 2nd matrix\n");
for(i=0;i<2;++i)
        for(j=0;j<2;++j)
        {
                scanf("%d",&b[i][j]);
        }
  for(i=0;i<2;++i)
        for(j=0;j<2;++j)
        {
                c[i][j]=a[i][j]+b[i][j];
        }
  printf("Sum Of Matrix:\n");
  for(i=0;i<2;++i)
  {
        for(j=0;j<2;++j)
                printf("%d\t",c[i][j]);
         printf("\n");
   }
return 0;
}
```

Module - 9: User Defined Functions

- **Introduction**-As a single person cannot performs all of life’s task alone, he has to rely on others. A mechanic repairs bikes, architects design houses or a store provides groceries every month. In a similar manner a single computer program cannot handle all the tasks by itself. It requests other program entities called `functions’ to get its task done.
- **What is a Function**-A function is a group of statements that perform a particular job. These statements are grouped together under a single name. This name can then be used to call the function repeatedly to perform the same function again and again.Example-

```
main()
{message();
printf(“\nIn main”);
}
message()
{printf(\nIn message”);
}
```

OUTPUT-

```
In message
In main
```

Here main() itself is a function which calls the function message(). `Call’ means the function main() passes control to the function message().This means that the main() function stops working for some time while the message function runs and the main() function resumes its work. A function gets called when its name is followed by a semicolon.

The program execution always begins with function named main(). The rest of the functions are executed in the order in which they are called in main(). A program can contain any number of functions which can be called any number of times.

Example-

```
main()
{printf("\nI am in main");
Italy();
Brazil();
}
Italy()
{printf("\nI am in Italy");
}
Brazil()
{printf("\nI am in Brazil");
}
```

OUTPUT-

```
I am in main
I am in Italy.
I am in Brasil.
```

- **Function Definition-**A function is defined when the name of the function is followed by a pair of braces in which one or more statements may be present. The function definition is thus used to tell the compiler what to do when the function is called. Example-

  ```
  Italy()
  {printf("\nI am in Italy");
  }
  ```

  Thus the definition of the function Italy tells the compiler to print `I am in Italy' when the function is called.

- **Reasons to Use Functions-** Writing functions helps us to avoid rewriting the same code over and over. Instead we can just call the function repeatedly. Using functions also makes writing a program easier as the program is divided into separate subprograms which can be written individually and then combined together to execute the whole program.

· **Types of Functions-**The two types of functions are

1. **Library functions**
2. **User Defined functions**

Library functions include the commonly used input/output functions like scanf(),printf() etc. These functions have already been defined by the people who wrote the compiler and are stored on the disk in what is called a library.

On the other hand user defined functions are defined by programmers like us. Examples include fuctions like message(), Italy(), etc.

· **Passing values between functions**-Example

```
main()
{int a, b, c, sum;
printf("\nEnter any three numbers");
scanf("%d%d%d", &a, &b, &c);
sum = calsum ( a, b, c);
printf("\nsum=%d", sum);
}
calsum( x, y, z)
int x, y, z;
{int d;
d=x+y+z;
return(d);
}
```

OUTPUT

```
Enter any three numbers 10 20 30
sum = 60
```

In this program the values for a, b, c are taken through main() function. When main() calls calsum(), the values of a, b, c are used in place of x, y, z respectively. This is known as passing values. Thus the values of a, b, c are passed to calsum by mentioning them in the parenthesis when the function is called. The variables in the parenthesis are known as arguments. The arguments given in the function definition are known as formal arguments and the ones used while calling the function are known as actual arguments. Any number and type of arguments can be passed to a function. But while calling the function the arguments must be given in the same order and type as in the function definition. Formal arguments can be declared in two ways.

```
calsum(x, y, z)
int x, y, z;
```

or

```
calsum(int x, int y, int z)
```

As seen in the above program we have to use a return statement to return values to the calling function. The return statement also terminates the function. By default all functions can return integers. If we wish the function to some other type of data, we need to specify the return type of the data. Example-

```
float square(int a)
{float s;
s=a*a;
return s;
}
```

If he called function doesn’t return anything we must mention it by using the keyword void.

```
void display()
{printf*(“Display function doesn’t return anything”);}
```

A function can return only one value at a time.

If the value of a formal argument is changed in the called function then the corresponding change does not take place in the changed function. Example-

```
main()
{int a =30;
fun (a);
printf ("\n%d",a);
}
fun(int b)
{b=60;
printf("\n%d", b);
}
```

OUTPUT

```
60
30
```

Thus even though the value of ‘b’ is changed in fun() the value of ‘a’ remains unaffected.

- A function prototype is a statement that gives the name, return type and type of variables passed to the function. Example-

  ```
  float square(int x);
  ```

  The prototype of square should be given in all the functions that call it. If a large number of functions call square() then it is better to give its prototype once, outside all the functions as this will enable any function to call it.

- **Scope rules-**Example-

  ```
  int a
  main()
  {int I=20;
  ```

```
display(i);
}
display(int j)
{int k=35;
printf("\n%d", j);
printf("\n%d", k);
printf("\n%d", a);
}
```

In this it is necessary to pass the variable ‘i’ to the function display() as ‘i’ is not in the scope of the function. A variable is said to be in the scope of a function if it is declared inside the function calling it. The variable or is said to have scope local to the function. If a variable is declared outside all the functions then it is said to have global variable and doesn’t need to be passed to the function. The variable ‘a’ is a global variable.

- **Calling Convention(optional)**- This indicates the order in which the arguments are passed to the function when the function is called. In C the arguments are passed from the right to the left.Example-

```
int a=1;
printf("%d%d%d", a, ++a, a++);
```

OUTPUT

```
3 3 1
```

The a++ is executed first and then the ++a and finally the a in the function call of printf().