BASIC OF C PROGRAMMING

RAUSHAN GUPTA

CONTENTS

- **Programming Concepts**
- **Some interview Questions**

❖ ABOUT C PROGRAMMING

- COMPUTER LANGUAGE :- Computer language is a language through which we communicate with computer to give set of instruction to the computer and get desire output.
- PROGRAMMING LANGUAGE(C LANGUAGE):- Programming language is a set of instruction in algorithm formate or step by step instruction given to the computer to do a certain job.
- SOFTWARE:- Software is a combination of set of program which help to do our work easier. It is standalone software like Turbo C++, Vlc player etc.
- HOW POPULAR 'C' LANGUAGE:-
  - ✓ 'C' language is a very good language to introduce itself to the programming world as it is. A simple processable programmable language which is capable to doing wonders.
  - ✓ Program written in C language take very less time to execute program. Initially C language was mainly used for coding system lavel program like during operating system.
  - ✓ But there are others application as well as which can be very well design and develop using C like : Text editor, Compiler, Networkdriven etc.
- WHY WE LEARN 'C' PROGRAMMING:-
  - ✓ C is the best language for many programming. So learning as main language will play an important role while studying other programming language.

    It shares this same concept data type, operators, control statement.
  - ✓ It is simple language and provide faster execution.

❖ HISTORY OF C

Here we can see that how C programming developed:-

- **ALGOL-60 :-** The ALGOL means Algorithmic language. It is develop by international group at Cambridge Institute in 1960. It is develop through the photon that is used on that time in the form of alternative.

- **CPL :-** The full form of CPL is Combine Programming Language. It is also developed at Cambridge Institute in year 1963.
- **BCPL :-** The full form of BCPL is Basic Combine Programming Language. It is developed on the basis of CPL by Martin Richards at Cambridge Institute in year 1967. It is typeless language. This language is mostly used for developing Operating System and Compiler.
- **B Language :-** The full form of B language is Basic Language. It is derived from BCPL language. It is developed by Ken Thompson at Bell Labourty in year 1969-70. It is also typeless language. This language is used for UNIX System development.
- **C Language :-** It is a combination of all above language which is developed by Dennis Ritchie at Bell Labourty in year 1972. Before this language there is no concept of data types, variables, etc in that language. But in C language all features of above language is added with new features are added like data types, variables, etc. This language is used for making UNIX System more advance.

  In 1989, through ANSI (American National Standard Institute) standard is published for C. Due to standard published, the name was given 'ANSI-C or C89'.

  After that in 1990, same standard in approved by ISO and name is given 'C90'.

  After that in 1995, the extension is released for inter-nationalization by ISO for that version and that version name is given 'C95'.

  After that in 1999, it is revised by ISO and the name is given 'C99'.

  After that in December 2011, the standard is approved for it and name is given 'C11'.

❖ OVERVIEW OF C

- INTEGRATED DEVELOPMENT ENVIRONMENT(IDE):- IDE software is used to setup the environment for developing program. It is a duplicate environment. Original environment is OS (Operating

System). IDE software comes after OS. There are so many IDE software is available in the markets like Turbo C++, Dev++,..etc.

For Blue Screen (Turbo C++):- Shortcut key for using Turbo C++ is:

1. Save :F2 or Fn+F2
2. Compile :Alt+F9 or Alt+c
3. Run :Ctrl+F9 or Alt+r
4. Open :F3 or Fn+F3
5. Cut :Shift+Delete
6. Copy :Ctrl+Insert
7. Paste :Shift+Insert
8. Clear :Ctrl+Delete
9. Undo :Alt+Backspace
10. Redo :Shift+Alt+Backspace

For Black Screen (Console window):- Shortcut key for using Turbo C++ is:

1. See output :Alt+F5

- LIBRARY:- It is a collection of pre-define function.

like :

stdio.h	=>printf();, scanf();
conio.h	=>clrscr();, getch();
math.h	=>sqrt();, pow();, sin();
dos.h	=>getdata();, gettime();

- HIGH LEVEL LANGUAGE:- High level language is a language which is used to day-to-day our life i.e. English. It is also called Source code.
- LOW LEVEL LANGUAGE:- Low level language is a language that is used by our computer system i.e. Binary code.

❖ FEATURES OF C

Features of C are as follows:-

- ✓ C is a simple language in this sense that it provides structure approach within the bracket (To break the problem into parts) which setup library function datatypes etc.

- ✓ C programming is a case sensitive language that means in C program upper letter is totally different from lower letter. Compiler understand capital letter is different and small letter is different.
- ✓ Machine independent, portable unlike assembly language C program can executed on different machine withsome specific changes or no modification.
- ✓ In middle level programming language, C is intended to do low level programming language.

  It also supports the features of high level language that is why it is known as middle level language.
- ✓ Memory Management:- It supports the features of dynamic memory allocation in C. We can threed the allocated memory anytime by free function.
- ✓ Pointer:-
  - C provides the features of pointer. We can directly interent with the memory by using the pointer.
  - Program written in C are efficient and fast.
  - C compiler compile the capability of an assembly with feature of high level language.

- WHY WE NEEDS TO USE C PROGRAMMING:-
  - ✓ It gives the basic concept of all programming language.
  - ✓ It is used to create Standalon software i.e. system software which run without internet connection.
  - ✓ It's execuation is fast compare to other programming language.
  - ✓ It caught market till now means it's demand in the market is so high.
  - ✓ Standalon software is highly demand in the market.
  - ✓ By using C language we can only develop Standalon application. We cannot develop web application.
- TYPES OF APPLICATION WHICH WE MADE FROM PROGRAMMING:-
  1. STANDALON APPLICATION OR SYSTEM APPLICATION OR DESKTOP APPLICATION
  2. WEB APPLICATION OR SERVER APPLICATION

- **STANDALON APPLICATION:-** It is also known as System Application and Desktop Application. Standalon application is a those type of application which can run without the help of internet connection. These types of application is widely used is the market, so that is most demandable. That's why C programming language covers market till now.
  e.g. Turbo C++, Media player, MS Word,..etc.

- **WEB APPLICATION:-** It is also known as server application. Web application is a those type of application which can run wiyh the help of internet connection. e.g. Facebook, Gmail, Instagram,..etc.

❖ HOW TO INSTALL C

Following are the steps to install Blue Screen Software (Turbo C++):-
Step1: In the browser type 'Turbo C++ downloads'
Step2: Go to the first website i.e. Turbo C++ or C for Windows7,8,8.1 and 10,32/64-bit full screen.
Step3: Click on Download TurboC++ 3.2 from here.
Step4: Click on Download, and downloading starts.
Step5: After downloading go to your My Computer, go to the downloads and double click on Turbo C++3.2.
Step6: Double click on Turbo C++.
Step7: Click on Run.
Step8: Click on Next.
Step9: Click on 'I accept the terms in the licence agreement.'
Step10: Click on Next.
Step11: Click on Install.
Step12: Click on Yes.
Step13: Check 'Launch the program is checked' if it is not checked clicl and checked it.
Step14: Click on Finish.
Step15: Close all open windows.
Step16: Double click on Turbo C++ icon.
Step17: Checked Full screen mode and then
Step18: Click on Start Turbo C++.

❖ PROGRAMMING STYLE

Programming style means the way of writing code in IDE software. Generally beginners write the code one below one which does not looks goods and we have to face some difficulty to understand and analyzing the programs. It also help to not to leave any opening and closing braces. We know that if we write small program, we can easily find errors but in large program in which programmer write program to develop any software ,in that program the programmer face difficulty to find errors. So programming style help him/her to easily understand and analyzing the program.

e.g.

Suppose we write program like this, we have to face difficulty to find errors in large program :

```
#include<stdio.h>
int main()
{
int a=45;
printf(“ The value of a is : %d”,a);
return 0;
}
```

So that I will show you how to write code:-

```
#include<stdio.h>
int main()
{
    int a;
    printf(“Enter a number.”);
    scanf(“%d”,&a);
    if(a>100)
    {
        printf(“The number is greater than 100.”);
    }
    else
    {
        printf(“The number is smaller than 100.”);
```

```
    }
return 0;
}
```

So that as you see this program give clear analysis about the program that each and every message is inside the main block and also some message is inside the if statement block which is printed when condition becomes true or when condition becomes false else block have also some message inside it which is get printed. Here we give tab spaces which give four spaces in the next line after opening braces of main function and also four more spaces in if statement and else statement. Mainly most of the programmer use tab spaces to give spaces. But it is not necessary if you want to give less or more spaces you can free to give. Some programmers use two spaces. After that we also understand clearly that every opening and closing braces are use properly or not. So that this is the benefits of writing code in programming style. And I know, to become a good programmer programming style is very necessary to learn and use in the code.

- TYPES OF PROGRAMMING STYLE:-

It is two types :-

1. End-of- line style
2. Next-line style

1. **End-of-line style :-** End-of-line style means when we start any function and statement, the opening braces is write in the same line and the block message is write in the next line, that style is known as End-of-line style.

e.g. Something like this…

```
#include<stdio.h>
int main()  {
        int a;
        printf("Enter a number.");
        scanf("%d",&a);
        if(a>100)  {
                printf("The number is greater than 100.");
        }
        else{
```

```
            printf("The number is smaller than 100.");
        }
    return 0;
    }
```

2. **Next-line style :-** Next-line style means when we start any function and statement, the opening braces is write in the next line and the block message is write in that next line, that style is known as Next-line style.
   e.g. Something like this…

```
#include<stdio.h>
int main()
{
    int a;
    printf("Enter a number.");
    scanf("%d",&a);
    if(a>100)
    {
        printf("The number is greater than 100.");
    }
    else
    {
        printf("The number is smaller than 100.");
    }
return 0;
}
```

❖ FIRST C PROGRAM

A program have always three thing without these we can't say written data is a program. These three things are:-

i. Identifiers
ii. Variable and
iii. Function

Now we are going to write our first program:

```
#include<stdio.h>

void main()

{

// This is single line comment.

/* This is multi-line comment.*/

printf(“ Good Morning”);

getch();

}
```

- Pre-processor (#include):- #include is the first C program. It also known as pre-processor ,the task of pre-processor is to initialize the environment of program to link the program that is with header files with the help of pre-processor. So when we type #include<stdio.h>, it inform to the compiler to include the stdio.h header files before executing the program.
- void:- When used as function return type the void keywords specify that the function does not return value.

  When we used a function parameters list, void specifies that the function takes no parameters

- main():- Main() function is a function that must be every C program contains. If a program contain one function i.e. must be main function. Everything inside the main function C program will be executed. Execution of every C program is starts from main function. If main function is not present in any program ,the program is not executed.
- Curly Bracesses {}:- The curly bracesses used just after the main function for open and closed the body of the main function or any function.
- Return Statement:- The return 0; statement return execution states to the operating system.

  If program return 0 to the compiler it means the program run successfully and if program return 1 to the compiler it means the program error and run unsuccessfully.

Syntax:

```
int main()
{
printf(“ message”,argument list);
return 0;
}
```

- printf() and scanf():- In C the printf() function are used for print the message on the console window and scan() function is used for take input from used in the program. In C language both function are build in library and pre-define in stdio.h header files.
  - **printf(“”) function:-** The printf() function is used for show output or message on the console window.

    Syntax of printf() function
    printf(“message and formate specifiers”,argument list);

    Formate specifiers can be :

integers(int)	:%d or %i
float	:%f
character(char)	:%c
string	: %s

    etc..
  - **scanf(“”) function:-** The scanf() function is used for take input from user in the program.

    Syntax of scanf() function
    scanf(“formate specifiers”, &argument list);
- getch():- getch() is used for hold the console window to show output to the user. It not must be necessary to used it in the program but we will used it to see output during run the program. It is mostly used in void returntype. We can also see output without using it in the program by pressing Alt+f5.

❖ HOW TO COMPILE AND RUN C PROGRAM

- HOW TO COMPILE AND RUN C PROGRAM:-

There are two ways to compile and run the C program by *Menu* and shortcut method.

1. **By Menu :-** Now click on menu then compile sub menu to compile the C program.

   Then again click on run menu then run the program
2. **By Shortcut :-** Press the Alt+F9 or Alt+c to compile the C program and press Ctrl+F9 or Alt+r to run the program.

- You can view the console screen to see output any time by pressing Alt+F5 keys.
- Compile using Text editor :- To download and use gcc compiler to run C coding. You can run gcc compiler to setup environment of gcc compiler and write the coding in notepad and compile the C coding in command prompt by typing ‘gcc program_name.c -o program_name’ and for run the coding type ‘program_name.exe’.

