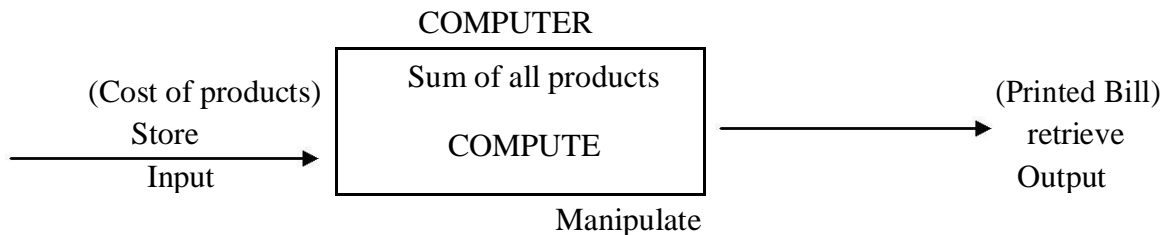


UNIT 1. INTRODUCTION TO COMPUTERS

1.1 Computer Systems:

“A Computer is an electronic device that performs the arithmetical and logical operations.”

For example: To Calculate a bill for a product in shop we use computer there we enter the cost of products. The computer calculates the sum of all products and gives a receipt.



The following are the objects of computer System

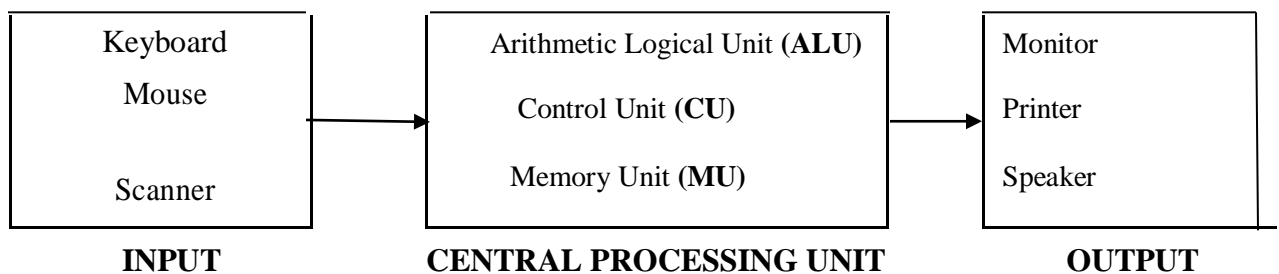
- User (A person who uses the computer)
- Hardware
- Software

1.1 Hardware

Hardware of a computer system can be referred as anything which we can touch and feel.

Example: Keyboard and Mouse.

Block Diagram of Computer system:-



1.INPUT: An input unit are the devices,that are used to interact with the computer system.

Ex:- Keyboard,Mouse.

2.CPU:-(Central Processing Unit)- It consists of the following units -

ALU: It performs the Arithmetic and Logical Operations such as +, -, *, /

(Arithmetic Operators)	
&&,	(Logical Operators)

CU: Every Operation such as storing, computing and retrieving the data should be governed by the control unit.

MU: The Memory unit is used for storing the data. The Memory unit is classified into two types. They are

1. Primary Memory
2. Secondary Memory

PRIMARY MEMORY: The following are the types of memories which are treated as primary

- a) **ROM:** It represents Read Only Memory that stores data and instructions even when the computer is turned off. The contents in the ROM can't be modified once if they are written. It is used to store the BIOS information.
- b) **RAM:** It represents Random Access Memory that stores data and instructions when the computer is turned on. The contents in the RAM can be modified any no. of times by instructions. It is used to store the programs under execution. The size of this memory is of fixed length.
- c) **CACHE MEMORY:** It is used to store the data and instructions referred by processor. It is like a temporary storage.

SECONDARY MEMORY: The following are the different kinds of memories

- a) **Magnetic Storage:** The Magnetic Storage devices store information that can be read, erased and rewritten a number of times. Example: Floppy Disks, Hard Disks, Magnetic Tapes
- b) **Optical Storage:** The optical storage devices that use laser beams to read and write stored data.

Example: CD(Compact Disk), DVD(Digital Versatile Disk).

3. **OUTPUT UNIT:-** It is used to display the output values on the screen.

Ex:- Monitor, LCD screen, speaker, printer.

1.1.2 COMPUTER SOFTWARE

Software of a computer system can be referred as anything which we can feel and see. **Example:** Windows, icons

Computer software is divided into two broad categories:

- (a) System software and
- (b) Application software.

System software manages the computer resources. It provides the interface between the hardware and the users.

Application software, on the other hand is directly responsible for helping users solve their problems.

1.1.2.1 System Software:

System software consists of programs that manage the hardware resources of a computer and perform required information processing tasks. These programs are divided into three classes: the operating system, system support, and system development.

