Unit - I

Programming Languages

<u>What is language?</u>

A language means a medium of communication from one person to another. It is exchange of facts ideas, opinions or emotions by two or more person. This language needed for carrying out the day to day activities of human life.

Marathi, Hindi, English etc are example of natural language used to communicate the people. Each language has its own set of alphabets, words to make meaningful instructions. All these natural language follows rules of spelling and grammar.

Similarly to communicate with computer for specific kind of work, which should be known by the computer system, we need the language called as programming language. This language is used for proper functioning of system; perform various operations such as arithmetical, logical according to the instructions given in the form of computer language.

<u>What is programming language?</u>

The computer doses not understand English so in order to interact with computer or to make a communication with the computer the programmer must know the language which the computer understands. The information and instruction which are to be carried out by the computer are written in the form of a program.

Programming language is a medium to communicate with a computer. The programming language consists of alphabets, symbols, words & sentences. The programming language written in such a way that the computer reads, understands and interprets each statement finally gives desired result according to the manner of program instructions.

Types of languages

1.1.Machine Language

1.2.Assembly language

1.3.High level languages

‘C’ is a general purpose, Structured Programming language. It is suitable for solving scientific and engineering problems as well as for business application. Over the past few years, C has become the most popular computer programming language. C’s speed and power have always made it a favorite language for operating systems, compilers, interpreters and word processors. Many powerful applications can be generated in C language economically and correctly using C language. C language encourages users to write additional library functions of their own to enhance the features of C language. One of the unique features of C language is that it is available on any make of computer and also available for a range of computers, including mainframe, mini and microcomputers.

Types of languages

As we human beings communicate with each other's in different language such as English, French, German or Arabic. Similarly to communicate with the computers we have to use specific language and for this purpose hundreds of languages have been invented in which few of them has gained international reputation. C language is one of them. Languages can be can be broadly categorized into three categories.

Machine Language (Low Level Language)

The most elementary and first type of computer, which was invented, was machine language. Machine language was machine dependent. A program written in machine language cannot be run on another type of computer without significant alterations. Machine language is sometimes referred to as the binary language; i-e, the language of 0 and 1, where 0 stands for the absence of electric pulse and 1 stands for the presence of electric pulse. Very few computer programs are actually written in machine language.

Assembly language

As computer became more popular, it became quite apparent that machine language programming was simply too slow and tedious for most programmers. Assembly languages are also called as low-level language. Instead of using the string of numbers, programmers began using English-like abbreviations to represent the elementary operation. The language provided an opportunity to the programmers to use English like words that were called MNEMONICS.

For Example

LDAA

ADD B

Where

LDAA = Load Accumulator with A

Add B = Add number B to the contents of the accumulator, leaving the result in accumulator.

High level languages

The assembly languages started using English like words, but still it was difficult to learn these languages. High level languages are the computers language in which it is much easier to write a program than the low-level languages. A program written in high-level language is just like giving instruction to person in daily life. It was in 1957 that a high level language called FORTRAN (Formula Translation) was developed by IBM, which was specially developed for Scientists and Engineers. Other high level languages are COBOL (Common Business Oriented Language), which is widely used for business data processing task. BASIC (Beginners All-Purpose Symbolic Instruction Codes), which is developed for the beginners in 1960s. The most powerful and widely used language is C language which is a general purpose programming language. You can use C language for almost any programming task. PASCAL PL/1 are other high level languages, which has gained widespread acceptance.

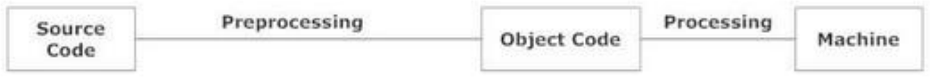
Compiler and Interpreter

We generally write a computer program using a high-level language. A high-level language is one which is understandable by us humans. It contains words and phrases from the English (or other) language. But a computer does not understand high-level language. It only understands program written in `0's` and `1's` in binary, called the machine code. A program written in high-level language is called a source code. We need to convert the source code into machine code and this is accomplished by compilers and interpreters. Hence, a compiler or an interpreter is a program that converts program written in high-level language into machine code understood by the computer.

The difference between an interpreter and a compiler is given below:

Compiler

Scans the entire program and translates it as a whole into machine code. It takes large amount of time to analyze the source code but the overall execution time is comparatively faster. Generates intermediate object code which further requires linking, hence requires more memory. It generates the error message only after scanning the whole program. Hence debugging is comparatively hard. Programming language like C, C++ use compilers.



Interpreter

Translates program one statement at a time. It takes less amount of time to analyze the source code but the overall execution time is slower. No intermediate object code is generated, hence are memory efficient. Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy. Programming language like Python, Ruby use interpreters.

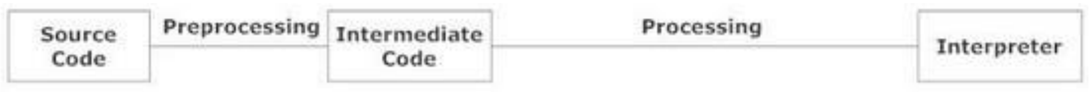


Figure: Interpreter

Chapter 2

Introduction to Programming in C

Introduction to C

C is a programming language developed at AT & T's Bell Laborites of USA in 1970. It was designed and written by a 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 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 languages like FORTRAN or PL/I or the newer ones like Pascal and APL.

Possibly why C seems so popular is because it is reliable, simple and easy to use. Out of the dozens of languages available, the prize of purity is often given to PASCAL.

Historical Development of C Language

C is a general-purpose language which has been closely associated with the UNIX operating system for which it was developed - since the system and most of the programs that run it are written in C. COBOL was being used for commercial applications, FORTRAN for Engineering and Scientific Applications and so on. At this stage people started thinking that instead of learning and using so many languages, each for a different purpose, why not use only one language. Which can program all possible applications. Therefore an international committee came out with a language called ALGOL 60. However, ALGOL 60 never really became popular because it seemed too abstract , too general. To reduce this abstractness and generality, a new language called combined Programming Language (CPL) was developed at Cambridge University. CPL was an attempt to bring ALGOL 60 down to earth. However, CPL turned out to be so big, having so many features, that it was hard to learn and difficult to implement.

Basic Combined Programming Language (BCPL), developed by Martin Richards at Cambridge University aimed to solve this problem by brining CPL down to its basic good features. But unfortunately it turned out to too less powerful and too specific. Around same time a language called B was written by Ken Thompson at AT & T's Bell Labs, as a further simplification of CPL. But like BCPL, B too and BCPL, added some of his own and developed C. Ritchie's main achievement is the restoration of the lost generality in BCPL and B, and still keeping it powerful.

Algorithm in Programming

In programming, algorithm is a set of well defined instructions in sequence to solve the problem.

Qualities of a good algorithm

Input and output should be defined precisely.

Each steps in algorithm should be clear and unambiguous.

Algorithm should be most effective among many different ways to solve a problem.

An algorithm shouldn't have computer code. Instead, the algorithm should be written in such a way that, it can be used in similar programming languages.

Write an algorithm to add two numbers entered by user.

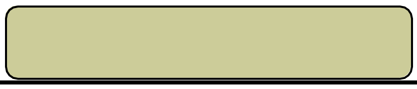
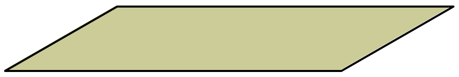
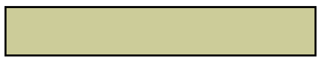
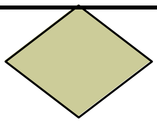
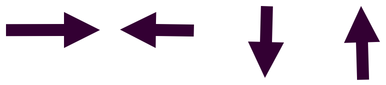
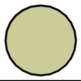
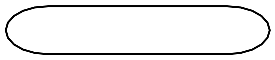
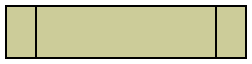
```
Step 1: Start
Step 2: Declare variables num1, num2 and sum.
Step 3: Read values num1 and num2.
Step 4: Add num1 and num2 and assign the result to sum.
        sum←num1+num2
Step 5: Display sum
Step 6: Stop
```

Write an algorithm to find the largest among three different numbers entered by user.

```
Step 1: Start
Step 2: Declare variables a,b and c.
Step 3: Read variables a,b and c.
Step 4: If a>b
            If a>c
                Display a is the largest number.
            Else
                Display c is the largest number.
        Else
            If b>c
                Display b is the largest number.
            Else
                Display c is the greatest number.
Step 5: Stop
```

Flow Chart

- Flowchart is the diagrammatic or pictorial representation of the algorithm for solving the problem. This is a tool used in the design of program. Flowchart consist of pictorial symbols. Flowchart helps a programmer to plan the procedure for the solution of a problem.
- Flowchart indicates the direction of flow of process, relevant operations and computations, point of decision and other information which is part of the solution.
- The symbols and their meaning used in flowchart are shown in the following table

Symbol	Meaning
	Start, Stop (begin, end)
	Input, Output
	Processing program instructions
	Decision
	Direction of flow
	Connector
	Loop
	Function symbol

Flowchart are not related to any programming language. Hence they can be implemented in any programming language.
Flowchart are easy to understand. The data flow of the program can be easily understood by flowcharts and errors in it can be easily recovered.
When we need to incorporate some more facilities within program, then it is easy to incorporate them through flowchart.

APPLICATIONS OF C LANGUAGE

C Programming is a best-known programming language. C Programming is near to machine as well as human so it is called as Middle-level Programming Language. C Programming can be used to do a verity of tasks such as networking related, OS related.

1. C language is used for **creating computer applications**
2. Used in writing Embedded software
3. Firmware for various electronics, industrial and communications products which use micro-controllers.
4. It is also used in developing verification software, test code, simulators etc. for various applications and hardware products.
5. For **Creating Compiler** of different Languages which can take input from other language and convert it into lower level machine dependent language.
6. C is used to implement different Operating System Operations. UNIX kernel is completely developed in C Language.

ADVANTAGES OF C LANGUAGE

- It contains a powerful data definition. The data type supported are characters, alphanumeric, integers, long integer, float, double. It also supports string manipulation in the form of character array.
- C supports a powerful set of operators.
- It also supports powerful graphics programming and directly operators with hardware. Execution of program is faster.
- An assembly code is also inserted into C programs.
- C programs such as compilers, Operating systems can be developed in C. For example the popular operating system UNIX is developed in C.
- The C language has 32 keywords and about 145 library functions and near about 30 header files.
- C works closely with machines and matches assembly language in many ways

Structure of C Programming

1) Documentation Section
2) Header Section [link section]
3) Definition Section
4) Global Declaration
5) Main Section
6) Sub Program Section

1) Documentation Section

We can write title of program in that section using comments line there are two types of comment line

a) **Single line comment:-**it is starting with // [double slash] this type of comment line are used for single line comment.

b) **Multi line Comment /* */:-T**his type of comment line is used to give additional information about the program in two or more then two lines this type of comment line is started with /* & end with '*/' there is no effect on the program by using comment line.

2) **Header Section[link section]**

In this section we can use the Header file such as # 'stdio.h' & 'conio.h' , string.h, such as Header file are used to give c linking support to the functions used in c program.

Stdio.h stands for slandered I/O file. Which support to standard I/O function as printf() & scanf().

Similarly conio.h is stand for console I/O hidder file which support to console I/O functions such as clrscr(); & getch().

3) **Definition Section**

In definition section we define constant value by using #defines ex:-#define pie 3.14 In above example to define constant value 3.142 variable.

4) **Global Declaration Section**

There are some variables that are used in more than one function such variables are called global variable they are declared in the global declaration section that in outside of all function.

5) **Main() Section**

Every C program have only one main() function is start with ({) & close with (}) main function have main two part i.e declaration part & execution part.

a) **Declaration part:-**
In this part we declare the variables before using program. In declaration part we declare variables with their particular data types such as int, float, char etc.

b) **Execution Part:-**
In this part of main() section we use execution statements in main() section every statement & function are terminated by (;).