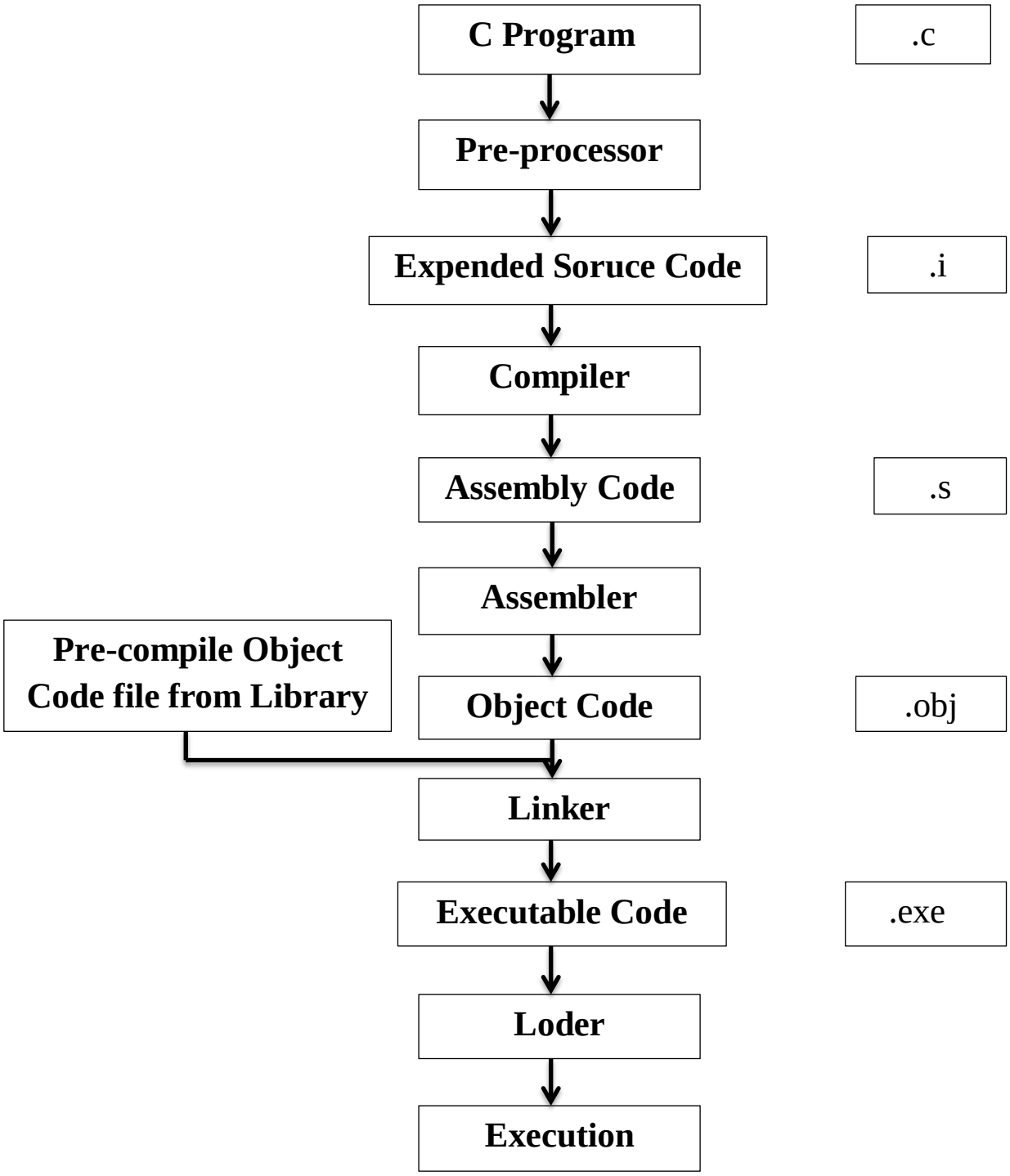
- COMPILATION PROCESS IN C:-

What is compilation?

=>The compilation is a process of converting source code into object code. It is done with the help of program. The compiler check the source code, check the error for syntstical or structural errors and if the source is error free then it generates the object code. The C compilataion process convert the source code taken as input source code into object code.

The compilation process can be divided into four steps i.e. Pre-processor, Compiler, Assembler, Linker.

Flow Diagram of Compilation Process



- **C program :-** First we write C program in Blue Screen software(Turbo C++) and save it with extension (.c).
- **Pre-processor :-** Then C program recived by Pre-processor and removed unwanted things like all commented line from program and converted into expended code.
- **Expended Source Code :-** Expended is a pure program which system can able to read all line or command given by the user. It's extension is (.i).

- **Compiler :-** And then Compiler expended code is recived by the compiler and compiler compile the program if there is error it show error or if there is no error , it converted into Assembly Code.
- **Assembly Code :-** Assembly code has extension (.s).
- **Assembler :-** Assembler recived the Assembly code and converted into Object Code.
- **Object Code :-** Object code is a code generated by the assembler. It extension is (.obj).
- **Linker :-** Linker link or combine the program object code with pre-compile object which is present is the library to execute the program and converted into Executable Code.
- **Executable Code :-** Executable code is the final and understable code which is able to understand by the system. It extension is (.exe).
- **Loder :-** Loder recived the executable code and lode it for execution.
- **Execution :-** Then finally the program going to be execute and gives desire output.

❖ 'C' SYNTAX

✓ C level language syntax specified rules for secquence of character to be written in C language.

✓ In simple language it state how to form statement in c language like how should the line of code starts, how it should end, where to use double cote (" ") and cerly braces, etc.

- SYNTAX RULES FOR 'C' PROGRAMMING :-
  - ✓ C is a case sensitive language. So all C instruction must be written in lower case letter.
  - ✓ All statement must be end with semi-colom.
  - ✓ White space is used in C to describe blank and tab.
  - ✓ White space is used in between keywords and identifiers.

❖ TOCKENS IN C

- ✓ A smallest individual unit in C program is known as tockens.
- ✓ It is most important elements to be used in creating a program in C.
- ✓ We can say that tockens in C is the building block or the basic components for creating a program in c language.

- CLASSIFICATION OF TOCKENS IN C :-
  1) Keywords
  2) Identifiers
  3) String
  4) Operators
  5) Constant
  6) Special Character/Symbols

❖ KEYWORDS AND IDENTIFIERS

- KEYWORDS IN C:-
  - ✓ Keywords are pre-reserve words that have special meaningin C language.
  - ✓ The meaning of C keywords has already be describe to the C compiler.
  - ✓ These meaning can't be change , thus keyword cannot be used as variable names because that would try to change the existing meaning of the keywords, which is not allowed.
  - ✓ There are 32 keywords available in C programming :-

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

Note: During writing program keywords become white colour. For understable keywords look like white colour.

- IDENTIFIERS IN C :-
  - ✓ In C identifiers are the names given to the variable, constant, function, and user define data.
  - ✓ These identifiers are define against a setup the rules.
  - ✓ **Rules of identifiers in C**
    - The first character of an identifiers should be either an alphabet or underscore and then it can be followed by any in identifiers

both upper case and lower case lettersare distinct. So we can say that identifiers are case sensitive.

- The length of the identifiers should not be more than 31 characters.
- Comma and blank space cannot be specified within an identifiers.
- Keywords are not allowed to be used as identifiers.
- When we declare a variable or any function in C language program to used it . We must provide a name to it which identified it throughout the program.

  For e.g.

  int a= 10;

  Here, a is name or identifiers which store the value 10.

Differenciate Between Keywords and Identifiers.

Keywords	Identifiers
i). Keywords is pre-define word. ii). It must be written in lower case. iii). It meaning pre-defining in C compiler. iv). It is combination of alphabetical character. v). Keywords does not contain	i). It is user define. ii). It can be written in both lower, upper case. iii). It meaning is not define in C compilers. iv). It is combination of alpha-numeric character. v). Identifiers can contain underscore characters.

❖ VARIABLE IN C

Naming of any address in C is called variable. Variable of name is memory location unlike constant variable is changeable. We can change value of variable during execution of program.

It is way to present memory location through symbol, so that it can easly identify.

Syntax: datatype along with identifiers.

Datatype variable_name;

For e.g. - int num;

We can also provide value while declaring the variable as given below:-

int num=20;

int num1=23;

Rules for defining variable:-

i. Variable name must not start with digits.
ii. Variable name can be start with alphabet or underscore(_).
iii. After first character it may be combination of alphabets and digits.
iv. Blank space not allowed.
v. Variable name should not start with keywords.
vi. Upper case and lower case treat as different . As C is case sensitive language.
For e.g. Here both are different
int a=10;
int A=20;

- DECLARING, DEFINING, AND INITIALIZATING A VARIABLE:-
  - **Declaration :-** Declaration of variable must be done before using in program.
    - ✓ First tells the compiler that what the variable name is.
    - ✓ It specified what types of data is hold until th variable is define the compiler does not have worry about allocating memory spaces.
  - **Defining :-** Defining a variable means the compiler has to how assign a storage to the variable because it will be used in the program.
    We can even define multiple variable of same datatype by single line by using comma.
    Note:- In C language definition and declaration for a variable takes place at the same time i.e. there is not different between declaration and definition.
  - **Initializing :-** Initializing a variable means to provide it with value. Variable can be define and initialize can be single line.

For e.g. int num=20;

- TYPES OF VARIABLE:-

  1) Local Variable
  2) Global Variable
  3) Static Variable
  4) Automatic Variable
  5) External Variable

- **Local Variable :-** The variable that is declared inside a function or block is called local variable. It must be declared at the start of the block.

  e.g.

```
void main()
{
int num=10;
printf("%d",num);
}
```

  We must have to initialize before use.

- **Global Variable :-** The variable that is declared outside a function or block is called global variable. It must be declared before the main function. Global variable can use whole of the program. It can call by any function to execute the function correctly.

  e.g.

```
int num=10;
void main()
{
printf("%d",num);
}
```

- **Static Variable :-** A variable declaring with static keyword. It gives updated value everytimes. It will occurs multiple function call.

  e.g.

```
#include<stdio.h>
#include<conio.h>
void display()
{
int a=2;
static int b=5;
```

```
a=a+1;
b=b+1;
printf("Value of a= %d\n",a);
printf("Value of b= %d\n",b);
}
int main()
{
clrscr();
display();
display();
display();
return 0;
}
```

Output:

```
Value of a= 2
Value of b= 5
Value of a= 2
Value of b= 6
Value of a= 2
Value of b= 7
```

If we call display function many times the local variable print value same for each function but static variable print the incremented value every times. (see in above example)

- **Automatic Variable :-** All the variable in C that are declare inside the block are called automatic variable. We can execute by declaring using auto keyword. It is by default available in local variable which do not declare with any keywords.

  e.g.

```
#include<stdio.h>
int main()
{
auto int a=4;
printf("Value of a=%d\n",a);
return 0;
}
```

Output:

Value of a=5

- **External Variable :-** To declare a external variable , to use extern keyword. We can share a variable in multiple C source files by using an external variable.

  Syntax:

  extern int num;

  Here an int type variable ‘a’ has been declared. But it has not been define in the memory means it not allowed to allocate any address in the memory or it not allowed to reserve space in the memory for that variable name. So it does not allow to initialize any value in that variable. If you want to define and initialize value in that variable, you have to again declare that same variable in the next line with datatypes.

  e.g.

```
#include<stdio.h>
extern int a;
int a=10;    //like this…
int main()
{
printf(“Value of a=%d”,a);
return 0;
}
```

  Output:

  Value of a=10

❖ OPERATORS IN C

- ✓ An operators is a symbol that tells the compiler to perform a certain mathematical an logical manipulation.
- ✓ Operators are used in program to manipulates data and variables.
- ✓ C operators can be classify into following types :-
  1. Arithmatic operators
  2. Relational operators
  3. Logical operators
  4. Bitwise operators
  5. Assignment operators
  6. Conditional operators and

7. Special operators
8. Unary operators and Ternary operators

- PRECEDENCE OF OPERATORS IN C :-

The precedence of operators specifies that which operators will be evaluated first and next the associativitys. Specifies the operators direction to be evaluates. It may be left to right or right to left.

- CLASSIFICATION OF PRECEDENCE OF OPERATORSS:-

Category	**Operators**	**Associativity**
Postifix	(),[],>=,++,--	Left to Right
Uniry	+,-,!,~,++,--,sizeof	Right to Left
Multiplicative	*,/,%	Left to Right
Shift	<<,>>	Left to Right
Relational	<<=,>>=	Left to Right
Equality	==,!=	Left to Right
Bitwise AND	&	Left to Right
Bitwise OR	|	Left to Right
Logical AND	&&	Left to Right
Logical OR	||	Left to Right
Conditional	?,:	Right to Left
Assignment	=,+=,-=,*=,/=,%=, >>=,<<=	Right to Left
Comma	,	Left

1. **Arithmatic operators:-** C supports all the basic arithmetic operators. The following tables show all the basic arithmetic operators.

