C Language Tutorial For Beginners (With Notes)

I have written these notes, practice sheets, and source code just for you. These are available for free to download and use. However, some bad people may start selling this material to some innocent students for some money! If you think this material helps, please share this video with everyone and spread the word that this video and notes are free!

C programming is the best way to learn to code and get your journey started as a programmer. This video is a 15 Hour long course that will teach you the C language from the basics to the very advanced level. We will make two fun games during the course which you can even play during your

free time. There are practice sets included along with Free Handwritten notes for all the chapters. You can download them from the links below.

Note: You can either download the notes, or you can read them on this site itself. PDF download links for all the chapters are at the end of the page

Chapter 0: Introduction

What is Programming?

Computer programming is a medium for us to communicate with computers, just like we use Hindi or English to communicate with each

other. Programming is a way for us to deliver our instructions to the computer.

What is C?

C is a programming language. C is one of the oldest and finest programming languages. C was developed by Dennis Ritchie in 1972.

Uses of C

C is a language that is used to program a wide variety of systems. Some of the uses of C are as follows:

Major parts of Windows, Linux, and other operating systems are written in C.
C is used to write driver programs for devices like Tablets, Printers, etc.
C language is used to program

embedded systems where programs need to run faster in limited memory. C is used to develop games, an area where latency is very important, i.e., a computer has to react quickly to user input.

Chapter 1: Variables, Constants, and Keywords:

Variables

A variable is a container that stores a ‘value.’ In the kitchen, we have containers storing rice, dal, sugar, etc. Similar to that variable in c stores the value of a constant. Example:

a = 3 a is assigned “3”
b = 4.7 b is assigned “4.7”
c = 'A' c is assigned “A”

Rules for naming variables in c:

1. The first character must be an

alphabet or underscore(_).

2. No commas or blanks are allowed.

3. No special symbol other than underscore is allowed

4. Variable names are case sensitive

Constants
An entity whose value doesn’t change is called a constant.

Types of constant
Primarily there are 3 types of constant:

1. Integer Constant -1,6,7,9

2. Real Constant -322.1,2.5,7.0
3. Character Constant ‘a’,’$’,’@’(must be enclosed within single inverted commas)
Keywords
These are reserved words whose meaning is already known to the compiler. There are 32 keywords available in c:

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

Our first C program
#include<stdio.h>

```
int main() {

printf(“Hello, I am learning C with
Harry”);
return 0;

}
```
File : first.c

The basic structure of a C program
All c programs have to follow a basic structure. A c program starts with the main function and executes instructions presents inside it. Each instruction terminated with a semicolon(;)

There are some basic rules which are applicable to all the c programs:

Every program's execution starts from the main function.
All the statements are terminated with a semi-colon.
Instructions are case-sensitive.
Instructions are executed in the same order in which they are written.
Comments
Comments are used to clarify something about the program in plain language. It is a way for us to add notes to our program. There are two types of comments in c:

Single line comment: //This is a comment.
Multi-line comment : /*This is multi-line comment*/

Comments in a C program are not executed and ignored.

Compilation and execution

A compiler is a computer program that converts a c program into machine language so that it can be easily understood by the computer.

A program is written in plain text. This plain text is a combination of instructions in a particular sequence. The compiler performs some basic checks and finally converts the program into an executable.

Library functions
C language has a lot of valuable library functions which is used to

carry out a certain task; for instance, printf function is used to print values on the screen.

printf(“This is %d”,i);

// %d for integers

// %f for real values

// %c for characters

Types of variables
Integer variables int a=3;
Real variables int a=7.7 (wrong as 7.7 is real) ; float a=7.7;
Character variables char a=’B’;

Receiving input from the user
In order to take input from the user and assign it to a variable, we use scanf function.

The syntax for using scanf:

scanf(“%d”,&i); // [This & is important]
& is the “address of” operator, and it means that the supplied value should be copied to the address which is indicated by variable i.

Chapter 1: Practice Set:
Q1. Write a c program to calculate the area of a rectangle:

a) using hardcoded inputs &

b) using inputs supplied by the user

Q2. Calculate the area of a circle and modify the same program to calculate the volume of a cylinder given its radius and height.

Q3. Write a program to convert Celsius (Centigrade degrees temperature to Fahrenheit)

Q4. Write a program to calculate simple interest for a set of values representing principle, no of years, and rate of interest.

Chapter 2: Instructions and Operators:

A C-program is a set of instructions. Just like a recipe - which contains instructions to prepare a particular dish.

Types of instructions:
1.Type declaration instruction

2. Arithmetic instruction

3.Control instruction

Type of declaration instruction:
int a;
float b;

other variations:

int i = 10; int j = i, int a = 2;
int j1 = a + j - i;

float b = a+3; float a = 1.1;
==>Error! As we are trying to use a before defining it.

int a,b,c,d;

a=b=c=d=30; ==> Value of a,b,c & d will be 30 each.
Arithmetic Instructions

Note:

1.No operator is assumed to be present

int i=ab (Invalid)

int i=a*b (valid)

2.There is no operator to perform exponentiation in c however we can use pow(x,y) from <math.h>(More later).