- a) The **operating system** provides services such as a user interface, file and database access, and interfaces to communication systems such as Internet protocols. The primary purpose

of this software is to keep the system operating in an efficient manner while allowing the users access to the system.

- b) **System support software** provides system utilities and other operating services. Examples of system utilities are sort programs and disk format programs. Operating services consist of programs that provide performance statistics for the operational staff and security monitors to protect the system and data.
- c) The last system software category, system **development software**, includes the language translators that convert programs into machine language for execution, debugging tools to ensure that the programs are error free and computer –assisted software engineering (CASE) systems.

1.1.2.2 Application software:

Application software is broken in to two classes: general-purpose software and application specific software.

- a) **General purpose software** is purchased from a software developer and can be used for more than one application. Examples of general purpose software include word processors, database management systems, and computer aided design systems. They are labeled general purpose because they can solve a variety of user computing problems.
- b) **Application –specific software** can be used only for its intended purpose. A general ledger system used by accountants and a material requirements planning system used by a manufacturing organization are examples of application-specific software. They can be used only for the task for which they were designed they cannot be used for other generalized tasks.

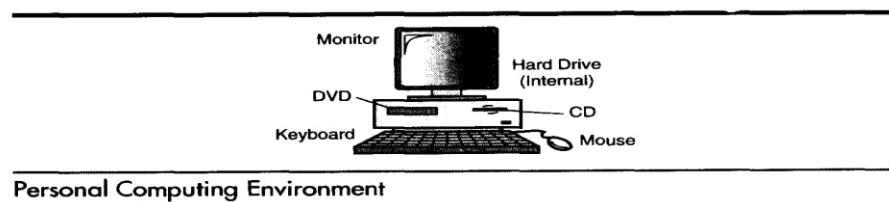
2. COMPUTING ENVIRONMENTS:

The word ‘**compute**’ is used to refer to the process of converting information to data. The following are the different kinds of computing environments available

- a) Personal Computing Environment
- b) Time Sharing Environment
- c) Client/Server Environment
- d) Distributed Computing Environment

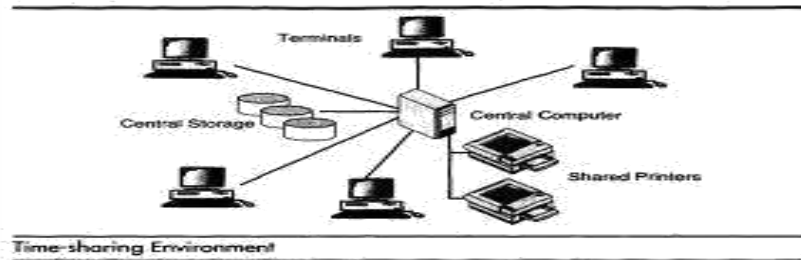
(a) PERSONAL COMPUTING ENVIRONMENT:

In 1971, Marcian E. Hoff, working for INTEL combined the basic elements of the central processing unit into the microprocessor. If we are using a personal computer then all the computer hardware components are tied together. This kind of computing is used to satisfy the needs of a single user, who uses the computer for the personal tasks. **Ex:** Personal Computer



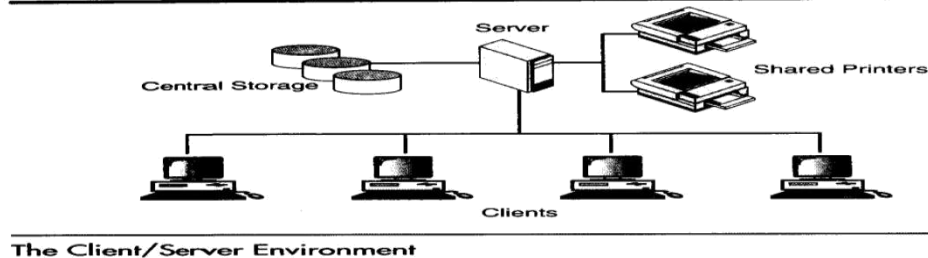
(b) TIME-SHARING ENVIRONMENT:

The concept of time sharing computing is to share the processing of the computer basing on the criteria time. In this environment all the computing must be done by the central computer. The complete processing is done by the central computer. The computer which ask for processing are only dumb terminals.



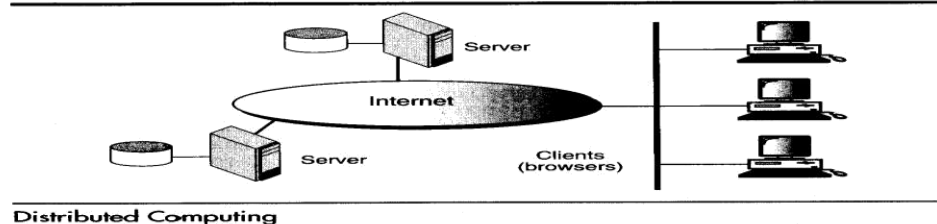
(c) CLIENT/SERVER ENVIRONMENT:

A Client/Server Computing involves the processing between two machines. A client Machine is the one which requests processing. Server Machine is the one which offers the processing. Hence the client is Capable enough to do processing. A portion of processing is done by client and the core (important) processing is done by Server.