Operators	**Description**
+	Add two operands
-	Subtract two operands
*	Multiply two operands
/	Divide numerators by denominator
%(modulus)	Remainder of division
++	Increment operators(It increases integers value by one).
--	Decrement operators(It decreases integers value by one).

2. **Relational Operators :-** The following tables shows relational operators.

Operators	Description
==	Checks two operands are equal
!=	Checks two operands are not equal
>	Checks if operands on the left is greater than operands on right.
<	Checks if operands on left side is smaller than right side.
>=	Checks left operands is greater than or equal to right operands.
<=	Checks left operands is less than or equal to right operands.

3. **Logical Operators :-** C supports three logical operators i.e. Logical AND, Logical OR, Logical NOT.

Operators	Description	Example
&&	Logical AND	(a&&b)
||	Logical OR	(a||b)
!	Logical NOT	(!a)

Logical AND :

A	B	=C
0	0	0
0	1	0
1	0	0
1	1	1

Logical OR :

A	B	=C
0	0	0
0	1	1
1	0	1
1	1	1

Logical NOT :

A	B
1	0

If 'A' is true then give false and vice-versa.

4. **Bitwise Operators :-** Bitwise operators perform manipulation of data at integers level . These operators also perform shifting of bits from right to left.

Bitwise operators are not apply to float or double data types.

Operators	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	Left shift
>>	Right shift

Left => Accending <<

Right => Decending >>

a^b

0	0	=	0
0	1	=	1
1	0	=	1
1	1	=	0

Bitwise shifting operators shift the bit value to the left operands specifies the value to be shifted and right operands specifies the number of position that the bits in the value have to be same precedence.

Example : a=0001000

b=2

a<<b=0100000

a>>b=0000010

5. **Assignment Operators :-** An equal to(=) is known as assignment operators.

e.g. a=10;

Operators	Description	Example
=	Assign value from right side operands to left side operands.	a=b;
+=	Add right operands to left operands and assign the result	a+=b; same as

	left	a=a+b;
-=	Subtract right side operands from left side operands and assign the result left operands.	a-=b; same as a=a-b;
*=	Multiply right operands to left operands and assign the result left operands.	a*=b; same as a=a*b;
/=	Divide right operands to left operands and assign the result left operands.	a/=b; same as a=a/b;
%=	Calculate remainder of two operands.	a%=b; same as a=a%b;

6. **Conditional Operators :-** It also known as ternary operators (?:). This syntax of ternary operators is :

   expression 1 ? expression 2 : expression 3
   condition ? true:false

   Explanation :- The first expression generally return true or false based on which it is decided either expression 2 will be executed or expression 3will be executed.

   If expression 1 return true then the expression 2 is executed and if expression 1 return false then the expression 3 is executed.

7. **Special Operators :-**

Operators	**Description**	**Example**
sizeof()	return sizeof variable and datatype also	sizeof(int)
&	return the address of variable or allocate address of variable	#
*p	pointer variable	*num;

8. Unary Operators and Ternary Operators:-

Difference between Unary operators and Ternary operators:-

Unary operators	**Ternary operators**
a++; (post-increment)	a=a+b;
a--; (post-decrement)	
++a; (pre-increment)	
--a; (pre-decrement)	

=>In case of pre-increment or decrement first will value evaluated then after assigning the value.

e.g.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a=10,b;
b=++a;
printf("%d%d\n",a,b);
return 0;
}
```

Output:

11
11

=>In case of post- increment or decrement, first value will be assigning and then evaluated.

e.g.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a=10,b;
b=a++;
printf("%d%d\n",a,b);
}
```

Output:

10
11

Example:

```
int a=10,y;
y=a++ + --a - a-- + --a;
  //10+10-10+8=18
```

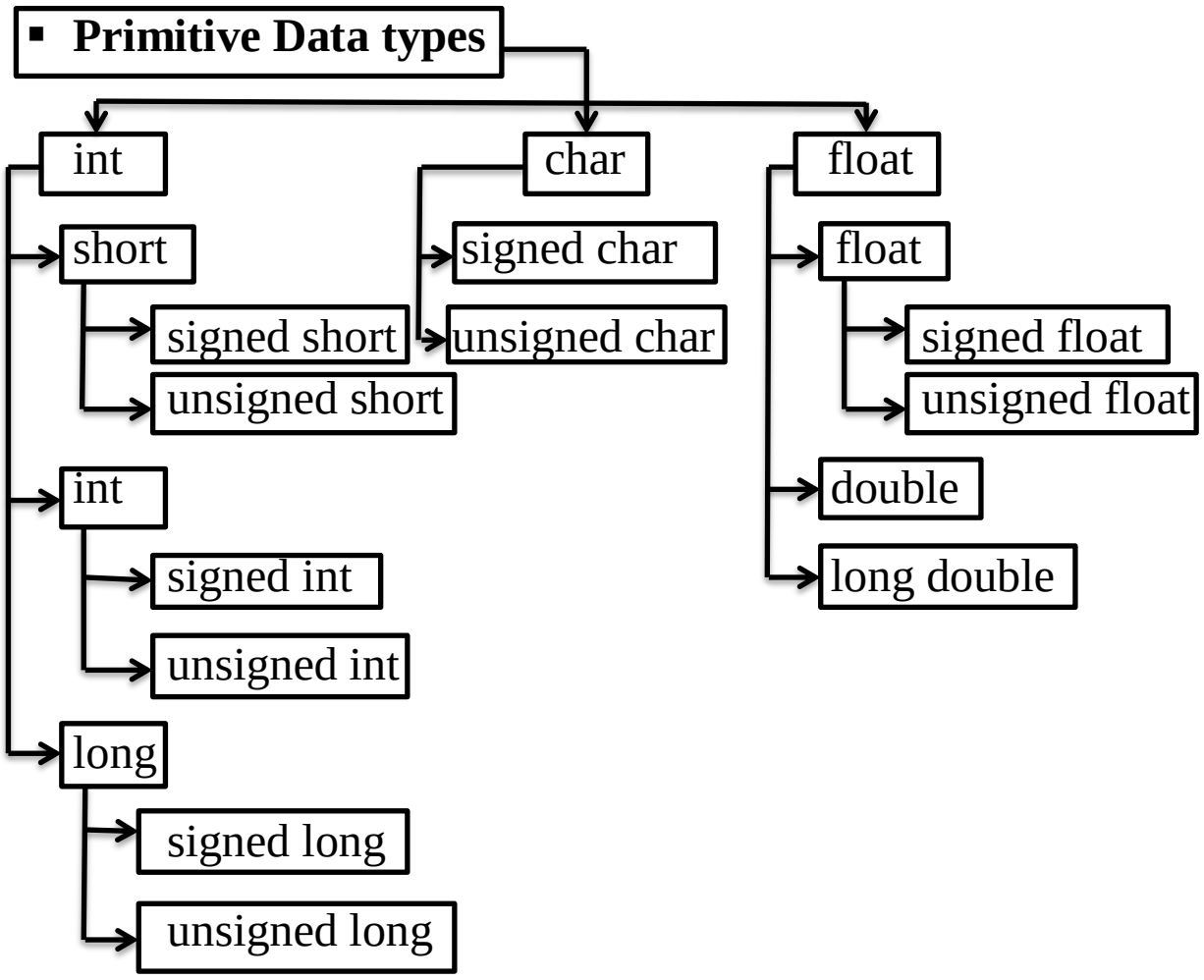
❖ DATA TYPES IN C

Data types specifies how to enter data into a program and whatever data we entered. C language has some pre-define setup data types to handle various kinds of data that we can use in our program.

- **Classification:**

Primitive	**Derived**	**User-define**
int	array	structure
char	string	union
long	pointer	typedef
void		enum

- **Primitive Data types**



a) **Int :-** It has name suggest an int variable is used to store whole number (0-9) or we can say integers numbers. It allocate space inside the memory is 2 bytes in 32 bits operating system and 4 bytes in 64 bits operating system.

b) **Char :-** It is used to store character types data inside it. It store single character and it allocate space inside the memory is 1 byte in almost all compiler.

c) **Float :-** It is used to store decimal numbers or real numbers. It allocate space inside the memory is 4 bytes.

Datatypes	**Memory size**	**Range**
Char	1 byte	-128 to 127
Signed char	1 byte	-128 to 127
Unsigned char	1 byte	256
Short	2 bytes	-32768 to 32767
Signed short	2 bytes	-32768 to 32767
Unsigned short	2 bytes	0 to 65535
Int	2 bytes	-32568 to 32567
Signed int	2 bytes	-32568 to 32567
Unsigned int	2 bytes	0 to 65535
Int	4 bytes	-2147483648 to 2147483647
Signed int	4 bytes	-2147483648 to 2147483647
Unsigned int	4 bytes	0 to 4294967295
Long	4 bytes	-2147483648 to 2147483647
Signed long	4 bytes	-2147483648 to 2147483647
Unsigned long	4 bytes	0 to 4294967295
Float	4 bytes	-2147483648 to 2147483647

Signed float	4 bytes	-2147483648 to 2147483647
Unsigned float	4 bytes	0 to 4294967295
Double	8 bytes	
Long double	10 bytes	

❖ CONSTANT IN C

- **CONSTANT :-** A constant is a variable which value never change.
  e.g. Hexadecimal, character,float,int.
- **TWO WAY TO DEFINE:-**
  There are two way to define constant :-
  i. Using const keyword
  ii. #define variable_name initialization

i. Using const keyword :- The const keyword is used to define the variable as a constant in the C program.

Syntax:

```
const char= 'j';
const in tab= 40;
const string= "C program";
```

e.g.

```
main()
{
const int num= 40;
printf("%d",num);
}
```

e.g.

```
main()
{
cons tint a=40;
a=a+1;
printf("%d",a);
}
```

Here this program gives error at compile time because we are trying to change value of the variable.

ii. #define preprocessor(macro):- #define is used to define constant or macro substitution. It support all types of data.

Syntax:-

```
#define a 5
```

```
main()
{
printf("%d",a);
}
```

e.g.

```
#include<stdio.h>
#define a 5
main()
{
printf("%d",a);
}
```

e.g.

```
#include<stdio.h>
#define a 5
main()
{
a=a+1;
printf("%d",a);
}
```

Here this program gives error at compile time because we are trying to change value of the variable.

❖ C INPUT/OUTPUT

Input means to provide the program some data to be used in program. Output means to display data on display screen or write a data to pointer or file.

C language provides many build in function to read and given input and display data on console screen where there is need to output the result.

- **Scanf() and printf():-** The standard input-output header file. The standard input-output header file name <stdio.h> contain.

  The definition of the function that is printf and scanf which are used to display output on screen and to take input from user respectively.

  e.g.

```
#include<stdio.h>
int main()
{
int num;
printf("Please enter a number\n");// display message on the screen
                                  and assigning the user to input a
```

value.

```
scanf("%d",&num);            // reading the entered value by the user.
printf("You entered : %d",num);  // display the number as output.
return 0;
}
```

In C we can also restric limit the number of digit or character that can be input or output by adding a number with the formate string specifiers.

e.g.

"%1d" // the first one is a single number digit.

"%3s" // It gives three character. Hence if you try to input that is 100. While scanf() has "%d", It will be take only for as input.

Note: printf("print the value");
Scanf("formate specifiers",variable_name); // it reads the data entered by the user and assign the data in the program.

- **getchar() function and putchar() function :-**

<u>getchar() function</u> :- getchar() function to get a character from the terminal and return. It is as an integers. These function reads only single character data type.

<u>putchar() function</u> :- The putcahr() function display the char passed to it on the screen same char. This function display only a single char data type.

If you want to display more than one char then use putchar() function used in loop.

- **gets() and puts() function :-**

<u>gets() function</u> :- gets() function is used to take input from user in the program. It is different from scanf() i.e. scanf() does not take string data as an input more than a word in a variable without looping. But gets() function have capacity to take and hold string data into a variable more than a word and a sentence into it without using loop.

It also we don't need to write formate specifiers into it. It always apply by default in all the program by the compiler. It is easy to use. It also contain space as a input and take data after space in input. But scanf () does not contain space as a input. scanf() does take data after space in input and we must have to write formate specifiers inside scanf() without formate specifiers it works.

Syntax:

```
main()
{
printf("   ");
gets();
}
```

e.g.

```
main()
{
char a;
printf("Enter a string more than a word.\n");
gets(a);
printf("%s",a);
return 0;
}
```

Output:

```
Enter a string more than a word.
Hai!! I am writing a program.
```

puts() function :- printf() and puts() function is almost same. There is a little bits differents from each other which does not make a major problem and the different is that when we use printf() in the program, we must use line break statement(\n) to go to the next line. But in puts() function, there is nothing like this, when we use puts() in the program, it go to the next line after print the statement automatically, we don't have to use line break statement (\n).

