Programming Enviroment ;-

Programming is the process of taking an algorithm and encripting it into a notation, so that it can be executed by a computer.

Programming is often the way that we create a representation for our solutions. Therefore, the Languate representation and the process of creating becomes a fundamental part of this dicipline.

Programming Enviroment in programming Language Provides a common set of execeting libraries. It allows you to use only one execution enviroment for your applications, regardless of the programming Language or system resource needs, because most system dependencies have been removed.

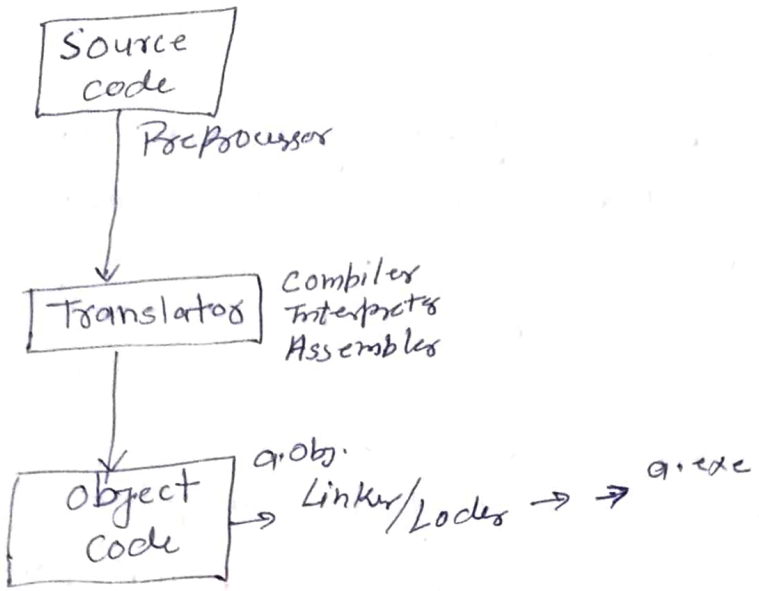
Programming environments provides to users different steps to create a good programming Language.

- A text editor to create computer programs.
- A compiler to compile the programs into binary format.
- An Interpreter to execute the programs directly.

"In a general sense, a programming enviroment combines hardware and software that allows a developer to build applications."

Developers typcally work in integrated development enviroments or IDEs. These connect user with all the features necessary to write and test their code correctly. Examples of IDEs, IntelliJ, Eclipse, NetBeans and Visual studio.

Program Execution



Algorithm :-

An algorithm is a finite step-by-step list of well defined instructions for solving a particular problem.

An algorithm named after ninth century mathematician Al-Khowarizmi is defined as follows:-

→ An algorithm is a set of rules for carrying out calculations either by hand or on a machine.

→ An algorithm is a sequence of computational steps that transform the input into the output.

→ An algorithm is a sequence of operations performed on data that have to be organized in data structures.

→ An algorithm is an abstraction of a program to be executed on a physical machine.

Characteristics of an Algorithm :-

1) **Unambiguous** :- Algorithm should be clear and unambiguous.

2) **Input** :- An algorithm should have 0 or more well defined ~~Outputs~~, ~~and should match~~ desired ~~output~~. Input.

3] Output :- An algorithm should have 1 or more well-define output.

4] Finiteness :- Algorithm must terminate after a finite number of steps.

5] Feasibility :- should be feasible with the available resource

6] Independent :- An algorithm should have step-by-step directions, which should be independent of any programming code.

How to Write an Algorithm ?

There are no well-defined standards for ~~writing~~ writing algorithms. Rather, it is problem and resource dependent. Algorithms are never written to support a particular programing code.

As we know that all programming languages share basic code constructs like loops (do, for, while), flow-control (If-else), etc. These common constructs can be used to write an algorithm.

Example :- Design an algo to add two number and display the result.

step1 - START
step2 - declare three integers a, b & c.
step3 - define values of a & b.
step4 - add values of a & b
step 5 - store output of step4 to c
step 6 - Print c.
step 7 - STOP.

Alternatively, the algorithm can be written as -

step1 - START ADD
step2 - get values of a & b.
step3 - c ← a+b
step4 - display c
step5 - STOP

Example-2 Write an algo to print a number even or odd.

Step1 - Start.

Step2 - Read a number to N.

Step3 - Divide the number by 2 and store the remainder in R.

Step4 - If R=0 then go to Step6.

Step5 - Print "N is odd" go to Step7.

Step6 - Print "N is even"

Step7 - Stop.

Example-3 Write an algo to finding the greatest of three numbers?

Step 1: Start

Step 2: Read three numbers A, B & C.

Step3: If A > B, then go to Step6

Step4: If B > C then print B & go to Step 8

Step5: Print C is greatest & go to Step 8

Step6: If A > C, then Print A is greatest & go to Step 8

Step 7: Print C is greatet

Step 8: end.

What is C

C is a Programming Language developed at AT & T's Bell Laboratories of USA in 1972. It was designed and written by man named Dennis Ritchie.

In the Late seventies C began to replace the more familiar Languages of that time like PL/I, ALGOL, etc. NO one pushed C. It wasn't made the 'official' Bell Labs and Language. Thus, without any advertisement, C's reputation spread and its pool of users grew. Ritchie seems to have been rather surprised that so many programmers preferred C to older Language like FORTRAN or PL/I, or the newer ones like Pascal and APL. But that's what happened.

History of C

1960	ALGOL	International Group
1967	BCPC	Martin Richards
1970	B	Ken Thompson
1972	Traditinal C	Dennis Ritchie
1978	K&R C	Kernighan & Ritchie
1989	ANSI C	ANSI committee
1990	ANSI/ISO C	ISO committee
1999	C99	standardization committee.

Characteristics of C!

→ C has become a popular programming Language because of its many features.

→ C is a general purpose Programing Language.

→ C is a structured Programming Language.

→ It has a rich set of operators.

→ It provide compact representation for expression.

→ It allows manipulation of internal processor registers.

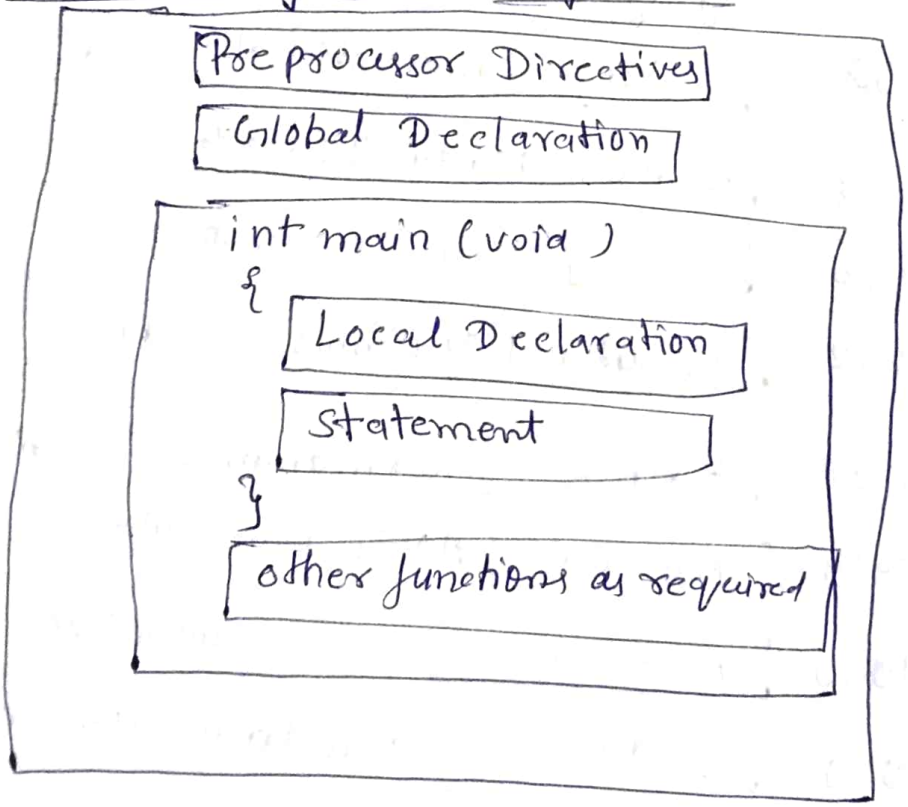
→ No grid format.

→ Portability.

→ Support rich set of data types.

→ Very less numbe of reserved words.

Structure of a C Program

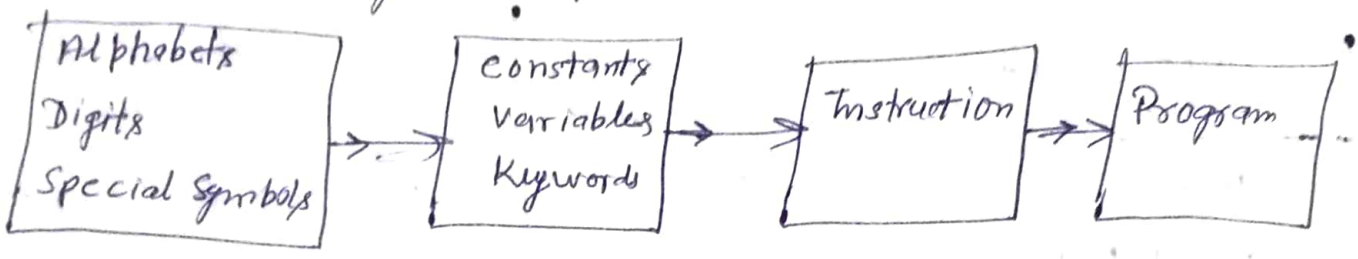


Getting started with C!-

steps in learing English Language

Alphabets → Words → Sentence → Paragraph

steps in learing C Language



Data Dytes!-

C Language is rich in its data type. storage representations and machine instructions to handle constants differ from machine to machine. The variety of data types available allow the programmer to select the type appropriate to the needs of the ~~all~~ appn as well as machine.

Data Types	Meaning	Size (bytes)
char	a character	1
int	an integer	2
float	a floating	4
double	a double precision real number	8
void	Values less	0

Data Types

Primary Data Type (Primtive)
- Character
- Integer
- float, double, void

Secondary Data types (Non Primtive – Non Linear / Learner)
- Array, Pointer
- Union, Enum etc

In C, there are three types of Datatypes

① Built in data types → Fundamental data type
② Derived data types → Derived out of fundamental data type
③ User define data types → by the user

Type Modifiers	Size (Bytes)	Range of Value
int	2	-32768 to 32767
signed int	2	-32768 to 32767
unsigned int	2	0 to 65535
short int	2	-32768 to 32767
long int	4	-2147483648 to 2147483647
float	4	3.4E+48 to 3.4E-48
double	8	-1.7E+308 to 1.7E308
char	1	-128 to 127
unsigned char	1	0 to 265
unsigned short int	2	0 to 65535
unsigned long int	4	0 to 4294967295
long double	10	-3.4E-4832 to 1.1E+4932

Backslash Space :-

Backslash constant	Meaning
\a	System alarm (bell)
\b	Back space
\f	Form feed
\n	New line
\t	Horizontal tab
\v	Vertical tab
\r	carriage return
\"	Double quote

\'	Apostrophe
\0	Null character
\\	Backslash character itself.