(d) DISTRIBUTED COMPUTING:

In distributed computing environment, there are many servers and clients that are connected to each other via internet. In this environment the computing functions of each server and client are integrated so as to provide reliable and scalable computing environment.



3. COMPUTER LANGUAGES:

Computer languages are used for writing programs for the computer. A computer language has been evolved over the years as shown in below table.

Years	Computer Languages
1940's	Machine Languages
1950's	Symbolic Languages
1960's	High Level Languages

3.1 MACHINE LANGUAGE:

Machine language is also known as **machine code or object code**. In the earliest days of computers, the only programming languages available were machine languages. Each computer has its own machine language which is made of streams of 0's and 1's. The instructions in machine language must be in streams of 0's and 1's. This is also referred as binary digits. These are so named as the machine can directly understood the programs

Advantages:

- i High speed execution
- ii The computer can understood instructions immediately
- iii No translation is needed.

Disadvantages:

- i Machine dependent
- ii Programming is very difficult
- iii Difficult to understand
- iv Difficult to isolate an error.

3.2 SYMBOLIC LANGUAGES (OR) ASSEMBLY LANGUAGES:

In the early 1950's Admiral Grace Hopper, a mathematician and naval officer, developed the concept of a special computer program that would convert programs into machine language. These early programming languages simply mirrored the machine languages using symbols or mnemonics to represent the various language instructions. These languages were known as **symbolic languages**. Because a computer does not understand symbolic language it must be translated into the machine language. A special program called an **Assembler** translates symbolic code into the machine language. Hence they are called as Assembly language.

(If a program is written in assemble language, then it comprises of sequence of instructions called **mnemonics**)

Advantages:

- i Easy to understand and use
- ii Easy to modify and isolate error
- iii High efficiency
- iv More control on hardware

Disadvantages:

- i Machine Dependent Language
- ii Requires translator
- iii Difficult to learn and write programs
- iv Less efficient

Example: 2 +3=5 PUSH 2,A
 PUSH 3,B
 ADD A,B
 PRINT C

3.3 HIGH-LEVEL LANGUAGES:

The symbolic languages greatly improved programming efficiency they still required programmers to concentrate on the hardware that they were using working with. Symbolic language was also very tedious because each machine instruction had to be individually coded. The desire to improve programmer efficiency and to change the focus from the computer to the problems being solved led to the development of high-level languages. Ex :- C,C++,Java etc.

Advantages:

- i. Easy to write and understand
- ii. Easy to isolate an error
- iii. Machine independent language
- iv. Easy to maintain Better readability
- v. Low Development cost.

Disadvantages:

- i Needs translator
- ii Requires high execution time
- iii Poor control on hardware .

Difference between High level Languages and Low level Languages:

	High Level Language	Low Level Language
1	Easily understood by the user because it uses statement written in simple English Language	Difficult to understood because it uses machine code that is easily understood by machine rather than a human
2	User-friendly	Machine-friendly
3	Programs are written without detailed knowledge about the internal working of computer.	Programs are written with direct involvement with actual register and interrupt interface to the hardware
4	Compilers and Interpreters are used for translating the instruction into assembly language or machine codes.	Assemblers are used for converting the instruction code (mnemonics) into machine code.
5	Programs written in the HLL are portable	Programs written in LL are non-portable i.e.,they are specific to a particular physical or llogical computer architecture.
6	The maintenance cost is very less	The maintenance cost is very high
7	It provides faster development time.	It provides development time which is 10 to 100 times less than HLL.
8	The memory utilization is very high.	The memory utilization is very low.
9	The execution time is high since the HLL instruction need to be converted into assembly instruction and the machine code.	The execution time is less since the assembly instruction needs to be converted directly into machine code.
10	Examples of HLL are C,C++,Java	Examples of LLL is assemble language like Micro-processor and Micro-controlller

• **Language Translators:**

These are the programs which are used for converting the programs in one language into machine language instructions, so that they can be executed by the computer.

- a) **Compiler:** It is a program which is used to convert the high level language programs into machine language.
- b) **Assembler:** It is a program which is used to convert the assembly level language programs into machine language.
- c) **Interpreter:** It is a program, it takes one statement of a high level language program, translates it into machine language instruction and then immediately executes the resulting machine language instruction and so on.