Syntax:

```
main()
{
```

```
puts(“  ”);
}
```

e.g.

```
main()
{
puts(“Hello World”);
puts(“I am writing program.”);
}
```

Output:

Hello World
I am writing program.

❖ C FORMATE SPECIFIERS

The formate specifiers is a string used in the formulated input and output. The formate string determine the formate of the input and output. The formate string always start with a %s character.

Formate specifiers	**Description**
%i or %d	It is used to print the signed integers value where signed integers means the variable that can hold both +ve and -ve value.
%u	It is used to print the unsigned integers value where the unsigned integers means the variable that can hold only +ve value
%o	It is used to print the octal unsigned integers where octal integers value always starts with zero(0) value.
%x(small x)	It is used to print the hexadecimal unsigned integers where hexadecimal integers value always starts with ox value. In this alphabet char are printed in small letter such as a,b,c,d,e.
%X(capital X)	used to print hexadecimal

%f	It is used to print decinal floating point values by default. It print six value after decimal.
%e or %E	It is used for scientific notation. It is also known as Mentessa or Exponent.
%g	It is used to print decimal floating point values and it uses fixed precision that means 102.45
%p	It is used to print the address in Hexa-decimal form.
%c	It is used to print unsigned character.
%s	It is used to print the string.
%ld	It is used to print long signed integers Value.

❖ ASCII VALUE IN C

ASCII value in C means as we know our computer understand binary number (i.e. 0 and1) and we also know only integers value is converted into binary number because binary conversion is a way to convert integers value into binary number and binary numbers into integers but this conversion is impossible for human beings to convert large data into binary numbers. So that high level language is used to give input and take output from computers.

So that according to the conversion only integers value is converted into binary numbers. So ASCII value means as we know we used character value to write program and use hign level language to run the computers. So ASCII value is a integers form of every character or symbols present in our system. It helps to write anything in computer to get desire output from computers in our language that human beings understand. So finally we can say anything we write it converted into integers form and then again converted into binary numbers or for output same process but it's vice-versa.

Here we are doing some example to understand this carefully know that how every character or symbols have ASCII value.

Here we are finding some ASCII value of any character or symbols.

Syntax:

```
int main()
{
char num;
printf("Enter any character or symbols to get ASCII value of this.\n");
scanf("%c",num);
printf("ASCII value of %c is : %d",num,num);
return 0;
}
```

e.g.

```
#include<stdio.h>
#iunclude<conio.h>
int main()
{
char ch= 'a';
printf("ASCII value of %c is : %d",ch,ch);
return 0;
}
```

Output:

ASCII value of a is : 97

Here one more example to find character or symbols from ASCII value.

e.g.

```
#include<stdio.h>
#iunclude<conio.h>
int main()
{
char ch= 65;
printf("Character or symbols of %d is : %c",ch,ch);
return 0;
}
```

Output:

Character or symbols of 65 is :A

❖ LITERALS IN C

Literals are constant value assign to the constant variable.

We can say that literals represents the fixed values that can not be changed or modify. It also contain memory but does not have reference in it's memory.

e.g.

const int num=20;

const int a=20;

Here 20 is a integer literals

- TYPES OF LITERALS :-

There are four types of literals i.e.

i. Integers literals
ii. Float literals
iii. Character literals
iv. String literals

1. **Integers literals :-** It is numeric literals that represent integers types value. It represent the value heigher fraction or it can be specifies following ways..
   i. Decimal number base 10 :- It is defined representing the digits by digits between 0 to 9.
      e.g. 547829
   ii. Octal number :- It can be defined by digits between 0 to 7.
      e.g. 01, 02, 03, 04, 05…
   iii. Hexa-decimal number :- It is defined as a number in which ox or oX is followed by the Hexa-decimal digits. i.e. digits from 0 to 9, alphabet character from small a to z, capital from A to Z.
2. **Float literals :-** It is such types of literals that which contain only floating point value and real number. These real number contain the number of parts integers, real part, exponential part and fractional part.
3. **Character literals :-** A character literals contain a string character enclose within singal quotation. It multiple character are assign to the variable then we need to a character array.

Note :- If we try to store more than one character in variable then the warning of a multiple character constant will be generated.

e.g.

```
#include<stdio.h>
int main()
{
char ch= ‘jk’;
printf(“%c”,ch);
return 0;
}
```

In this program in the above code we have used two character ‘jk’. i.e. ‘jk’ within single quote. So, this statement generate a warning represents a character literals. It can be represented by ‘j’,’k’.

We can also used ASCII value in integers to represent a character literals.

e.g. The character value of ASCII value of 65 is: A

4. **String literals :-** String literals represent multiple character enclosed within double quotes. It contains additional character(“\n”). i.e. null character, which gets automatically inserted this null character specifies the termission of the string.

We can use the (+) to connected two string.

❖ STATIC IN C

- ✓ The variable declared with the static keywords is called static variables.
- ✓ It returns the updated value every time, how much we call the function.

e.g.

```
#include<stdio.h>
#include<conio.h>
void show()
{
int a=10;
static int b=23;
a=a++;
b=b++;
```

```
printf("%d\n",a);
printf("%d\n",b);
}
void main()
{
clrscr();
show();
show();
show();
getch();
}
```

Output:

```
11
24
11
25
11
26
```

- ✓ If we call show function many times the local variables print the same variable for each function called.
- ✓ But the static variable will print the each value that incremented or decremented in the block.

❖ PROGRAMMING ERRORS IN C

Errors are the problem that occured in the program, which makes the behavior of the program abnormal.

Programming error are also known as the bugs or faults and the process of removing of these bugs is known as debugging.

- TYPES OF ERRORS :-
  1. Syntax error/Compilation error
  2. Runtime error
  3. Linker error
  4. Logical error
  5. Semantric error

1. **Syntax errors:-** Syntax error are also known as compilation errors. As they occured at compilation time or we can say that the syntax error are thrown by the compiler. These errors are mainly occur due to the mistake while typing or do not follow the syntax of the specified programming language.

   e.g.

   ```
   void main()
   {
   int a=10
   printf("%d",a);
   }
   ```

   In the above example we observe the code throws the error that a is undeclared.

   Missing ; at int a=10

2. **Runtime errors :-** Sometimes the errors exists during the execution time even after successful compilation known as runtime errors.

   e.g.

   ```
   #include<stdio.h>
   int main()
   {
       int a=10;
       int b=a/0;
       printf("%d",b);
       return 0;

   }
   ```

3. **Linker errors :-** Linker errors are mainly generated when the executable files of the program not created, this can be happen either due to wrong function prototyping or uses of the wrong header file.

   e.g.

   ```
   #include<stdio.h>
   int Main()      //Here are errors
   {
   int a=10;
   printf("%d",a);
   return 0;
   ```

}

4. **Logical errors:-** Logical error is an error that leads to an undesired output. These errors produce incorrect output but they are error free. They are known as logical errors.

   e.g.

   ```
   #include<stdio.h>
   int main()
   {
   int a= -10;
   printf("%u",a);
   return 0;
   }
   ```

5. **Semantric errors :-** Semantric errors are the errors that occurred when the statements are not understable by the compiler.

   e.g.

   ```
   int num1=10;
   int num2=20;
   int sum=0;
   num1+num2=sum;
   ```

   or

   ```
   sum+=10; ==> sum=sum+10;
   ```

❖ CONTROL STATEMENT AND DECISION MAKING

Decision making is about deciding the order of execution of statements based on certain conditions or repeat group of statement until certain specified conditions are made. C language handles decision making by supporting the following statements :

- If statement
- Switch statement
- Conditional operator statement/Loops is C
- Goto statement

- IF STATEMENT :- The if statement is used to check some given condition and perform some operations depending upon the

correctness of that condition. It is mostly used in the senerio where we need to perform the different operation for the different conditions.

Syntax:

```
If(expression)
{
//code
}
```

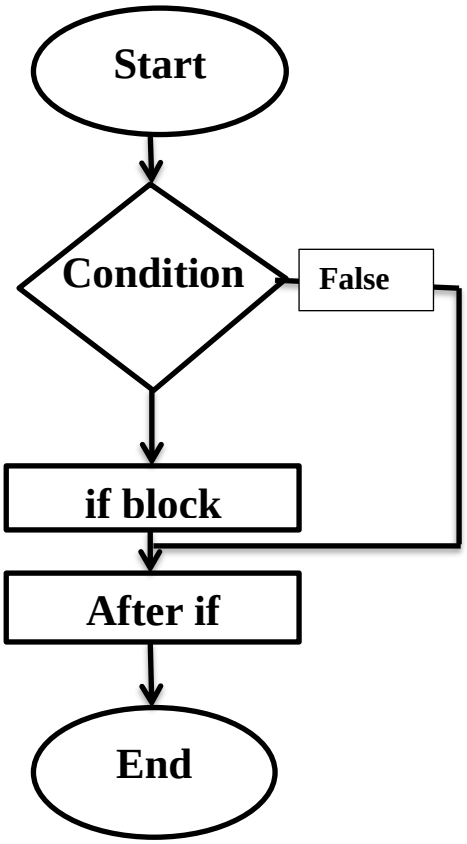
e.g.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int num;
printf("Enter a positive number.\n");
scanf ("%i",&num);
if(num==10)      //Here…
{
printf("given number is equal to 10");
}
printf("okay no issues");
return 0;
}
```

Note: Here we can use semi-colomn but when we only use if statement in the program if we use else statement in that program it gives error saying 'missplace use else'.

And if we not use curly braces after if statement we can only print one line of if statement when condition is true and second line is print when condition is false. When we have to print multiple line in if statement when condition is true, we must have to use curly braces.

Flowchart of if statement :



- **if-else statement :-** The if-else statement is used to perform two operation for a single condition. The if-else statement is an execution to the if statement using which, we can perform two different operation i.e. one is for the correctness of the condition and other is for incorrectness of the condition.

  Note: The if and else block cannot be executed simultaneously.

  If the given condition is satisfied then only if block is executed.

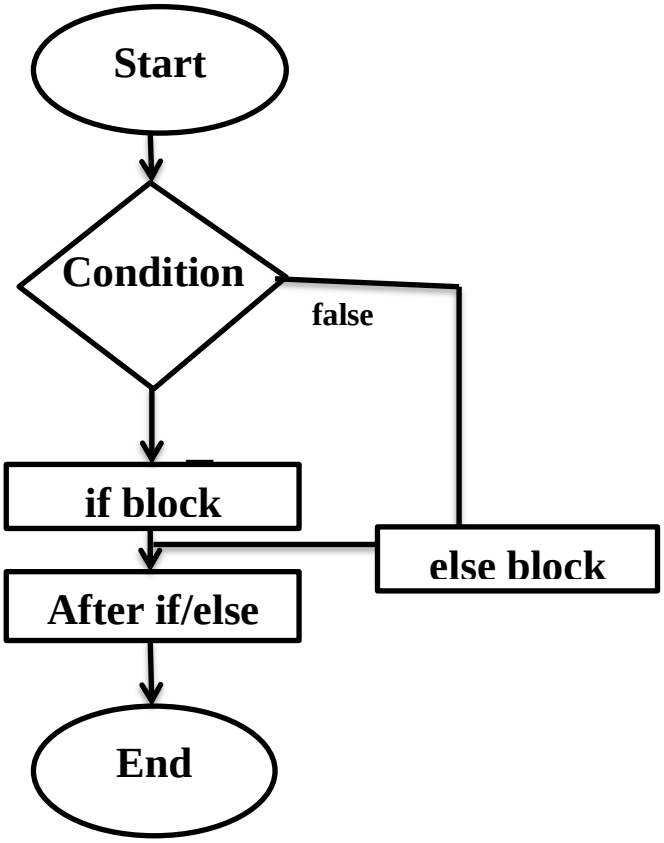
  Syntax:

```
if(condition/expression)
{
//code to be executed if condition is true;
}
else
{
//code to be executed if condition is false;
}
```

e.g.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int num;
printf(“Enter a positive number.\n”);
scanf (“%i”,&num);
if(num==10)      //Here…
{
printf(“given number is equal to 10”);
}
else
printf(“okay no issues”);
return 0;
}
```

Flowchart



- **if else-if ladder statement :-** The if else-if statement is an expression to the if else statement. It is used in the senerio where there are multiple cases to be perform of different conditions. In this statement if a condition is true then statement define in the if block will be executed, otherwise if some other condition is true then the statement define in else if block will be executed. At last if non of the condition is true then the statement define in the else block will be executed.