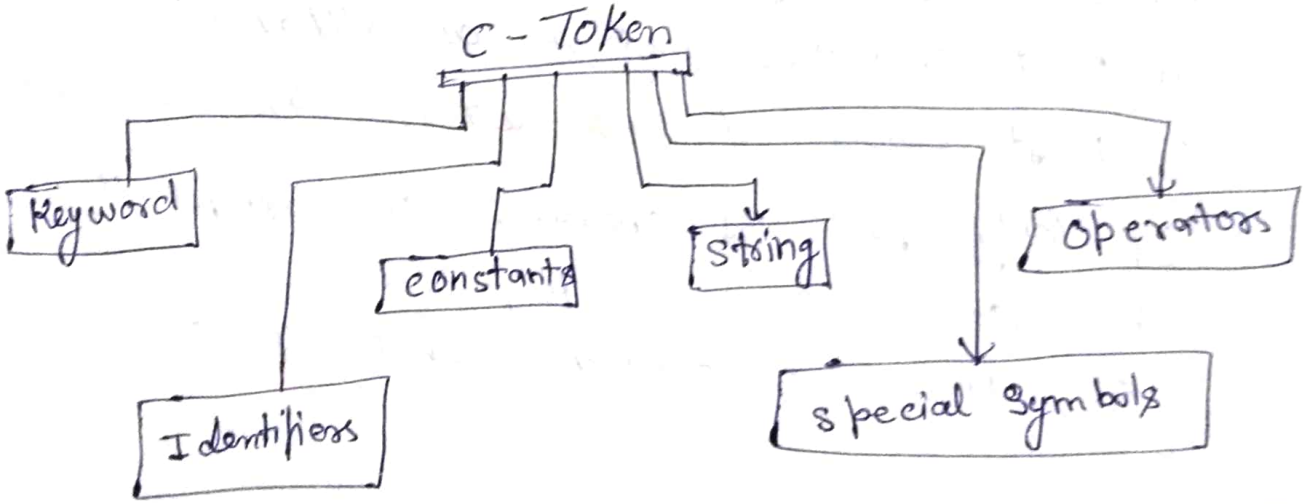
Data type Modifier!-

① Signed!- This can applied to integer variables. The deafault declaration assumes a signed number. The signed modifies can also applied with a char type, to create a small inter.

② Unsigned!- This can be used for both integer and character. It is use to create unsigned integer and it can also be used in combination with long or short. The high order bit of a signed integer is used as a signed flag. If the high order bit is zero, then the integer is treated negative.

③ Long!- This is applied to int data type, when applied to int, it essentially doubles the length in bits of the data type.

④ Short!- This make the int half., but most compiler have 16-bit int and since ANSI



A smallest individual unit of a C program is known as token.

Keywords:- Kyword are those word whose meaning have fixed and is known to it compiler. We cannot change the meaning of the kyword, If we ~~ty~~ try to do so an error message will occur.

List of Keywords -32

auto	short	Signed	If
break	struct	switch	return
case	unsigned	Void	static
char	Continue	default	Union
const	else	enum	while.
double	for	goto	
float	Long	register	
int	Volatile	~~zi~~ sizeof	
do	extern	typedef	

Identifiers:- Identifiers refer to the name of variable, function and arrays.

Rules for Identifiers —

1] First character must be an alphabet

2] Must consist of only letters, digit or underscore

3] Only first 31 charaters are significant.

4] can-not use a kyword.

5] Must not contain white space.

CONSTANTS :-

Constants in C refer to fixed values do not change during the execution of a program.

Type —

Integers constants

Floating point constants (Real constants)

Character constants (Single character string)

String constants

Integer constants :- These are whole number without any fractional part.

charaterisics —/Rule

(i) It must have at least digit.

(ii) It must not contain a decimal point.

(iii) It may either have + or - sign.

(iv) When no sign is present it is assumed to be positive.

(v) Commas and blanks are not permitted.

* The range of constant depends upon the compilers like Turbo C or Turbo C++ the range -32768 to 32767

Eg #

#

```
Void main()
{  Printf("Integer value \n");
   Printf("%d %d %d \n", 32767, 32767+1, 32767+10);
   Printf("\n")
  Printf("Long integer value \n");
  Printf("%ld %ld %ld \n", 32767L, 32767L+1L,
                                   32767L+10L);
  getch();
}
```

Output

Integer values -

32767 -32768 -32769

Long integer values

32767 32768 —

Real Constants / Floating Point constants :

1) A real constant must have at least one digit
2) must have decimal point.
3) It could be either positive or negative.
4) Default sign is positive.
5) No commam or blanks are allowed

Single Character Constants :-

A Single character constant contains a single character enclosed within a pair of single quote mark.

eg
```
Printf ("%d", 'a');
Printf ("%c", '97');
Printf ("%d", A);
Printf ("%c", 65);
```

String Constants :-

A string constant is a sequence of characters enclosed in double quotes.

eg = "Hell"

Variable :- A variable is a data name that may be used to store a data value. A variable may take different values at different times during execution.

Rules for Forming Variable Names :-

→ The first character of a variable must be an alphabet or an underscor.

→ All successding characters consists of letters and digits.

→ Both uppercase and lowercase varibles are significant in C.

→ Writing the varible name in lowercase is a good programming practice

→ Keywords should not be used as variable.

→ special characters are not allowed.

Valid	Invalid
Saurabh	
total_sum	
a_01	

Declaration of varibles

1] The type of data to be stored by the variable.

2] The name of variable.

Eg
```
int i, j;
float x, y;
char choice;
```

Floating Point Types

The C Standard recognizes three floating point types

1. Real
2. Imaginary
3. Complex.

Real:- The real type holds values that consist of an integeral and a fractional part, such as 49.32. The C Language supports three different size of real types float, double and long double.

Imaginary:- C define an imaginary type, An imaginary number is used extensively in mathematics and engineering. An Imaginary number is a real number multiplied by the square root of -1 ($\sqrt{-1}$). ~~The imaginary type, like the~~

Complex:-

C defines is complex type, which is implemented by most comilers. A complex number is a combination of a real and imaginary number.

Introduction of Input / Output

Be in ~~any~~ any program, it requires data to operate upon, so that it could process it and produce the result as per requirements. One method of providing data to the program to assign value to the variable within the program and ~~at~~ access the variable whenever required.

This method suits well for the situation where data is fixed type and happens to very small in volume. You cannot afford to deal with this method If data such that rollNo, name, address, subject, marks etc. of fifty student has to be proceed. It quite obvious that for a such ~~of~~ situations, program should accept the data from other source like, Keyboard etc. Well C- Language provide wide variety of Input/Output function for accepting the data from keyboard in the program and display the result on screen.

The statement #include <stdio.h>, include the content of standard input-output file, std.h at compile time. which contain the definition of stdin, stdout and stderr.

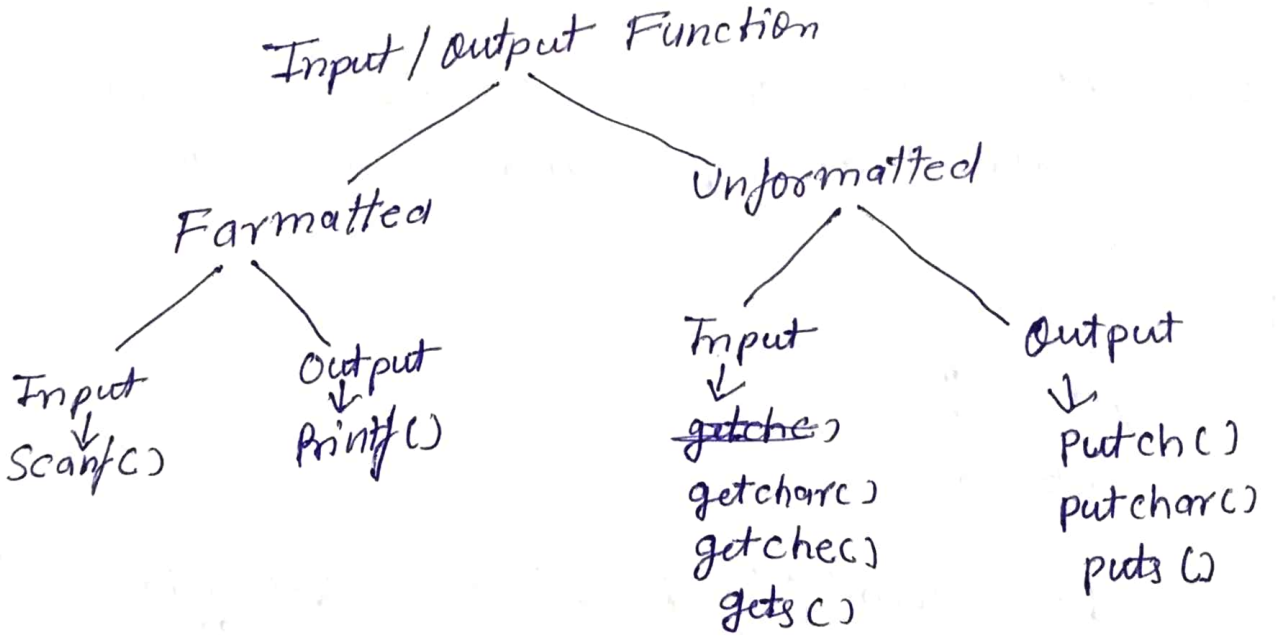


Fig - Type of Input/Output Functions

Formatted Functions -

① Formatted Input statement :-

The scanf() function for input operation :- It is used for formatting input from standard input device that is Keyboard.

The syntax of the function scanf() is:

```
scanf ("control string", address-list);
```

Where —

Control string :- It is sequence of one are more character group. Each charater group is combination of % symbol and one of the conversion character. The control strings specifies type of value which are to be supplied to the variables.

Address List :- address of memory location where the value of the input varibles should be stored

eg

```
int x;
float y;
char ch;
```

```
scanf ("%d %f %c", &x, &y, &c);
```

Format specification string

List of address of variable

character Group	Meaning
%c	data items is a Single character
%d	data item is a decimal character
%e	data item is a floating point value
%f	data item is a floating point values
%g	"
%h	short integer
%i	decimal, Hexadecimal or Octal integer.
%o	octal Integer
%s	string followed by a white space
%u	unsigned decimal integer
%x	hexadecimal integer.

ld → ~~long double~~

Formatted Output statement! -

Function Printf() display all type of data on the screen.

Format

Printf ("control string", variable);

where control string → specifies the type and format of the value to display.

variable → A variable to be displayed.

eg
```
Printf ("welcom")
Printf ("%d", a);
Printf ("%d %f", a,b);
Printf ("multiplication of three num = %d", mul);
Printf ("ln x = %f ln y = %d", x, y);
```

Unformatted Functions :-

① Unformatted Input statement :-

These statement primarily concerned with reading the character type data from the keyboard.

Header file — #include <stdio.h>

getchar() :- It return the character just entered from the standard input unit. Mostly is keyboard.

```
char_var = getchar();
```

where **char_var** → It is character type variable which an accepted character is assigned.

eg
```
#include <stdio.h>
void main()
{
    char letter;
    letter = getchar();
}
```

gets() :- This function reads in ~~every thing~~ everything you enter from the keyboard until the enter key or return is pressed.