Comparison between a Compiler and Interpreter

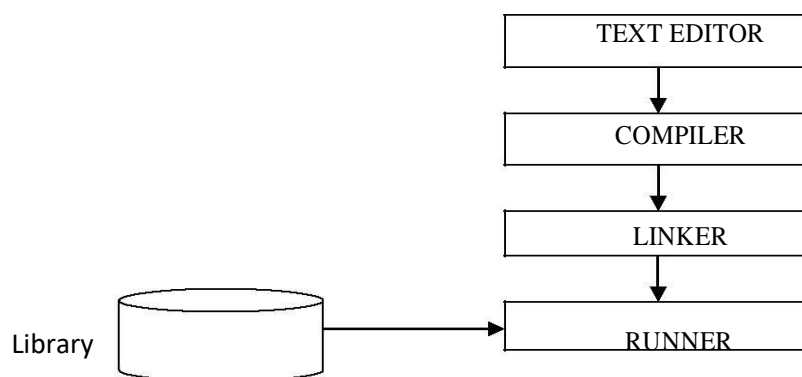
COMPILER	INTERPRETER
A Compiler is used to compile an entire program and an executable program is generated through the object program	An interpreter is used to translate each line of the program code immediately as it is entered
The executable program is stored in a disk for future use or to run it in another computer	The executable program is generated in RAM and the interpreter is required for each run of the program
The compiled programs run faster	The Interpreted programs run slower
Most of the Languages use compiler	A very few languages use interpreters.

4. CREATING AND RUNNING PROGRAMS

The procedure for turning a program written in C into machine Language. The process is presented in a straightforward, linear fashion but you should recognize that these steps are repeated many times during development to correct errors and make improvements to the code.

The following are the four steps in this process :

- i Writing and Editing the program
- ii Compiling the program
- iii Linking the program with the required modules
- iv Executing the program.



4.1 Writing and Editing Programs

The software used to write programs is known as a text editor. A text editor helps us enter, change and store character data. Once we write the program in the text editor we save it using a filename stored with an extension of .C. We refer this file as source code file.

4.2 Compiling Programs

The code in a source file stored on the disk must be translated into machine language. This is the job of the compiler. The Compiler is a computer program that translates the source code written in a high-level language into the corresponding object code of the low-level language. This translation process is called **compilation**. The entire high level program is converted into the executable machine code file. The Compiler which executes C programs is called as C Compiler. **Example:** Turbo C, Borland C, GCC etc.

4.4 Executing Programs

To execute a program we use an operating system command, such as run, to load the program into primary memory and execute it. Getting the program into memory is the function of an operating system program known as the **loader**. It locates the executable program and reads it into memory. When everything is loaded the program takes control and it begins execution.

5. PROGRAM DEVELOPMENT METHOD

- a) Specifying and analyzing the problem statement.
- b) Designing an Algorithm
- c) Coding and Implementation
- d) Debugging
- e) Testing and validating
- f) Documentation and maintenance

- a) **Specifying and analyzing the problem statement:** the problem which has to be implemented into a program must be thoroughly understood before the program is written. Problem must be analyzed to determine the input and output requirements of the program. A problem statement is created with these specifications.
- b) **Designing an Algorithm:** with the problem statement obtained in the previous step, various methods available for obtaining the required solution are analyzed and the best method is designed into algorithm.
- c) **Coding and implementation:** the actual problem is written in the required programming language with the help of information depicted in flow charts and algorithms.
- d) **Debugging:** there is a possibility of occurrence of errors in programs. These errors must be removed to ensure proper working of programs. Hence solving the program without errors is known as **debugging**.

Types of errors that may occur in the program are:

- ✓ **Syntactic Errors (Compilation Errors):** These errors occur due to the usage of wrong syntax for the statements.
 - ✓ **Runtime Errors:** These errors are determined at the execution time of the program.
 - ✓ **Logical Errors:** These errors occur due to incorrect usage of the instructions in the program
- e) **Testing and Validating:** Testing and Validation is performed to check whether the program is producing correct results or not for different values according to user requirement.

- f) **Documentation and Maintenance:** Documentation is the process of collecting, organizing and maintaining, in written the complete information of the program for future references. Maintenance is the process of upgrading the program according to the changing requirements.

6. ALGORITHM / PSEUDOCODE

Algorithm is a method of representing the step by step logical procedure for solving a problem. It is a tool for finding the logic of a problem.

Algorithm Properties:

- a) **Finiteness:** an algorithm must terminate in a finite number of steps.
- b) **Definiteness:** Each step of an algorithm must be clear and easy to understand.
- c) **Effectiveness:** Each step must be effective, in the sense that should be primitive
- d) **Generality:** The algorithm must be complete in itself so that it can be used to solve all problems of a specific type for any input data.
- e) **Input / Output:** Each algorithm must take zero, one or more quantities as input data and produce one or more output values.

6.1 ADVANTAGES OF ALGORITHMS:

- a) It provides the core solution to a given problem. The solution can be implemented on a computer system using any programming language of user's choice.
- b) It facilitates program development by acting as a design document or a blueprint of a given problem solution.
- c) It ensures easy comprehension of a problem solution as compared to an equivalent computer program.
- d) It eases identification and removal of logical errors in a program.
- e) It facilitates algorithm analysis to find out the most efficient solution to a given problem.