Type conversion
An Arithmetic operation between

int and int ==> int
int and float ==> float
float and float ==> float

5/2 --> 2 5.0/2 --> 2.5 // IMPORTANT!!
2/5 --> 0 2.0/5 --> 0.4 // IMPORTANT!!

NOTE:

int a = 3.5; //In this case, 3.5 (float) will be denoted to a 3 (int) because a cannot store floats.

float a = 8; // a will store 8.0 [8-->8.0(Promotion to float)]
Quick Quiz:

Question- int k=3.0/9 value of k? and why?

Solution- 3.0/9=0.333, but since k is an int, it cannot store floats & value 0.33 is demoted to 0.

Operator Precedence in C
3*x-8y is (3x)-(8y) or 3(x-8y)?

In the c language, simple mathematical rules like BODMAS no longer apply.

The answer to the above question is provided by operator precedence & associativity.

Operator precedence

The following table list the operator priority in C

Priority	Operators
1st	* / %
2nd	+ -
3rd	=

Operators of higher priority are evaluated first in the absence of parenthesis.

Operator associativity

When operators of equal priority are present in an expression, the tie is taken care of by associativity

x * y / z => (x * y) / z
x / y * z => (x / y) * z

*, / follows left to right associativity.
Control instructions
Determines the flow of control in a program.

Four types of control instruction in C are:

1. Sequence Control Instruction

2. Decision Control Instruction

3. Loop Control Instruction

4. Case-Control Instruction

Chapter 2: Practice Set

Q1. Which of the following is invalid in c?

1. int a; b=a;

2. int v=3^3;

3. char dt= '21 Dec 2020' ;

Q2. What data type will 3.0/8 – 2 return?

Q3. Write a program to check whether a number is divisible 97 or not.

Q4. Explain step by step evaluation of 3*x/y-z +k

Where x=2, y=3, z=3 and k=1

Q5. 3.0+1 will be:

Integer
Floating number
Character

Chapter 3: Conditional Instructions
Sometimes we want to watch comedy videos on youtube if the day is Sunday. Sometimes we order junk food if it is our friend's birthday in the hostel. You might want to buy an umbrella if it's raining and you have the money. You order the meal if dal or your favorite bhindi is listed on the menu.

All these are decisions that depend on conditions being met.

In ‘C’ language, too, we must be able to execute instructions on a condition(s) being met.

Decision-making instructions in C
If-else statement
Switch statement
If-else statement
The syntax of an if-else statement in c looks like this:

```
if ( condition to be checked) {

Statements-if-condition-true ;

}

else{
```

statements-if-condition-false ;

}

Code Example

```
int a=23;
if (a>18){
printf(“you can drive\n”);
}
```

Note that else block is not necessary but optional.

Relational Operators in C

Relational operators are used to evaluate conditions (true or false) inside the if statements. Some examples of relational operators are:

== equals to
>= greater than or equal to
> greater than
< less than
<= less than or equal to
!= not equal to
Important Note: '=' is used for an assignment, whereas '==' is used for an equality check.

The condition can be any valid expression. In C, a non-zero value is considered to be true.

Logical Operators
&&, ||, and ! are the three logical operators in C. These are read as “and,””or,” and “not.” They are used to provide logic to our c programs.

Use of logical operators:

1. && (AND) is true when both the conditions are true

“1 and 0” is evaluated as false

“0 and 0” is evaluated as false

“1 and 1” is evaluated as true

2. || (OR) is true when at least one of the conditions is true. (1 or 0 = 1)(1 or 1 = 1)

3. ! returns true if given false and false if given true.

!(3==3) evaluates to false

!(3>30) evaluates to true

As the number of conditions increases, the level of indentation increases. This reduces readability. Logical operators come to the rescue in such cases.

Else if clause
Instead of using multiple if statements, we can also use else if along with if, thus forming an if-else if-else ladder.

```
if {

// statements ;

}

else if { //statements;

}
```

```
else {  //statements;

}
```

Using if-else if-else reduces indents. The last “else” is optional. Also, there can be any number of “else if.”

Last else is executed only if all conditions fail.

Operator Precedence

Priority	Operator
1st	!
2nd	*,/,%
3rd	+,-
4th	<>,<=,>=
5th	==,!=
6th	&&

7th ||
8th =

Conditional operators
A shorthand “if-else” can be written using conditional or ternary operators.

Condition ? expression-if-true ; expression-if-false
Here, '?' and ':' are Ternary operators.

Switch case-control instruction
Switch-case is used when we have to make a choice between the number of alternatives for a given variable.

Syntax,

Switch(integer-expression)

{

Case c1:

Code;

Case
c2: //
c1,c2,c3 are constants

Code; //
Code is any valid C code

Case c3:

Code;

default:

Code;

}
The value of integer-expression is matched against c1,c2,c3.....if it matched any of these cases, that case along with all subsequent “case” and “default” statements are executed.