6) Sub program section:-

In sub program section we can use user defined function. User defined function are function which are defined by user.

Simple C Program

/* This is my first 'c' program */

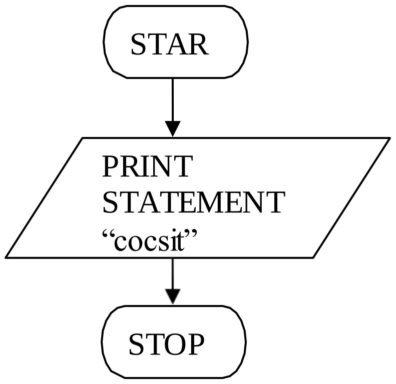
```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
printf("Wel come C programming");
getch();
}

//Write a program two addition of two numbers
#include<stdio.h>
#include<conio.h>
Void main()
{
Clrscr()
Printf("The addition of two number is %d",20+30);
Getch();
}
```

//Write A Program to display the name of college.

Algorithm:-

1. Start
2. Print statement "cocsit"
3. stop



Flowchart:-

Program:-

/* Write A Program to display the name of college.*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
     clrscr();
     printf("*****out/put****");
     printf("Hello cocsit");

     getch();
}
          *****out/put****
          Hello cocsit
```

The Character set

Every language has it's own character set ex;- In English language there are 26 alphabets are used to create word, sentence, paragraph.
Similarly 'C' language has it's own character sets. The character set of 'C' language is nothing but combination of alphabets, digits & special symbols 'C' inserts the uppercase letter A—Z lower case latter a---z the digits 0 9 & special character. The alphabets & digits together are called as alphanumeric character.
These characters are used as building blocks from basic program element.
Following table shows the valid alphabets, digits & special symbol allowed in 'C'

Alphabets

Uppercase -

A	B	C	---------------------	Z.

Lowercase -

a	b	C	---------	Z

Digits:-

0	2	3	4	5	6	7	8	9

Special Characters

,	Comma
;	Semi column
:	Column
?	Question Mark
"	Double quotation

|	Vertical Bar
\	Back Slash
/	Slash
~	Tile
_	Under score
$	Dollar sign
#	Hash
%	Percentage
&	Ampere Sign
*	Astric
-	Minus
+	Plus
<	Less sign/ opening angle
>	Grater sign/ closing angle
(	Left parentheses
)	Right parentheses
[	Left Bracket
]	Right Bracket
{	Left Brace
}	Right Brace

White spacious

The ‘C’ compiler ignores white space unless they are a part of string constant white space may be used to separate words, white space include blank space, horizontal tab.

Horizontal tab	-	/t
New line	-	/n

By using this character set we can create program, function name, variable name. ‘C’ uses certain combination of characters such as, /b,/n,/t. to represent special condition such as back space, new line & horizontal tab these characters combination are known as escape sequence.

IDENTIFIERS AND KEYWORDS

Identifers

- Identifiers are names that are given to various program elements such as variables, function & array name. there are certain rules we should be follow while running & identifiers are
- Identifier consists of letters & digits in any order except the first character should be letter
- Both uppercase & lowercase letters are permitted although lowercase letter are commonly used.

- Uppercase & lowercase letters are not interchangeable [an uppercase letters is not equivalent to corresponding lowercase letter
- The underscore [_] character can be included & it is considered as a letter.
- An identifiers name up to 8 characters many compilers allow an identifiers name up to 31 characters.
- Following names are valid identifiers x, xyz, sum, nau, basic_sal.
  First character must be latters.

The following names are valid identifiers

X	y12	sum_1	_tempera
Names	area	text_rate	TABLE

The following names are not valid identifiers

7h	The first character must be a letter
X”	Illegal characters (“).
Rder-no	Illegal charaters (-).
Rror flag	Illegal characters (blank space).

C Keywords

Keywords are the words whose meaning has already been explained to the C compiler. The keywords cannot be used as variable names because if we do so we are trying to assign a new meaning to the keyword, which is not allowed by the computer.

Auto	Break	Case	Char	Const	Continue
Default	do	Double	Else	Enum	extern
Float	far	For	Goto	If	Int
Long	near	Register	Return	Short	Signed
Static	struct	Switch	Typedef	Union	Unsigned
Void	while				

There are 32 keywords used in C all keywords must be written in lower case.

Some compilers may also include some or all of the following keywords.

Ada	Far	Near	Asm	Fortran	Pascal
Entry	Huge				

Variables

A variables is data name that may be used to store the data value these variables name are giving to location in the memory of computer where we are storing the data there are some rules available for creation and deceleration of variables.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c;
clrscr();
printf("Enter the number");
scanf("%d",&a,&b);
c=a+b
printf("The addition is %d",c);
getch();
}
```

Types of variables

In C, a quantity which may change during program execution is called a variable. Variable names are names given to locations in the memory of computer where different constants are stored. This location can contains integer, real or character constants. This is because a constant stored in a location with a particular type of variable name can hold only that type of constant. Ex:- a constant stored in a memory location with an integer variable name must be an integer constant. One stored in location with a real variable name must be a real constant and the one stored in location with a character variable name must be a character constant.

Rules for Constructing Variable Name.

a) A variable name is any combination of 1 to 8 alphabets, digits or underscores. Some compilers allow variable names whose length could be up to 40 characters.
b) The first character in the variable name must be an alphabet.
c) No commas or blanks are allowed within a variable name.
d) No special symbol other than an underscore (as in gross_sal) can be used in a variable name.

```
Ex:-        int I,net_sal;

            Float a;
            Char code;-
```

Data type

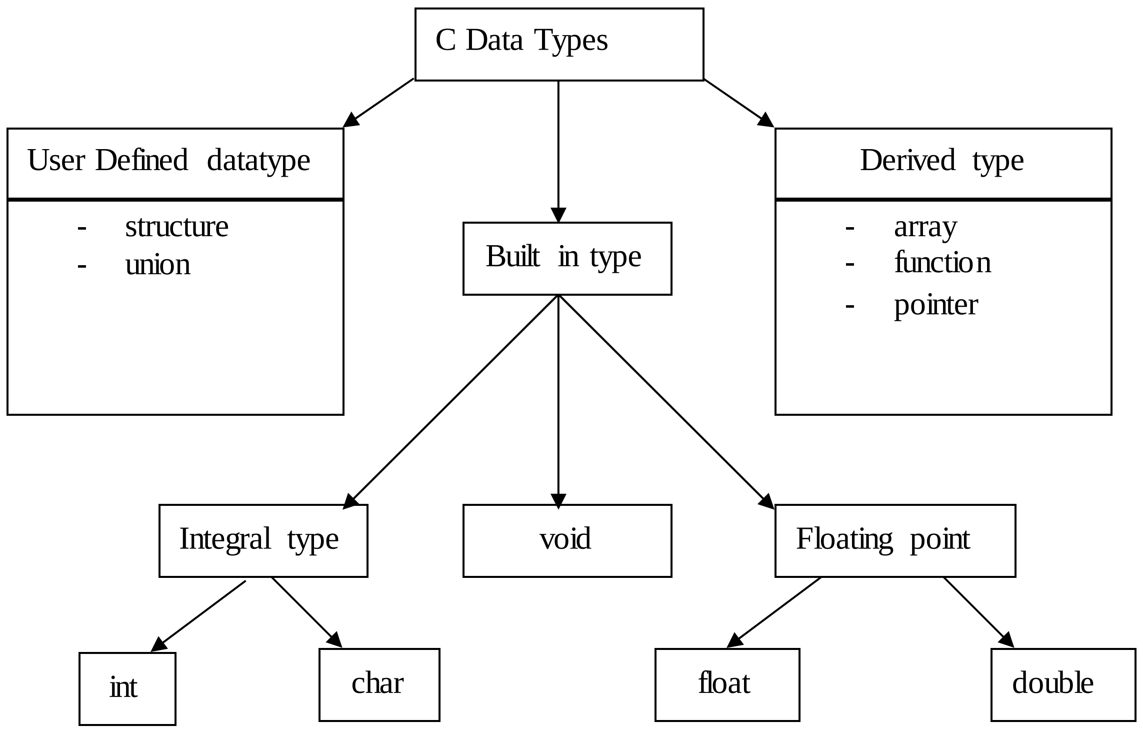
C language is rich in data type like other languages 'C' language has also it's own data types data types are used to define or declare the type [data type] of particular variable, constant, function etc.

In C language there are three classes of data types.

1) Primary data type.
2) Derived data type.
3) User-defined data types.

Data type are always written in small letters each data types has it's own feature depending upon a condition the particular data types is used in the name of data type we cannot use in valid space.

Data Type	**Range**	**Size in Bite**	**Size in Byte**	**Format String**
Char	-128 to 127	8	1	%c
Unsigned char	0 to 255	8	1	%c
Int	-32768 to 32767	16	2	%i or %d
Unsigned int	0 to 65535	16	2	%u
long int	-2147483648 to 2147483647	32	4	%ld
Unsigned long int	0 to 4294967295	32	4	%lu
Float	3.4e-38 to 3.4e+38	32	4	%f or %g
Double	1.7e-308 to 1.7e+308	64	8	%lf
Long double	3.4e-4932 to 1.1 e+4932	80	10	%lf



1) **Primary Data Type:-** it is also known as standard of or built in data type all compiler support fundamental data type. Primary data type is divided into 3 type.

Int [integer]:-

Integer are defined by keyword 'int' integer are store for no we cannot store decimal point no in that data type integer occupies one word of storage i.e 2 bytes or 16 bits. The range of integer data type is -32768 to 32767 'C' has support to classes of integer storage namely int shor int & long int variable than they increases the range of value as will as size of it. 32 bit word length can store an integer ranging from - 2,147,483,648to 2147483647.

Ex:- Int a=5; int b=6;

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,c;
    a=10;
    b=20;
    c=a+b;
```

```
        printf("\n  A addition  of two number  is %d",c);
        getch();
    }
```

Float

The real no are known as floating value. These no contain functions part. This is defined by keyword 'float' float value requires 4 bytes or 32 beats. The range of float is 3.4e-38 to 3.4e38.

Ex:- float pie=3.14;

```
#include<stdio.h>
#include<conio.h>
{
        float a,b,c;
        clrscr();
        a=3.2;
        b=1..5;
        c=a+b;
        printf("\n  Addition  of two number  %f',c);
        getch();
}
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int sub1=40,sub2=50,sub3=60,sum;
        float avg;
        clrscr();
        sum=sub1+sub2+sub3;
        printf("\n  Addition  of three subject is %d",sum);
        avg=sum/3;
        printf("\n  average  of three subject is %f',avg);
        getch();
}
```

character (char)

Generally those data types are used when we have to storage a single character or string to variable. It requires 1 byte or 8 beats to store in memory. It is defined by keyword as 'char' the range of char is -128 to 127.

Ex:-char name;
Name='a';
Char name[15]="Bhosle";

Double

It is very similar to float data type the major difference between float & double data type is size & range of the double data type is always greater than float data type size of double data type is 64 beats, 8 bytes range of double data type is 1.7e-308 to 1.7e +308 .

Void Type

The Void type has no value. This is usually used to specify the type of functions. The type of a function is said to be void when it does not return any value to the calling function. It can also play the role of a generic type, meaning that it can represent any of the other standard types.

User defined data type

These data type are defined by using keyword 'typedef' it takes following format.

Syntax :- typedef type identifier name

Ex:- typedef int mark
The example will demonstrate the use of user defined data type.