6.2 DISADVANTAGES OF ALGORITHMS:

- a) In large algorithms the flow of program control becomes difficult to track.
- b) Algorithms lack visual representation of programming constructs like flowcharts; thus understanding the logic becomes relatively difficult.

6.3 WRITING AN ALGORITHM

An algorithm can be written in English, like sentences and using mathematical formulas. Sometimes algorithm written in English like language is Pseudo code.



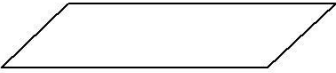
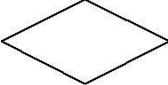
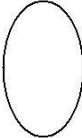
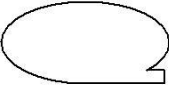
Examples

- 1) Finding the average of three numbers
 1. Let a, b, c are three integers
 2. Let d is float
 3. Display the message "Enter any three integers:"
 4. Read three integers and stores in a, b, c
 5. Compute the $d = (a + b + c) / 3.0$
 6. Display "The avg is:", d
 7. End.

7. FLOW CHART

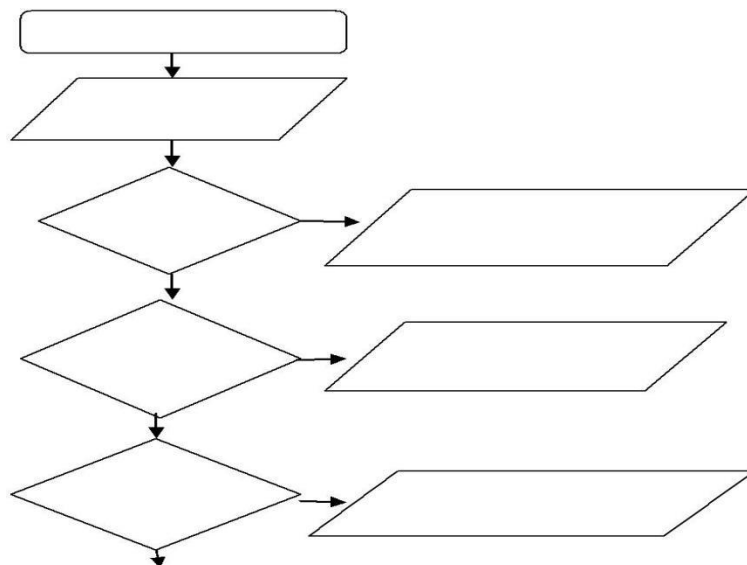
The pictorial representation of an algorithm using some shapes and symbols is known as flowchart. A flow chart is a set of symbols that indicate various operations in the program. For every process, there is a corresponding symbol in the flow chart. Once an algorithm is written, its pictorial representation can be done using flow chart symbols. In other words, a pictorial representation of a textual algorithm is done using a flowchart.

7.2 Symbols in a Flow chart

SYMBOLS	DESCRIPTION
	Start or end of the program
	Computational steps or processing function of a program
	Input or output operation
	Decision making and branching
	Connector or Joining of two parts of program
	Comments

Pseudo code:

Greatest of three
 Read a, b, c;
 if (a>b) and (a>c) print a is greater
 if (b>a) and (b>c) print b is greater
 if (c>a) and (c>b) print c is greater



8. COMPUTER NUMBER SYSTEMS

What are the Number Systems?

Number systems are the technique to represent numbers in the computer system architecture, every value that we are saving or getting into/from computer memory has a defined number system. Computer architecture supports following number systems.

- a) **Binary number system**
- b) **Decimal number system**
- c) **Hexadecimal (hex) number system**

9. BINARY NUMBER SYSTEM

A Binary number system has only two digits that are **0 and 1**. Every number (value) represents with 0 and 1 in this number system. The base of binary number system is 2, because it has only two digits.

Example: Binary Number: **10101₂**

Calculating Decimal Equivalent:

Step	Binary Number	Decimal Number
Step 1	10101 ₂	$((1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2	10101 ₂	$16 + 0 + 4 + 0 + 1_{10}$
Step 3	10101 ₂	21_{10}
Note: 10101 ₂ is normally written as 10101 .		

10. DECIMAL NUMBER SYSTEM

Decimal number system has only ten (10) digits from **0 to 9**. Every number (value) represents with 0,1,2,3,4,5,6, 7, 8 and 9 in this number system. The base of decimal number system is 10, because it has only 10 digits.

