1.4 PROGRAMMING LANGUAGES:

- Computer programming languages are mainly divided into three categories:

Machine Language: Is a programming language that a particular computer can understand. Also known as **Low-Level language**.

ML is a collection of very detailed instructions that are meant to control the internal circuitry(hardware) of a particular computer.

In that case, every different CPU(computer) has its own unique machine language.

This language is coded in series of binary (0's and 1's).

Every different type of computer has its own unique instruction set. Thus a machine language program written for one type of computer cannot be run on another type of computer.

- **Assembly languages :** This lying between machine languages and high-level languages.

Assembly languages are similar to machine languages, but there are much easier to program in. because they allow programmer to substitute names for numbers whereas machine language consists of only numbers.

In assembly language, a programmer writes instructions using symbolic codes which are meaningful abbreviations

e.g :

A-for addition C-for compare

- L-for load M-for multiply

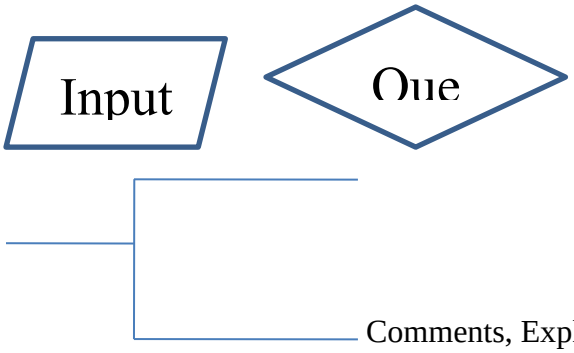
- **High-Level Language**: These are programming languages such as C, FORTRAN or PASCAL that enables a programmer to write programs that are more or less independent of a particular type of computer.

Such languages are considered to be high level language because they are closer to human(natural) languages.

Today, almost all computer programs are written in high-level language, whose instruction set is more compatible with human languages.

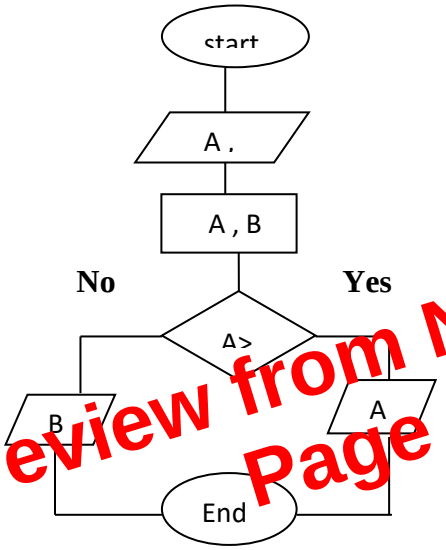
High-level language offers **three** significant advantages over machine language:

- Simplicity



FLOWCHART EXAMPLES:

1. A flowchart to compare two numbers and display the greatest:



CHAP2: C FUNDAMENTALS

2.0..WHAT IS C

- **C** is a general-purpose structured computer programming language.
- This means that you can use **C** to create lists of instructions for a computer to follow.
- **C** is one of the most popular programming languages developed in 1972 by Dennis Ritchie.

2.1..FEATURES OF C:

" Let's look at the elements of the program. **The #include is a preprocessor**" directive that tells the compiler to put code from the header called stdio.h into our program before actually creating the executable. By including header files, you can gain access to many different functions--both the **printf** and **getchar** functions are included in stdio.h. The semicolon is part of the syntax of C. It tells the compiler that you're at the end of a command. You will see later that the semicolon is used to end most commands in C.

The next imporant line is int main(). This line tells the compiler that there is a function named main, and that the function returns an integer, hence int. The "curly braces" ({ and }) signal the beginning and end of functions and other code blocks. If you have programmed in Pascal, you will know them as BEGIN and END. Even if you haven't programmed in Pascal, this is a good way to think about their meaning.