```
#include<stdio.h>
#include<conio.h>
typedef int mark
Void main()
{
        mark sub1,sub2,sub3,sum,avg;
        printf("\nEnter the mark of three subject");
        scanf("%d%d%d",&sub1,&sub2,&sub3);
        sum=sub1+sub2+sub3;
        avg=sum/3;
        printf("\nThe total of three subject is%d",sum);
        printf("\nThe average of three subject is %d",avg);
        getch();
}
```

here marks indicate integer data type but sub1, sub2,sub3 indirectly marks referees to standard data type int.

Derived data type

Some time these derived data type are also called user defined data type which is derived from fundamental data type. The derived data type are

ex:- array, pointer, function.

Escape Sequences

These are the special characters starting with '\' which makes the computer to line out of the normal sequence.

Computer has some default sequences e.g computer always prints data in a line unless we ask the computer to create a new line etc. the following are the escape sequences.

Constant	Meaning
\n	New line
\t	Horizontal tab
\v	Vertical tab
\a	Alert (bell)
\b	Back space
\f	Form feed

Constant	Meaning
\r	Carriage return
\0	Null
\'	Single quote
\"	Double qutoe
\?	Question mark
\\	Back slash

//Write a program to swap two value by using third variable

```
#include<stdio.h>
#include<conio.h>

void main()
{
int a,b,c;
clrscr();
printf("\nEnter the First number');
scanf("%d",&a);
printf("\nEnter the Second Number");
scanf("%d",&b);
c=a;
a=b;
b=c;
printf("\n After swap A=%d",a);
printf("\n After swap B=%d",b);
getch();
}
          *****out/put****
          Enter the First number 23
          Enter the Second Number 55
          After swap A= 55
          After swap B= 23
```

//Write a program to swap two value by using third variable

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b;
clrscr();
printf("\nEnter the First number');
scanf("%d",&a);
printf("\nEnter the Second Number");
scanf("%d",&b);
a=a+b;
b=a-b;
a=a-b;
printf("\n After swap A=%d",a);
printf("\n After swap B=%d",b);
getch();
}
          *****out/put****
          Enter the First number 23
          Enter the Second Number 55
          After swap A= 55
          After swap B= 23
```

/*write a program to reveres the given number*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
int no,rev,a1,a2,a3;
clrscr();
printf("Enter the number");
scanf("%d",&no);
a1=no%10;
no=no/10;
a2=no%10;
no=no/10;
a3=no;
rev=100*a1+10*a2+a3;
printf("Revers number is %d",rev);
getch();
}
          *****out/put****
          Enter the number 123
          Entered number is 321
```

Operators :-

An operators is a symbol which tells the computer to perform mathematic or logical operation C supports following 8 types of operators

1) Arithmetic operators
2) Relational operators
3) Logical operators
4) Assignment operators
5) Increment & Decrement operators
6) Conditional operators
7) Bit-wise operators
8) Special operators

1) Arithmetic operators

C Language supports all arithmetic operators this operators provides same function as in other languages this operators can operate on any built in data type in C following list gives the arithmetic operators with meaning.
+,-,*,%

Integer division ignores frication parts in the result c doesn’t support an operators for exponential

A+B
A and b is operands

//Addition of two number

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a;
float b,c;
clrscr();
printf("\nEnter the two number");
scanf("%d%f",&a,&b);
c=a+b;
printf("Addition of two number is %f",c);
getch();
}
```

2) Relational operators

If we want to compare two values then Relational operators are used C supports six Relational operators

1	>
2	>=
3	<
4	<=
5	= =
6	!=

The simple format of Relational operators as follows

Expression Relational operator's expression

The value of real expression is dissention statement if

<u>//Write a program to use various relational operators and display their return values.</u>

```
#include<stdio.h>
#include<conio.h>
void main()
{
        clrscr();
        printf("\nCondition  : Return Values \n");
        printf("\n10!=10        : %5d",10!=10);
        printf("\n10==10        : %5d",10==10);
        printf("\n10>=10        : %5d",10>=10);
        printf("\n10<=10        : %5d",10<=10);
        printf("\n10!=9         : %5d",10!=9);
        getch();
}
```

3) logical operators

C language support three logical operators

&&	Logical And
||	Logical or
!	Logical Not

a<b && x==30

a>b || b>c

Logical 'AND' (&&)		
Condition1	**Condition 2**	**Result**
True	False	False
False	True	False
True	True	True
False	False	False

Logical 'OR' (&&)		
Condition1	**Condition 2**	**Result**
True	False	True
False	True	True
True	True	True
False	False	False

Logical 'NOT' (!)	
True	False
False	True

4) Assignment Operators

The assignment operators are used to assign values or result of the expression to variables C support for the assignment operators that is

=

ex:-a=10;

5) Increment and Decrement operators

Prefix Increment and Decrement operators

In the prefix operators the values of variables is Increment and Decrement First and then assigned to the expression

++a --a
a=3
y=++a
y=4
postfix operators
a post fix operators first assign the values to variables on left side and then increment the operand

```
a=5;
y=a++;
y=6
```

6) Conditional operators

Conditional operators
Expirations ? expression2 : expression3

```
? :
a=3;
b=4;
x=(a<b)? a:b;
```

//write a program to find out the given number greater or not by using conditional operators.

```
#include<stdio.h>
#include<conio.h>
Void main()
{
        Int a,b;
        Printf("Enter two number");
        Scanf("%d%d",&a,&b);

        a>b?printf("A is greater number"):Printf("B is greater number");

        getch();
}
```

in above expression a is less then b if this condition is true then value of a will be to assign to the expression and if given condition is false then value of b it will be assign to the expression

7) Bit-wise operators

There operators are used for manipulation of data at bit level we can use these operators for testing the bits, sift right and left we can't apply the bit wise operators with float and double following list gives the operators and there meaning

This operator move bit patterns either to the left or to the right. The shift operators are represented by Left shift << and Right shift >>.

<< Left
10

0	0	0	0	1	0	1	0
0	0	1	0	1	0	0	0

>>Right
10

0	0	0	0	1	0	1	0
		0	0	0	0	1	0

8) Special operators

Comma operators this operator is used to link or connect the related expression
Values(a=3,b=4,a+b)
X=7
3 is assign to variables a 4 will be assign to b after that x=7 addition of two numbers

// Write a program to use of comma (,) operator

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
printf(" Subtraction = %d\n Addition =%d",5-3,5+6);
getch();
}
```

Input And Output Statement

Introduction

Reading the data from the input devices and displaying the results on the screen, are the two main tasks of any program. To perform these tasks user friendly C has a number of input and output functions. When a program needs data, it takes the data through the input functions and sends results obtained through the output functions. The input/output functions are the link between the user and the VDU.

There are number of I/O functions in C based on the data types. The input/output functions are classified in to two types

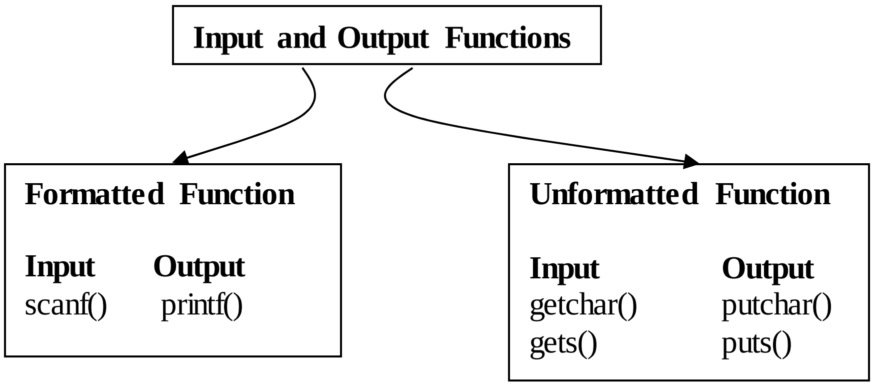
4) Formatted functions
5) Unformatted function.

1) **<u>Formatted Function:-</u>**

The formatted input/output functions read and write all types of data values. They require conversion symbol to identify the data type. They can be used for both reading and writing of all data values. The formatted functions return the values after execution. The return value is equal to the number of variables successfully read/ written.

2) **<u>Unformatted function:-</u>**

The formatted input/output functions only work with the character data type. There is no need to convert the data. In case values of other data types are passed to these functions, they are treated as the character data. The unformatted function also return values, but the return value of unformatted function is always the same.



<u>Formatted Function:-</u>

<u>printf():-</u>

This function is used to display result on the screen it can be used to display any combination of numerical value as will as char or string. It requires conversion symbol and variable names to print the data. The conversion symbol and variable names should be same in number.

Syntax:-

```
printf("<Format string>" ,arg1,arg2,….);
```

Ex:-

```
void main()
{
  int a=3;
  float b=5;
  char c='b';
  printf("%d    %f     %c",a,b,c);
}
```

//display the ascii value

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int y=65;
  clrscr()
  printf("%c    %d",y,y);
  getch();
}
```

scanf():-

The input data or information can be enter to the computer through a standard input device by using scanf() function. This function can be used to enter any combination of numerical value, single char, or strings this function return the number of data item that have been successful we can declare the scanf() function as follow.

```
scanf("<Format string>"  ,&arg1,&arg2,….);
```

In the format string various format specifies are used

Ex:- scanf("%d",&a);

The argument1,argument2,……are the argument at that represent that indusial data item in the format string these argument are return of variables. The scanf() statement requires '&' operator called address operator. The address operators print the memory location of the variable. Here in the scanf() statement the role of '&' operator is to indicate the memory location of the variable. So that value read would be placed at that location.

Ex:-

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a,b,c;
        clrscr();
        scanf("%d%d",&a,&b);
        c=a+b;
        printf("%d",c);
        getch();
}
```

Getchar() Function:-

Single character can be entered into the computer using ‘c’ library function getchar(); the get character function is a part of standard I/O function. It written a single character from a standard input device [keyboard] the function does not require any argument a pair of empty parentheses must follow the word getchar(). The getchar take following form

```
Variable name=getchar();
```

Variable name is a valid C name that has been declared as char type. When this statement is encountered, the computer waits until a key is pressed and then assign this character as a value to getchar function. Since getchar is used on the right-hand side of an assignment statement,

Syntax:-

```
Char name;
Name=getchar();
```

Ex:-

```
Void main()
{
        char c;
        clrscr();
        c=getchar();
        putchar(c);
        getch();
}
```

Putchar() Function:-

This function prints one character on the screen at a time which is read by the standard input device such as keyboard.

Syntax:-

```
putchar(name);
```

Ex:-

```
Void main()
{
        char c;
        clrscr();
        c=getchar();
        putchar(c);
        getch();
}
```

puts() function:-

This function is used to display the string or collection of character to the output device such as monitor.

Syntax:-

```
Char name[20];
puts(name);
```

Ex:-

```
#include<stdio.h>
#include<conio.h>
void main()
{
        char name[20];
        clrscr();
        printf("\nEnter the name");
        gets(name);
        printf("My name is %s",name);
        getch();
}
```

gets() function:-

This function is used to accepting any string or collection of character from input device such as keyboard.

Syntax:-