Each position represents a specific power of the base 10. For example, the decimal number 1234 consists of the digit 4 in the units position, 3 in the tens position, 2 in the hundreds position, and 1 in the thousands position, and its value can be written as

$$\begin{aligned}
 &(1 \times 1000) + (2 \times 100) \\
 &+ (3 \times 10) + (4 \times 1) \\
 &(1 \times 10^3) + (2 \times 10^2) + \\
 &(3 \times 10^1) + (4 \times 10^0) \\
 &1000 + 200 + 30 + 4 \\
 &1234
 \end{aligned}$$

11. HEXADECIMAL NUMBER SYSTEM

A Hexadecimal number system has sixteen (16) alphanumeric values from **0 to 9** and **A to F**. Every number (value) represents with 0,1,2,3,4,5,6, 7, 8, 9, A, B, C, D, E and F in this number system. The base of hexadecimal number system is 16, because it has 16 alphanumeric values. Here A is 10, B is 11, C is 12, D is 14, E is 15 and F is 16.

Example: Hexadecimal Number:

19FDE₁₆ Calculating
Decimal Equivalent:

Step	Binary Number	Decimal Number
Step 1	19FDE ₁₆	$((1 \times 16^4) + (9 \times 16^3) + (F \times 16^2) + (D \times 16^1) + (E \times 16^0))_{10}$

Step 2	19FDE ₁₆	$((1 \times 16^4) + (9 \times 16^3) + (15 \times 16^2) + (13 \times 16^1) + (14 \times 16^0))_{10}$
Step 3	19FDE ₁₆	$65536 + 36864 + 3840 + 208 + 14_{10}$
Step 4	19FDE ₁₆	106462_{10}

Note : 19FDE₁₆ is normally written as 19FDE

12. NUMBER SYSTEM CONVERSIONS

There are three types of conversion:

a) Decimal Number System to Other Base

[for example: Decimal Number System to Binary Number System]

b) Other Base to Decimal Number System

[for example: Binary Number System to Decimal Number System]

c) Other Base to Other Base

[for example: Binary Number System to Hexadecimal Number System]

12.1 DECIMAL NUMBER SYSTEM TO OTHER BASE

To convert Number system from **Decimal Number System** to **Any Other Base** is quite easy; we have to follow just two steps: Divide the Number (Decimal Number) by the base of target base system (in which we want to convert the number: Binary (2), and Hexadecimal (16)).

Decimal to Binary Conversion				Result
Decimal Number is : (12345) ₁₀				
2	12345	1	LSB	
2	6172	0		
2	3086	0		
2	1543	1		
2	771	1		
2	385	1		
2	192	0		
Binary Number is (11000000111001) ₂				

Decimal to Hexadecimal Conversion				Result	
Example 1: Decimal Number is : (12345) ₁₀				Hexadecimal Number is (3039) ₁₆	
16	12345	9	LSB		
16	771	3			
16	48	0			
8	3	3	MSB		

SHORTCUT METHOD - BINARY TO HEXADECIMAL**Steps**

Step 1 - Divide the binary digits into groups of four starting from the right. **Step 2** - Convert each group of four binary digits to one hexadecimal symbol.

Example: Binary Number : **10101₂**

Calculating hexadecimal Equivalent:

Steps	Binary Number	Hexadecimal Number
Step 1	10101 ₂	0001 0101
Step 2	10101 ₂	¹ 10 ⁵ 10
Step 3	10101 ₂	15 ₁₆

Binary Number : **10101₂** = Hexadecimal Number : **15₁₆**

1. INTRODUCTION TO C PROGRAMMING

C is a general-purpose computer [programming language](#) developed in 1972 by [Dennis Ritchie](#) at the [Bell Telephone Laboratories](#) for use with the [Unix operating system](#). C is a structured programming language, which means that it allows you to develop programs using well-defined *control structures* (you will learn about *control structures* in the articles to come), and provides *modularity* (breaking the task into multiple sub tasks that are simple enough to understand and to reuse). C is often called a [middle-level language](#) because it combines the best elements of low-level or machine language with high-level languages.

Applications of C Language:- The following is a partial list of areas where C language is used:

- Embedded Systems
- Systems Programming
- Artificial Intelligence
- Industrial Automation
- Computer Graphics
- Space Research

Features of C Language/Characteristic of C

- 1) **Simple:** C is a simple language in the sense that it provides structured approach (to break the problem into parts), rich set of library functions, data types etc.
- 2) **Machine Independent or Portable:** Unlike assembly language, c programs can be executed in many machines with little bit or no change. But it is not platform-independent.
- 3) **Mid-level programming language:** C is also used to do low level programming. It is used to develop system applications such as kernel, driver etc. It also supports the feature of high level language. That is why it is known as mid-level language.
- 4) **Structured programming language:** C is a structured programming language in the sense that we can break the program into parts using functions. So, it is easy to understand and modify.
- 5) **Rich Library:** C provides a lot of inbuilt functions that makes the development fast.

- 6) **Memory Management:** It supports the feature of dynamic memory allocation. In C language, we can free the allocated memory at any time by calling the free () function.
- 7) **Speed:** The compilation and execution time of C language is fast.
- 8) **Pointer:** C provides the feature of pointers. We can directly interact with the memory by using the pointers. We can use pointers for memory, structures, functions, array etc.
- 9) **Recursion:** In c, we can call the function within the function. It provides code reusability for every function.
- 10) **Extensible:** C language is extensible because it can easily adopt new features.