The printf function is the standard C way of displaying output on the screen. The quotes tell the compiler that you want to output the literal string as-is (almost). The '\n' sequence is actually treated as a single character that stands for a newline (we'll talk about this later in more detail); for the time being, just remember that there are a few sequences that, when they appear in a string literal, are actually not displayed literally by printf and that '\n' is one of them. The actual effect of '\n' is to move the cursor on your screen to the next line. Again, notice the semicolon: it is added onto the end of all lines, such as function calls, in C.

The next command is getchar(). This is another function call: it reads in a single character and waits for the user to hit enter before reading the character. This line is included because many compiler environments will open a new console window, run the program, and then close the window before you can see the output. This command keeps that window from closing because the program is not done yet because it waits for you to hit enter. Including that line gives you time to see the program run.

Finally, at the end of the program, we **return a value** from main to the operating system by using the return statement. **This return value** is important as it can be used to tell the operating system whether our program succeeded or not. A return value of 0 means success.

The final brace closes off the function. You should try compiling this program and running it. You can cut and paste the code into a file, save it as a .c file, and then compile it.

Example ‘a’, ‘8’, ‘’ etc.

Character constants have integer values known as **ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) values. For example, the statement **printf(“%c %d”, 65, ‘B’)** will display the characters ‘**A**’ and **66**.

2) String constants

String constants are sequence of characters enclosed within a double quote marks. The string may be a combination of all kinds of symbols.

Example “Hello”, “a”,”UNILAK”

2.6 VARIABLES

When a program is executed, many operations are carried on the data. The data types are integers, real or character constants. These data are stored in the memory and at the time of execution different operations are performed on them.

A variable is a data name used for storing a data value. Its value may be changed during the program execution. The variables value keeps on changing during the execution of a program. In other words, a variable can be assigned different values at different times during the execution of a program.

A variable name may be declared based on the meaning of the operation. Some meaningful variable names are as follows.

Example height, average, sum etc.

Rules for defining variables

1) They must begin with a character without spaces but underscore is permitted.

short int b=2; When a variable is declared without short or long keyword, the default is short-signed int.	**long int c;**

b) Integers, signed and unsigned

Difference between signed and unsigned integers

Signed integer	Unsigned integer
Occupies 2 bytes in memory	Occupies 2 bytes in memory
Range: -32 768 to 32 767	Range: 0 to 65 535
Control string is **%d** or **%I**	Control string **%u**
By default signed int is short-signed int.	By default unsigned int is short unsigned int.
There are also long signed integer having range from - 2 147 483 648 to 2 147 483 647	There are also long unsigned int with range 0 to 4 294 967 295
Example: **int a=2;** **short int b=2;**	Example: **unsigned long b;** **unsigned long int c;** When a variable is declared as unsigned, the negative range of the data type is transferred to positive i.e. doubles the largest size of the possible value. This is due to on declaring unsigned int; the 16th bit is free and not used to store the sign of the number.

2. Char, Signed and Unsigned

Difference between signed and unsigned char

(?).

Write a program to use the conditional operator with two values.

```
#include <stdio.h>

#include <conio.h>


main()

{

clrscr();

printf(“Result as a value =%d”,2= = 3?4:5);

}
```

Output:

Result: 5

Explanation: In the above program the condition 2= =3 is false. Hence, 5 is printed.

Write a program to use the conditional operator with two statements.

```
#include <stdio.h>

#include <conio.h>


main()

{

clrscr();

3>2?printf(“True”):printf(“False);

}
```

is ambiguous. What will happen in this case? The answer is that the C compiler has a convention about the way in which expressions are evaluated: it is called operator precedence. The convention is that some operators are stronger than others and that the stronger ones will always be evaluated first. Otherwise, expressions like the one above are evaluated from left to right: so an expression will be dealt with from left to right unless a strong operator overrides this rule. Use parentheses to be sure. A table of all operators and their priorities is given in the reference section.