Quick Quiz: Write a program to find the grade of a student given his marks based on below:

90-100 A
80-90 B
70-80 C
60-70 D
<70 F
Important notes

We can use switch case statements even by writing in any order of our choice

Char values are allowed as they can be easily evaluated to an integer
A switch can occur within another, but in practice, this is rarely done

Chapter 3- Practice Set

What will be the output of this program?

```
int a=10;

if(a=11)
printf(“I am 11”);
else
printf(“I am not 11”);
```

Write a program to find out whether a student is pass or fail; if it requires a total of 40% and at least 33% in each subject to pass. Assume 3 subjects and take marks as input from the user.

Calculate income tax paid by an employee to the government as per the slabs mentioned below:

Income Slab	Tax
2.5L-5.0L	5%
5.0L-10.0L	20%
Above 10.0L	30%

Note that there is no tax below 2.5L.
Take income amount as an input from the user.

Write a program to find whether a year entered by the user is a leap year or not. Take the year as input from the user.
Write a program to determine whether a character entered by the user is lowercase or not.
Write a program to find the greatest of four numbers entered by the user.

Chapter 4 - Loop Control Instruction

Why loops?

Sometimes we want our programs to execute a few sets of instructions over and over again, for eg. Printing 1 to 100, first 100 even numbers, etc.

Hence loops make it easy for a programmer to tell the computer that a given set of instructions must be executed repeatedly.

Types of Loops: Primarily, there are three types of loop in c language:

1.While loop

2.do-while loop

3.for loop

We will look into this one by one

While Loop
While(condition is true) {

// Code // The block keeps executing as long as the condition is true

// Code

}
An example:

int i=0;
while (i<10){
printf(“The value of i is %d”,i); i++;
}
Note:

If the condition never becomes false, the while loop keeps getting executed. Such a loop is known as an infinite loop.

Quick Quiz: Write a program to print natural numbers from 10 to 20 when the initial loop counter i is initialized to 0.

The loop counter need not be int, it can be a float as well.

Increment and decrement operators
i++ (i is increased by 1)

i-- (i is decreased by 1)

printf("—i=%d",--i);

This first decrements i and then prints it

printf(“i--=%d”,i--);
This first prints i and then decrements it

+++ operators does not exists => Important
+= is compound assignment operator just like -=, *=, /= & %= =>Also important
Do-while loop:
The syntax of do-while loop looks like this:

do {

//code;

//code;

}while(condition)
Do-while loop works very similar to while loop

While -> checks the condition & then executes the code

Do-while -> executes the code & then checks the condition

**do-while loop = while loop which executes at least once

For Loop
The syntax of for loop looks like this:

```
for( initialize; test; increment or
decrement)

{

//code;

//code;

}
```

Initialize -> setting a lop counter to an initial value

Test -> checking a condition

Increment -> updating the loop counter

An example:

```
for(i=0;i<3;i++)
```

```
{
printf(“%d”,i);
printf(“\n”);
}
```

Output:

0

1

2

Quick Quiz: Write a program to print first n natural numbers using for loop.

A case of Decrementing for loop

```
for(i=5; i; i--)

printf(“%d\n”,i);
```

This for loop will keep on running until

i becomes 0.

The loop runs in the following steps:

i is initialized to 5
The condition “i” (0 or none) is tested
The code is executed
i is decremented
Condition i is checked, and the code is executed if it's not 0.
& so on until i is non 0.
Quick Quiz: Write a program to print n natural numbers in reverse order.

The Break Statement in C
The break statement is used to exit the loop irrespective of whether the condition is true or false. Whenever a “break” is encountered inside the

loop, the controls are sent outside the loop.

Let us see with an example:

```
for (i=0; i<1000; i++){
printf("%d\n",i);
if (i==5){
break;
}
}
```

Output: 0, 1, 2, 3, 4, 5 and not 0 to 100

The continue statement in c

The continue statement in c is used to immediately move to the next of the loop. The control is taken to the next iteration, thus skipping everything below continue inside the loop for that iteration.

Let us look at an example:

```
int  skip=5;
int i=0;
while(i<10){
if(i  != skip)
continue;
else
printf(%d”,i);
}
```

Output: 5 and not 0.....9

Notes:

1. Sometimes, the name of the variable might not indicate the behavior of the program.

2. Break statement completely exits the loop

3. Continue statement skips the particular iteration of the loop

Chapter 4 - Practice Set

Write a program to print the multiplication table of a given number n.

Write a program to print a multiplication table of 10 in reversed order

A do-while loop is executed:

At least once

At least twice

At most once

4. What can be done using one type of loop can also be done using the other two types of loops – True or False.

5. Write a program to sum the first ten natural numbers using a while loop.

6. Write a program to implement program 5 using for and do-while loop.

7. Write a program to calculate the sum of the numbers occurring in the multiplication table of 8.(Consider 8x1 to 8x10)

8. Write a program to calculate the factorial of a given number using for loop.

9. Repeat 8 using a while loop.

10. Write a program to check whether a given number is prime or not using loops.

11. Implement 10 using other types of

loops.

Project 1: Number Guessing Game
Problem: This is going to be fun!! We will write a program that generates a random number and asks the player to guess it. If the player's guess is higher than the actual number, the program displays "Lower number please." Similarly, if the user's guess is too low, the program prints "Higher number please."

When the user guesses the correct number, the program displays the number of guesses the player used to arrive at the number.