```
Char name[20];
Name=gets();
```

Ex:-

```
#include<stdio.h>
#include<conio.h>
void main()
{
        char name[20];
        clrscr();
        printf("\nEnter the name");
        gets(name);
        printf("\n My name is ");
        puts(name);
        getch();
}
```

Conversion Specifiers used in c

Data type	Conversion Symbol
Integer	%d or %i
Unsigned	%u
Long signed	%ld
Long unsigned	%ld
Unsigned hexadecimal	%x
Unsigned octal	%o
Float	%f or %g
Double	%lf
Char	%c
Unsigned char	%c
String	%s

Chapter 3
Control Structure

Introduction

The statements in a program are executed line by line similarly in the way we read a text book. The method to read a textbook is straight forward, which starts from first line to the end of line and from top to bottom.

Similarly, in the program the statements are read and executed sequentially line-by-line unto the end of program. In some time, we may select one of the options from the available options by testing some condition. Depending on the problem, the flow of the program is fixed.

C Language supports decision making to control the flow and execution of statements in any order. Such statements are decision or control statements. Decision-making is one of the important aspects of the computer. For making decisions computer needs a condition i.e logical expression and decision-making statement. For building conditions, we need the operators.

Expressions in decision-making statements normally consist of arithmetic, relational, logical or conditional operators.

Decision making structures

- The if statement
- The if- else statement
- Nested if – else statement

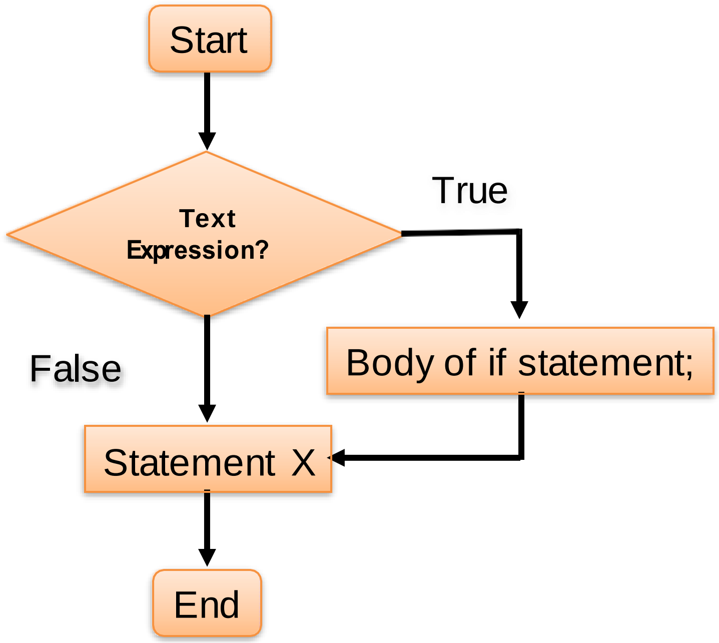
The if statement

If it is an keyword, The if statement is a powerful decision making statement used to control the flow of execution of statements. First condition is checked. If condition is true than it execute the true statement block control transfer to statement x. If the condition is false then without executing the true block the control is goes directly to the next statement x. The if statement always used in conjunction with expressions. These expressions evaluate true of false.

The general format of a simple if statement is

```
if (test expression)
{
        body of if statement;
}
        Next statement x;
```

The body of if may have one statement or group of statement where each statement is separated by semicolon. If the test expression is true, the true statement block will be executed otherwise computer will skip the block of statement and the execution will jump to the next statement immediately after the end of if statement.



Simple If statement flow chart

```
If ( a>b)
{
Printf(%d\n",a);
}

Void main()
{
        Int a,b;
        Printf("\nEnter the Number a & b");
        Scanf("%d%d",&a,&b);
        if(a>b)
                Printf("A is grater number");
        If(b>a)
                Printf("B is grater number");
        Getch();
}
```

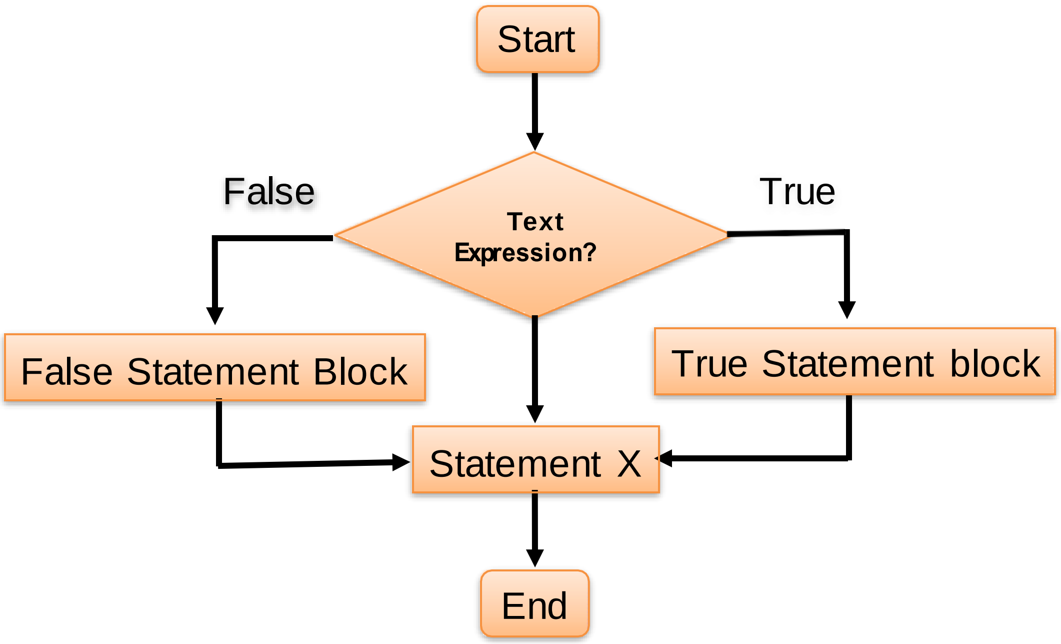
- The test expression (condition) should not end with semicolon(;).
- The test expression without expression arguments is not acceptable.
- The test expression must be always enclosed within a pair of brackets().

If – else statement

The if – else statement is an extension of the simple if statement. In this statement first check the condition if condition is true then body of if (true) statement is executed otherwise body of else (false) statement is executed. In this statement either body of if or body of else is executed and then only the control moves to the next statement.

Syntax:-

```
if (condition)
{
        Body of true statement block;
}
else
{
        Body of false statement block;
}
```



Flow Chart of if – else Statement

Ex:-

```
if (a>b)
{
        Printf("\n A is grater then B");
}
else
{
```

```
                Printf("\nA  is less then B");
        }
```

This statement provides the alternates in both the situations, if condition is true the body of if is executed if condition is false the body of else is executed.
Here computer can not proceed without executing either body of if or body of else.

//Write a program to find grater number within two numbers

```
    void main()
    { int a,b;
        printf("\nEnter the Number a & b");
        scanf("%d%d",&a,&b);
        if(a>b)
        {
            printf("A is grater number");
        }
        else
        {
            printf("B is grater number");
        }getch();
    }
```

// Write a program to find given number is even or odd

```
    void main()
    { int num;
        printf("\nEnter the any number");
        scanf("%d",&num);
        if(num%2==0)
        {
            printf("The given number is even number");
        }
        else
        {
            printf("The given number is odd number");
        }   getch();
    }
```

//Write a program to find given year is leap year or not

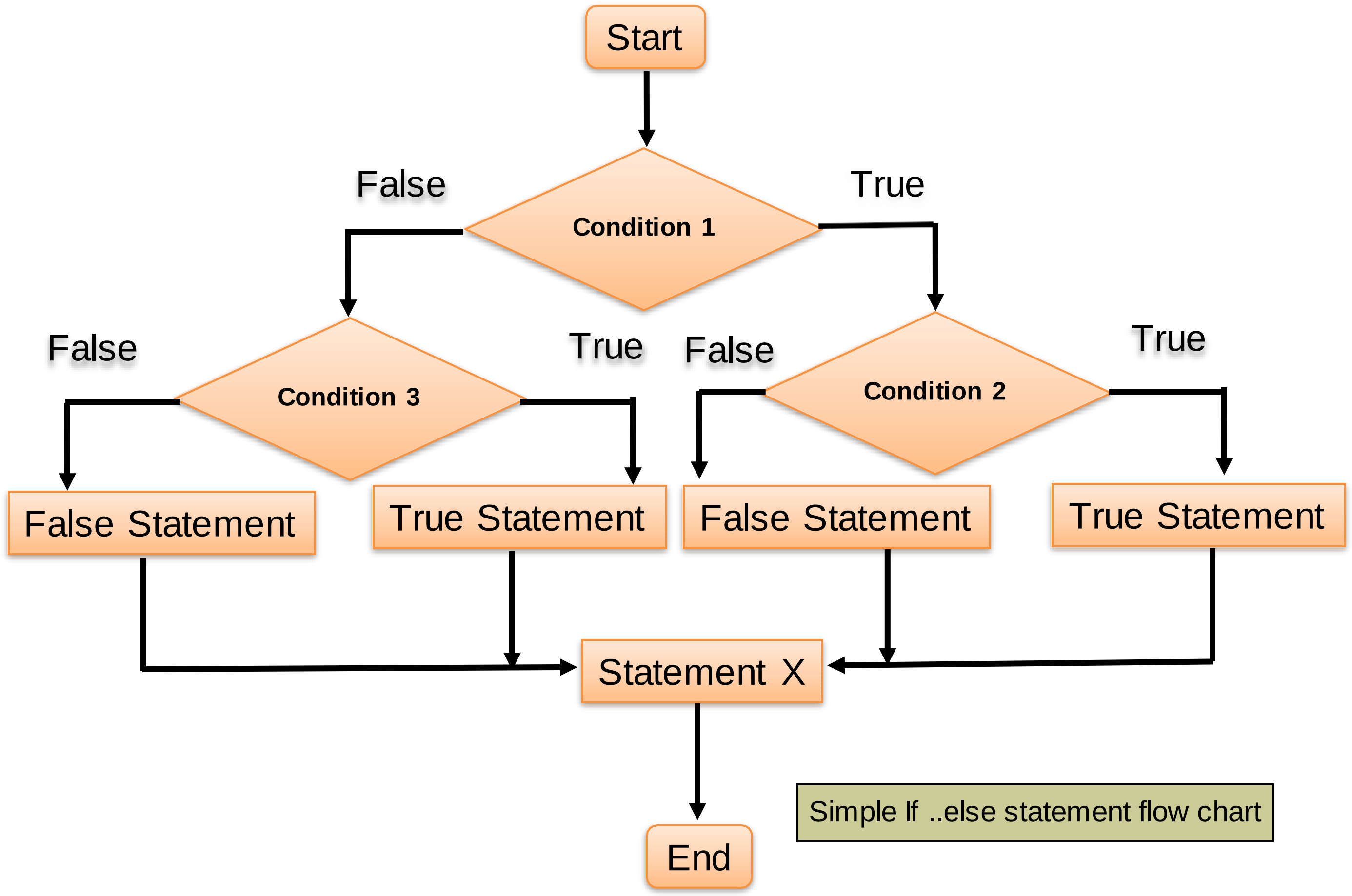
```
    void main()
    { int year;
    printf("\nEnter the year");
    scanf("%d",&year);
        if(year%4==0)
        {
            printf("The given year is leap year");
        }
        else
        {
            printf("The given year is not leap year");
        }getch();
    }
```

Nested if –else statement

It is perfectly all right if we write an entire if-else construct within either the body of the if statement or the body of an else statement. This is called as nesting of if. In this kind of statement no of logical condition are checked. For executing various statement. Nested if-else can be chained with one another If the condition is check if condition is false then control passes to else statement block where condition is again checked with if statement. This process continue become condition is true if condition is true then execute the true statement block otherwise last else statement block will be execute.

Syntax:-

```
if(Condition 1)
{
        if(Condition 2)
        {
                Body of True Statement Block
        }
        else
        {
                Body of False Statement Block
        }
}
else
{
        if(Condition 3)
        {
                Body of True Statement Block
        }
        else
        {
                Body of False Statement Block
        }
}
```



Flow Chart of Nested if – else Statement

```
Ex-
if(a>b)
{
        if(a>c
        {
        printf("a  is  grater");
        }
        else
        {
        printf(c  is  grater");
        }
}
```

```
else
{
        if(b>c)
        {
        printf("B  is grater");
        }
        else{
        printf("c  is  grater");
        }
}
```

In that first check the condition. If the given condition1 is true then again check the condition 2 if the given condition 2 is true then execute the true statement block and control transfer to statement x. if the given condition 2 is false then without executing the true statement block execute the false statement block of condition 2 and control transfer to statement x. if given condition 1 is false then control transfer to else statement again check the condition 3 if the condition 3 is true then execute the true statement and control transfer to statement x. if the given condition 3 is false then execute the false statement block and control transfer to statement x.

When this structure gets executed and as soon as any one of the statement gets executed, immediately the chain breaks & computer goes next statement. There are many possible forms of nesting. It always depends on our problem.

Write a program that display the class obtained by student if percentage of the student is input through the keyboard.