Everything means sequence of all printable ASCII characters.

```
gets (string);
```

where - **String :-** is a sequence of character & its of type char.

eg
```
#include <stdio.h>
void main()
{
  char name[25];
  Printf ("enter your name");
    gets (name);
}
```

(2) Unformatted Output Function :-

Putchar() :- It is display the single character.

eg
```
#include <stdio.h>
Void main()
{
  char ch;
    Putchar (ch);
}
```

Puts() :- It Prints string of character :

```
#include <stdio.h>
void main()
{
  char message[50];
    Puts (message);
}
```

Q1. Ramesh's basic salary is input through Keyboard, his dearness allowance (da) is 40% of basic salary and house rent (~~Hr~~ (Hra)) allowance is 20% of basic salary. write a Progrogram in c to calculate his gross salary.

Sol^n
```
#include <stdio.h>
# include <conio.h>
void main()
 {
   float, bs, ~~har~~ hra, da, gs;
   Printf("\n Enter Ramesh's basic salary is= ");
   Scanf("%f", &bs);
   hra = bs*20/100;
   da = bs*40/100;
  Printf("\n House rent allowance = %f ", hra);
  Printf(" \n Dearness allowance = %f", da);

    gs = bs + hra + da;
    Printf("\nGross Salary = %f", gs);
    getch();
  }
```

Q2 Write A Program in c to swap two number. using third variable and without third variable.

Sol^n
```
# include <stdio.h>
# include <conio.h>
 Void main();
  { clrscr();
   int a, b, temp;
   Printf(" \n Enter two value a and b ");
    Scanf("%d%d", &a, &b);
   Printf(" \n Before swap the value is =");
   Printf(" a is = %d and b is = %d", a, b);}
```

```
        t = a;
        a = b;
        b = t;
    Printf ("\n after swap the value is:-");
    Printf (" a is = %d and b is = %d", a, b);
      getch();
    }
```

Q3 WAP in c to swap two no without using third variable

```
a = 10
b = 20
```

```
a = a + b
b = a - b
a = a - b
```

```
a = 30
b = 30 - 20 = 10
a = 30 - 10 = 20
```

Q4 WAP to Find area of circle.?

Sol"

```
#
#
void main()
{ clrscr()
  float r, a_o_c;
  float pie = 3.14;
   Printf("\n Enter the radius of a circle");
    scanf("%f", &r);
    a_o_c = pie * r * r;
    Printf("\n area of a circle = %f", a_o_c);
    getch();
  }
```

Q5 WAP to read two number and display the number and on their rounded off to 3 place after the decimal point.

Sol"

```
#
#
void main()
{ clrscr()
  float x, y, sum;
  Printf("Enter the value of x and y");
  Scanf("%f %f", &x, &y);
   Sum = x+y;
   Printf("\n x = %10.3f", x);
   Printf("\n y = %10.3f", x);
   Printf("\n sum = %10.3f", sum);
    getch();
  }
```

Q6 WAP to read any character from Keyboard and to display the ASCII number associated with it.

Sol^n

```
#
void main();
{ clrscr()
  char ch;
  Printf("Enter any character");
  Scanf("%c", &ch);
  // ch = getche();
  Printf("\n ASCII number of charater is %c is %d", ch, ch);
   getch();
}
```

Q7 WAP to ~~read a~~ change ~~lower~~ ch small character to capital character.

Q8 Area of a triangle is given by the formula.

$$A = \sqrt{S(S-a)(S-b)(S-c)}$$

where a, b and c are sides of a triangle and $S = (a+b+c)/2$. WAP in C to compute the area of a triangle given the value of a, b, and.

Sol^n

```
# include <stdio.h>
# include <conio.h>
# include <math.h>
 void main()
{ clrscr();
 float a, b, c, area;
 Printf("\n enter the value of, a, b & c");
 scanf("%f %f %f", &a, &b, &c);
    S = (a+b+c) /2 ;
  area = sqrt( s*(s-a)*(s-b)*(s-c)) ;
  Printf("\n S=%f \n area = %f", S, area);
   getch();
}
```

Q9 WAP in C to receive an integer and find the octal equivalent.

Soln
```
# include <stdio.h>
# include <conio.h>
void main()
{
  clrscr();
  int a;
  Printf("enter the value of a");
  Scanf("%d", &a);
  Printf("%o", a);
  getch();
}
```

Q10: WAP in C to receive an integer and find the hexadecimal number.

Soln
```
Printf("\n Hexadecimal = %x", a);
```

Q11 Write a program to solve the roots of the given quadratic equation.

$ax^2 + bx + c = 0$.

Soln
```
#
#
# include <Math.h>
void main()
{
  float a, b, c, root, x1, x2;
  Printf("\n Enter the value of a, b and");
  Scanf("%f %f %f", &a, &b, &c);
  root = sqrt(b*b - 4*a*c);
  x1 = (-b + root) / 2*a;
  x2 = (-b - root) / 2*a;
  Printf("\n root = %f", root);
  Printf("\n x1 = %f", x1);
  Printf("\n x2 = %f", x2);
  getch();
}
```

Q12 WAP to convert lower case character to uppercase characters.

Soln
```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
void main()
{
  int lower, upper;
  lower = getchar();
  upper = toupper(lower); // tolower()
  putchar(upper);
  getch();
}
```

Q13 Temperature in a city in Fahrenheit degree is input through keyboard. Write a program to convert the temprecture into centigrate degree.

Soln
```
Printf("Enter the temprature in Fahrenheit");
scanf("%f", F);
c = 5*(F-32)/9;
Printf("%f", c);
```

Top-down approach :-

C programming uses top down approach to solve a problem. Top down approach start with high-level design and ends with the low-level implementation.

In top down approach, we use following approach to solve any problem.

1. First we will make a high level design of a problem statement
2. After that we will write the main function.
3. From the main function we will call the sub function.
4. Later we will implement (code) all the sub function based on requirements. That's why c is called the top down approach.

eg
```
int main()
{ int a,b;
scanf("%d%d", &a, &b);
add (a,b);
Sub (a,b);
mul (a,b);
div (a,b);
 return 0;
}
```

Stepwise refinement:-

Stepwise refinement is the idea that software is developed by moving through the levels of abstraction, beginning at higher levels and, incrementally refining the software through each level of abstraction, providing more details at each increment.

At higher levels, the software is merely its design models; at lower levels there will be some code; at the lowest level the software has been completely developed.

At the early steps of the refinement process the software engineer does not necessarily know how the software engineer does not necessarily know how the software will perform what it needs to do. This is determined at each successive refinement step, as the design and the software is elaborated upon.

Refinement can be seen as the compliment of abstraction. Abstraction is concerned with hiding lower levels of details; it moves from lower to higher levels. Refinement is the moment from higher levels of details to lower levels. Both concepts are necessary in developing software.

Storage classes :—

To fully define a variable, one needs to mention not only its 'type' but also its 'storage class'.

A variable name identifies some physical location within the computer where the string of bits representing the variable's value is stored.

There are basically two kind of location in a computer where such a value may be kept—

(i) Memory and (ii) Register.

→ A variable's storage class tell us:

(1) where the variable would be stored.

[2] What will be the initial value of the variable, if initial value is not specifically assigned.

3] What is the scop of variable. {In which function the value of the variable would be available.}

4] What is the Life of the variable. eg How long would the variable exist.

Type—

There are four storage classes in C.

1] Automatic storage class

2] Register storage class

3] static storage class

4] External storage class.

Automatic storage class: (Auto)

local variable known only to the function in which it is declared. Default is auto.

Storag :-	Memory
Default initial value:-	Garbage value
Scop :-	Local to the block in which the variable is defined.
Life :-	Till the control remains within the block, in which the variable is defined.

eg 1

```
# include <std io.h>
# include <con io.h>
int main()
{ auto int i, j;
  printf("%d", );
  printf(" %d %d ", i, j);
    return 0;
  getch();
}
```

eg 2

```
#
#
void main()
{
  auto int i = 1;
   {
      auto int i = 2;
      {
        auto int j=3;
         printf("%d", i);
      }
```

```
        Printf("%d", i);
    }
    Printf("%d\n", i);
    getch();
}
```

Register storage Class :- (register)

Local variable which is stored in the register.

storage — CPU Registers

Default initial value — Garbage value.

Scope — Local block in which the variable is defined.

Life — Till the control remains withing the block in which the variable is defined.

eg

```
void main()
{ register int i;
  for (i=1; i<=10; i++)
      Printf("%d\n", i);
  getch();
}
```

Static storage class :- (Static)

Local variable which exists and retains its value even after the control is transferred to the calling function.

storage — Memory

Default initial value - Zero.

Scop - Local to the block in which the variable is defined.

Life - Value of the variable persists btw different function calls.

```
#
#
void increment();
int main()
{
  increment();
  increment();
  increment();
   return 0;
}
void increment()
{ auto int i=1;
  Printf("%d\n",i);
    i = i+1;
}
```

o/p

1
1
1

```
#
void increment();
int main()
{
  increment();
  increment();
  increment();
    return 0;
}
void increment()
{
  static int i=1;
   Printf("%d\n",i);
     i = i+1;
}
```

o/p

1
2
3

External storage class :- (extern)

Global variable Known to all function in the life

Storage — Memory.

Default initial value — Zero

Scop — Global

Life — As long as the program's execution doesn't come to an end.

```
#include<stdio.h>
#include<conio.h>
int i;
void increment();
 void decrement();
int main()
{
```

```
Printf ("\n i= %d", i);
        increment();
        increment();
        decrement();
        decrement();
        return 0;
    }
void increment()
{   i = i+1;
    Printf ("on incrementing i= %d \n", i);
}
 void decrement()
    {   i = i-1;
        Printf ("on decrementing i = %d \n", i);
    }
```

Operators & Expression

C Language offers many type of operators -

1. Arithmetic Operators
2. Assignment Operators
3. Relational operators
4. Logical operators
5. Bit wise operators
6. Conditional Operators (ternary operators)
7. Increment / decrement operators
8. Special operators.

1. Arithmetic Operators in C! -

C Arithmetic operators are used to perform mathematical calculation like addition, subtraction, multiplication divison and modulus in C program.

Arithmetic Operators	Operations	Example
+	Addition	A+B
-	Subtraction	A-B
*	multiplication	A*B
/	Division	A/B
%	Modulus	A%B

Example! - To illustrates the basic arithmetic expression.

Soln #

```
void main()
{ clrscr();
  int x, y, sum, sub, mult, div, remain;
  Printf(" Enter the value of x and y");
  scanf("%d%d", &x, &y);
```

```
sum = x+y;
sub = x-y
mul = x*y
div = x/y
remain = x%y;
Printf("\n Sum =%d \n Subtraction = %d \n Multiplication
    =%d \n Division =%d \n Remainder =%d",
    sum, sub, mul, div, remain);
getch();
}
```

Assignment operators in C:

In programs, values for the variables are assigned using assignment operators.