Hints:

Use loops
Use a random number generator.
Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

int main(){
int number, guess, nguesses=1;
srand(time(0));
number = rand()%100 + 1; //
Generates a random number between
1 and 100
// printf("The number is %d\n",
number);
// Keep running the loop until the
number is guessed
do{
printf("Guess the number between 1
```

```
to 100\n");
scanf("%d", &guess);
if(guess>number){
printf("Lower number please!\n");
}
else if(guess<number){
printf("Higher number please!\n");
}
else{
printf("You guessed it in %d attempts
\n", nguesses);
}
nguesses++;
}while(guess!=number);

return 0;
}
```

Chapter 5 - Functions and Recursions

Sometimes our program gets bigger

in size, and its not possible for a programmer to track which piece of code is doing what.

The function is a way to break our code into chunks so that it is possible for a programmer to reuse them.

What is a function?
A function is a block of code that performs a particular task. A function can be reused by the programmer in a given program any number of times.

Example and syntax of a function:

```
#include<stdio.h>
void  display();                    // Function
prototype
int main(){
int a;
```

```
display();                    // Function
call
return(0);
}

void display(){               //
Function definition
printf("Hi I am display");
}
```

Function prototype:

Function prototype is a way to tell the compiler about the function we are going to define in the program.

Here void indicates that the function returns nothing.

Function call:

Function call is a way to tell the

compiler to execute the function body at the time the call is made.

Note that the program execution starts from the main function in the sequence the instructions are written.

Function definition:

This part contains the exact set of instructions that are executed during the function call. When a function is called from main(), the main function falls asleep and gets temporarily suspended. During this time, the control goes to the function being called when the function body is done executing main() resumes.

Quick Quiz: Write a program with three functions,

Good morning function which prints "Good Morning."
Good afternoon function which prints "Good Afternoon."
Good night function, which prints "Good night."
main() should call all of these in order 1 - 2 - 3.

Important Points:

Execution of a c program starts from main()
A c program can have more than one function
Every function gets called directly or indirectly from main()
There are two types of functions in c.

Let's talk about them.
Types of Functions:
Library functions: Commonly required functions grouped together in a library file on disk.
User-defined functions: These are the functions declared and defined by the user.
Why use functions?
To avoid rewriting the same logic again and again
To keep track of what we are doing in a program
To test and check logic independently
Passing values to functions:
We can pass values to a function and can get a value in return from a function

int sum(int a, int b)

The above prototype means that sum is a function which takes values a(of type int) and b(of type int) and returns a value of type int

Function definition of sum can be:

```
int   sum(int a, int b){

int c;                                  => a and b are parameters

c=a+b;

return c;

}
```

Now we can call sum(2,3) [here 2 and 3 are arguments]; from main to get 5 in return.

int d=sum(2,3); => d becomes 5

Note:

1. Parameters are the values or variable placeholders in the function definition. Ex: a & b

2. Arguments are the actual values passed to the function to make a call. Ex: 2 & 3

3. A function can return only one value at a time.

4. If the passed variable is changed inside the function, the function call doesn’t change the value in the calling function.

```
int change(int a){

a=77;                    => Misnomer

return 0;

}
```

change is a function which changes a to 77. No, if we call it from main like this.

```
int b=22;

change(b);                    => The value of b remains 22

printf("b is %d",b);          => prints "b is 22"
```

This happens because a copy of b is paused to the change function.

Quick Quiz: Use the library function to calculate the area of a square with side a.

Recursion
A function defined in C can call itself. This is called recursion.

A function calling itself is also called a recursive function.

Example of Recursion:

A very good example of recursion is factorial

factorial(n) = 1x 2 x 3.....x n

factorial(n)= 1 x 2 x 3.....n-1 x n

factorial(n)= factorial of (n-1) x n
Since we can write factorial of a number in terms of itself, we can program it using recursion.

// Function to calculate factorial using recursion

```
int factorial(int x){
int f;
if(x==0||x==1)
return 1;
else
f=x * factorial(x-1);
return f;
}
```

How does it work?

Important Notes:

1. Recursion is sometimes the most direct way to code an algorithm

2. The condition which doesn’t call the function any further in a recursive function is called the base condition.

3. Sometimes, due to a mistake made by the programmer, a recursive function can keep running without returning, resulting in a memory error.

Chapter 5- Practice Set
Write a program using functions to find the average of three numbers.

Write a function to convert Celcius temperature into Fahrenheit.
Write a function to calculate the force of attraction on a body of mass m exerted by earth. (g=9.8m/S2)
Write a program using recursion to calculate the nth element of the Fibonacci series.
What will the following line produce in a C program: printf("%d %d %d\n",a,++a,a++);
Write a recursive function to calculate the sum of first n natural numbers.
Write a program using functions to print the following pattern(first n lines):

*

***

*****

Chapter 6 - Pointers

A pointer is a variable that stores the address of another variable.

j is a pointer

j points to i.

The "address of" (&) operator

The address of operator is used to obtain the address of a given variable

If you refer to the diagrams above

&i=> 87994

&j=>87998

Format specifier for printing pointer address is ‘%u’

The "value of address" operator (*)
The value at address or * operator is used to obtain the value present at a given memory address. It is denoted by *

*(&i) = 72

*(&j) = 87994
How to declare a pointer?
A pointer is declared using the following syntax,

int *j; => declare a variable j of type int-pointer

j=&i   =>store address of i in j
Just like pointer type integer, we also

have pointers to char, float, etc.

int *ch_ptr; -> pointer to integer

char *ch_ptr; -> pointer to character

float *ch_ptr -> pointer to float
Although it's a good practice to use meaningful variable names, we should be very careful while reading & working on programs from fellow programmers.

A Program to demonstrate Pointers:

```
#include<stdio.h>
int main()
{
int i=8;
```

```
int *j;
j=&i;
printf(“Add i=%u\n”,&i);
printf(“Add i=%u\n”,j);
printf(“Add j=%u\n”,&j);
printf(“Value i=%d\n”,i);
printf(“Value i=%d\n”,*(&i));
printf(“Value i=%d\n”,*j);
return 0;
}
```

Output:

Add i=87994

Add i=87994

Add j=87998

Value i=8

Value i=8

Value i=8
This program sums it all. If you understand it, you have got the idea of pointers.

Pointers to a pointer:
Just like j is pointing to i or storing the address of i, we can have another variable, k which can store the address of j. What will be the type of k?

int **k;

k= &j;

We can even go further one level and create a variable l of type int*** to store the address of k. We mostly use

int* and int** sometimes in real-world programs.

Types of function calls
Based on the way we pass arguments to the function, function calls are of two types.

Call by value -> sending the values of arguments.
Call by reference -> sending the address of arguments
Call by value:

Here the values of the arguments are passed to the function. Consider this example:

int c = sum(3 , 4); => Assume x=3 and y=4
If sum is defined as sum(int a, int b), the values 3 and 4 are copied to a and b. Now even if we change a and b, nothing happens to the variables x and y.

This is call by value.

In C, we usually make a call by value.

Call by reference:

Here the address of the variable is passed to the function as arguments.

Now since the addresses are passed to the function, the function can now modify the value of a variable in calling function using * and &

operators. Example:

```
void swap(int *x, int *y)
{
int temp;
temp= *x;
*x = *y;
*y = temp;
}
```

This function is capable of swapping the values passed to it. If a=3 and b=4 before a call to swap(a,b), a=4 and b=3 after calling swap.

```
int main()
{
int a=3;    // a is 3 and b is 4
int b=4;
swap(a,b)
return 0;    // now a is 4 and b is 3
}
```

Chapter 6 - Practice Set

Write a program to print the address of a variable. Use this address to get the value of this variable.

Write a program having a variable i. Print the address of i. Pass this variable to a function and print its address. Are these addresses same? Why?

Write a program to change the value of a variable to ten times its current value. Write a function and pass the value by reference.

Write a program using a function that calculates the sum and average of two numbers. Use pointers and print the values of sum and average in main().

Write a program to print the value of a variable i by using the "pointer to

pointer" type of variable.
Try problem 3 using call by value and verify that it doesn’t change the value of the said variable.

Chapter 7 - Arrays

An array is a collection of similar elements.

One variable => Capable of storing multiple values

Syntax,

The syntax of declaring an array looks like this:

int marks[90]; => integer array

char name[20]; =>
character array or string

float percentile[90]; => float
array
The values can now be assigned to
marks array like this:

marks[0]=33;

marks[1]=12;
Note: It is very important to note that
the array index starts with 0.

Accessing Elements
Elements of an array can be accessed
using:

scanf(“%d”,&marks[0]); => Input first value

printf(“%d”, marks[0]); => Output first value of the array

Quick Quiz: Write a program to accept marks of five students in an array and print them to the screen.

Initialization of an array
There are many other ways in which an array can be initialized.

int cgpa[3]={9,8,8} => Arrays can be initialised while declaration

float marks[]={33,40}
Arrays in memory
Consider this array:

Int arr[3]={1,2,3} => 1 integer = 4 bytes
This will reserve 4x3=12 bytes in memory. 4 bytes for each integer.

1 2 3
62302 62306 62310
{Array in Memory}

Pointer Arithmetic
A pointer can be incremented to point to the next memory location of that type.

Consider this example

int i = 32;
int *a = &i; ==> a =87994 (address = 87994)
a++; ==> now a = 87998

char a = 'A';
char *b = &a; ==> b = 87994
b++; ==> now b = 87995

float i = 1.7;
float*a = &i; ==> Address of i or a = 87994
a++; ==> Now a = 87998
Following operations can be performed on pointers:

Addition of a number to a pointer.
Subtraction of a number from a pointer
Subtraction of one pointer from another
Comparison of two pointer variables

Quick Quiz: Try these operations on

another variable by creating pointers in a separate program. Demonstrate all four operations.

Accessing arrays using pointers
Consider this array,

If ptr points to index 0, ptr++ will point to index 1 & so on...

This way we can have an integer pointer pointing to the first element of the array like this:

int * ptr = & arr[0]; => or simply arr

ptr++;

*ptr => will have 9 as it’s value
Passing arrays to functions
Arrays can be passed to the functions like this

printArray(arr,n); =>
function call

void printarray(int *i,int n);
=> function prototype

or

void printarray(int i[] ,int n);
Multidimensional arrays
An array can be of 2 dimension / 3 dimension / n dimensions.

A 2-dimensional array can be defined as:

```
int arr [3][2] ={
{1,4}
{7,9}
{11;22}
};
```

We can access the elements of this array as arr [0] [0], arr [0] [1] & so on...

At arr [0] [0] value would be 1 and at arr [0] [1] value would be 4.

2-D arrays in Memory

A 2-d array like a 1-d array is stored in contiguous memory blocks like this:

Quick Quiz: Create a 2-d array by taking input from the user. Write a display function to print the content of this 2-d array on the screen.

Chapter 7- Practice Set

Create an array of 10 numbers. Verify using pointer arithmetic that (ptr+2) points to the third element where ptr is a pointer pointing to the first element of the array.

If S[3] is a 1-D array of integers, then *(S+3) refers to the third element:

True

False

Depends

Write a program to create an array of 10 integers and store a multiplication table of 5 in it.

Repeat problem 3 for a general input provided by the user using scanf()

Write a program containing a function that reverses the array passed to it.

Write a program containing functions

that counts the number of positive integers in an array.
Create an array of size 3x10 containing multiplication tables of the numbers 2,7 and 9, respectively.
Repeat problem 7 for a custom input given by the user.
Create a three-dimensional array and print the address of its elements in increasing order.

Chapter 8 - Strings

A string is a 1-d character array terminated by a null('\0') => {this is null character}

The null character is used to denote string termination, characters are stored in contiguous memory

locations.

Initializing Strings
Since string is an array of characters, it can be initialized as follows:

char s[]={‘H’,’A’,’R’,’R’,’Y’,’\0’}
There is another shortcut for initializing strings in c language:

char s[]=”HARRY”; => In this case C adds a null character automatically.
Strings in memory
A string is sorted just like an array in the memory as shown below

Quick Quiz: Create a string using " " and print its content using a loop.

Printing Strings

A string can be printed character by character using printf and %c.

But there is another convenient way to print strings in C.

```
char st[] = ”HARRY”;

printf(“%s”,st);    => prints the entire string
```

Taking string input from the user

We can use %s with scanf to take string input from the user:

```
char st[50];

scanf(“%s”,&st);
```

scanf automatically adds the null

character when the enter key is pressed.

Note:

The string should be short enough to fit into the array.
scanf cannot be used to input multi-word strings with spaces.
gets() and puts()
gets() is a function that can be used to receive a multi-word string.

```
char st[30];

gets(st);   => the entered string is stored in st!
```

Multiple gets() calls will be needed for multiple strings.

Likewise, puts can be used to output a

string.

puts(st); =>Prints the string and places the cursor on the next line

Declaring a string using pointers
We can declare strings using pointers

char *ptr= ”Harry”;
This tells the compilers to store the string in the memory and the assigned address is stored in a char pointer.

Note:

Once a string is defined using char st[]= ”harry”, it cannot be initialized to something else.
A string defined using pointers can be reinitialized. => ptr=”rohan”;

Standard library functions for Strings

C provides a set of standard library functions for strings manipulation.

Some of the most commonly used string functions are:

strlen() - This function is used to count the number of characters in the string excluding the null ('\0') character.

```
int length=strlen(st);
```

These functions are declared under <string.h> header file.

strcpy() - This function is used to copy the content of second string into first string passed to it.

char source[ ]= ”Harry”;

char target[30];

strcpy(target,source); => target now contains “Harry”
Target string should have enough capacity to store the source string.

strcat() - This function is used to concatenate two strings

char s1[11]= ”Hello”;

char s2[ ]= ”Harry”;

strcat(s1,s2); => s1 now contains “Hello Harry” <No space in between>
strcmp() - This function is used to compare two strings. It returns: 0 if

strings are equal

Negetive value if first strings mismatching character's ASCII value is not greater than second string's corresponding mismatching character. It returns positive values otherwise.

strcmp(“For”, “Joke”); => positive value

strcmp(“Joke”, “For”); => Negative value

Chapter 8 - Practice Set
Which of the following is used to appropriately read a multi-word string-
Gets()
Puts()
Printf()

Scanf()
Write a program to take a string as an input from the user using %c and %s. Confirm that the strings are equal.
Write your own version of strlen function from <string.h>
Write a function slice() to slice a string. It should change the original string such that it is now the sliced strings. Take m and n as the start and ending position for slice.
Write your own version of strcpy function from <string.h>
Write a program to encrypt a string by adding 1 to the ASCII value of its characters.
Write a program to decrypt the string encrypted using the encrypt function in problem 6.
Write a program to count the occurrence of a given character in a

string.
Write a program to check whether a given character is present in a string or not.

Chapter 9 - Structures

Arrays and Strings => Similar data (int, float, char)

Structures can hold => dissimilar data

Syntax for creating Structures

A C Structure can be created as follows:

```
struct employee{

int code;                    => this
```

declares a new user-defined datatype

float salary;

char name[10];

}; •

semicolon is important

We can use this user-defined data type as follows:

struct employee e1; // creating a structure variable

strcpy(e1.name,"harry");

e1.code=100;

e1.salary=71.22;

So a structure in c is a collection of variables of different types under a single name.

Quick Quiz: Write a program to store the details of 3 employees from user-defined data. Use the structure declared above.

Why use structures?
We can create the data types in the employee structure separately but when the number of properties in a structure increases, it becomes difficult for us to create data variables without structures. In a nutshell:

Structures keep the data organized.
Structures make data management easy for the programmer.
Array of Structures
Just like an array of integers, an array of floats, and an array of characters, we can create an array of structures.

struct employee facebook[100];
=>an array of structures
We can access the data using:

facebook[0].code=100;

facebook[1].code=101;
.....and so on.

Initializing structures
Structures can also be initialized as follows:

struct employee
harry={100,71.22,”Harry”};

struct employee
shubh={0}; // All the elements set to 0
Structures in memory

Structures are stored in contiguous memory locations for the structures e1 of type struct employee, memory layout looks like this:

In an array of structures, these employee instances are stored adjacent to each other.

Pointer to structures
A pointer to the structure can be created as follows:

```
struct employee *ptr;
```

```
ptr=&e1;
```

Now we can print structure elements using:

printf(“%d”,*(ptr).code);

Arrow operator
Instead of writing *(ptr).code, we can use an arrow operator to access structure properties as follows

*(ptr).code or ptr->code

Here -> is known as an arrow operator.

Passing Structure to a function
A structure can be passed to a function just like any other data type.

void show(struct employee e);
=>Function prototype

Quick Quiz: Complete this show function to display the content of the

employee.

Typedef keyword
We can use the typedef keyword to create an alias name for data types in c.

typedef is more commonly used with structures.

```
struct complex{
float real;                    // struct
complex c1,c2; for defining complex
numbers
float img;
};

typedef struct complex{
float real;
float img;                     // ComplexNo
c1,c2; for defining complex numbers
```

}ComplexNo;

Chapter 9- Practice Set

Create a two-dimensional vector using structures in C.

Write a function SumVector which returns the sum of two vectors passed to it. The vectors must be two-dimensional.

Twenty integers are to be stored in memory. What will you prefer- Array or Structure?

Write a program to illustrate the use of an arrow operator -> in C.

Write a program with a structure representing a Complex number.

Create an array of 5 complex numbers created in problem 5 and display them with the help of a display function. The values must be taken as an input from the user.

Write problem 5's structure using typedef keyword.
Create a structure representing a bank account of a customer. What fields did you use and why?
Write a structure capable of storing date. Write a function to compare those dates.
Solve problem 9 for time using typedef keyword.

Chapter 10 - File I/O

The random access memory is volatile and its content is lost once the program terminates. In order to persist the data forever, we use files.

A file data stored in a storage device.
A-C program can talk to the file by

reading content from it and writing content to it.

File pointer
The “File” is a structure that needs to be created for opening the file. A file pointer is a pointer to this structure of the file.

File pointer is needed for communication between the file and the program.

A file pointer can be created as follows:

FILE *ptr;
ptr=fopen(“filename.ext”,”mode”);
File opening modes in C

C offers the programmers to select a mode for opening a file.

Following modes are primarily used in c File I/O

Types of Files
There are two types of files:

Text files(.txt, .c)
Binary files(.jpg, .dat)
Reading a file
A file can be opened for reading as follows:

```
FILE *ptr;
ptr=fopen(“Harry.txt”,”r”);
```

int num;
Let us assume that “Harry.txt” contains an integer

We can read that integer using:

fscanf(ptr,”%d”,&num); =>
fscanf is file counterpart of scanf

This will read an integer from the file in the num variable.

Quick Quiz: Modify the program above to check whether the file exists or not before opening the file.

Closing the file
It is very important to close file after read or write. This is achieved using fclose as follows:

fclose(ptr);

This will tell the compiler that we are done working with this file and the associated resources could be freed.

Writing to a file
We can write to a file in a very similar manner as we read the file

```
FILE *fptr;
fptr=fopen(“Harry.txt”,”w”);
int num=432;
fprintf(fptr,”%d”,num);
fclose(fptr);
```

fgetc() and fputc()
fgetc and fputc are used to read and write a character from/to a file.

fgetc(ptr); => Used to

read a character from file

fputc(‘c’,ptr); => Used to write character 'c' to the file
EOF: End of File
fgetc returns EOF when all the characters from a file have read. So we can write a check like below to detect the end of file.