```
void main()
{
float per;
printf("\nEnter the percentage");
scanf("%f",&per);
If(per<0 && per>100)
{
printf("Invalid input");
exit();
}


if(per<40)
        printf("Fail");
else
{
        if(per<50)
                printf("Pass");
        else
                {
                if(per<60)
                        printf("Second class class");
                else
                        printf("First class");
                }
}
```

Write a program find out entered key is upper case or lower case.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char x;
clrscr();
printf("\nEnter any char");
scanf("%c",&x);
if((x>=65&&x<=90)||(x>=97&&x<=122))
{
if(x>=65&&x<=90)
printf("it is an upper case alphabet");
else
printf("It is lower case alphabet");
}
else
printf("It is not alphabet");
getch();
}
```

Positive number and negitive number

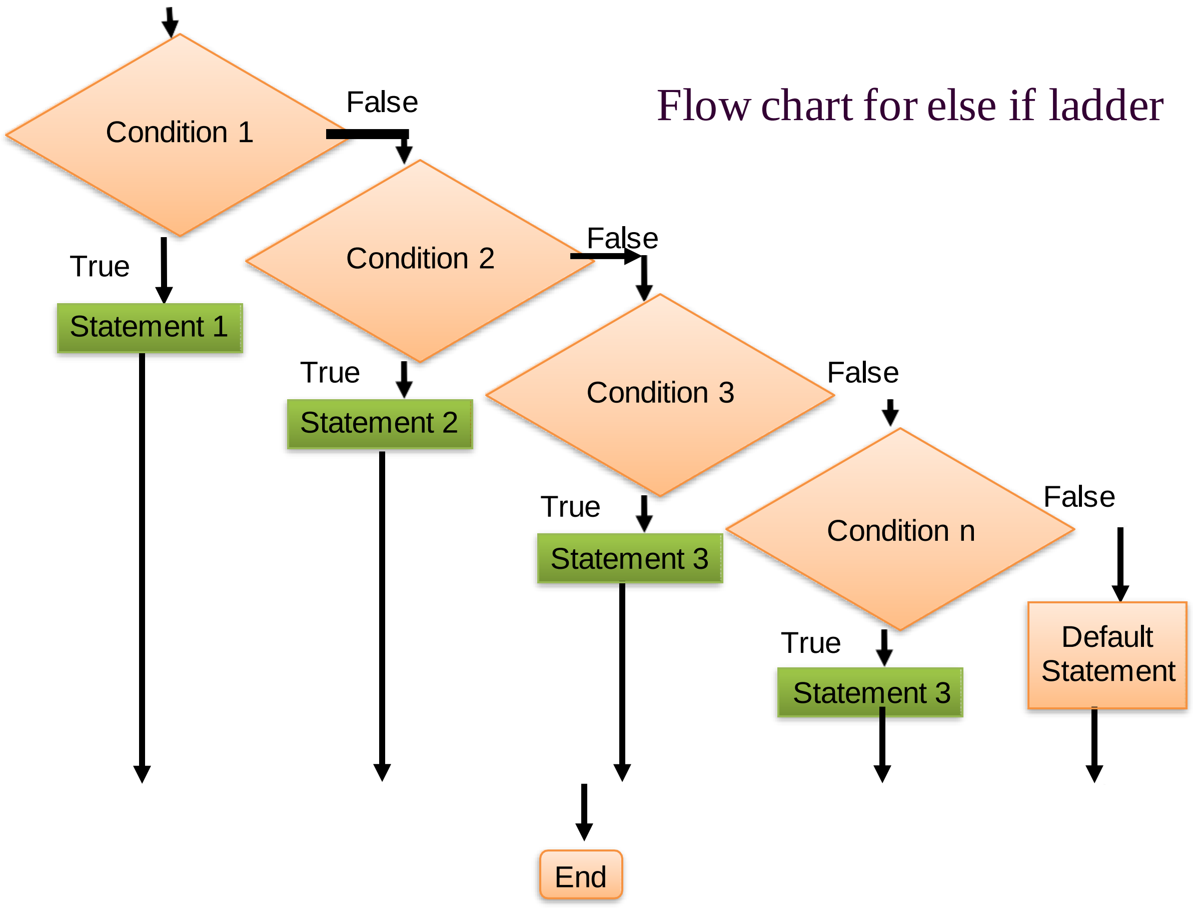
```
    #include<stdio.h>
    #include<conio.h>
    void main ()
    {
        int num;
        printf("\nenter any number");
        scanf("%d",&num);
        if(num>0)

        {
        printf("given number is positive");
        }
        else
        {
        if(num<0)
        {
        printf("given num is negative");
        }
        else
        {
        printf("Zero");
        }
        }
        getch();}
```

else—if ladder

There is a negative side to the program. Even if the first condition turns out to be true, still all other condition are checked. This will increase the time of execution of the program. This can be avoided using the else if clause.
When multiple decision are taken then else if is used when we used else if ladder, can else is associated with if.
Syntax

```
If(Condition 1)
        statement 1;
else if(condition 2)
        statement 2;
else if(condition 3)
        statement 3;
 else if(condition n)
        statement n;
else
        default statement;
statement x;
```



Flow chart for else if ladder

To find the grade

```
Void main()
{
        Float per;
        Printf("Enter the value");
        Scanf("%f",&per);
        If(per>80)
        {
        Printf("grade is merit");
        }
        Else if(per>70)
        {
        Printf("Distinction");
        }
        Else if(per>60)
        {
        Printf("Grade A");
        }

        else if(per>50)
        {
        printf("Grade is B");
        }
        else if(per>=35)
        {
        printf("Grade is C");
        }
        else
        {
        printf("Failed");
        }
        printf("try try but don't cry, one day you will success");
        getch();
}
```

Write a program to Larger number within five number.

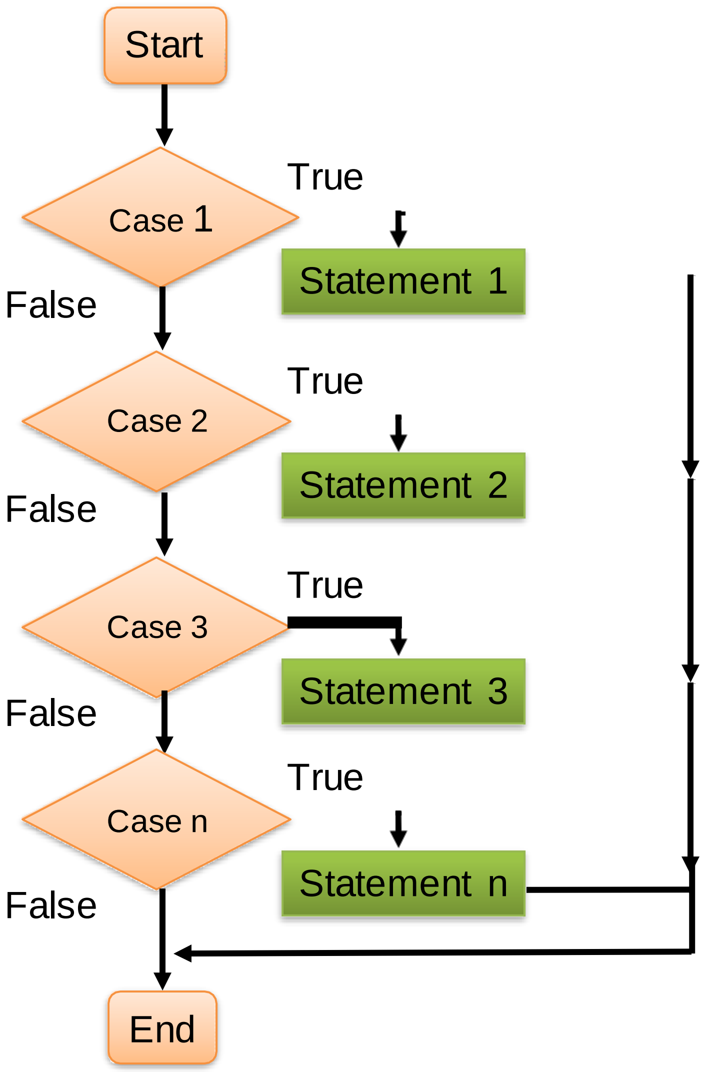
```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n1,n2,n3,n4,n5,larg,small;
    clrscr();
    printf("\nEnter the number");
    scanf("%d%d%d%d%d",&n1,&n2,&n3,&n4,&n5);
    larg=n1;
    if(n1>larg)
    larg=n1;
    if(n2>larg)
    larg=n2;
    if(n3>larg)
    larg=n3;
    if(n4>larg)
    larg=n4;
    if(n5>larg)
    larg=n5;
    printf("\nthe largest number within five number is %d",larg);
    small=n1;
    if(n1<small)
    small=n1;
    if(n2<small)
    small=n2;
    if(n3<small)
    small=n3;
    if(n4<small)
    small=n4;
    if(n5<small)
    small=n5;
    printf("\nthe smallest number within five number is %d",small);
    getch();}
```

Switch statement

By using switch statement, which have multiple collection but (alternative increases) complexity increase and program become easy to read and debug. Switch statement is multi-case statement. The keyword switch statement value of condition match with label Syntax:

```
switch(expression)
{
        case value 1:
                statement 1;
                break;
        case value 2:
                statement 2;
                break;
        Default:
                Default block;
}
```

Switch flow chart



If condition is doesn’t match with label of case (1) than it checks condition value is match with label of case (2). If that value is match than execute statement block of case (2) and control transfer to statement x and so on---
We have to remember that, case labels are single character constants.

Explanation of switch statement

Expression: it is an integer expression or character.
Value1, value 2 : this are constants or constants expression known as case label.
Block 1, block 2: this are statement which may content one or more statement.
Case label must be end with colon(:).
default: the default is an optional case which it will execute expression value, doesn’t match with any case value.

- In a switch there can be either variable or expression
- If it is a variable it must be either integer or character
- If it is an expression it must be an arithmetic expression.
- There can be any number of cases.
- Every case should have an unique value.
- These cases can be written in any sequence.
- In a case there can be any number of statement.
- It is possible to have the nested switch.
- After every case break statement is compulsory.
- Default is a case which gets selected when none of the case value matches with the result of expression.
- Default case is optional.