  Syntax:-

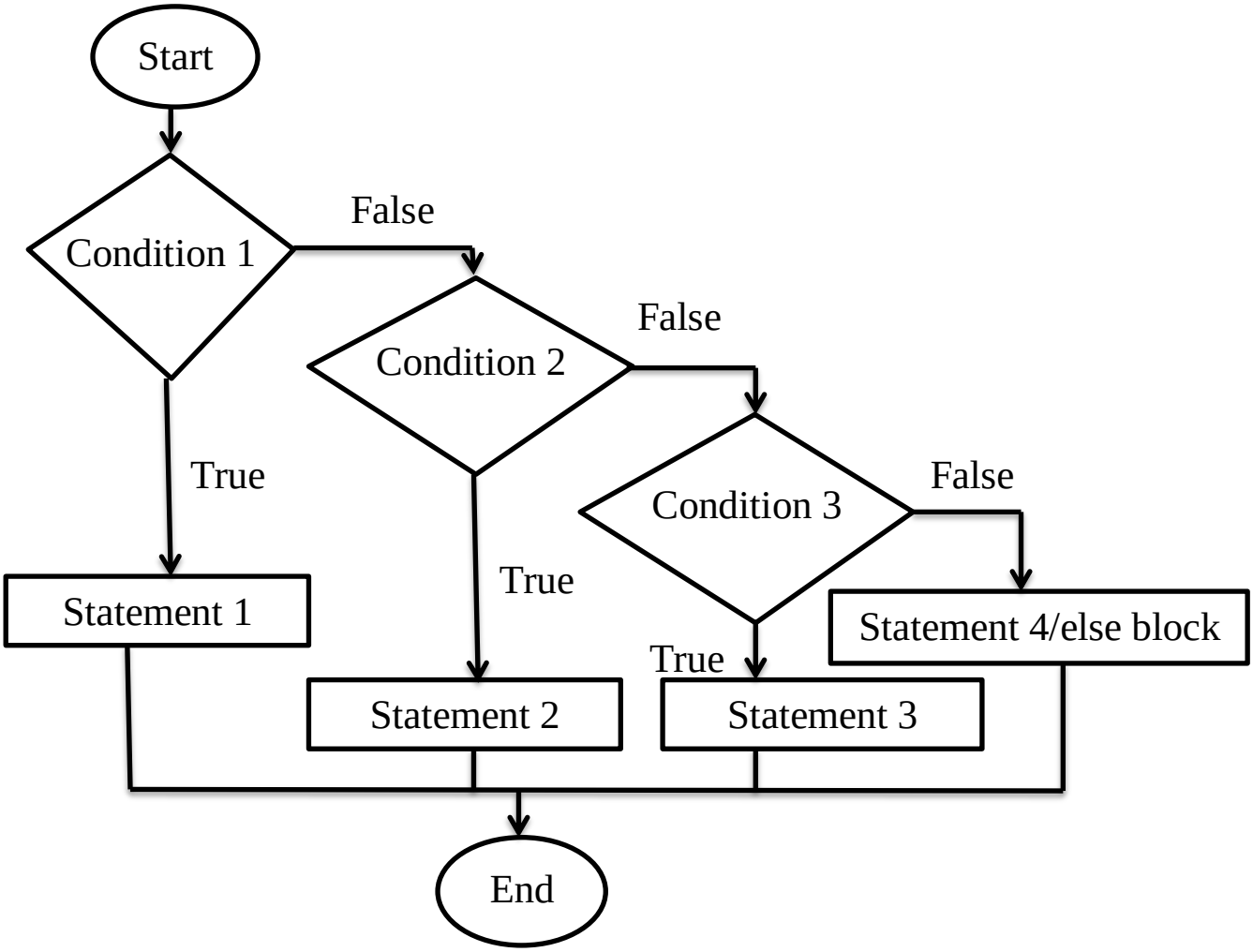
```
if(condition 1)
{
//if condition is true then it will be executed.
}
else if(condition 2)
{
//code
}
else if(condition 3)
{
//code
}
else
{
//this message is printed
}
```

  e.g.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a;
printf("Enter a number.");
scanf("%d",&a);
if(a==10)
{
printf("%d is equal to 10",a);
```

```
else if (a==20)
{
printf("%d  is equal to 20",a);
}
else if (a==30)
{
printf("%d  is equal to 30",a);
}
else
{
printf("Please enter again.");
}
return 0;
}
```

Flowchart:



- **Nested if..else statement :-** if else statement is inside the another if else statement is called Nested if..else statement.

  Syntax:-

```
if(condition 1)
{
        if(condition 2)
        {
        //code
        }else{
                //code
                }
}else{
        if(condition 3)
        {
        //code
        }else{
                //code
                }
}
```

e.g.

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a,b,c;
printf("Enter three number.");
scanf("%d%d%d",&a,&b,&c);
if(a>b)
{
        if(a>c)
```

```
        {
        printf("%d is greater.",a);
        }else{
                printf("%d is greater.",c);
                }
}else{
        if(b>c)
        {
        printf("%d is greater.",b);
        }else{
                printf("%d is greater.",c);
                }
        }
```

Explaination:-

If condition is false then block3 will be executed otherwise the execution continues and enters inside the first if to perform the check for next if block, where if condition 2 will be executed.

Note:-

1. In if statement a single statement can be include without enclosing it into curly braces({..}).

e.g.

```
int num=7;
if(num>0)
printf("Yes it's greater than 0.");
```

In above example no curly braces are required,but if we have more than one statement inside if block then we must enclosed then inside the curly braces.

2. == must be used to comparision the expression of if condition, if you use = the expression will always return true,because it perform assignment not comparision.
3. Other than zero(0) all other values are considered as true.
   e.g.
   if(10)
   printf("Please write down all these things.");

   In the above example 'Please write down all things' will print.

- SWITCH STATEMENT :-Switch statement is a control statement that allows us to choose only one choice among of many them. The expression in switch to return an integers value which is than compare to the values present in different cases. It executed that block of code which matches the case value. If there is number match then default block is executed.

- **Syntax:**

```
switch(expression)
{
case value-1:
//code;
break;
case value-2:
//code;
break;
.
.
case value-n;
//code;
break;
default :
//code/message;
```

```
break;
}
```
e.g.
```
#include<stdio.h>
#include<conio.h>
int main()
{
        int choice;
        clrscr();
        printf("Enter a numeric digit for days.");
        scanf("%d",&choice):
        switch(choice)
        {
          case 1:
          printf("Today is Sunday.");
          break;
              case 2:
              printf("Today is Monday.");
              break;
                    case 3:
                    printf("Today is Tuesday.");
                    break;
                          case 4:
                          printf("Today is Wednesday.");
                          break;
                                case 5:
                                printf("Today is Thrusday.");
                                break;
                                      case 6:
                                      printf("Today is Friday.");
                                      break;
                                        case 7:
                                        printf("Today is Saturday.");
                                        break;
default:
printf("Invalid input!!  Please enter number within 1-7");
```

```
getch();
return 0;
        }
}
```

Rules for switch statement

- ✓ The switch expression must be of an integers type and character types.
- ✓ The case value must be an integers and character type constant.
- ✓ The case value can only used inside the switch statement.
- ✓ The case level values must be end with a coloum.
- ✓ The break statement are used to exits the switch block is not necessary to use break after each case block. But if we don't use it then all the community blocks of code will get executed after the matching block.
- ✓ Default case is executed when none of these mentions case matches the switch expression.
- ✓ The default case can placed in the switch case.
- ✓ Using default case is optional.
- ✓ We can write default any where in the statement.

- LOOPS IN C :-Loops are used to execute a set up statement repeatedly until a particular condition is satisfied.
- TYPES OF LOOPS :-
  1. While loops
  2. For loops
  3. Do loops

1. **While loops :-** It is also known as pre-tested loop or entry controller loop. A while loop allows a part of the code to be executed multiple times depending upon given condition. It is also known as entry controller loop because it takes entry first and then print the statement.

   It is completed in three steps:-

   i. Variable initialization (e.g. int x=1;)
   ii. Condition (e.g. while(x<=5))
   iii. Modification (incrementation/Decrementation) (e.g. x++ or x—or x=x+2)

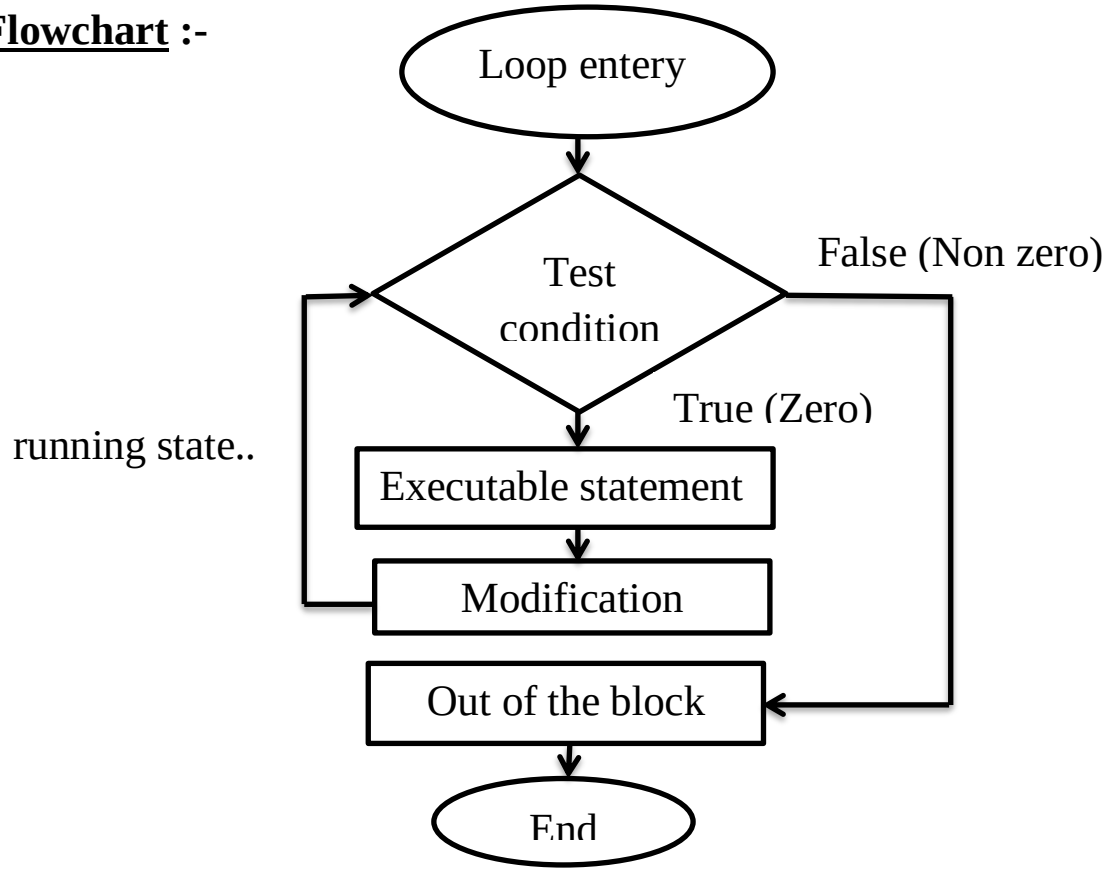
Syntax:

```
while(condition)
{
statements;
modification;
}
```

e.g.

```
#include<stdio.h>
int main()
{
        int i=1;
        while(i<=10)
        {
        printf("%d",i);
        }
return 0;
}
```

Flowchart :-



2. **For loops :-** Loops are used to execute a set up statement repeatedly until a particular condition is satisfied. We can say, it is an open ended loop.