For example If the value 10 is to be assigned for the variable "sum". It can be assigned as sum = 10;

Operators		Example	Explanation
Simple assignment operator	=	sum = 10	10 is assigned to variable sum.
Compound assignment operators	+=	sum += 10	sum = sum+10
	-=	sum -= 10	sum = sum-10
	*=	sum *= 10	sum = sum*10
	/=	sum /= 10	sum = sum/10
	%=	sum %= 10	sum = sum%10
	&=	sum &= 10	sum = sum&10
	^=	sum ^= 10	sum = sum^10

Relational Operators in C

Relational operators are used to find the relation between two variable.

eg to compare the values of two variable in C program

Operators	Example	Description
>	x>y	x is greater than
<	x<y	x is less than y
>=	x>=y	x is greater than or equal to y.
<=	x<=y	x is less than or equal to y.
==	x==y	x is equal to y.
!=	x!=y	x is not equal to y.

eg

```
void main()
{
int m=40, n=20
if (m==n)
 {
 Printf("m and n are equal");
 }
 else
 {
 Printf("m and n are no equal");
 }
 getche();
}
```

Logical operators in C :-

These operations are used to perform logical operation on the given expressions.

	Operators	Name	Example	Description
(1)	&&	logical AND	(x>5)&&(y<5)	It returns true when both condition are true.

(2)	||	logical OR	(x>=10)||(y>=10)	It returns true when at-least one of the condition is true.
(3)	!	logical NOT	!(x>5)&&(y<5)	It reverses the state of the operand "((x>5)&&(y<5))" If "((x>5)&&(y<5))" is true logical NOT operato makes it false.

Example Write a Program to illustrate the logical ANDing & and ORing operations.

Soln

```
#
#
Void main()
{ clrscr();
  int x, y, z;
    x = y;
    y = 3;
    z = x && y;
    y = x || y || z;
    x = x && y || z;
  Printf("x = %d \n y = %d \n z = %d", x, y, z);
  getch();
}
```

O/P

```
1
1
1
```

Bit Wise Operators in C:-

These operators are used to perform bit operations. Decimal values are converted into binary values which are the sequence of bit and bit wise operators work on these bit.

Operator - Symbol	Operator - name
&	Bitwise - AND
|	Bitwise OR
~	Bitwise N
^	XOR
>>	~~Left shift~~ Right shift
<<	Left shift.

Truth Table

x	y	x | y	x & y	x ^ y	~x
0	0	0	0	0	1
0	1	1	0	1	1
1	0	1	0	1	0
1	1	1	1	0	0

eg Consider $x = 40$, $y = 80$

$x = 00101000$

$y = 01010000$

$x \& y = 00000000 = 0$ (decimal)

$x | y = 01111000 = 120$ (decimal)

$x \wedge y = 00010100 = 20$ (decimal)

$x << 1 = 01010000 = 80$ decimal

$x >> 00010100 = 20$ (decimal)

eg!
```
void main()
{
 clrscr();
 usigned int x = 128, y = 32;
 x = x>>1;
 Printf (" After right-shifting by 1 x= %d ", x);
  y = y<<1;
 Printf(" After Left shifting by 1 y = %d", y);
 getch()
}
```

Conditional OR Ternary Operator!-

Conditional operators return one value If condition is true and returns another value If condition is false. The type of operator is also called as ternary operator.

Syntax!- (condition ? true_value : false_value);

Example- (A>100 ? 0 : 1)

If A is greater than 100, 0 is return else 1 is returned. This is equal to if else condition statements.

eg
```
#
#
void main()
{
 int x=1, y;
 y = (x==1 ? 2 : 0);
 Printf( " x value is %d \n", x);
 Printf ( " y value is %d", y);
 getch()
}
```

o/p

x value is 1
y value is 2.

Increment / decrement Operators :-

Increment operators are use to increase the value of the variable by one and decrement operators are used to decrease the value of the variable by one in C programs.

Increment operator : ++var_name ; (or) var_name++ ;

Decrement operator : --i ; i-- ;

Example of Increment operator :-

```
#
#=
void main()
{
  clrscr();
  int a,b;
  Printf(" Enter the value of a and b");
  Scanf("%d %d", &a, &b);
  Printf(" a = %d\n", a++);
  Printf("\n b = %d", ++b);
  getch();
}
```

O/P

Enter the value a & b

3, 6

a = 3

b = 7

++int_var → indicate pre increment. The value of int_var must be incremented before it is being used.

int_var++ → indicate post increment. It use the value of int_var first and then increment.

eg a = ++b

This is equivalent to two assignment statement.

b = b++ ; and then

a = b ;

where as

a = b++ ;

This is equivalent to

a = b ;

and

a = b++ ;

a = b ++

Example of Decrement Operator :-

```
#
#
void main()
{ clrscr();
  int a,b;
  Printf("Enter the value of a and b");
  Scanf("%d %d", &a, &b);
  Printf("\n a=%d", a--);
  Printf("\n a=%d", --b);
  getch()
}
```

o/p

Input 6, 9

a = 6

b = 8

** --a ⇒ decrement the value of int_var first and then use it.

a-- ⇒ use the value of int_var and then decrement it

Special operators in c!-

Operators	Description
&	This is used to get the address of the variable. Example:- &a will give address of a.
*	This is used as pointer to a variable. Example- *a where * is pointer to a variable.
sizeof()	This gives the size of the variable. Example- sizeof(char) will give us 1.

Example Program for & and * Operator in c;-

```
#
#
void main()
{
  int *ptr, q;
    q = 50;
  ptr = &q ;    //Address of q is assigned to ptr
  Printf ("%d", *ptr); //display q's value using ptr
  getch();                variable.
}
```

Example of sizeof() operator in c!-

```
#
#
void main()
{
  int a;
  char b;
```

```
float c;
double d;
Printf("\n Storage size for int data type %d", sizeof(a));
Printf("\n Storage size for char datatype %d", sizeof(b));
Printf("\n Storage size for float datatype %d", sizeof(c));
Printf("\n Storage size for double datatype %d", sizeof(d
getche)
}
```

C Operator Precedence and Associativity

C Operators in order of precedence (highest to lowest). Their associativity indicates what order operators of equal precedence in an expression are applied.

Operator	Description	Associativity
()	Parentheses (Function call)	Left-to-right
[]	Brakets (array subscript)	
.	Member selection via object name	
->	Member Selection via Pointer	
++ --	Postfix increment/decrement	
++ --	Pre increment / dicrement	Right-to-left
+ -	Unary plus/minus	
! ~	logical negation / bitwise complement	
(type)	Cast (convert value to temporary value of type)	
*	Indirection or dereference operator	
&	Address (of operand)	
Sizeof	Determine size in bytes on this implementation	
* / %	Multiplication / division / modulus	Left-to-Right
~~+ -~~	~~Unary plus/minus~~	
+ -	Addition / Subtraction	
<< >>	Bitwise shift Left, Bitwise shift right	
< <=	Relational less than / less than or equal to	
> >=	Relational greater than / greater than or equal to	
== !=	Relational is equal to / is not equal to	
&	Bitwise AND	
^	Bitwise exclusive OR	
|	Bitwise inclusive OR	

&&	Logical AND	
||	Logical OR	
?:	Ternary conditional	
=	Assignment	Right-to-Left
+= -=	Addition / Subtraction assignment	
*= /=	Multiplication / division assignment	
%= &=	Modulus / bitwise AND assignment	
^= |=	Bitwise exclusive / inclusive OR assignment	
<<= >>=	Bitwise shift Left / right assignment	
,	Comma (separate expression)	Left-to-right

Mixed Mode Operation :-

Expressions that involves elements of type real and integer are called mixed mode operation on the expression. If an expression containing both integer and floating point elements, is evaluated then result will be of type float.

If this value is to be assigned to some identifier, then we should know the data type of that identifier. If the identifier is an int the and the resulting value to be assigned is a float, then the floats franctional value must be truncated. The operators that are used in forming mixed mode expressions are basic arithmetic operators.

Example

```
#include <stdio.h>
#include <conio.h>
void main()
{
  clrscr();
  int a,b;
  float x,y;
  printf("Enter the value of a,b,x and y \n");
  scanf("%d %d %f %f", &a, &b, &x, &y);
    a = (x/y) + (a/b)
    b = (a*x) / y;
    x = (a-b)
  printf("a = %d \n b = %d \n x = %f ", a, b, x);
    getch();
}
```

O/P

4, 2, 2 3 F/P

a = 2
b = 2
x = 2.0

Data Type Conversion or Type casting

In some application, we may often want to change the data type of the variable, when we declare some variable as int, the desired output may be float or vice-versa. In such situations, we change the nature of the data stored in the variable. This process is known as data type conversion, it also called type casting.

The general form of type casting is

```
(datatype)variable
```

where data type → Any basic data type and must be written within the parentheses.

Example:-

```
int K = 2;
float term;
    term = 1/K;
```

Here, the content of term will be

term = 1/K
= 1/2
= 0 (Result of integer division)

it should be = term = 1/(float) K;

Associativity :-

Associativity is used to determine the orders in which operator with the same precedence are evaluated in a complex expression

Ex. Type -

(i) Left to Right
(ii) Right to Left

Associativity is applied when we have more than one operator of the same precedence level in an expression.

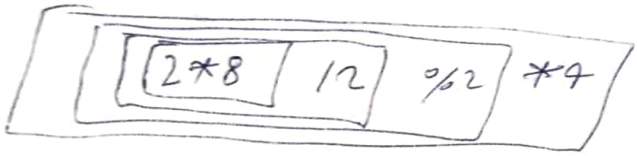
Left to Right Associativity :-

Left to right associativity means that the left operand must be unambiguous.

Unambiguous means it must not be involved in evaluation of any other sub-expression.

eg ((2*8)/2) %2)*4

A graphical representation of this expression is shown in below -



Right to Left Associativity :-

Right to left associativity the right operand must be unambiguous. Several operators have Right to Left associativity as shown in the precedence table.

Decision making and Branching

C Language possesses such decision-making capabilities by supporting the following statements.

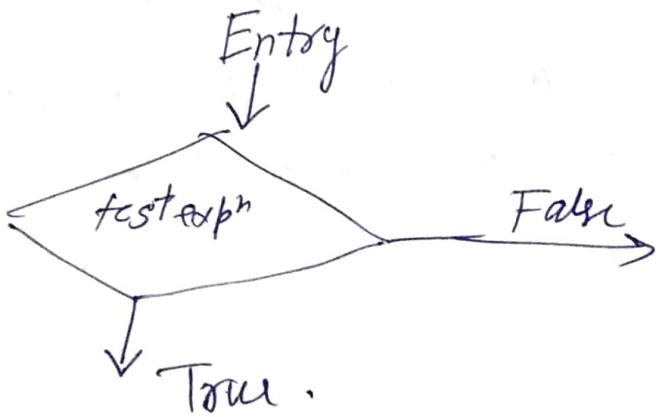
1. If statement
2. switch statement
3. conditional operator statement
4. goto statement.

If statement :

It is basically a two-way decision statement and is used in conjunction with an expression. It takes the following form.

If (test expression)

It allows the computer to evaluate the expression first & and then, depending on whether the value of the expression is true or false. It transfers the control to a particular statement. This point of program has two paths to follow on for the true condition and other for the false condition.