```
while(1){
ch=fgetc(ptr);       // When all the content of a file has been read, break the loop
if(ch==EOF){
break;
}
//code
}
```

Chapter 10 - Practice Set

Write a program to read three integers from a file.
Write a program to generate a multiplication table of a given number in text format. Make sure that the file is readable and well-formatted.
Write a program to read a text file character by character and write its content twice in a separate file.
Take name and salary of two employees as input from the user and write them to a text file in the following format:
name1, 3300

name2, 7700
Write a program to modify a file containing an integer to double its value.
If old value = 2, then new file value = 4

Project 2: Snake, Water, Gun

Snake, Water, Gun or Rock, Paper, Scissors is a game most of us have played during school time. (I sometimes play it even now :))

Write a C program capable of playing this game with you.

Your program should be able to print the result after you choose Snake/ Water or Gun.

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
```

```
int snakeWaterGun(char you, char
comp){
// returns 1 if you win, -1 if you lose
and 0 if draw
// Condition for draw
// Cases covered:
// ss
// gg
// ww
if(you == comp){
return 0;
}

// Non-draw conditions
// Cases covered:
// sg
// gs
// sw
// ws
// gw
// wg
```

```
if(you=='s' && comp=='g'){
return -1;
}
else if(you=='g' && comp=='s'){
return 1;
}

if(you=='s' && comp=='w'){
return 1;
}
else if(you=='w' && comp=='s'){
return -1;
}

if(you=='g' && comp=='w'){
return -1;
}
else if(you=='w' && comp=='g'){
return 1;
```

```
}

}
int main(){
char you, comp;
srand(time(0));
int number = rand()%100 + 1;

if(number<33){
comp = 's';
}
else if(number>33 && number<66){
comp='w';
}
else{
comp='g';
}

printf("Enter 's' for snake, 'w' for water
and 'g' for gun\n");
scanf("%c", &you);
```

```
int result = snakeWaterGun(you,
comp);
if(result ==0){
printf("Game draw!\n");
}
else if(result==1){
printf("You win!\n");
}
else{
printf("You Lose!\n");
}
printf("You chose %c and computer
chose %c. ", you, comp);
return 0;
}
```

Chapter 11 - Dynamic Memory Allocation

C is a language with some fixed rules of programming. For example:

changing the size of an array is not allowed.

Dynamic Memory Allocation:
Dynamic memory allocation is a way to allocate memory to a data structure during the runtime we can use DMA function available in C to allocate and free memory during runtime.

Function for DMA in C
Following functions are available in C to perform dynamic memory allocation:

malloc()
calloc()
free()
realloc()
malloc() function
Malloc stands for memory allocation.

It takes number of bytes to be allocated as an input and returns a pointer of type void.

Syntax:

The expression returns a NULL pointer if the memory cannot be allocated.

Quick Quiz: Write a program to create a dynamic array of 5 floats using malloc().

calloc() function
calloc stands for continuous allocation.

It initializes each memory block with a

default value of 0.

Syntax:

ptr = (float*) calloc(30*sizeof(int)) // Allocates Contiguous space in memory for 30 blocks
If the space is not sufficient, memory allocation fails and a NULL pointer is returned.

Quick Quiz: Write a program to create an array of size n using calloc() where n is an integer entered by the user.

free() function
We can use free() function to allocate the memory.

The memory allocated using calloc/ malloc is not deallocated automatically.

Syntax:

free(ptr); => Memory of ptr is released

Quick Quiz: Write a program to demonstrate the usage of free() with malloc().

realloc() function

Sometimes the dynamically allocated memory is insufficient or more than required.

realloc is used to allocate memory of new size using the previous pointer and size.

Syntax:

ptr = realloc(ptr,newSize);
ptr = realloc(ptr, 3* sizeof(int)) //ptr now points to this new block of memory, which is capable of storing 3 integers

Chapter 11 - Practice Set
Write a program to dynamically create an array of size 6 capable of storing 6 integers.
Use the array in Problem 1 to store 6 integers entered by the user.
Solve problem 1 using calloc().
Create an array dynamically capable of storing 5 integers. Now use realloc so that it can now store 10 integers.
Create an array of the multiplication table of 7 up to 10 (7x10=70). Use realloc to make it store 15

numbers(from 7x1 to 7x15).
Attempt problem 4 using calloc().

Download Chapter-wise Free Notes + Code + Practice Sheets Here:
Click here to download the notes for Chapter 0 - Introduction.
Click here to download the notes For Chapter 1 - Variables, Constants & Keywords.
Click here to download the notes For Chapter 2 - Instructions & Operators.
Click here to download Chapter 2 - Practice Set.
Click here to download the notes For Chapter 3 - Conditional Instructions.
Click here to download Chapter 3 - Practice Set.
Click here to download the notes For Chapter 4 - Loop Control Instructions.

Click here to download Chapter 4 - Practice Set.
Click here to download the notes For Project 1 - Guess The Number.
Click here to download the notes For Chapter 5 - Functions & Recursions.
Click here to download Chapter 5 - Practice Set.
Click here to download the notes For Chapter 6 - Pointers.
Click here to download Chapter 6 - Practice Set.
Click here to download the notes For Chapter 7 - Arrays.
Click here to download Chapter 7 - Practice Set.
Click here to download the notes For Chapter 8 - Strings.
Click here to download Chapter 8 - Practice Set.
Click here to download the notes For

Chapter 9 - Structures.
Click here to download Chapter 9 - Practice Set.
Click here to download the notes For Chapter 10 - File I/O
Click here to download the notes For Chapter 10 - Practice Set.
Click here to download the notes For Project 2 - Snake, Water, Gun.
Click here to download the notes For Chapter 11 - Dynamic Memory Allocation.
Click here to download the notes For Chapter 11 - Practice Set