   Syntax:-

   for(initialization;condition;modification)
   {
   Statements;
   }

   The for loop is executed as follows :-

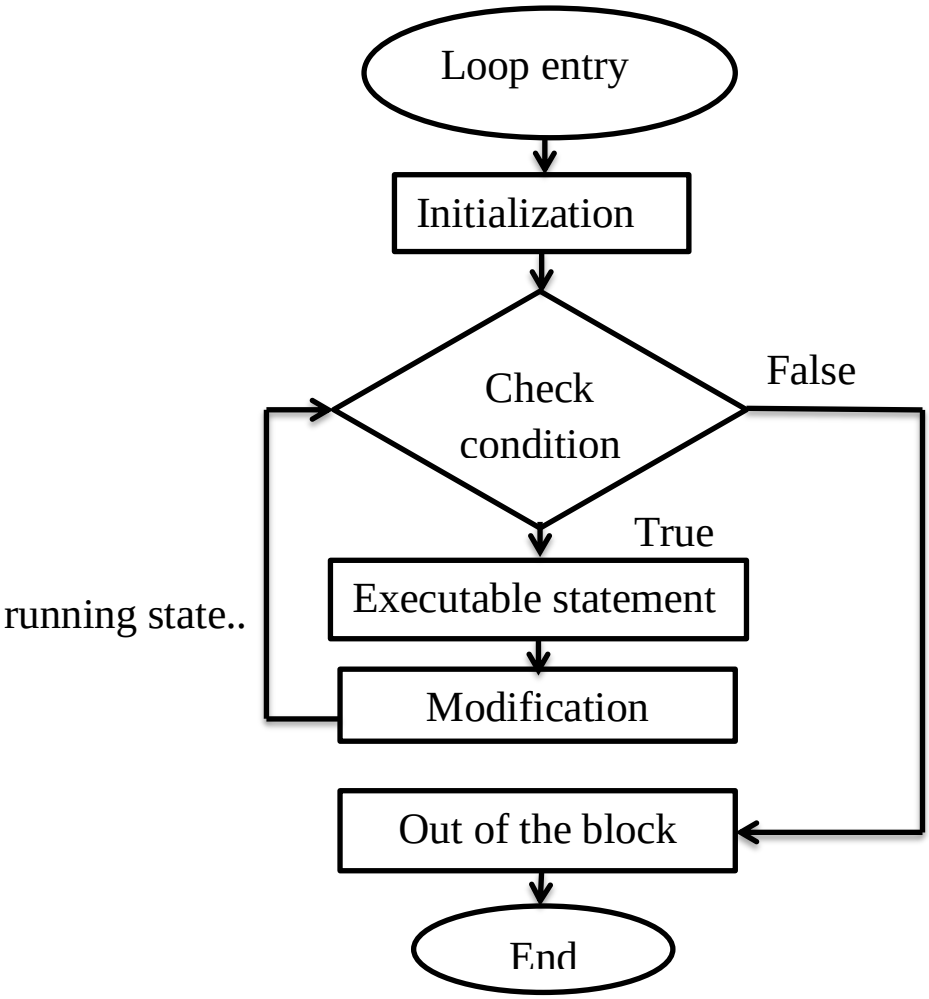
   i. It first, evaluate the initialization code.
   ii. Then it check the condition.
   iii. If the given condition is true then it execute the for loop body.
   iv. Then it evaluate the increment or decrement condition and again follows from steps 2. When the condition becomes false, it exist the loop.

   e.g.

   #include<stdio.h>

```
int main()
{
    int I,num;
    printf("Enter the number so that I can display a table.\n");
    scanf("%d",&num);
    for(i=1;i<=10;i++)
    {
        printf("%d",(num*i));
    }
return 0;
}
```

Flowchart :-



running state..

- **Nested for loop in C :-** Nested for loop means for loop inside for loops.

  Syntax:-

```
for(initialization;condition;modification)
{
     for(initialization;condition;modification)
    {
        inner loop statements;
    }
outer loop statements;
}
```

  e.g.

```
#include<stdio.h>
int main()
{
        int i,j,n;
        printf("Enter number of rows \n");
        scanf("%d".&n);
        for(i=1;i<=n;i++)
        {
            for(j=i;j=n;j--)
            {
                  printf("%d",j);
            }
        printf("\n");
        }
}
```

3. **Do while loop :-** In some situation, it is necessary to execute i.e. body of the loop before testing the condition. This types of situation can be handle with the help of do while loop. Do statement evaluate the body of the loop first and at the end the condition is checked using while statement. It means that the body of the loop will be executed at least once even through the static condition inside while is initialized to be false. It is also known as exit controller loop because it print the statement once first and

then check the condition. If the condition is true, it execute the statement repeatedly and condition become false it exit.

Note: We can do all program of while and for loop from do-while loop but we can't do all program of do-while loop from for and while loop because do-while loop print statement first and then check the condition. Suppose we have to do one program in which we have to print one statement first and then check the condition if it is true, go to looping process. Like this program we can't do it from while and for loop. So that we can say.

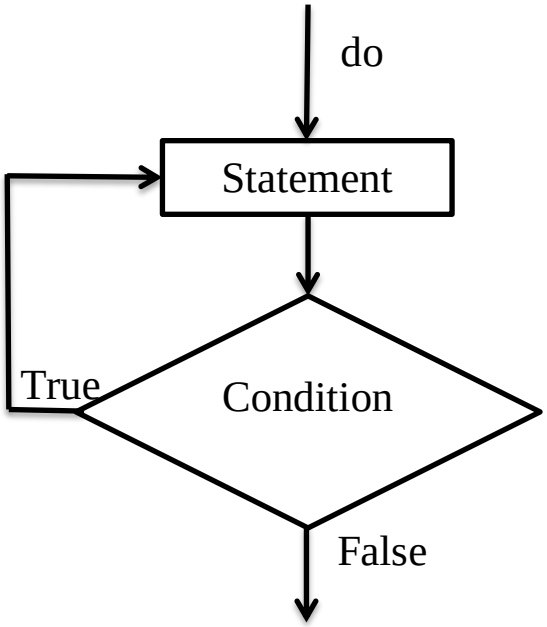
Syntax:-

```
do
{
   Statements;
}
while(condition);
```

e.g.

```
main()
{
   int i=2;
   do
   {
         printf("%d",i);
         i++;
   }
   while(i<=10);
}
```

Flowchart:-



- GOTO STATEMENT :- It is also known as jumping statement or jump statement. This statement is used to transfer the program control to a pre-define label. This statement can be used to repeat some part of the code for a particular condition. It jumping from one statement to another statement.

  Syntax:

  label:
  statements;
  goto label;

  e.g.

```
#include<stdio.h>
int main()
{
        int num,i=1;
        printf("Plz enter the number whose table u want to see.\n");
        scanf("%i",&num);
        table:
        printf("%i x %i = %i\n",num,i,num*i);
        i++;
        if(i<=10)
        goto table;

return 0;
```

}

Explanation :- goto means it directly jump to statement from where you want to execute repeatedly. Suppose you enter a name like label in the above syntax and in which you execute some statement inside it and you want to execute or call these statement repeatedly, simply write goto label to execute it on some other place inside the program. It work like loop but it call from any other place but in loop we can't do like this.

- INFINITE LOOP IN C :- An infinite loop is a looping construct that does not terminate the loop and execute the loop forever until the memory get full. It is also known know as indefinite loop or endless loop. We can create an infinite loop through various loop structure. i.e.
  i. for loop
  ii. while loop
  iii. do- while loop
  iv. goto statement and
  v. c macro

1. **Syntax for infinite for loop :-**

```
for( ; ; )
{
// Statements;
}
```

e.g.

```
#include<stdio.h>
int main()
{
        for( ; ; )
        {
                printf("Welcome to C community.");
        }
return 0;
}
```

2. **Syntax for infinite while loop :-**

```
while(1)   // Put any non-zero number because it gives condition true.
{
// Statements;
}
```

e.g.

```
#include<stdio.h>
int main()
{
        while(1)
        {
                printf("Welcome to C community.");
        }
return 0;
}
```

3. **Syntax for infinite do-while loop :-**

```
do
{
// Statements;
}
while(1);
```

e.g.

```
#include<stdio.h>
int main()
{
        do
        {
                printf("Welcome to C community.");
        }while(1);
return 0;
}
```

4. **Syntax for infinite goto statement :-**

```
infinite_loop
// Statements;
goto infinite_loop
```

e.g.

```
#include<stdio.h>
int main()
{
        hello:
        for( ; ; )
        {
                printf("Welcome to C community.");
        }
        goto hello;
return 0;
}
```

5. **Syntax for infinite macros :-**

```
#define infinite for( ; ;)   //Here you can use any looping.
main()
{
        infinite
        {
        // Statements;
        }

}
```

e.g.

```
#include<stdio.h>
#define infinite for( ; ; )
int main()
{
        infinite
        {
                printf("Welcome to C community.");
        }
return 0;
}
```

- BREAK STATEMENT IN C :- The break is a keyword in C which is used to bring the program control out of the loop. The break statement is used inside loops or control statements.

Syntax:-

```
switch(expression)
{
        case 1:
        // Statements;
        break;
        .
        .
        default:
        //Statements;
        break;
}
```

e.g.

```
#include<stdio.h>
int main()
{
        int num=2;
        switch(num)
        {
                case 1:
                printf("one \n");
                break;
                case 2:
                printf("Two \n");
                break;
                default:
                printf("default value \n");
                break;
                case 3:
                printf("Three \n");
                break;
        }
}
```

- CONTINUE STATEMENT :- Continue statement is used to bring the program control to the beginning of the loop. This statement skip

some line of code inside the loop and continue with the next execution.

Syntax:-

loop statement
continue;
some more statement which is to be skipped by the compiler.

e.g.

```
#include<stdio.h>
int main()
{
        int num=0;
        while(num<=10)
        {
                printf("%i\n",num);
                continue;
                num++;
        }
}
```

❖ FUNCTIONS

A function is a set of instruction. It is taking the input (i.e arguments) and processing the data, after processing the data it produce the output. In other words, dividing a large program into the basic building blocks known as function. Functions contains set of Statements enclosed by curly braces. There are many situations where we might need to write same line of code for more than once in a program. This may lead to unnecessary repetition of code. So, C language provides an approach in which you can declare and define a group of statements once in the form of a function and it can be called and used whenever required. Every C program has at least one function, which is main() function.

- C FUNCTION CAN BE CLASSIFIED INTO TWO CATEGORIES.

1. Library function
2. User-defined function

1. **Library functions :-** Library functions are the functions which are already defined in C library such as printf(), scanf(), gets(), puts(), strcat(), floor() etc. You just need to include appropriate header files to use these functions. These are already declared and define in c libraries.
2. **User-defined function :-** User defined-functions are those functions which are defined by the programmer at the time of writing program.It reduces the complexity of a big program and optimize the code.

- **ADVANTAGE OF FUNCTION IN C:-**

There are the following advantages of c functions.

- ✓ Function makes your code reusable. You just have to call the function by its name to use it, whenever required.
- ✓ It makes the program more readable and easy to understand.
- ✓ By using functions, we can avoid rewriting same code again and again in a program.
- ✓ We can call C functions multiple times in a program.

- FUNCTION ASPECTS :-

There are three aspects available in C function.

1. Function declaration
2. Function definition
3. Function calling

1. **Function Declaration:** Like any variable, a function must be declared before its used. The function declaration tells the compiler about a function name, function return type and function parameters. In C program function must declared globally.

- General syntax for function declaration is :-

return_type function_name (parameter_list);

- FUNCTION DECLARATION CONSISTS OF THREE PARTS:-

1. return_type
2. function name
3. parameter list

1. **return_type:-** A function may return a value. The return_type is the data type of the value of the function returns. In case your function doesn't return any value, the return type would be void.
2. **function name:-** Function name is an identifier and it specifies the name of the function. The function name will be your any valid C identifier.
3. **parameter list:-**The parameter list declares the type and number of arguments that the function expects when it is called. Also, the parameters will receive the argument values when the function is called.

2. **Function Definition:** Function definition contains the actual statements which are to be executed. It is the most import aspects to which the control comes when the function is called. The function body contains a collection of statements that define what the function does.

   - General syntax for function definition is:

     return_type function_name(parameter_list)

     {

     //function body;

     }

3. **Function calling:** To use a function, you will have to call that function to perform the defined task. When a program calls a function, the program control is transferred to the called function. A called function perform a defined task and when its return statement is executed or when its function ending closing brace is reached, it returns the program control back to the main program. To call a function, you simply need to pass the required parameters along with function name. The parameter list must not be differ in function calling and function declaration. We must pass same number of functions as it is declared in the function declaration. If the function returns a value, then you can store the returned value.

   - General syntax for function calling:

     function_name (arguments);

Example:-

```
#include<stdio.h>
int multiply(int,int);        /*Function declaration */
int main()
{
        int num1,num2,res;     /*local variable declaration*/
        printf("Enter two number\n");    //print message
        scanf("%i%i",&num1,&num2);   //taking user input for 'n'
        res = multiply(num1,num2);       /*Function call*/
        printf("multiplication of %i and %i is:%d\n",num1,num2,res);
return 0;
}
int multiply(int num1,int num2)        /*Function definition*/
{
        return num1*num2;
}
```

Output:-

```
Enter two number
45
87
multiplication of 45 and 87 is:3915
```

- FUNCTION CALLING ASPECTS:-

In every programming language there are four different aspects of function calls available.

1) Function without return_type and without arguments.

2) Function without return_type and with arguments.
3) Function with return_type and without arguments.
4) Function with return_type and with arguments.