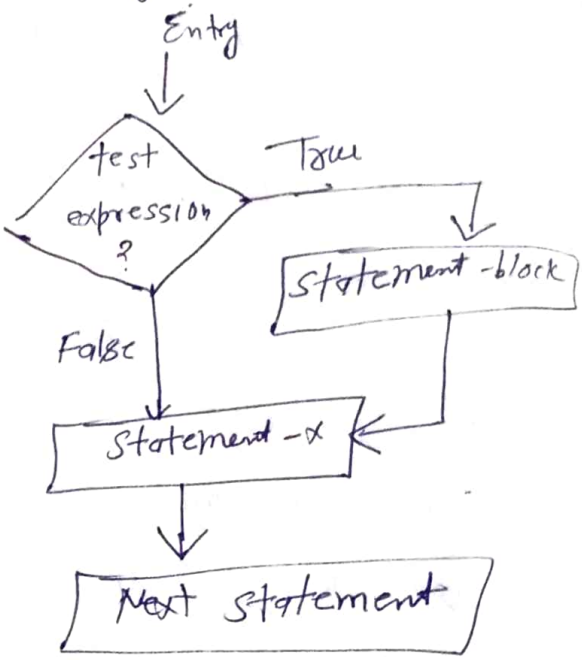
Type of If statement :-

1. Simple If statement
2. If ... else statement
3. Nested if ... else statement
4. else if ladder.

1. Simple If statement :-

The general form of a simple If statement is

```
If (test expression)
{
    statement-block;
}
statement-x;
```



Flowchart of Simple If control

The statement-block may be a single statement or a group of statements. If the test expression is true, the statement-block will be executed; otherwise the statement-block will be skipped and execution will jump to the statement-x.

Example Two integer numbers are input through the keyboard. Write a program find out the greater number.

Soln
```
#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    int x, y;
    Printf("Enter the value of x and y \n");
    Scanf("%d %d", &x, &y);
    If (x>y)
    { Printf("x is greater \n");
    }
    If (y>y)
    {
        Printf("y is greater \n");
    }
    getch();
}
```

Exmp2 One integer number is input through the keyboard. Write a program to find out the number is even or odd.

```
#
#
Void main()
{ int a;
 Printf("Enter the integer number");
  scanf("%d", &a);
  If (a%2 ==0)
      Printf("even number");
  If (a%2 != 0)
     Printf("odd number");
} getch();
```

2. If... Else statement :-

The If... else statement is an extension of the Simple If statement. The general form is

```
If (test expression)
  {
    True-block statement(s)
  }
else
  {
    False-block statement(s)
  }
statement-x
```

If the test expression is true, then the true-block statement(s) are executed. In other case, either true-block or false-block will be executed, not both.

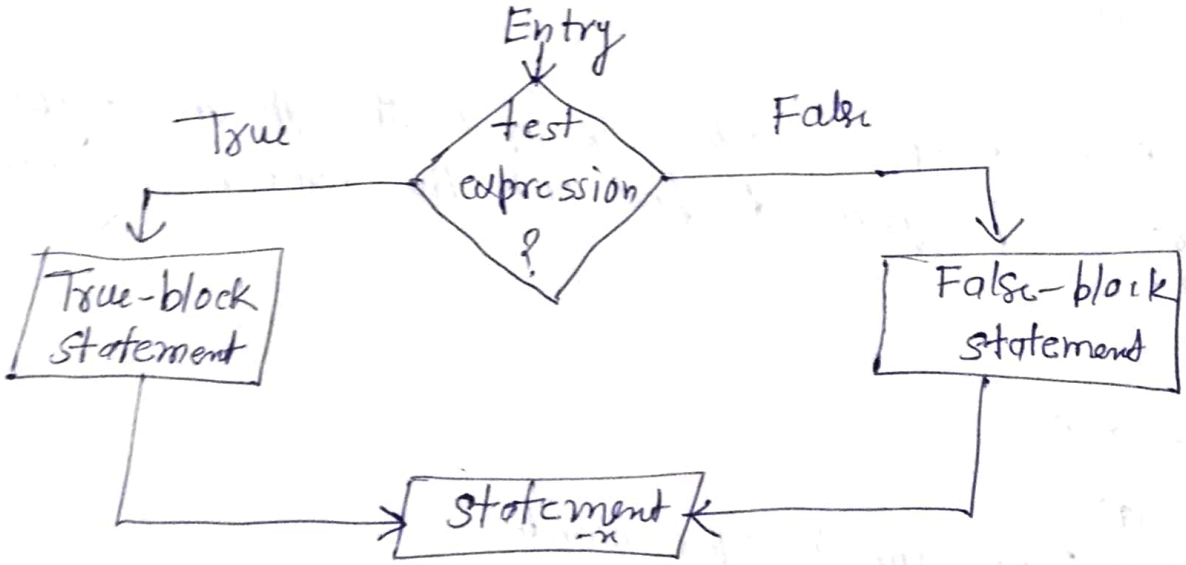


Fig Flowchart of If....else Control

Example:- Any year is input through the keyboard.
write a program to find out year is leap year or not.

Solution:-
```
#include <stdio.h>
#include <conio.h>
void main()
{
 int a;
 Printf(" Enter the year \n");
 Scanf("%d", &a);
 If ( a%4==0)
   {
     Printf (" year is a leap year");
   }
   else
    {
       Printf("year is not leap year");
    }
    getch();
}
```

Example:- Two inter numbers are input through the keyboard, write a program to find out the greatest number.

Solution:-
```
##
##
Void main()
{ Int a,b;
  Printf (" Enter the value of a and b\n");
  Scanf("%d %d", &a, &b);
  If ( a>b)
    {  Printf (" a is greater ");
    }
  else
      {
          Printf (" b is greater");
      }
  getche();
}
```

Nested- If...else statement:-

When a series of decision are involved, we may have to use more the one "If else statement in nested form as shown below.

If the condition-1 is false, the statement-3 will be executed, otherwise it continues to perform the second test. If the condition-2 is true, the statement-1 will be evaluated, otherwise the statement-2 will be evaluated and then the control is transferred to the statement-x.

```
If ( test condition-1)
  {
    If (test condition-2)
      {
        statement-1 ;
      }
      else
       {
         statement-2 ;
       }
  }
else
   {
     statement-3 ;
   }
statement-x
```

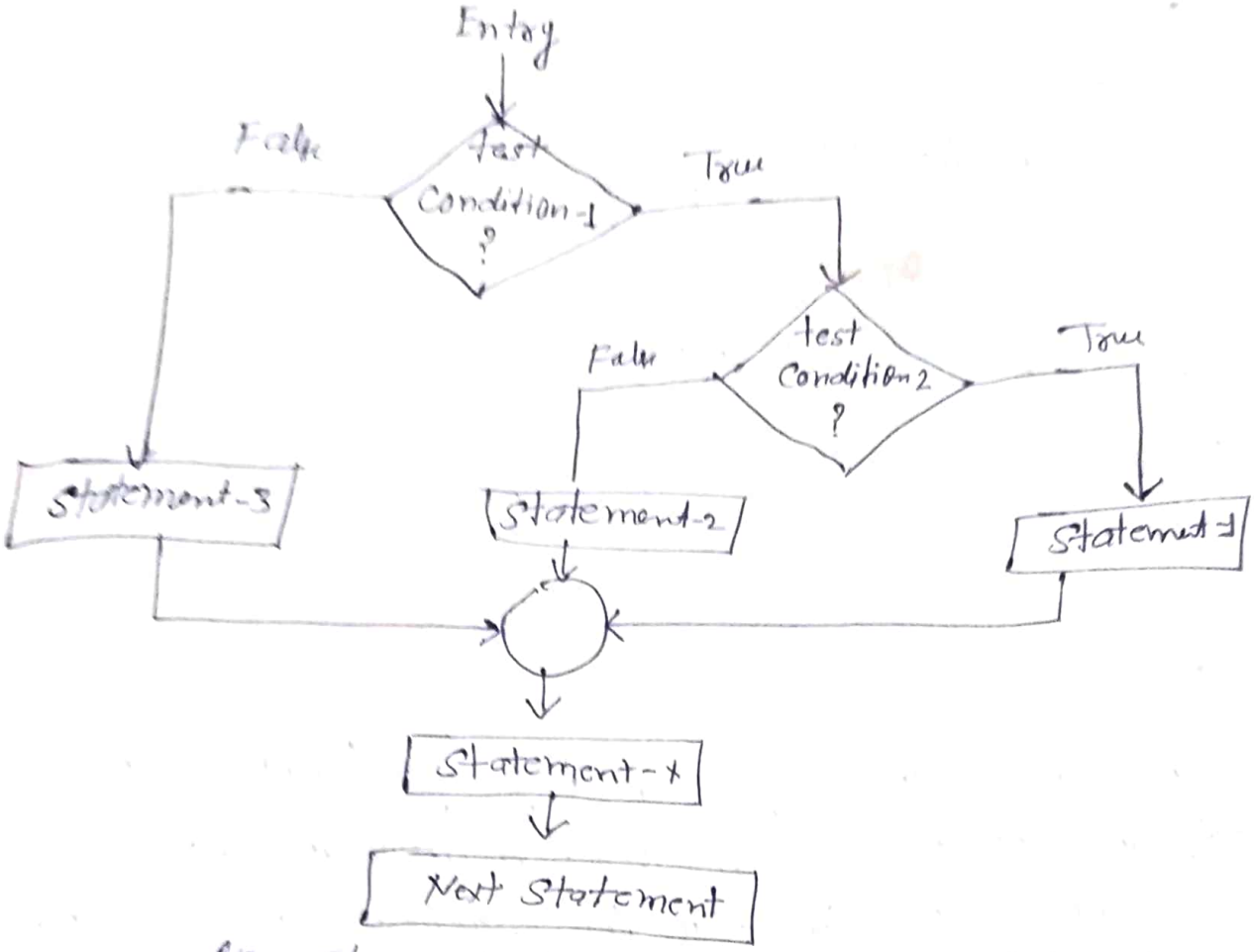


Fig Flow-Chart of nested If--else statement.

Question W.A.P in C to print the largest of the three numbers using nested If else statement.