Write a program that reads a number between 1 to 7 and display the day name

```
main()
{
Int day;
Printf(“Enter a number between 1 to 7 \n”);
Scanf(“%d”,&day);
Switch(day)
{
Case 1:printf(“Monday\n”);
Break;
Case 2:printf(“Tuesday\n”);
Break;

Case 3:printf(“Wednesday\n”);
Break;
Case 4:printf(“Thursday\n”);
Break;

Case 5:printf(“Friday\n”);
Break;
Case 6:printf(“Saturday\n”);
Break;
Case 7:printf(“Sunday\n”);
Break;
Default:printf(“Monday\n”);
}
```

Write a program add,sub,multi & division of two number

```
void main()
{
int a,b,n,p;
flat c;
clrscr();
xyz:printf(“\nenter two number”);
scanf(“%d%d”,&a,&b);
printf(“\n 1 addition of two number”);
printf(“\n 2 subtraction of two number”);
printf(“\n 3 multiplication of two number”);
printf(“\n 4 division of two number”);
printf(“\nenter your choice”);
scanf(“%d”,&n);

switch(n)
{
case 1:c=a+b;
    printf(“\naddition of two number is%f”,c);
        break;
case 2:c+a-b;
        printf(“\subtraction of two number is%f”,c);
        break;

case 3:c+a*b;
        printf(“\multiplication of two number is%f”,c);
        break;
case 4:c+a-b;
        printf(“\division of two number is%f”,c);
        break;
default:
        printf(“\nentered choice incorrect”);
```

```
break;
}
printf(“\ndo  you want to continue”);
printf(“\nenter  6 yes 7: no”);
scanf(“%d”,&p);

swatch(p)
{
case 6:goto xyz;
case 7:exit();
}
getch();
}

#include<stdio.h>
#include<conio.h>
#include<process.h>
void main()
{
int a,b,c,n,d;
clrscr();
xyz:printf("\nEnter  the number  a");
scanf("%d",&a);
printf("\nEnter  the number  b");
scanf("%d",&b);
printf("\n\n  1 Addition");
printf("\n\n  2 Subtraction");
printf("\n\n  3 Multiplication");
printf("\n\n  4 Divission");
printf("\n\n  Enter  the  choice");
scanf("%d",&n);
switch(n)
{
case 1:c=a+b;
printf("\n\nAddition  of two number  is%d",c);
break;
case 2:
c=a-b;
printf("\n\nsubtraction  of two number  is%d",c);
break;
case 3:
c=a*b;
printf("\n\nmulti  of two number  is%d",c);
break;
default:
```

```
c=a/b;
printf("\n\ndivision of two number is%d",c);
}
printf("\n\nEnter 6 to Yes and enter 7 No");
scanf("%d",&d);
switch(d)
{
case 6:goto xyz;
case 7:exit(0);
break;
}
getch();
}
```

The output of the following code is_____________

```
#include <stdio.h>
int main()
{
int c=1;
switch( c ){
case 1:
printf("1");
default:
printf("0");
}
return 0;
}
```

Answer: 10

Looping Control Structures

In the program so many times we come across the situation where we want to execute some part of the program again and again. For some particular number of time. In such situations we make use of loop statements. These are the statements which can execute a block of statements for some particular number of time.

The loops are used for repetitive execution of statement in block of loop as long as the condition is true, all statements in a block of loop are executed repeatedly. When looping condition become false. The control moves to the next statement immediately after the end of loop block. Here the condition is a valid logical statement. The logical statement determines the condition under which the loop should continue repeating.

So when we want to repeat some part or procedure for some particular number of times then we use loops.

In this language there are three loops

- while loop
- do-while loop
- for loop
- break statement, continue statement, goto statement

Entry control loop

In this loop, first condition is checked, if the condition is true, than it executes the body of loop. Again the condition is checked, if the condition is true, it execute body of loop . So on , this processes continues until the given condition become false. Once condition become false, than it transfer control to statement x without executing the body of loop

Ex -While loop,For loop

Exit control loop

In this loop, first body of loop is executed, then the condition is checked. If the condition is true, than it execute body of loop. Again condition is checked, if it is true the body of execute. So on this process continue until the given condition become false. Once condition become false. It control transfer to statement x without executing the body of loop.

In exit control loop, the condition may true or false, at least once the body of loop is executed

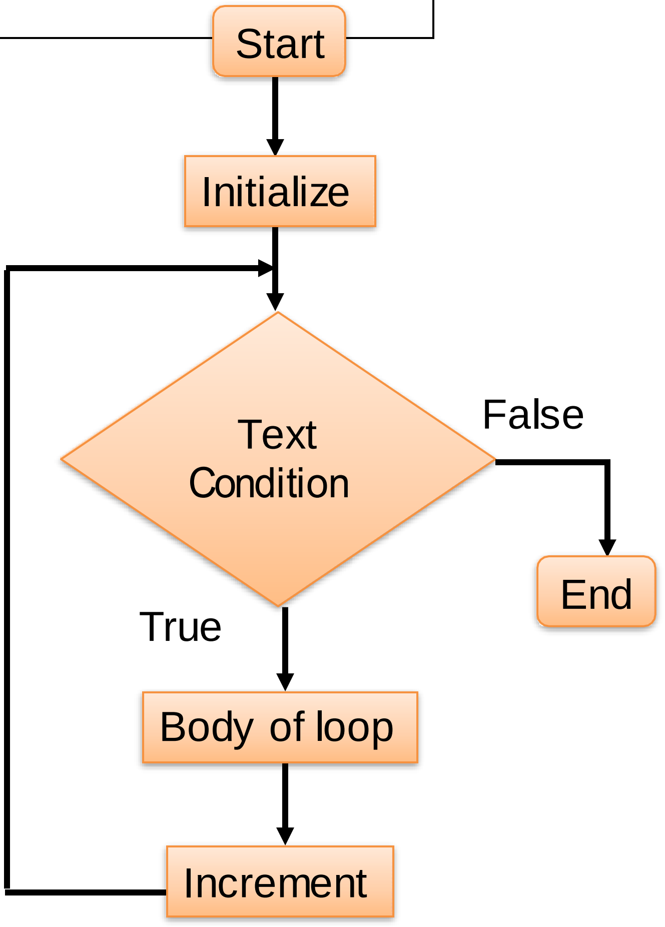
Ex-do-while

While loop

the while loop is one of the entry control loop. In this loop the condition is given at the top. First the check condition and if the condition is true then execute body of loop. The body of while may have one statement or more than one statement, if there is a single statement no need of brackets'{''}'. If there are more one statement then brackets are compulsory. Again the check condition and if it is true then execute body of loop this process still continues of condition become false if condition become false does not execute the body of loop and control goes to statement x. This loop should be used when condition is more important. Syntax:

```
While(condition)
{
        Body true statement block;

}
```



flow chart for while loop

Ex:-
```
void main()
{
        int i=1;
        clrscr();
        while(i<=100)
                {
                printf("%d\n",i);
                i=i+1;
                }getch();
}
void main()
{
        int i=2;
        clrscr();
        while(i<=100)
                {
                printf("%d\n",i);
                i=i+2;
                }
getch();
}
```

Do while

There is a minor difference between the working of while and do-while loops this difference is the place where the condition is tested. The while tests the condition before executing any of the statements within the while loop. The do-while tests the condition after having executed the statements within the loop.

Syntax:-

```
Do
{
        Body of the loop
}
While(test-condition);
```

Since the test condition is execute at the bottom of the loop, the do –while construct provides an exit controlled loop and therefore the body of the loop is always executed at least once.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>

    void main()
    {
        int n,k,m;
        clrscr();
        scanf("%d",&k);
        n=1;
        do
        {
                m=pow(n,2);
                printf("%d\n",m);
                n++;
        } while(m<=(pow(k,2)));
getch();
}
```

//display 5,10,15----

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int n,k;
        clrscr();
        scanf("%d",&k);
        n=5;
        while(n<=k)
        {
                printf("%d\n",n);
                n=n+5;
        }
 getch();
}
```

//display 7,14,21----

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int n=7;
        clrscr();
        while(n<=70)
        {
                printf("%d\n",n);
                n=n+7;
        }
 getch();
}
```

//display 10,20,30----

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int n,k;
        clrscr();
        scanf("%d",&k);
        n=10;
        while(n<=k)
        {
                printf("%d\n",n);
                n=n+10;
        }       getch();
}
```

//display 1,2,4,7,11----

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int n,k,d;
        clrscr();
        scanf("%d",&k);
        n=1;
        d=1;
        while(n<=k)
        {
                printf("%d\n",n);
                n=n+d;
                d=d+1;
        }       getch();
}
```

//display 1,2,4,8----

```
#include<stdio.h>
#include<conio.h>
void main()
{               int n,k;
        clrscr();
        scanf("%d",&k);
        n=1;
        while(n<=k)
        {
                printf("%d\n",n);
                n=n*2;
        }
getch();        }
```

//sum of 1 to 10 number

```
        #include<stdio.h>
        #include<conio.h>
        void main()
        {
                int i=1,sum=0;
                clrscr();
                while(i<=10)
                {
                        printf("%d\n",i);
                        sum=sum+i;
                        i++
                }
        Printf(“sum  of I to 10 number%d”,sum);
        getch();
        }
```

//multiplication table

```
        #include<stdio.h>
        #include<conio.h>
        void main()
        {
                int I,j;
                clrscr();
                while(i<=10)
                {
                        j=1;
                        while(j<=10)
                        {
                                printf("%d\t",i);
                                j++
                        }
                        printf(“\n”);
                        i++;
                }
        Printf(“sum  of I to 10 number%d”,sum);
        getch();
        }
```

Difference between while and do-while

While	Do—while
Condition is at the top	Condition is at the bottom.
No necessity of brackets if there is single statement in body.	Brackets are compulsory even if there is a single statement.
There is no semicolon at the end of while.	The semicolon is compulsory at eh end do-while.
Computer executes the body if and only if condition is true.	Computer executes the body at least once even if condition is false
This should be used when condition is more important.	This should be used when the process is important.
This loop is also refered as entry controlled loop.	This loop is also refered as exit controlled loop.

while(n<=10)
{
printf("%d\n",n);
n++;
}
	do
	{
	printf("%d\n",n);
	n++;
	}while(n<=100);

```
void main()
{
        int i=1;
        clrscr();
                do
                {
                printf("%d\n",i);
                i=i++;
                } while(i<=100);
getch();
}
```

//display 5,10,15----

```
	#include<stdio.h>
	#include<conio.h>
	void main()
	{
		int n,k;
		clrscr();
		scanf("%d",&k);
		n=5;
		do
		{
			printf("%d\n",n);
			n=n+5;
		} while(n<=k);
	getch();
	}
```

//display 7,14,21----

```
	#include<stdio.h>
	#include<conio.h>
	void main()
	{
		int n=7;
		clrscr();
		do
		{
			printf("%d\n",n);
			n=n+7;
		} while(n<=70);
	getch();
	}
```

//display 10,20,30----

```
	#include<stdio.h>
	#include<conio.h>
	void main()
	{
		int n,k;
		clrscr();
		scanf("%d",&k);
		n=10;
		do
```

```
                {
                        printf("%d\n",n);

                         n=n+10;
                 } while(n<=k);
        getch();
        }
```

//display 1,2,4,7,11----

```
        #include<stdio.h>
        #include<conio.h>
        void main()
        {
                int n,k,d;
                clrscr();
                scanf("%d",&k);
                n=1;
                d=1;
                do
                {
                        printf("%d\n",n);
                        n=n+d;
                        d=d+1;
                } while(n<=k);
        getch();
        }
```

//display 1,2,4,8----

```
        #include<stdio.h>
        #include<conio.h>
        void main()
        {
                int n,k;
                clrscr();
                scanf("%d",&k);
                n=1;
                do
                {
                        printf("%d\n",n);
                        n=n*2;
                } while(n<=k);
        getch();
```

```
          }
```

//display 1,2,4,8----

```
          #include<stdio.h>
          #include<conio.h>
          void main()
          {
                    int n,km;
                    clrscr();
                    scanf("%d",&k);
                    n=1;
                    do
                    {
                              m=pow(n,2);
                              printf("%d\n",m);
                              n++;
                    } while(m<=(pow(k,2)));
          getch();
          }
```

//multiplaction table

```
                    #include<stdio.h>
                    #include<conio.h>
                    void main()
                    {
                              int i=1,j;
                              clrscr();
                              while(i<=5)
                              {
                                        j=1;
                                        while(j<=5)
                                        {
                                                  printf("%d\t",i*j);
                                                  j++;
                                        }
                                        printf("\n\n");
                                        i++;
                              }
                     getch();
                    }
```

Display output as

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i=1,j;
        clrscr();
        while(i<=5)
        {
                j=1;
                while(j<=i)
                {
                        printf("*\t");
                        j++;
                }
                printf("\n\n");
                i++;
        }
        getch();
}
```

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

For loop

This statement is used when calculations are to be carried out between initial and final value with step. The for loop condition consist of three actions namely initialization, testing and incrementing/ decrementing. Each action should be separated by semicolon(;) also this loop statement is used when number of execution time is known in advance.

It is an entry control loop having syntax as follow.