1. **Example of function without return_type and without arguments.**

Q. Addition of Two numbers using function :-

```
#include<stdio.h>
void addAll(void);/*Function declaration without return_type and
                    without arguments.*/
int main()
{
    addAll();      // function calling
return 0;
}
void addAll()      //function definition
{
int num1,num2,sum=0;
printf("Enter 2 numbers\n");
scanf("%i %i",&num1,&num2);
sum = num1+num2;
printf("\n Sum of %i and %i is: %i",num1,num2,sum);
}
```

Output:-

```
Enter 2 numbers
85
45
```

Sum of 85 and 45 is: 130

2. **Example of function without return_type with arguments:-**

Q. Addition of Two numbers using function :-

```
#include<stdio.h>
void addAll(int,int);/*Function declaration without return_type with
                                arguments.*/
int main()
{
    int num1,num2;
    printf("Enter 2 numbers\n");
    scanf("%i %i",&num1,&num2);
    addAll(num1,num2);     //function calling
return 0;
}
void addAll(int a,int b)   //function definition
{
    int sum=0;
    sum=a+b;
    printf("sum = %d",sum);
}
```

Output:-

```
Enter 2 numbers
56
12
sum = 68
```

3. Example of function with return_type and without arguments:-

Q. Addition of Two numbers by using function:-

```
#include<stdio.h>
float addAll(void);   /*Function declaration with return_type and
                              without arguments.*/
int main()
{
        float add;
        add = addAll();      //function call
        printf("Sum=%g",add);
}
float addAll(void)        //function definition
{
        float num1,num2;
        printf("Enter 2 nos\n");
        scanf("%g%g",&num1,&num2);
        return num1+num2;
}
```

Output:-

```
Enter 2 nos
48.87
78.27
Sum=127.14
```

4. Example of function with return_type and with arguments:-

Q. Addition & multiplication of two numbers.

```
#include<stdio.h>
float addAll(float,float);   /*Function declaration with return_type and
                                with arguments.*/
float multiply(float,float);
int main()
{
    float num1,num2,res,res1;
    printf("Enter two numbers \n");
    scanf("%g %g", &num1,&num2);
    res = addAll(num1,num2);
    res1 = multiply(num1,num2);
    printf("sum=%g",res);
    printf("\n Multiplication=%g",res1);
return 0;
}
float addAll(float num1,float num2)
{
    return num1+num2;
}
float multiply (float num1,float num2)
{
    return num1 * num2;
}
```

Output:-

```
Enter two numbers
```

52.47

44.76

sum=97.23

Multiplication=2348.56

In the above code while declaring the function, we have declared two parameters of float type. Therefore, while calling the function, we need to pass two arguments, else we will get compile time error([Error]: too few arguments to function ‘addAll’). And the two arguments passed should be received in the function definition, which means that the function header in the function definition should have the two parameters to hold the argument values. The name of the variables while declaring, calling and defining a function may or may not be same.

- RETURNING A VALUE FROM FUNCTION:-

A function may or may not return a value. But if it does, we must use the return statement to get the result. return statements also ends the function execution, so it must be the last statement of any function. If you write any statement after the return statement, it won’t be executed. The data type of the value returned using the return statement should be same as the return type mentioned at function declaration and definition. If any of it mismatches, you will get compile time error ([Error: conflicting types for ‘addAll’]).

Call by value and Call by reference in C:-

We all know that functions are called by their names then what is the use of call by value and call by reference here? Well if the function does not have any arguments, then to call a function you can directly use its name. But for functions with arguments, you can call a function in two different ways i.e.

1) Call by value
2) Call by Reference

1. **Call by value:-** Call by value means calling a function by value. In this method, the value of the actual parameters is copied to the formal parameters. As we are working with the copy values of the actual parameter

so we can not modify the value of the actual parameter by the formal parameter. The actual parameter is the argument which is used in the function call whereas formal parameter is the argument which is used in the function definition. Here different memory is allocated for actual parameters and formal parameters since the value of the actual parameter is copied into the formal parameter.

Example of Call by value in C:-

Q. Wap two numbers using call by value.

```
#include<stdio.h>
void swap(int,int);     //function declaration
int main()
{
        int num1,num2;
        printf("Enter two numbers \n");
        scanf("%i %i",&num1,&num2);
        printf("Before swapping the value of num1 = %i and num2 = %i
            \n",num1,num2);
        swap(num1,num2);      //passing value in the function
        printf("within main\n");
        printf("After swapping value of num1= %i and num2=%i \n",
            num1,num2);    /*here the value of actual parameters do
            not change by changing the formal parameters. So it will
            return num1=10 and num2=20*/
        return 0;
}
void swap(int num1, int num2)     //function definition
{
```

```
        int temp=num1;
        num1 = num2;
        num2 = temp;
        printf("Within called function\n");
        printf("After swapping the value of num1= %i and num2=%i
                \n",num1,num2);
}
```

Output:-

```
Enter two numbers
10
20
Before swapping the value of num1 = 10 and num2 = 20
Within called function
After swapping the value of num1= 20 and num2=20
within main
After swapping value of num1= 10 and num2=20
```

In the above example, the actual parameter num1 and num2 is not changed because we are passing the argument by value. So a copy of num1 and num2 is passed to the function, which is updated during function execution, and that copied value in the function is destroyed when the called function job is over. So the variable num1 and num2 in the main() is never changed.

2. **Call by reference:-** Call by reference means calling a function by reference i.e. we pass the address (reference) of a variable as the actual parameter to any function. The value of the actual parameter can be modified by changing the formal parameters since the address of the actual parameter is passed. Here the memory allocation is similar for all both actual parameters and formal parameters.

Example of Call by reference in C:-

Q. Swap two numbers using call by reference.

```
#include<stdio.h>
void swap(int* , int*);    //function declaration
int main()
{
        int num1,num2;
        printf("Enter two numbers \n");
        scanf("%i %i",&num1,&num2);
        printf("Before swapping the value of num1 = %i and num2 = %i
              \n",num1,num2);
        swap(&num1,&num2);//passing reference in the function
        printf("within main\n");
        printf("After swapping value of num1= %i and num2=%i \n",
                    num1,num2);
return 0;
}
void swap(int* num1, int* num2)     //function definition
{
        int temp;
        temp= *num1;
        *num1 = *num2;
        *num2=temp;
        printf("Within called function\n");
        printf("After swapping the value of num1= %i and num2=%i
```

```
\n",*num1,*num2);
}
```

Output:-

Enter two numbers

10

20

Before swapping the value of num1 = 10 and num2 = 20

Within called function

After swapping the value of num1= 20 and num2=10

within main

After swapping value of num1= 20 and num2=10

In the above example, the actual parameter num1 and num2 is changed because all the operation in the function are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.

- RECURSION IN C:-

Any function which call itself is called recursive function, and such function calls are called recursive calls. It involves several numbers of recursive calls. However, it is important to impose a termination condition of recursion. Recursion can not be applied to all the problems.

Example 1:-

Q. Calculate the factorial of a number using recursion.

```
#include<stdio.h>
int fact(int);
int main()
{
    int res,num;
```

```
        printf("Enter a number\n");
        scanf("%i",&num);
        res  = fact(num);
        printf("Factorial of %d is %d",num,res);
    return 0;
    }
    int fact(int num)
    {
        int res;
        if(num==0)
        res=1;
        else
        {
        res = num*fact(num-1);
        }
        return res;
    }
```

Output:-

```
Enter a number
6
Factorial of 6 is 720
```

Example 2:-

Q. Find the nth term of the Fibonacci series.

```
#include<stdio.h>
int fib(int);
```

```
int main()
{
    int num,i,temp=0;
    printf("Enter a number\n");
    scanf("%d",&num);
    printf("See the Fibonacci series:\n");
    for(i=1;i<=num;i++)
    {
    printf("%d",fib(temp));
    temp++;
    }
return 0;
}
int fib(int num)
{
    if(num==0)
    {
    return 0;
    }
    else if (num==1)
    {
    return 1;
    }
    else
    {
```

```
        return fib(num-1)+fib(num-2);
        }
}
```

Output:-

Enter a number

8

See the Fibonacci series:

011235813

- MEMORY ALLOCATION OF RECURSION:-

Each recursive call creates a new copy of that function in the memory. Once some data is returned by the function, the copy is removed from the memory. Since, all the variables and other appliance declared inside function get stored in the stack, therefore a separate stack is maintained at each recursive call. Once the value is returned from the corresponding function, the stack gets destroyed.

❖ ARRAY

Arrays are the derived data type in c which can store primitive data types such as int, double, float, char etc. It has the capability to store the collection of derived data types such as structure, pointers etc. It is the collection of similar type of data items stored at contiguous memory locations. As we know, arrays are the collection of homogeneous data items so, instead of declaring individual variables, such as num_0, num_1, num_2, ……, num_n, you can declare one array variable such as numbers and use num[0], num[1], num[2],…., num[n] to represent individual variables. The array is the simplest data structure where each data elements can be randomly accessed by using its index number. C array is beneficial if you have to store list of Employee or Students names, marks of students or to store list of numbers of characters etc.

Elements of the array are stored at contiguous memory locations where the lowest address corresponds to the first element and the highest address to the last element.

num[0] [First Element]	num[1]	num[2]	num[3]	num[n] [Last Element]

- ADVANTAGE OF ARRAYS:-
  - ✓ In arrays, the elements can be accessed randomly by using the index number.
  - ✓ It represent multiple data items of the same type using a single name.
  - ✓ To sort the elements of the array, we need a few lines of code only.
  - ✓ Using arrays, other data structures like linked lists, stacks, queues, trees etc can be implemented.
- DISADVANTAGE OF ARRAYS:-
  - ✓ Whatever size define at the time of declaration of the array we can't exceed the limit.
  - ✓ Insertion and deletion are quite difficult in an array as the elements are stored in consecutive memory locations and the shifting operation is costly.
- DECLARATION OF ARRAY:-

Like any other variable, arrays must be declared before they are used. To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array.

Syntax:

```
type array_name[array_Size];
```

Example:

```
int arr[5];
```

In the above example, **int** is the data type, **arr** is the name of the array and 5 is the size of the array. It means array **arr** can only contain 5 elements of int type. As we know, **index** of an array starts from 0 to size-1 so first

element of arr will be stored at **arr[0]** and the last element will occupy **arr[4].**

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]

- INITIALIZATION OF ARRAY:-

  After an array is declared it must be initialized. otherwise, it will contain garbage value. An array can be initialized at either **Compile time** or at **Runtime**.

- COMPILE TIME ARRAY INITIALIZATION:-

  Like variable, we can initialize the c array at the time of declaration.

  Syntax:

  ```
  data_type array_name[size]={List of values};
  ```

  Example:

  ```
  int arr[5]={10,5,7,25,6};

  float arr[5]={21.47,45.42,46.87,88.47,94.27};
  ```

- RUNTIME ARRAY INITIALIZATION:-

  An array can also be initialized at runtime using **scanf().**

  Example:

  ```
  #include<stdio.h>

  int main()

  {

      int arr[10];

      int i,n;

      printf("Enter size of the array\n");

      scanf("%i",&n);
  ```

```
        printf("Enter %i elements to the array\n",n);
        for(i=0;i<n;i++)
        {
                scanf("%i",&arr[i]);
        }
        printf("See the array elements\n");
        for(i=0;i<n;i++)
        {
                printf("%i\t",arr[i]);
        }
        return 0;
}
```

Output:-

```
Enter size of the array
5
Enter 5 elements to the array
45
78
54
12
82
See the array elements
45      78      54      12      82
```

- TWO DIMENSIONAL ARRAY IN C:-

Double sub-scripted values are known as 2D array. The 2D array is organized as matrices which can be represented as the collection of rows and columns.

➢ DECLARATION OF TWO DIMENSIONAL ARRAY:-

Syntax:

```
data_type array_name[row_Size][column_Size];
```

Example:

```
int arr[2][3];
```

arr[0][0]	arr[0][1]	arr[0][2]
arr[1][0]	arr[1][1]	arr[1][2]

➢ INITIALIZATION OF 2D ARRAY IN C:-

In the one dimensional array, we don't need to specify the size of the array if the declaration and initialization are being done simultaneously. However, this will not work in 2D arrays. We will have to define at least the second dimension of the array. Like 1D array, it can be initialized at either **Compile time** or at **Runtime**.