Unary Operator Precedence

Unary operators are operators which have only a single operand: that is, they operate on only one object. For instance:

```
++ -- + - &
```

The precedence of unary operators is from right to left so an expression like:

```
*ptr++;
```

would do ++ before *.

Special Assignment Operators ++ and --

C has some special operators which cut down on the amount of typing involved in a program. This is a subject in which it becomes important to think in C and not in other languages. The simplest of these perhaps are the increment and decrement operators:

++

increment: add one to

--

decrement: subtract one from

These attach to any variable of integer or floating point type. (character types too, with care.) They are used to simply add or subtract 1 from a variable. Normally, in other languages, this is accomplished by writing:

variable = variable + 1;

In C this would also be quite valid, but there is a much better way of doing this:

variable++; or

```
int x,y,z;

printf("\n Enter three numbers:");

scanf("%d %d %d",&x,&y,&z);

printf("\n Largest out of three numbers is:");

if(x>y&&x>z)

printf("x=%d",x);

else if(y>x&&y>z)

printf("y=%d",y);

else

printf("z=%d",z);

getch();

}
```


Output:

Enter three numbers: 10 20 30

Largest out of three numbers is: z=30

3.0 Iteration:

Iteration simply means repeating or reiterating the same sequence of commands. And it is more commonly referred to as Looping.

What is a loop?

A loop is defined as a block of statements which are repeatedly executed for a certain number of times.

Steps involved in looping

```
getch();

}
```

Output:

Numbers from 1 to 100 not divisible by 2,3&5:

1 7 11 13 17 19 23 29 31 37 41 43 47 49 53 59 61 67 71 73 77 79 83 89 91 97.

Total Numbers: 26

Example3:

Write a program to display the stars as shown below.

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



```
#include<stdio.h>
main()
{
clrscr();
int x,i,j;
printf("How many lines stars (*) should be printed ?:");
scanf("%d",&x);
for(i=1;i<=x;i++)
```

```
{
   for(j=1;j<=i;j++)
   {
   printf("*");
   }
printf("\n");
}
getch();
```

Output:

How many lines stars (*) should be printed?:5

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


Example4:

Write a program to display the series of numbers as given below.

```
1
1 2
1 2 3

3 2 1
```

```
        printf("\n %d memory locations are reserved for ten 'long' elements",sizeof(l));

        getch();

         }
```

Output:

```
The type of 'int' requires 2 bytes
The type of 'char' requires 1 bytes
The type of 'long' requires 4 bytes
20 memory locations are reserved for ten 'int' elements
10 memory locations are reserved for ten 'char' elements
40 memory locations are reserved for ten 'long' elements
```

Data type & their required bytes.

Data type	Memory Requirement
Character	1 byte
Integer	2 bytes
Float	4 bytes
Long	4 bytes
Double	8 bytes

Character arrays are called **strings**. There is a slight difference between an integer array and a character array: in character array NULL ('\0') character is automatically added at the end where as in integer or other types of arrays, no character is placed at the end.

The NULL character acts as an end of the character array. By using this ***NULL*** character compiler detects the end of the character array. When a compiler reads the NULL character '\0' it ends a character array.

Write a program to display character array with their address.

```
#include <stdio.h>

 #include <conio.h>
```

```
                        if(i==a[j])

                        printf("%3d",a[j]);

                           }

 }

}
```

Output:

```
 Enter how many numbers:3
7 9 3

 Numbers in ascending order:  3  7  9
 Numbers in descending order:  9  7  3
```


Write a program to sort the numbers in ascending order by comparison method

```
#include <stdio.h>

#include <conio.h>

main()

{

int i,j,temp,a[10],n;

printf("\n Enter how many numbers to sort:");

scanf("%d",&n);

for(i=0;i<n;i++)

scanf("%d",&a[i]);

for(i=0;i<n-1;i++)

{
```