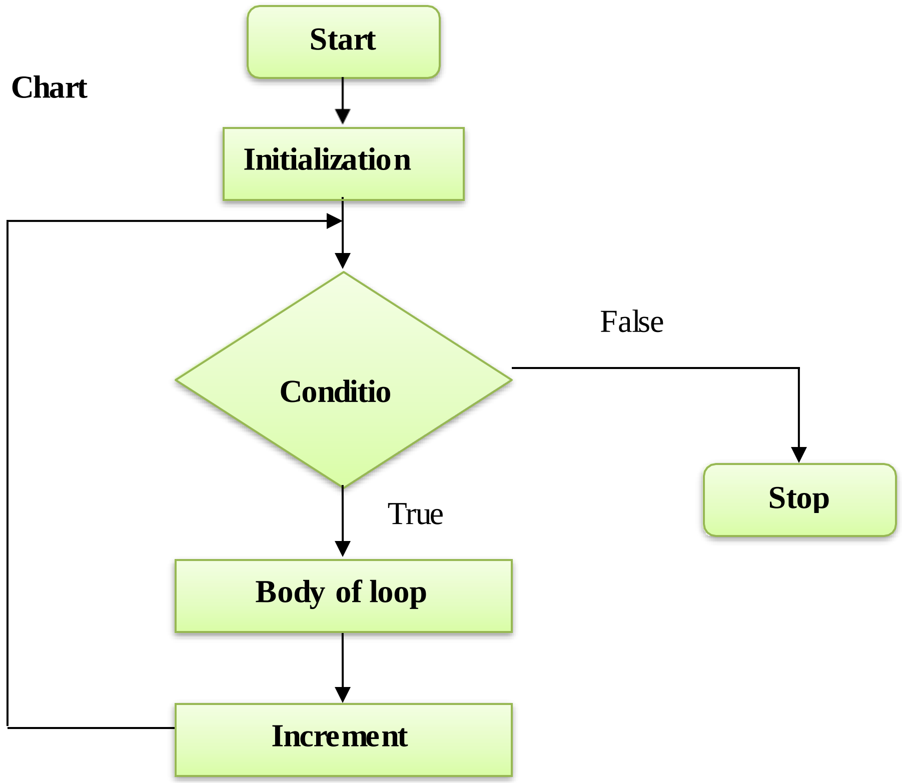
Syntax-

```
for(initial value;condition;increment/decrement)
{
        body of for loop;
}
```

In this loop the initial value, condition, increment/decrement is specified in the same line. Here computer starts with the initial value, checks the condition. If the condition is true computer executes the body and automatically goes up, it takes the increment or decrement automatically and it continues. Remember, if the body of for continue a single statement, no need of brackets. But if there are more than one statement in the body then they must be enclosed between brackets.

Here every thing is specified in the same line and hence we can tell the number of repetitions it can take. When we know the number of repetitions in advance than we should go for the for loop.

Flow Chart



```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i;
        for(i=1;i<=10;i++)
        {
                printf("%d",i);
        }
        getch()
}
```

Break Statement

The break statement is used for two different purpose it can be used to terminated a case in the switch statement that is to terminate the given loop immediately and it by pass the normal loop condition test.
we often come across situations where we want to jump out of a loop instantly, without wetting to get back to the conditional test, the keyword break allows us to do this. when break is encountered inside any loop, control automatically passes to the first statement after the loop. A break is usually associated.
Ex:-

```
        Switch()
        {
                Case 1:
                Break;
                Case 2:
                Break;
                Default:
        }
ex:-    void main()
        {
        int I;
                for(i=1;i<=10;i++)
                {
                        if(i==5)
                        {
                                break;
                        }
                        printf("%d",i);
                }
                getch();
        }
```

Continue Statement:-

This statement is used to when execution of further statement. This stops execution of next statement ?& transfer control to the start of loop for further execution of loop. This statement should be used only within the loop.

Ex:- program to determine continue statement.

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

in the above ex I is initialized to 1 then it will check (i==5) if it is true it will execute printf() statement. After execution I is incremented by 1 & again executes printf() statement. This process continues till the condition i<=10 becomes true.

Goto Statement:

This statement transfers the control from one statement to other statement in the program which may not be in the sequence. The general form of the goto statement is

```
goto label;
------
------
label:
```

towards jump

```
label: statement

-------
-------
goto label;
```

backward jump

where goto requires a level in order to identify the place where the branch is to be made label. It is an valid variable name & must be followed by a (:) the program either before or after the goto label statement as shown below.

Forward jump:

If the label is placed after goto label statement will be skipped such a jump is known as forward jump.

Backward jump:

If the label is before the goto statement a loop will form & some statement will be executed repeatedly such a jump is known as backward jump.

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void main ()
{
        int a,b;
        clrscr();
        xyz:printf("enter the number");
        scanf("%d",&a);
                if(a%2==0)
                {
                        printf("given num is even ");
                }
                else
                {
                        printf("\nthe given num is odd");
                }

        printf("\nif you want to continue");
        printf("\n Enter 6 to yes 7 to no");
        scanf("%d",&b);
        if(b==6) goto xyz;
        if(b==7) exit(0);
        getch();
}
```

Chapter 4
Arrays

What is an Array?

So far we have used only the fundamental data types. Namely char, int, float, double. Although these types are very useful, they are constrained by the fact that a variable of these types can only on value at any given time. Therefore, they can be used only to handle limited amounts of data. In many applications, however, we need to handle a large volume of data in terms of reading, processing and printing. To process such large amounts of data items. C supports a derived data type known as array that can be used for such application.

An array is a group of similar type of data elements stored in contiguous memory location. These data element are addressed by common name. an array is declared by writing the data types. Followed by the name, followed by the size in brackets.

Ex-int a[2]

For understanding the arrays properly, let us consider the following program:

```
main( )
 {
 int x ;
x = 5 ;
 x = 10 ;
printf ( "\nx = %d", x ) ;
 }
```

This program will print the value of **x** as 10. Why so? Because when a value 10 is assigned to **x**, the earlier value of **x**, i.e. 5, is lost. Thus, ordinary variables are capable of holding only one value at a time. There are situations in which we would want to store more than one value at a time in a single variable.

Suppose we wish to arrange the percentage marks obtained by 100 students in ascending order. In such a case we have two options to store these marks in memory:

Construct 100 variables to store percentage marks obtained by 100 different students, i.e. each variable containing one student’s marks.

Construct one variable (called array or subscripted variable) capable of storing or holding all the hundred values.

A simple reason for this is, it would be much easier to handle one variable than handling 100 different variables. There are certain logics that cannot be store without use of an array. An array is a collective name given to a group of ‘similar quantities’. These similar quantities could be percentage marks of 100 students, or salaries of 300 employees, or ages of 50 employees. What is important is that the quantities must be ‘similar’. Each member in the group is referred to by its position in the group.

Ex: per = { 48, 88, 34, 23, 96 }

In C the fourth number is referred as per[3]. This is because in C the counting of elements begins with 0 and not with 1. Thus, in this example per[3] refers to 23 and per[4] refers to 96. In general, the notation would be per[i], where, i can take a value 0, 1, 2, 3, or 4, depending on the position of the element being referred. Here per is the subscripted variable (array), whereas i is its subscript.

Thus, an array is a collection of similar elements. These similar elements could be all ints, or all floats, or all chars, etc. Usually, the array of characters is called a ‘string’, whereas an array of ints or floats is called simply an array. Remember that all elements of any given array must be of the same type. i.e. we cannot have an array of 10 numbers, of which 5 are ints and 5 are floats.

```
Void main ( )
{
        int avg, sum = 0 ; int i ; int marks[30] ; /* array declaration */
        for ( i = 0 ; i <= 29 ; i++ )
        {
                printf ( "\nEnter marks " ) ;
                scanf ( "%d", &marks[i] ) ; /* store data in array */
        }
        for ( i = 0 ; i <= 29 ; i++ )
                sum = sum + marks[i] ; /* read data from an array*/
                avg = sum / 30 ;
        printf ( "\nAverage marks = %d", avg ) ;
        getch();
}
```

Array Declaration

To begin with, like other variables an array needs to be declared so that the compiler will know what kind of an array and how large an array we want. In our program we have done this with the statement:

Int marks[5];

Here, int specifies the type of the variable, just as it does with ordinary variables and the word marks specifies the name of the variable. The [5] however is new. The number 5 tells how many elements of the type int will be in our array. This number is often called the ‘dimension’ of the array. The bracket ([ ]) tells the compiler that we are dealing with an array.

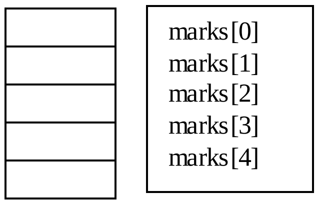
One-Dimensional Arrays:-

A list of items can be given one variable name using only one subscript and such a variable is called a single-subscripted variable of a one-dimensional array.

For ex- if we want to represent a set of five numbers, say(33,55,66,77,88)by an array variable number, then we may declare the variable number as follow

int marks[5] ;

the computer reserves five storage locations as shown below.



The vale to the array elements can be assigned as follows.
marks[0]=33;
marks[1]=55;
marks[2]=66;
marks[3]=77;
marks[4]=88;

<u>Declaration of one-Dimensional Arrays</u>

Like any other variable, arrays must declared before they are used so that the compiler can allocate space for them in memory.
Syntax:

```
Type variable-name[size];
```

The type specifies the type of element that will be contained in the array, such as int, float, or char and the size indicates the maximum number of elements that can be stored inside the array.

Ex-float height[50];

Declares the height to be an array containing 50 real elements. Any subscripts o to 49 are valid.

Ex- int n[12];

Here declares the n as an array to contain a maximum of 12 integer constants.
The c language treats character strings simply as arrays of characters. The size in a character string represents the maximum number of characters that the string can hold.

Char n[12];

Here declare the variable n as a character array variable that can hold a maximum of 10 characters. Suppose we read the following string constant into the string variable name.

“Bhosle Smita”

Each character of the string is treated as an element of the array name and is stored in the memory as follow.

n[0]	n[1]	n[2]	n[3]	n[4]	n[5]	n[6]	n[7]	n[8]	n[9]	n[10]	n[11]	n[12]
↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
‘B’	‘h’	‘o’	‘s’	‘l’	‘e’	‘ ’	‘S’	‘m’	‘I’	‘t’	‘a’	‘\0’

When the compiler sees a character string. It terminates it with an additional null character. The element n[12] holds the null character ‘\0’. When declaring character arrays, we must allow one extra element space for the null terminator.

Initialization of one – Dimensional Arrays

After an array is declared, its elements must be initialized. Otherwise, then will contain garbage value. An array can be initialized at either of the following stages.

We can initialize the elements of arrays in the same way as the ordinary variables when they are declared.

Syntax:

```
Data type array-name[size]={list of values};
```

The values in the list are separated by commas
Ex-int number[3]={0,0,0};
Here declare the variable number as an array of size 3 and will assign zero to each elements. If the number of values in the list is less than the number of elements will be set to zero automatically.

Float total[5]={0.0,15.75,-10};
Here initialize the first three elements to 0.0, 15.75 and -10 and the remaining two elements to zero.
We also declare the array as variant. In that no need to enter the size of array. Ex int a[]={1,3,4}; that is called the variant in that we can store the number of element. Similar, We can also declare the character array but we must end the string with ‘\0’ ex-char name[]={‘b’,’h’,’o’,’s’,’l’,’e’,’\0’}’

Accessing Elements of an Array

Once an array is declared, let us see how individual elements in the array can be referred. This is done with subscript, the number in the brackets following the array name. This number specifies the element's position in the array. All the array elements are numbered, starting with 0. Thus, marks[2] is not the second element of the array, but the third. In our program we are using the variable i as a subscript to refer to various elements of the array. This variable can take different

values and hence can refer to the different elements in the array in turn. This ability to use variables as subscripts is what makes arrays so useful.

Entering Data into an Array

Here is the section of code that places data into an array:

```
for ( i = 0 ; i <= 29 ; i++ )
{
        printf ( "\nEnter marks " ) ;
       scanf ( "%d", &marks[i] ) ;
}
```

The for loop causes the process of asking for and receiving a student's marks from the user to be repeated 30 times. The first time through the loop, i has a value 0, so the scanf( ) function will cause the value typed to be stored in the array element marks[0], the first element of the array. This process will be repeated until i

becomes 29. This is last time through the loop, which is a good thing, because there is no array element like marks[30].

In scanf( ) function, we have used the "address of" operator (&) on the element marks[i] of the array, just as we have used it earlier on other variables (&rate, for example). In so doing, we are passing the address of this particular array element to the scanf( ) function, rather than its value; which is what scanf( ) requires.

Reading Data from an Array

The balance of the program reads the data back out of the array and uses it to calculate the average. The for loop is much the same, but now the body of the loop causes each student's marks to be added to a running total stored in a variable called sum. When all the marks have been added up, the result is divided by 30, the number of students, to get the average.