Solution:- #1

```
#
void main()
{
  float a, b, c;
  Printf(" Enter three values a, b & c \n");
  Scanf(" %f %f %f", &a, &b, &c);
  If ( a > b)
    {
      If ( a > c)
        {
          Printf (" %f \n", a);
        }
      else
        {
          Printf (" %f \n", c);
        }
    else
```

```
{
    If (c>b)
        Printf ("%f", c);
    else
        Printf ("%f in", b);
    }
} getch();
```

Else If Ladder! -

There is another way of putting Ifs together when multipath decisions are involved. A multipath decision is a chain of ifs in which the statement associated with each else is an If. It takes the following general form —

```
If ( condition 1 ) :
    statement-1;
elseif ( condition 2 )
    statement 2;
elseif ( condition -3)
    statement -3 ;
elseif ( condition n )
    statement n ;
else
default - statement ;
statement-x;
```

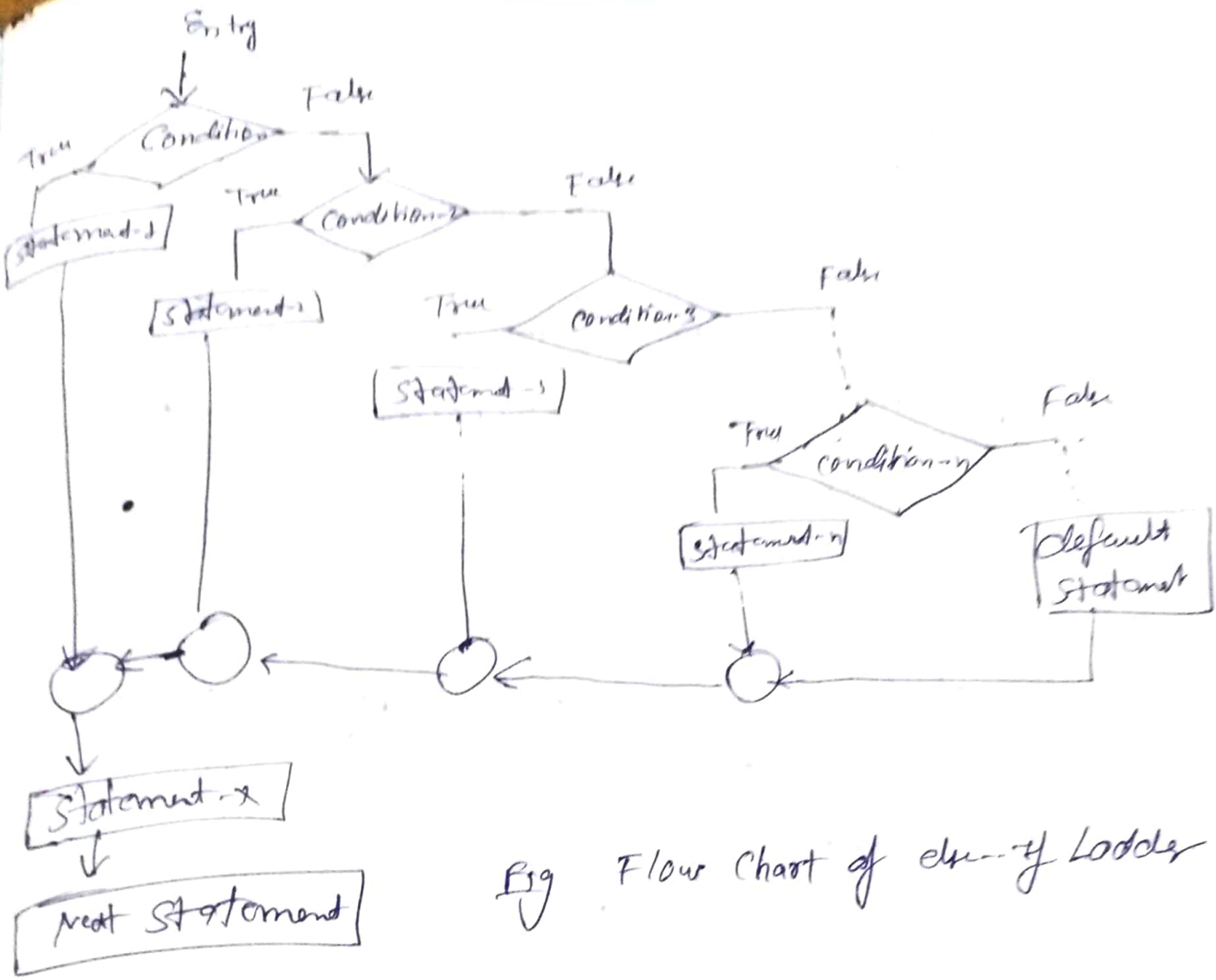


Fig Flow Chart of else-if Ladder

Example:- The program reads the customer number and power consumed and prints the amount to be paid by the customer.

Solution:-

```
main()
{
  Int units, custnum;
  float charges;
  Printf("Enter customer No. and units consumed \n");
 Scanf("%d %d", &custnum, &units);
  If (units <= 200)
     charges = 0.5 * units;
  else if (units <= 400)
```

```
        Charges = 100 + 0.65 * (Units - 200);
    else if (Units <= 600)
        Charges = 230 + 0.8 * (Units - 400);
    else
        Charges = 390 + (Units - 600);
    Printf("\n Customer No: %d: Charges = %2f \n",
            Custhum, charges);
}
```

Program An electric power distribution company charges its domestic consumers as follow—

Consumption Units	Rate of charges
0 - 200	Rs 0.5 Per Unit
201 - 400	Rs 100 plus Rs 0.65 per Unit excess of 200
401 - 600	Rs 230 plus Rs. 0.80 per Unit excess of 400
601 and above	Rs 390 plus Rs 1.00 per Unit excess of 600

Read the customer number and power consumed and prints the amount to be paid by the customer.

Example 1 WAP in C to Find largest from three numbers given by User to explain working of if-else-if statement or ladder.

```
#
void main()
{
    int a, b, c;
```

```
Printf("Enter three numbers ~~given~~ \n");
Scanf("%d%d%d", &a, &b, &c);
  If (a>b && a>c)
    {
      Printf("largest = %d", a);
    }
    else if (b>a && b>c)
      {
        Printf("Largest = %d", b);
      }
    else
      {
        Printf("Largest = %d", c);
      }
    getch();
  }
```

Example 2:- C Program to print weekday based on given number.

Solution:-

```
#
#
void main()
  {
    int day;
    Printf("Enter day number:");
    Scanf("%d", &day);
  If (day == 1)
    { Printf("Sunday.");
  }
  else if (day = 2)
      {
        Printf("Monday.");
      }
```

```
else if (day == 3)
  {  Printf (" Tuesday.");
  }
else if (day == 4)
    {
      Printf (" Wednesday.");
    }
else if (day == 5)
    {
      Printf (" Thursday.");
    }
else if ( day == 7)
    {
      Printf (" Invalid day.");
    }
  getche();
getche();
}
```

Example: C Program to Find largest number from four given numbers.

```
#
#
void main()
{  float a, b, c, d, lg;
   Printf (" Enter four numbers : \n");
   Scanf (" %f %f %f %f ", &a, &b, &c, &d);
   If (a > b && a > b && a > d)
     {   lg = a;
     }
   else if (b > a && b && b > d)
        {  lg = b;
        }
```

```
else if (c>a && c>b && c>d)
    {
      lg = c;
    }
  else
    {
      lg = d;
    }
printf(" Largest = %f", lg);
  getch();
}
```

Switch() Statement :-

Switch statement provides means for checking a given expression and selecting one course of action among various action available.

Switch statement ~~is~~ comprises of two sections.

1. This section is identified by switch keyword and contain the expression that is to be evaluated
2. This section ~~is~~ comprises of multiple case statement, which clearly associate all possible outcomes of the expression and the action to be formed in each cases.

A Switch statement is used when there is a choice as to which code to execute, depending on the value of a constant expression. Different cases are presented, and are checked one by one to see if one case matches the value of the constant expression.

If a case matches, that block of code is executed.
If none of the case match the default code is executed.

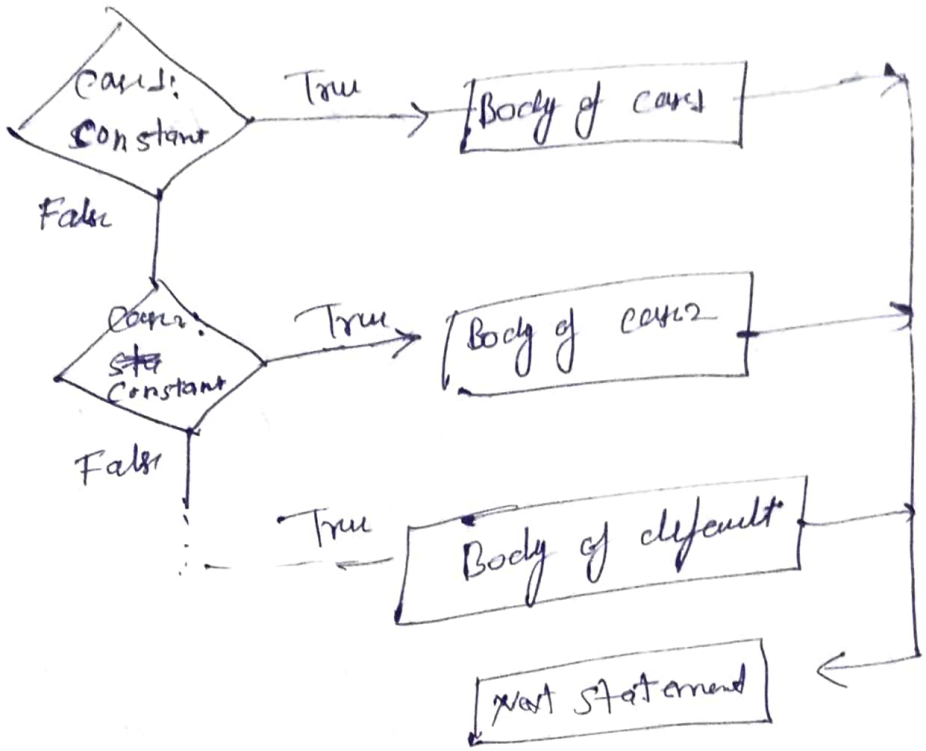
The switch case statement is a multiple branch selection statement, which successively test the value of an expression against a list of integer constant. A switch statement allow you to choose a block of statements among several alternative.

The general syntax of the switch statement is —

```
switch (expression)
{
   case1 :
   statement1;
    break;
  case2:
   statement2;
     break;
  case3:
    statement3;
      break;
  .  .  .  .  .  .
  .  .  .  .  .  .
  default :
   statement n;
}
```

The break statement when encountered at the end of the each statement ~~causes~~ causes the control of execution to terminate and transfer the control to next statement following the closing curly brace of the switch statement,

If there is no break at the end of a particular sequence of statement for a particular case label then the statement belong to the next case get execution



The following rules should be used for switch statement —

(i) The expression value must be an integer, hence the type can be int or char.

(ii) Case should always be followed by an integer constant, character constant or constant expn.

(iii) All cases should be distinct.

(iv) The block of statement under default is executed when none of the case match the value of the expression.

(v) Default can optionally be present.

(vi) case and default can occur in any order.

(viii) The break statement causes an explicit exit from the switch statement. If it is not present after executing the case which match the value of expression, the following other cases are executed, as C treats the cases just a labels.

The Break statement :-

It causes an exit from the switch body. Control goes to the first statement following the end of the switch statement. If the break statement is not used, the control passes to the next case constant and the remaing statement in the switch statement in the switch construct will also be executed.

Example— Write a program to find out the number of days in a months.

Solution :-

```
#
#
Void main()
{
  clrscr()
  int a;
  Printf("enter the number");
  Scanf("%d", &a);
  Switch(a)
  {
    case 1:
    Printf("January days 31");
      break;
  case 2:
      Printf("Febrüary deg = 28 or 29");
        break;
```

```
        default :
          Printf (" wrong choice ") ;
          break ;
      }
  } getch() ;
```

Example-2 Write a program to find out the week day using Switch statement.