➢ COMPILE TIME INITIALIZATION OF 2D ARRAY:-

```
int arr[2][3]={{1,2,5},{4,6,7}};
```

Example 1:

```
#include<stdio.h>
int main()
{
    int arr[2][3]={{10,20,30},{60,40,50}};
    int i,j;
    printf("See the array elements:\n");
```

```
        for(i=0;i<2;i++)
        {
            for(j=0;j<3;j++)
            {
            printf("arr[%d][%d]=%d\n",i,j,arr[i][j]);
            }
        }
        return 0;
    }
```

Output:

See the array elements:

arr[0][0]=10

arr[0][1]=20

arr[0][2]=30

arr[1][0]=60

arr[1][1]=40

arr[1][2]=50

Example 2:

int arr[][3];={{10,20,30},{60,40,50}};

```
    #include<stdio.h>
    int main()
    {
        int arr[][3]={{10,20,30},{60,40,50}};
        int i,j;
        printf("See the array elements:\n");
```

```
        for(i=0;i<2;i++)
        {
            for(j=0;j<3;j++)
            {
            printf("arr[%d][%d]=%d\n",i,j,arr[i][j]);
            }
        }
        return 0;
}
```

Output:

See the array elements:

arr[0][0]=10

arr[0][1]=20

arr[0][2]=30

arr[1][0]=60

arr[1][1]=40

arr[1][2]=50

In the above example, we have not assigned any row value that means we can initialize any number of rows. But we must have to specify number of columns, else it will give a compile time error.

Example 3:

```
#include<stdio.h>
int main()
{
    int arr[2][]={{10,20,30},{60,40,50}};
```

```
    int i,j;
    printf("See the array elements:\n");
    for(i=0;i<2;i++)
    {
        for(j=0;j<3;j++)
        {
        printf("arr[%d][%d]=%d\n",i,j,arr[i][j]);
        }
    }
    return 0;
}
```

Output:

It will generate compile time error i,e **[Error] array type has incomplete element type.**

- **RUNTIME INITIALIZATION OF 2D ARRAY:-**

```
#include<stdio.h>
int main()
{
    int arr[2][3],row,col;
    printf("Enter array elements\n");
    for(row=0;row<2;row++)
    {
        for(col=0;col<3;col++)
        {
            scanf("%i",&arr[row][col]);
```

```
            }
        }
        printf("\n See the array elements\n");
        for(row=0;row<2;row++)
        {
            for(col=0;col<3;col++)
            {
                printf("%i\t",arr[row][col]);
            }
            printf("\n");
        }
        return 0;
}
```

Output:

```
Enter array elements
15
45
78
95
82
42
See the array elements
15      45      78
95      82      42
```

- PASSING ARRAYS TO A FUNCTION IN C:-
  - **PASSING INDIVIDUAL ARRAY ELEMENTS TO FUNCTION:-**

```
#include<stdio.h>
void display(int ,int );
int main()
{
    int arr[]={2,6,4,8};
    display(arr[1],arr[2]);
    return 0;
}
void display(int num1,int num2)
{
    printf("%d\n",num1);
    printf("%d\n",num2);
}
```

Output:

6

4

  - **WAY TO DECLARE A FUNCTION THAT RECEIVES AN ARRAY AS AN ARGUMENT:-**

First way

```
return_type function_name (type array_name[])
```

Second way

```
return_type function_name (type array_name[size])
```

Third way

return_type function_name (type *array_name)

Example 1 : Sort an Array

```
/*Function to sort an array*/
#include<stdio.h>
void sort_array(int[]);    // function prototyping
int main()
{
    int arr[8]={10,45,12,78,43,88,77,132};
    sort_array(arr);     //function calling
    return 0;
}
void sort_array(int a[])    //function definition
{
    int i,j,temp;
    for(i=0;i<8;i++)
    {
        for(j=i+1;j<8 ;j++ )
        {
            if(a[j]<a[i])
            {
            temp=a[j];
            a[j]=a[i];
            a[i]=temp;
            }
```

```
            }
        }
        printf("see the elements after sorting\n");
        for(i=0;i<8;i++)
        {
        printf("%d\t",a[i]);
        }
    }
```

Output:

see the elements after sorting

10 12 43 45 77 78 88 132

Example 2:

```
/* Passing two dimensional array*/
#include<stdio.h>
void matrixdisplay(int num[2][2]);
int main()
{
    int num[2][2],row,col;
    printf("Enter 4 elements to the array\n");
    for(row=0;row<2;row++)
    {
        for(col=0;col<2;col++)
        {
            scanf("%d",&num[row][col]);
        }
```

```
        }
        matrixdisplay(num);//passing 2D Array
        return 0;
}
void matrixdisplay(int num[2][2])
{
        int row,col;
        printf("See the matrix\n");
        for(row=0;row<2;row++)
        {
                for(col=0;col<2;col++)
                {
                        printf("%d\t",num[row][col]);
                }
                printf("\n");
        }
}
```

Output:

```
Enter 4 elements to the array
12
45
72
94
See the matrix
12      45
```

72 94

- RETURNING ARRAY FROM THE FUNCTION:-

As we know that, a function can not return more than one value. However, if we try to write the return statement as return x,y to return two values (x,y), the function will return the last mentioned value. In some cases, we may need to return multiple values from a function. In such cases, an array is returned from the function. Returning an array is similar to passing the array into the function, The name of the array is returned from the function.

Syntax:

```
return_type* function_name()
{
//statements;
return array_type;
}
```

Example:

```
/* Returning array from the function:- */
#include<stdio.h>
int* sort_array(int[]);     // function prototyping
int main()
{
        int arr[8]={10,45,12,78,43,88,77,132};
        int *res= sort_array(arr),i;      //function calling
        printf("see the elements after sorting\n");
        for(i=0;i<8;i++)
        {
        printf("%d\t",*(res+i));
```

```
        }
        return 0;
}
int* sort_array(int a[])     //function definition
{
        int i,j,temp;
        for(i=0;i<8;i++)
        {
                for(j=i+1;j<8 ;j++ )
                {
                        if(a[j]<a[i])
                        {
                        temp=a[j];
                        a[j]=a[i];
                        a[i]=temp;
                        }
                }
        }
        return a;
}
```

Output:

see the elements after sorting

10 12 43 45 77 78 88 132

- SOME IMPORTANT PROGRAM OF ARRAY:-

Q. WAP to find the largest and second largest number from an array.

```
#include<stdio.h>
int main()
{
	int arr[20];
	int i,j,large,slarge,n;
	printf("Enter size of an array\n");
	scanf("%d",&n);
	printf("Enter %d elements to an array\n",n);
	for(i=0;i<n;i++)
	{
		scanf("%d",&arr[i]);
	}
	large=arr[0];
	slarge=arr[1];
	//logical part
	for(i=0;i<n ;i++)
	{
		if(arr[i]>large)
		{
			slarge=large;
			large=arr[i];
		}
		else if(arr[i]>slarge && arr[i]!=large)
		{
			slarge=arr[i];
```

```
            }
        }
        printf("largest value is: %d\n",large);
        printf("Second largest value is: %d",slarge);
        return 0;
    }
```

Output:

```
Enter size of an array
5
Enter 5 elements to an array
42
874
54
32
584
largest value is: 874
Second largest value is: 584
```

Q. WAP to insert an element to the array.

```
#include<stdio.h>
int main()
{
    int arr1[10];
    int i,loc,n,key;
    printf("Enter number of array \n");
    scanf("%d",&n);
```

```
printf("\nEnter %d elements in the array\n",n);
for(i=0;i<n;i++)
{
        scanf("%d",&arr1[i]);
}
printf("Enter value that u want to insert\n");
scanf("%d",&key);
printf("Enter location where u want to store the elements\n");
scanf("%d",&loc);
printf("\n See the arr1 elements that u inserted\n");
for(i=0;i<n;i++)
{
        printf("%d \t", arr1[i]);
}
for(i=n+1;i>=loc;i--)     //shift
{
        arr1[i+1]=arr1[i];
}
if(loc<=n)
{
arr1[loc]=key;
printf("\n see the elements after insertion\n");
for(i=0;i<=n;i++)
{
        printf("%d\t",arr1[i]);
```

```
        }
        }else
        printf("\n Invalid location!!! Please enter again :)");
        return 0;
    }
```

Output:

```
Enter number of array
4
Enter 4 elements in the array
10
20
30
40
Enter value that u want to insert
500
Enter location where u want to store the elements
3
See the arr1 elements that u inserted
10      20      30      40
see the elements after insertion
10      20      30      500     40
```

Q. WAP to delete a specified index from the array.

```
#include<stdio.h>
int main()
{
```

```
        int a[50],size,pos,i;
        printf("Enter size of the array\n");
        scanf("%d",&size);
        printf("Enter %d elements\n",size);
        for(i=0;i<size;i++)
        {
                scanf("%d",&a[i]);
        }
        printf("Enter position to delete\n");
        scanf("%d",&pos);
        if(pos<=0 || pos>size)
        {
                printf("Invalid position!!!");
        }else{
                for(i=pos;i<size;i++)
                {
                        a[i]=a[i+1];
                }
                size--;
        for(i=0;i<size;i++)
        {
                printf("%d\t",a[i]);
        }
        }
}
```

Output:

Enter number of array

4

Enter 4 elements in the array

10

20

30

40

Enter value that u want to insert

500

Enter location where u want to store the elements

3

See the arr1 elements that u inserted

10 20 30 40

see the elements after insertion

10 20 30 500 40

Q. Addition of 2 matrices.

```
#include<stdio.h>
int main()
{
    int arr1[30][30],arr2[30][30],sum[30][30];
    int r,c,n1,n2;
    printf("Enter size of array1\n");
    scanf("%d",&n1);
    printf("\n Enter size of array2\n");
```

```
scanf("%d",&n2);
printf("\n Enter %d elements in the array\n",(n1*n2));
for(r=0;r<n1;r++)
{
        for(c=0;c<n2;c++)
        {
        printf("Enter arr1[%d][%d] :- ",r,c);
        scanf("%d",&arr1[r][c]);
        }
}
printf("\n Enter %d elements in the array\n",(n1*n2));
for(r=0;r<n1;r++)
{
        for(c=0;c<n2;c++)
        {
        printf("Enter arr2[%d][%d] :- ",r,c);
        scanf("%d",&arr2[r][c]);
        }
}
for(r=0;r<n1;r++)     //logic
{
        for(c=0;c<n2;c++)
        {
                sum[r][c]=arr1[r][c]+arr2[r][c];
        }
```

```
    }
    printf("\n See the 1st matrix\n");
    for(r=0;r<n1;r++)
    {
        for(c=0;c<n2;c++)
        {
            printf("%d\t",arr1[r][c]);
        }
            printf("\n");
}
printf("\n See the 2nd matrix\n");
    for(r=0;r<n1;r++)
    {
        for(c=0;c<n2;c++)
        {
            printf("%d\t",arr2[r][c]);
        }
            printf("\n");
}
    printf("\n See the resultant matrix\n");
    for(r=0;r< n1;r++)
    {
        for(c=0;c<n2;c++)
        {
            printf("%d\t",sum[r][c]);
```

```
            }
                printf("\n");
    }
}
```

Output:

```
Enter size of array1
2
Enter size of array2
3
Enter 6 elements in the array
Enter arr1[0][0] :- 10
Enter arr1[0][1] :- 20
Enter arr1[0][2] :- 30
Enter arr1[1][0] :- 40
Enter arr1[1][1] :- 50
Enter arr1[1][2] :- 60

Enter 6 elements in the array
Enter arr2[0][0] :- 70
Enter arr2[0][1] :- 80
Enter arr2[0][2] :- 90
Enter arr2[1][0] :- 100
Enter arr2[1][1] :- 25
Enter arr2[1][2] :- 48
```

See the 1st matrix

10 20 30

40 50 60

See the 2nd matrix

70 80 90

100 25 48

See the resultant matrix

80 100 120

140 75 108

Q. Transpose matrix.

```
#include<stdio.h>
int main()
{
    int a[3][3],t[3][3],r,c;
    for(r=0;r<3;r++)    //taking input
    {
        for(c=0;c<3;c++)
        {
        printf("Enter elements of [%d][%d]-:",r,c);
        scanf("%d",&a[r][c]);
        }
    }
    for(r=0;r<3;r++)    //logic
```

```
        {
            for(c=0;c<3;c++)
            {
            t[c][r]=a[r][c];
            }
        }
        printf("\n See the user matrix\n");
        for(r=0;r<3;r++)
        {
            for(c=0;c<3;c++)
            {
            printf("%d\t",a[r][c]);
            }
        printf("\n");
        }
        printf("See the transpose matrix\n");
        for(r=0;r<3;r++)
        {
            for(c=0;c<3;c++)
            {
            printf("%d\t",t[r][c]);
            }
        printf("\n");
        }
    return 0;
```

}

Output:

Enter elements of [0][0]-:10

Enter elements of [0][1]-:20

Enter elements of [0][2]-:30

Enter elements of [1][0]-:40

Enter elements of [1][1]-:50

Enter elements of [1][2]-:60

Enter elements of [2][0]-:70

Enter elements of [2][1]-:80

Enter elements of [2][2]-:90

See the user matrix

10 20 30

40 50 60

70 80 90

See the transpose matrix

10 40 70

20 50 80

30 60 90