TOKENS :

The smallest individual unit in a program is called “*Token*”. Tokens are basic building blocks of C Programming.

Token Example :

Sno	Token Type	Example 1	Example 2
1	Keyword	Do	While
2	Constants	number	Sum
3	Identifier	-76	89
4	String	“HTF”	“PRIT”
5	Special Symbol	*	@
6	Operators	++	/

i) C-Language keywords:

Keywords are those words whose meaning is already defined by Compiler. It cannot be used as Variable Name. There are **32** Keywords in C. C Keywords are also called as **reserved words**.

Ex:-

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

ii) IDENTIFIERS

Identifiers are the names we can give to entities such as variables, functions, structures etc. Identifier names must be unique. They are created to give unique name to a C entity to identify it during the execution of a program. For example:

```
int money;

double account Balance; Here, money and accountBalance are identifiers
```

Also remember, identifier names must be different from keywords. We cannot use **int** as an identifier because **int** is a keyword.

iii) VARIABLES

Variable is a name of memory location where we can store any data. In C, a variable must be declared before it can be used. Variables can be declared at the start of any block of code, but most are found at the start of each function. A declaration begins with the type, followed by the name of one or more variables.

Syntax: DataType Name_of_Variable_Name;

For example: int a, b, c;

Scope of Variables:

- a) Local Variables
- b) Global Variables

A) Local Variables:

A variable that is declared inside the function or block is called **local variable**, and can only be used within that function only. It must be declared at the start of the block.

```
{  
int a,b,c;  
}  
void fun1()  
{  
int x,y,z; }
```

Program: /* Example of Local Variable */

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

```
Void main()
```

```
{  
/* local variable  
declaration */ int  
a, b,c;  
/* actual  
initializatio  
n */ a = 10;  
b = 20;  
c = a + b;  
printf ("value of a = %d, b = %d and c =  
%d\n", a, b, c);  
getch();  
}
```

Output: value of a = 10, b = 20 and c = 30

Here all the variables a, b and c are local to main() function.

B) Global Variable:

A variable that is declared outside the function or block is called **global variable**. It is available to all the functions. It must be declared at the start of the block. The variable is not declared again in the body of the functions which access it.

A program can have same name for local and global variables but value of local variable inside a function will take preference. Following is an example:


```
#include <stdio.h>
/* global variable
declaration */
int g = 20;
void main ()
{
    /* local variable
    declaration */ int
    g = 10;
    clrscr();
    printf ("value of g
    = %d\n", g);
    getch();
}
```

When the above code is compiled and executed, it produces the following result: **Output:** value of g=10

DATA TYPES IN C: -

In C, variable (data) should be declared before it can be used in program. Data types are the keywords, which are used for assigning a type to a variable.

Data types in C

1) Fundamental Data Types or Primitive Data Types

1.1 Integer .

1.2 Floating point.

1.3 Character.

1.4 Double.

1.1 Integer data types:

Integer data type allows a variable to store numeric values. “**int**” keyword is used to refer integer data type.

The storage size of **int** data type is 2 bytes. It varies depend upon the processor in the CPU that we use. If we are using 16 bit processor, 2 byte (16 bit) of memory will be allocated for int data type. Ex- int x,y;

1.2 Floating datatype:

Float data type allows a variable to store decimal values. Storage size of float data type is 4 bytes. This also varies depend upon the processor in the CPU as “**int**” data type. We can use up-to 6 digits after decimal using float data type. For example, 10.456789 can be stored in a variable using float data type. Ex- float avg;

1.3 Character Type:

A single character can be defined as a defined as a character type of data. Characters are usually stored in 8 bits of internal storage. The qualifier signed or unsigned can be explicitly applied to char. While unsigned characters have values between 0 and 255, signed characters have values from –128 to 127. Ex: char name[];

1.4 double:

Double data type is also same as float data type which allows up-to 10 digits after decimal. The range for double datatype is from 1E-37 to 1E+37. Ex- double usn;

sizeof() function in C:

sizeof() function is used to find the memory space allocated for each C data types.

Program Ex:-

```
#include <stdio.h>
void main()
{
    int a;
    char b;
    float c;
    printf("Storage size for int data type:%d \n",sizeof(a));
    printf("Storage size for char data type:%d \n",sizeof(b));
    printf("Storage size for float data type:%d \n",sizeof(c));
}
```

CONSTANTS

Constant in C means the content whose value does not change at the time of execution of a program.

Declare constant

const keyword are used for declare a constant. **Syntax:**const int height = 100;

Example:-

```
#include<stdio.h>
void main()
{
    const a=10;
    printf("%d",a);
    a=20;
    getch();
}
```

In C, constants can be classified as:

- 1.Integer constants.
- 2.Floating-point constants
- 3.Character constants
- 4.String constants
- 5.Enumeration constants.