Soln

```
#
#
Void main()
{
  clrscr() ;
  int n ;
  Printf (" Enter any number of day ") ;
  Scanf ("%d", &n) ;
  switch (n)
  {
     case 1:
      Printf (" In Sunday ") ;
       break;
     Case 2:
      Printf (" In Monday ") ;
       break :
     Case 3 :
       Printf (" In Tuesday ") ;
        break ;
     Case 4 :
        Printf (" In Wednesday ") ;
        break ;
```

```
case 5:
  Printf (" In Thursday ");
    break;
case 6:
  Printf (" In Friday ");
     break;
case 7:
  Printf (" In Saturday ");
     break;
default:
  Printf (" value is not valid ");
}
  getch();
}
```

Programming Loop and Iteration

Many time we require that a group of instructions are to be executed until some Logical conditions are satisfied. It is known as Looping.

A Loop in a program essentially consist of two part :-

One is called the body of the Loop and second is known as the control statement. The control statement perform a Logical a Logical test whose result is either a true or false. If the result of this Logical test is true then the statement contain in the body of the Loop. Otherwise Loop is terminated.

The control statement can be place either before or after the body of the Loop. If the control statement is placed before the body of the Loop, it is called the Entry-Controlled Loop. If the control statement is writted after the body of the Loop, it is called the Exit-Control Loop. C support both type of Loop statement.

Generally, a Loop has three part that have different purpose —

→ The first one is initialization expression which is declare before entering into the Loop.

→ Secondly, test expression is an expression whose Mrouth value decides whether the Loop body will be executed or not.

→ Third is the updates expression which changes the value of the control variable and the last part is the body of Loop which contains the statements that are executed repeatedly.

C provide three type of Loop control structure—

① The while Loop
② The do-while Loop
③ The for Loop

The while Loop! -

while Loop in c starts ~~whic~~ with the reserved word while, followed by a relational expression in parenthesis. This is followed by ~~the~~ a single statement or a block of statement enclosed ~~with it get executed~~ in braces to define the begining and the end of the block.

Following point, need to be taken care for proper execution of while Loop—

1. The expression should be so framed that its value should change in each iteration of loop.
2. If the expression is ill-framed and its value doesn't change within the loop, Loop will never terminate. The Loop which do not terminate are called infinite Loop.
3. Initially if the expression is found false, Loop will not execute even once.

~~The while~~

This is used to execute a set of statements repeatedly as long as specified condition is true.

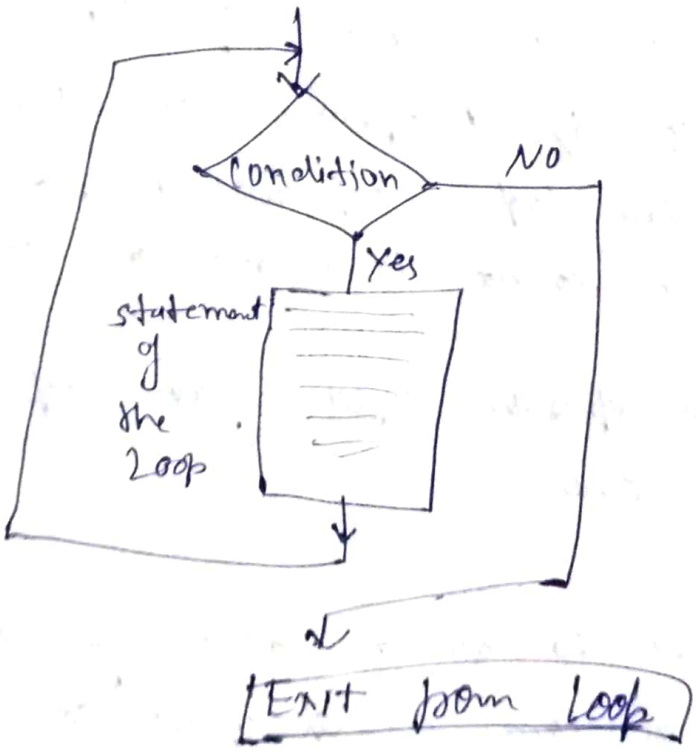
The syntax of the while Loop can be expressed as—

```
while (exp)
  {
    statement;
  }
```

where exp = Logical expression that results in either true or false.

statement:- may be simple or compound statement.

When the compiler enter a while block, it first tests, if the condition is true. If it is, then all the statement in the while block are executed. At the end of the block, the compiler come back to the begning of the while block and re-executes the statement in the while-block, If the while condition is still true. The compiler exists the while block, if the while condition is not true.



Q WAP in C to display the result when an integer number is input through the keyboard. Find the table of the ith integer number.

Soln
```
# include<conio.h>
# include<stdio.h>
{
   clrscr() ;
   int  n, i, m ;
    Printf (" Enter integer number") ;
    scanf (" %d", &n) ;
    i = 1;
    while ( i <= 10)
    {
       m =  m*i ;
       Printf (" \n %d * %d = %d" , n, i, m ) ;
         i++ ;
    }
      getch() ;
}
```

Q2 Write a program to evaluate the factorial of a integer number.

Soln
```
void main()
{
   int   n, i ;
   Long int fact = 1 ;
   Printf (" Enter inter number") ;
   scanf ("%d ", &n) ;
   i = 1 ;
   if ( n > 1)
     while ( i <= n )
       {   fact = fact * i ;
           i++ ;
```

```
    printf("\n factorial value = %ld", fact);
    getch();
}
```

Q3 Write a program to find the area of a circle.

Sol^n

```
Void main()
{
    float radius, pie = 3.14, area;
    printf(" enter the radius of a circle");
    scanf("%f", &radius);
    while (radius != 0.0)
    {
        if (radius < 0.0)
            printf("\n Area is not possible");
        else
            area = pie * radius * radius;
            printf("\n Area of a circle = %f", area);
    }
    getch();
}
```

Q4 WAP to Find Out the sum of First n integer numbers.

Sol^n

```
Void main()
{
    int n, i, sum = 0;
    printf(" Enter the value of n");
    scanf(" %d", &n);
    i = 1;
    while (i <= n)
    {
        sum = sum + i;
        i++;
    }
```

```
        Printf (" \n Sum = %d", Sum) ;
        getch();
    }
```

Q5 Write a program to print even number starting from 1 to 20 and their squares.

Sol^n

```
void main()
{
    int start=2, limit = 20, Sq;
    while ( start <= limit)
    {
        Printf (" \n Number = %d", start);
        Sq = start * start;
        Printf (" \n Square = %d", Sq);
        start = start +2;
    }
    getch();
}
```

The do-while Loop :-

This is used to execute a set of statement repeatedly, until the logical test result is false. This is called the past test Loop. Because, the Loop for repetition is made at the end of each pass.

The do while Loop is another repetitive Loop used in a program. Whenever one is certain about a ~~set~~ test condition, the do-while Loop can be used. ~~at~~ as it enters into the loop at least once and then ~~the~~ checks whether the given condition is true or false.

The structure of the do-while Loop in C is given below —

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

Syntax of do-while Loop —

Where — do and while → are keywords.

statement → may be simple or compound statement

condition → Logical expression results in true or false.

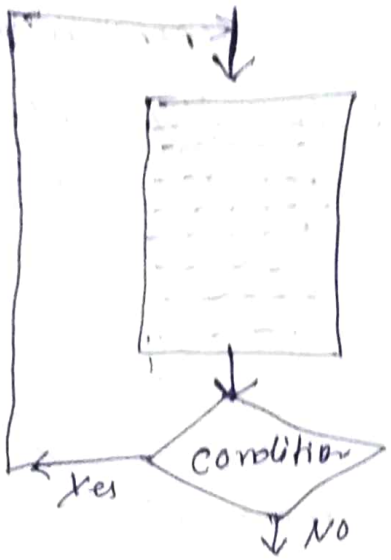
To Remembered —

(i) It is executed at least once.

(ii) It is executed till the condition remains true and the control comes out of the loop when the condition becomes false.

(iii) There must be some Loop terminating condition inside the body of the Loop to avoid infinite Looping.

After the first pass, the statement is repeatedly executed value is true. Otherwise, the control comes out of the do-while and continues with the next executable statement.



Q. WAP to calculate the factorial of an integer no.

Soln

```
void main()
{
   long int fact = 1;
   int n, i;
   Printf("Enter the no");
   Scanf("%d", &n);
     i = 1;
   if (n>1)
   do
   {
      fact = fact * i;
        i++;
   }
while (i<=n);
Printf("\n Factorial value = %ld", fact);
  getch();
}
```

Q2 WAP to display the result when an integer number is input through the Kyjboard. Find out the table of an integer no. (Using do-while loop)

Soln

```
#
#
Void maine)
{
int   i, n, m ;
Printf(" Enter the number");
Scanf (" %d", &n);
i= 1;
do
{
  m = n * i ;
  Printf (" %d %d = %d", n, i, m) ;
     itt ;
}
while (i<=10);
  getche) ;
}
```

Q3 Write a program when a number is input through the Keyboard and find the sum of digits of the number.

Soln

```
Void maine)
{
  Int num, n, r, sum ;
  Printf(" Enter the number for finding the sum of
  its digit ");
  Scanf(" %d", &num) ;
    Sam = 0 ;
    n = num ;
```

```
do
{
    r = n%10;
    sum = sum+r;
    n = n/10;
}
while (n!=0)
Printf("\n Sum of digit of %d is %d", num, sum);
getch();
}
```

The for-Loop:-

The while and do-while-Loops are used when the number of iteration is not known. The for-loop is used when the number of iteration is known in advance.

For Loop consist of three expressions. The first expression is used to initialize the index value, the second is used to check whether or not the loop is to be continued again and the third is to change the index value for further alteration.

The for Loop in C is more powerful when compared to other Loops.

The structure of the for_loop ~~in its simple form~~

```
for ( exp1 ; condition ; exp2)
```

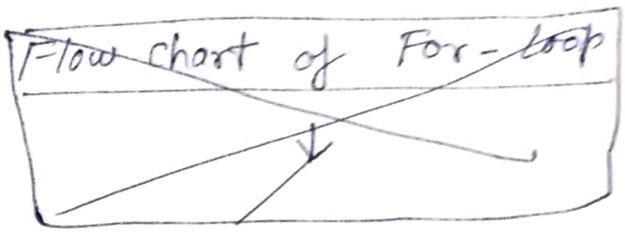
where

1. The for-Loop start with the reserved word for
2. exp1 - an assignment expression which initializes Loop index.

2. Condition → the condition which decides either to re-enter or exit form the Loop.

3. exp2 → an assignment expression which can change the loop index.