```
Ex:    for ( i = 0 ; i <= 29 ; i++ )
       sum = sum + marks[i] ;
       avg = sum / 30 ;
       printf ( "\nAverage marks = %d", avg ) ;
```

An array is a collection of similar elements. The first element in the array is numbered 0, so the last element is 1 less than the size of the array. An array is also known as a subscripted variable. Before using an array its type and dimension must be declared.

However big an array its elements are always stored in contiguous memory locations. This is a very important point which we would discuss in more detail later on.

/*Write a program to Display Given No using array */

```
        #include<stdio.h>
        #include<conio.h>
        void main()
        {
        int a[10],i;
        clrscr();
        printf("\n\n\nEnter The 10 Element");
        for(i=0;i<10;i++)
        {
                scanf("%d",&a[i]);
        }
         printf("\n\nDisplay all Entred num ");
        for(i=0;i<=9;i++)
        {
                printf("\n%d",a[i]);
        }
        getch();
        }
```

Two – Dimensional Arrays

So far we have discussed the array variables that can store a list of values. There could be situations where a table of values will have to be stored. A two dimensional array can be used to represent of rows and columns and is also known as a matrix. The matrix in mathematics is a two dimensional array. Ex- rows and column. The element of two dimensional array is indicated by the row and column number in which it is located. The element of matrix is written as a[i][j] where I stands for row and j stands for column.

Syntax:

    Data type array name[row_size][column_size];

```
Ex- int a[2][2]={{22,44,55},{33,55,44},{43,23,54}};
      Int b[2][2];
```

To store the data in memory in this form.

[0][0] 22	[0][1] 44	[0][2] 55
[1][0] 33	[1][1] 55	[1][2] 44
[2][0] 43	[2][1] 23	[2][2] 54

Initializing Two-Dimensional Array:-

Like one-dimensional array, two- dimensional arrays may be initialized by following their declaration with a list of initial values enclosed in braces.

```
Example;-  Int table[2][3]={0,0,0,1,1,1};
```

Initializes the element of the first row to zero and the second row to one. The initialization is done row by row. The above statement can be equivalently written as

```
int table[2][3]={{0,0,0},{1,1,1}};
```

By surrounding the elements of the each row by braces. We can also initialize a two-dimensional array in the form of a matrix as shown below.

```
int table[2][3]={
                        {0,0,0},
                        {1,1,1}
                };
```

Entering values to Two-Dimensional array using scanf function.

The value can be entered to a 2-d array using scanf function in the nested for statement where the outer loop generates the row number and inner loop generate column number.

```
Examples:- int a[3][3];
for(i=0;i<=2;i++)
{
        for(j=0;j<=2;j++)
        {
                Scanf("%d",&a[i][j]);
        }
}
```

Where a is two dimensional arrays the I is an different from 0 to 2 and j is different from 0 to 2. Hence matrix of 3*3 orders is read by this statement.

Reading values to Two-Dimensional array using printf function.

The value can be read to a 2-d array using printf function in the nested for statement where the outer loop generates the row number and inner loop generate column number.

```
Examples:- int a[3][3];
for(i=0;i<=2;i++)
{
        for(j=0;j<=2;j++)
        {
                printff("%d",a[i][j]);
        }
}
```

/*Program to find sum of two Matrices By using Array.*/

```
#include <stdio.h>

#include <conio.h>

void main()

{

        int mat1[3][3],mat2[3][3],matsum[3][3],i,j;
```

```
clrscr();

printf("\t---------OUTPUT-------- \n");
printf("\n\t  Enter Values  in 3x3 Matrix-I  :\n");
for(i=0;i<3;i++)
{
            for(j=0;j<3;j++)
            {
                        printf("\t  Enter  the  [%d][%d]th  Element  :",i,j);
                        scanf("%d",&mat1[i][j]);
            }
}
printf("\n  \tEnter  Values  in 3x3 Matrix-II  :\n");
for(i=0;i<3;i++)
{
`           for(j=0;j<3;j++)
            {
                        printf("\tEnter  the  [%d][%d]th  Element  :",i,j);
                        scanf("%d",&mat2[i][j]);
             }
}
for(i=0;i<3;i++)
{
            for(j=0;j<3;j++)
            matsum[i][j]  = mat1[i][j]  + mat2[i][j];
}
 printf("\n  The  values  in Matrix-I  are :\n");
 for(i=0;i<3;i++)
```

```
        {
                for(j=0;j<3;j++)
                printf("\t  [%d][%d] = %d ",i,j,mat1[i][j]);
                printf("\n");
        }
        printf("\n  The  values  in Matrix-II  are :\n");
        for(i=0;i<3;i++)
        {
                for(j=0;j<3;j++)
                printf("\t  [%d][%d] = %d ",i,j,mat2[i][j]);
                printf("\n");
        }
        printf("\n  The  values  in Addition  of Two  Matrices  are :\n");
        for(i=0;i<3;i++)
        {
                for(j=0;j<3;j++)
                printf("\t  [%d][%d] = %d ",i,j,matsum[i][j]);
                printf("\n");
        }
      getch();
      }
```

/*Write a program to Display Given No is Ascending Order*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a[10],i,j,k;
        clrscr();
        printf("\n\n\nEnter  The  10 Element");
        for(i=0;i<10;i++)
        {
        scanf("%d",&a[i]);
        }
        for(i=0;i<9;i++)
        {
                for(j=i+1;j<=9;j++)
                {
                        if(a[i]>a[j])
                        {
                        k=a[i];
                        a[i]=a[j];
                        a[j]=k;
                }
        }
}
        printf("\n\nDisplay  all  Entred  num  in  ascending");
        for(i=0;i<=9;i++)
        printf("\n%d",a[i]);
        getch();
}
```

//Sub of two matrix

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int  a[2][2],b[2][2],c[2][2];  int  i,j,n;
        clrscr();

        printf("enter  your  first  matrix  A=\n");
        for(i=0;i<2;i++)
        for(j=0;j<2;j++)
        scanf("%d",&b[i][j]);
        printf("\n  Enter  your secon matrix  b=\n");
        for(i=0;i<2;i++)
        for(j=0;j<2;j++)
        scanf("%d",&b[i][j]);
        for(i=0;i<2;i++)
        for(j=0;j<2;j++)
        c[i][j]=a[i][j]-b[i][j];
        printf("\n  subtraction  of matrix  a & b=\n");
        for(i=0;i<2;i++)
        {
        for(j=0;j<2;j++)
        {
        printf("%d\t",c[i][j]);
        }
        printf("\n");
        }
getch();
}
```

<u>Multi-Dimensional Array:-</u>

C allows arrays of three or more dimensions. The exact limit is determined by the compiler. The general form of a multi-dimensional array is

Data type array_name[s1][s2][s3]……[sn];

Where s1 is the size of the ith dimension.

Ex:- int a[3]5][12];

A is a three – dimensional array declared to contain 180 integer type element. Similarly table is a four- dimensional array containing 300 elements of floating-point type.

Passing Array Elements to a Function

Array elements can be passed to a function by calling the function by value, or by reference. In the call by value we pass values of array elements to the function, whereas in the call by reference we pass addresses of array elements to the function.

```
/* Demonstration of call by value */
main( )
{
 int i ;
int marks[ ] = { 55, 65, 75, 56, 78, 78, 90 } ;
for ( i = 0 ; i <= 6 ; i++ )
display ( marks[i] ) ;
}
display ( int m )
{
printf ( "%d ", m ) ;
}
```

And here's the output...

```
55 65 75 56 78 78 90
```

Here, we are passing an individual array element at a time to the function **display( )** and getting it printed in the function **display( ).** Note that since at a time only one element is being passed, this element is collected in an ordinary integer variable **m**, in the function **display( ).** And now the call by reference.

```
/* Demonstration of call by reference */
main( )
{
 int i ;
int marks[ ] = { 55, 65, 75, 56, 78, 78, 90 } ;
for ( i = 0 ; i <= 6 ; i++ )
disp ( &marks[i] ) ;
}
disp ( int *n )
{
 printf ( "%d ", *n ) ;
}
```

output...

```
55 65 75 56 78 78 90
```

Here, we are passing addresses of individual array elements to the function display( ). Hence, the variable in which this address is collected (n) is declared as a pointer variable. And since n contains the address of array element, to print out the array element we are using the 'value at address' operator (*).

Read the following program carefully. The purpose of the function disp( ) is just to display the array elements on the screen. The program is only partly complete. You are required to write the function show( ) on your own. Try your hand at it.

```
main( )
{
 int i ;
int marks[ ] = { 55, 65, 75, 56, 78, 78, 90 } ;
for ( i = 0 ; i <= 6 ; i++ )
disp ( &marks[i] ) ;
}
disp ( int *n )
{
 show ( &n ) ;
}
```

<u>String:-</u>

A string is a sequence of characters that is treated as a single data item. We have used string in a number of examples in the past. Any group of characters defined between double quotation marks is a string constant.

<u>Declaring and initializing string variable:-</u>

C does not support string as a data type. However, it allows us to represent string as character arrays. In c therefore, a string variable is any valid c variable name and is always declared as an array of character. The general form of declaration of string variable is.

```
Char string_name[size];
```

The size determines the number of characters in the string_name.

```
Ex:-  char city[10];
      Char name[30];
```

When the compiler assigns a character string to a character array, it automatically supplies a null character ('\0') at the end of the string. Therefore the size should be equal to the maximum number of character in the string plus one. Character array may be initialized when they are declared. C supports the following two forms to initialize the character string.

```
Ex:-    char city[9]=”Mumbai”;
        char city[9]={‘M’,’u’,;m’,’b’,’a’,’i’,’\0’};

ex

main()
{
        Char name[6]=”Bhosle”;
        Clrscr();
        for(i=0;i<=6;i++)
        printf(“My name is %s”,name[i]);
        getch();
}
```

Reading String:-

The familiar input function scanf can be used with %s format specification to read in a string of characters.

```
        Char address[10];

        Scanf(“%s”,address);
```

The problem with the scanf function is that it terminates its input on the first white space it finds. A white space includes blanks, tabs and new lines.

Ex New York

Then only the string “New” will be read into the array address, since the blank space after the word ‘New’ will terminate the reading of string. The scanf function automatically terminates the string that is read with a null character and therefore the character array should be large then the string. In that the ampersand (&) is not required before the variable name.

Writing String:-

We have used extensively the printf function with %s format to print strings to the screen. The format %s can be used to display an array of characters that is terminated by the null character.

Ex:- printf(“%s”,name);

Can be used to display the entire contents of the array name.

```
Main()
{
Int c,d;
Char string[]=”Cprogramming”;
Printf(“\n\n”);
For(c=0;c<=11;c++)
{
D=c+1;
Printf(“|%-12.*s|\n”,d,string);
}
For(c=11;c>=0;c--)
{
D=c+1;
Printf(“|%-12.*s|\n”,d,string);
}
Printf(“..................................“);
}
Getch();
}
```