4. Semicolons!- exp1, condition and exp2 must be separated by semicolons.

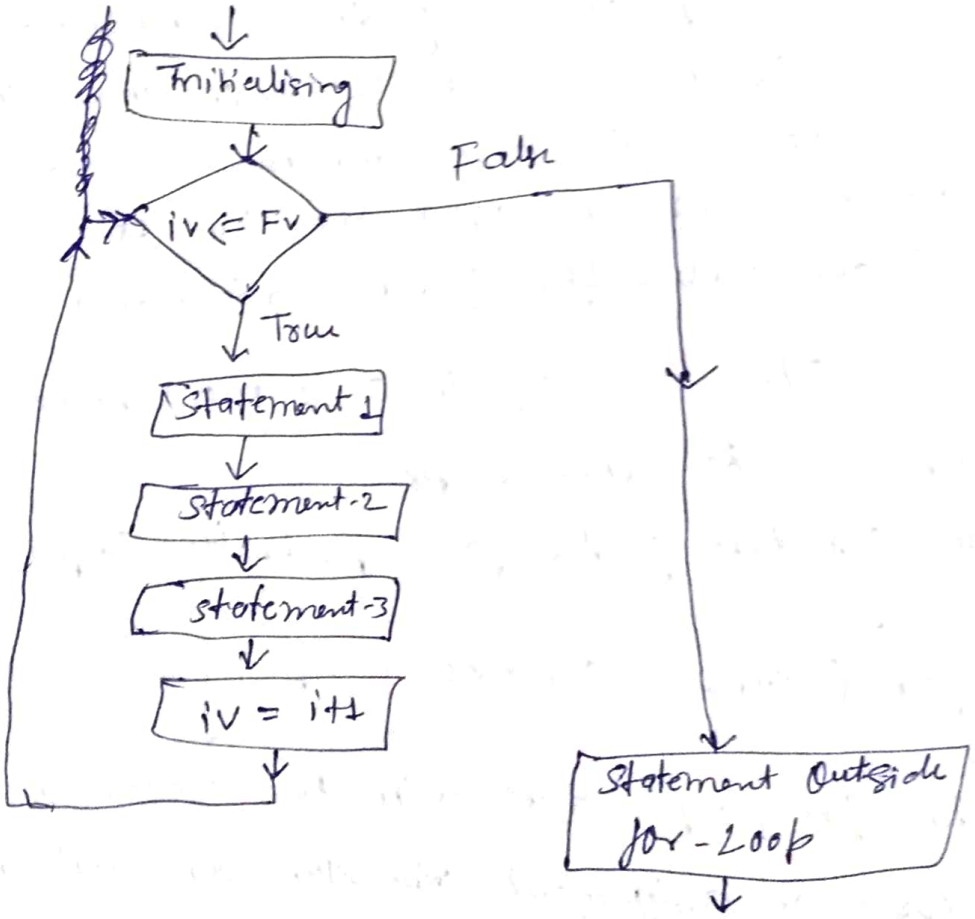


$$\text{Number of iteration} = \frac{Fv - iv + si}{si}$$

where Fv = Final value

iv = initial value

si = step increment



→ Initial Expression!- It is executed only once when the Loop first starts.

→ Test Expression!- It involves relational operators. It is executed every time through the Loop before the body of the Loop executed. If the condition is true, the body of the Loop executed If false, the control comes out of the loop.

→ Increment/Decrement!- It is always executed at the end of the Loop, after the body of the Loop.

Example!- Write a program to find out the factorial value of given number.

Soln

```
#
#
void main()
{
  int i,n;
  Long int fact;
  Printf("enter the number");
  Scanf("%d", &n);
  if (n<0)
    { Printf("Factorial no defined");
    }
  else
  {
    fact = 1;
    for (i=1; i<=n; i++)
    { fact = fact * i;}
  } Printf("\n Factorial value = %d", fact);
  }
}getch();
```

Exapl-2 Write a program to find out the sum of the first 50 integer values.

Soln

```
#
#
void main()
{
  int i, sum = 0;
 for ( i = 1 ; i <=50 ; i++)
   {
      sum = sum + i ;
   }
  Printf (" \n  sum = %d", sum) ;
    getch();
}
```

Example-3 Write a program to find out the sum of n natural number.

Soln

```
#
#
void main()
{
  int i, n, sum=0;
  Printf ("enter an integer number") ;
  Scanf (" %d", &n) ;
   for (i=1; 1<n ; i++)
     {
        sum = sum+i;
     }
  Printf (" \n sum of %d natural number = %d", sum) ;
    getch();
}
```

Nested for Loop :-

In nested for loop, one loop is placed inside the other for Loop.

If there are many data items to be processed against a set of elements repeatedly, then a single 'for' statement is not adequate. So we must used a nested for loop. If one loop is completely placed within the other, it is known as a nested for loop.

Syntax :-

```
for (exp1 ; condition1 ; exp2)
  {
    for (exp3 ; condition2 ; exp4)
     {
       statement 1 ;
       statement 2;
     }
  }
```

Example

```
*
* *
* * *
* * * *
```

```
1
1 2
1 2 3
1 2 3 4
```

```
        1
      2 3 2
    3 4 5 4 3
  4 5 6 7 6 5 4
```

The break statement :-

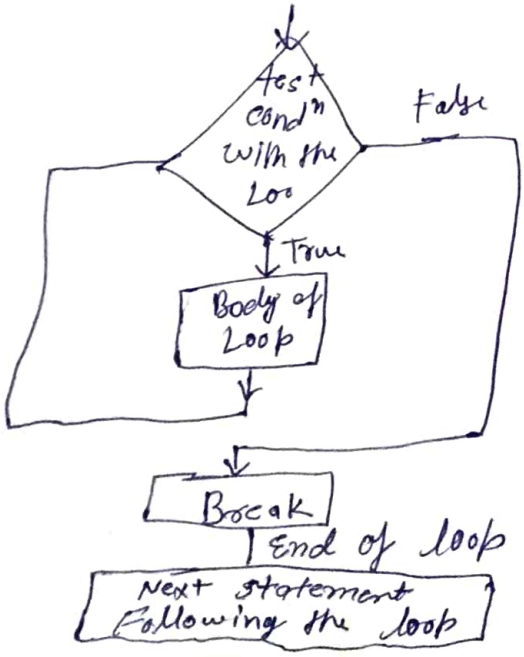
We have already read about break while discussing the switch statement. It is use to exit from a Loop or a switch control, by passing to the first statement after the loop on a switch.

The break statement ~~can be~~ terminate the loop. As soon as it gets executed, Program control comes out of the loop and the first statement after the loop get executed.

The following figure shown break at work —

```
main
{
  int i
for (i=1; i<=10; i++)
  {
    if (i==5)
      break;
    else
      Printf ("%d", i);
  }
    Printf ("Hello");
}
```

Take control here



Example:- Write a program to search a number b/w 1 to 50.

Soln

```
#
#
Void main()
{
  int i, n;
  Printf(" Enter the number to search between 1 to 50");
  Scanf(" %d", &n)
  Printf(" \n Search is:");
  for("\n
  for (i=1; i<=50; i++)
    {
      Printf(" \n %d", i);
        If (i==n)
          {
          Printf("\n At last Reached the desired No");
           break;
          }
    }
  getch()
}
```

2. WAP in C to accept the sex code and find out the person ~~or~~ is male or ~~fe~~ Female.

Soln

```
void main()
{  char  Sex-code;
   Printf(" Enter the Sex code of a person");
   scanf(" %c", & Sex-code);
  switch ( Sex-code)
   {
      case 'M':
           case 'm':
        Printf(" Male");
           break;
```

```
Case 'F':
Case 'f':
    Printf("\n Female");
    break;
default:
    Printf("\n Invalid choice");
    break;
}
getch()
}
```

Example:- Write a program to find out number is prime or not (using while loop and break statement).

Soln

```
#include<math.h>
#include<stdio.h>
#include<conio.h>
void main()
{
int n, test, remainder, Prime=1;
Printf("Enter the number");
Scanf("%d", &n);
    test = 2;
    while (test <= sqrt(n))
    {
        remainder = n % test;
        If (remainder == 0)
        {
            Prime = 0;
            break;
        }
    else (Prime)
    Printf("\n %d is a prime number\n");
    else
    Printf("\n %d is not prime number\n");
    getch();
}
```

The continue statement:-

The continue statement is used to repeate the some operation once again even if it checks the error.
The syntax of the continue statement is:

```
continue;
```

This is similar to break statement, but is encountered less ~~frequency~~ frequently. It only works within loops where its effect is to ~~for~~ force an immediate jump to the loop control statement.

The continue statement is used for the inverse operation of the break statement.

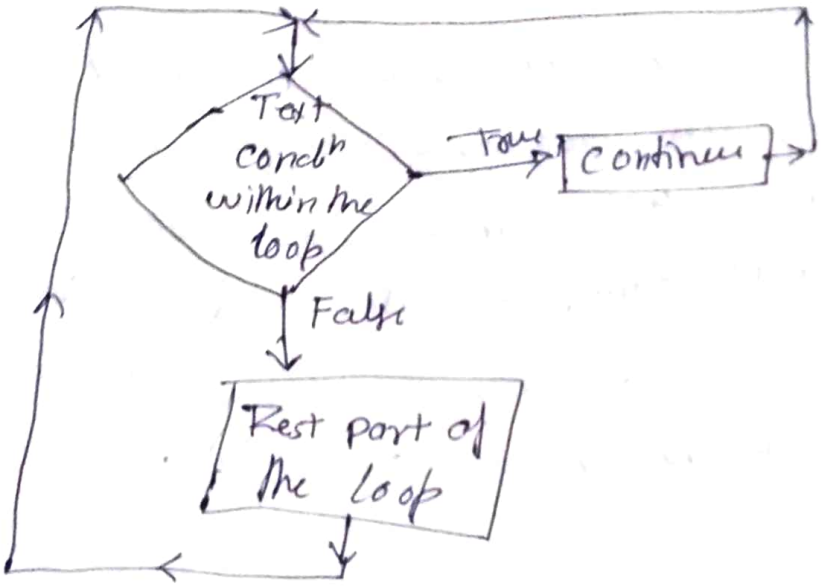
~~Syntax~~:

```
main()
{ int i;
for (i=1; i<=10; i++)
   {  if (i==5)
      continue;
      else
      Printf("\n %d", i);
   }
   Printf("\n Hello");
}
```

The 'continue' statement is used for the next iteration of the loop to take skipping any code in b/w. Hence it is use to terminate the current iteration and continue ~~to the~~ with the next iteration of the loop.

For the loop, 'continue' cause the updation and then the conditional test portions of the loop to execute.

For the while and do-while, Program control passes to the conditional tests.



Example Write a program using using continue statement.

```
#
#
Void main()
 {
    int i,j ;
    for (i=1 ; i<=3 ; i++)
      {
        for (j=1 ; j<=3 ; j++)
          {
            if (i!=j)
              continue ;
              Printf("\n %d", i, j);
          }
      }
    getch() ;
 }
```

The goto statement :-

The goto statement is used to alter the program execution sequence by transferring the control to some part of the program.

The Syntax of goto statement is —

```
goto statement label ;
```

Type — (i) Unconditional goto
(ii) Conditional goto

Unconditional goto :-

The unconditional goto statement is used ~~to~~ just to transfer the control from one part of the program to the other part without checking any condition.

Example :-

```
#
#
void main()
{
  start :
  Printf ("welcome");
   goto start;
   getch();
}
```

Conditional goto :-

The conditional goto is used to transfer the control of the execution from one part of the program to the other in certain conditional cases.

eg

```
void main()
{
  int a, b ;
  Printf (" Enter the value of a and b ");
  scanf ("%d %d", &a, &b) ;
```

```
if (a>b)
  goto out 1;
  else
  goto out 2;
    out1:
  Printf ("\n largest is %d", a);
  out 2 :
  Printf ("\n Largest is %d", b);
    getch()
}
```

The exit() statement!-

C provides a way to leave a program early with the exit() function.

Syntax!- exit (status) ;

where - status is an int variable or constant.

example!-

```
Void main ()
{
  Printf (" C programming is intresting ");
  exit (0);
  Printf (" C is easy to learn ");
  Printf (" It is powerful Language ");
    getch();
}
```