1.Integer Constant:

Integer constants are the numeric constants (constant associated with number) without any fractional part or exponential part. There are three types of integer constants in C language: decimal constant (base 10), octal constant (base 8) and hexadecimal constant (base 16).

Decimal digits: 0 1 2 3 4 5 6 7 8 9

Octal digits: 0 1 2 3 4 5 6 7

Hexadecimal digits: 0 1 2 3 4 5 6 7 8 9 A B C D E F.

For example:

Decimal constants: 0, -9, 22 etc

Octal constants: 021, 077, 033 etc

2.Floating-point constants

Floating point constants are the numeric constants that has either fractional form or exponent form. For

Example:

-2.0
0.0000234
-0.22E-5
.

3.Character constants

Character constants are the constant which use single quotation around characters. For **Eg:** 'a', 'l', 'm', 'F' etc.

Escape Sequences

Sometimes, it is necessary to use newline (enter), tab, quotation mark etc. in the program which either cannot be typed or has special meaning in C programming. In such cases, escape sequence are used.

For ex: \n is used for newline. The **backslash**(\) causes "escape" from the normal way the characters are interpreted by the compiler.

4.String constants

String constants are the constants which are enclosed in a pair of double-quote marks. **For example:**

"good"//string constant
"" //null string constant
""//string constant of six white space
"x" //string constant having single character.
"Earth is round \n"//prints string with newline

5.Enumeration constants

Keyword **enum** is used to declare enumeration types. **For example:**

```
enum color {yellow, green, black, white};
```

Here, the variable name is color and yellow, green, black and white are the enumeration constants having value 0, 1, 2 and 3 respectively by default.

OPERATORS:

An **operator** is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise etc. There are following types of operators to perform different types of operations in C language.

1. Arithmetic Operators
2. Assignment Operators
3. Relational Operators.
4. Logical Operators
5. Bitwise Operators
6. Ternary or Conditional Operators
7. Increment/Decrement Operators
8. Special Operator.

1. Arithmetic Operators:

C Arithmetic operators are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus in C programs. **Arithmetic Operators**

Arithmetic Operator

Operation

Example

- + Addition or Unary Plus :A+B
- Subtraction or Unary Minus : A-B

* Multiplication :A*B

/ Division :A/B

% Modulus Operator :A%B

Here A,B are known as **operands**. The modulus operator is a special operator in C language which evaluates the remainder of the operands after division.

Example:

```
#include<stdio.h>
void main()
{
    int a, b, sum, sub, mul, div, mod;
    clrscr();
    printf("enter the two numbers");
    scanf ("%d %d", &a, &b);
    sum = a+b;
    printf("\n The Addition is = %d", sum);
    sub = a-b;
    printf("\n The Subtraction is = %d", sub);
    mul = a*b;
    printf("\n The Multiplication is = %d", mul);
    div = a/b ;
    printf("\n The division is = %d", div);
    mod = a%b;
    printf("\n Thu modulus is = %d", mod);
    getch();
}
```

2.Assignment Operators:

In C programs, values for the variables are assigned using assignment operators. For **example**, if the value “10” is to be assigned for the variable “sum”, it can be assigned as “**sum = 10;**”

3.Relational Operators

Relational operators are used to find the relation between two variables. i.e. to compare the values of two variables in a C program.

A simple relational expression contains only one relational operator and takes the following form.

exp1 relational operator exp2

Where **exp1** and **exp2** are expressions, which may be simple constants, variables or combination of them. Relational expressions are used in decision making statements of C language such as if, while and for statements to decide the course of action of a running program.

Example program for relational operators in C:

In this program, relational operator (==) is used to compare 2 values whether they are equal are not. If both values are equal, output is displayed as ” values are equal”. Else, output is displayed as “values are not equal”.

4. Logical Operators

These operators are used to perform logical operations on the given expressions. There are 3 logical operators in C language. They are, logical AND (&&), logical OR (||) and logical NOT (!).

Operators	Name	Example	Description
&&	logical AND	(x>5)&&(y<5)	It returns true when both conditions are true
	logical OR	(x>=10) (y>=10)	It returns true when at-least one of the condition is true
!	logical NOT	!((x>5)&&(y<5))	It reverses the state of the operand.

5 Bitwise Operator

Bitwise AND

In C/C++/Java, the **&** operator is bitwise AND. If any of the bit is **0** then Resultant AND is **0**. The following above chart that defines & defining AND on individual bits.

☐ **Bitwise OR**

If both bits are 0 then Resultant OR is 0

☐ **Bitwise NOT**

Bitwise NOT flips all of the bits

☐ **Bitwise XOR**

The **^** operator is **bitwise XOR**. If Two bits are *same* Then Resultant XOR is **0**. If Two bits are *different* Then Resultant XOR is **1**

6. Conditional (ternary) operators

Conditional operators return one value if condition is true and returns another value if condition is false. This operator is also called as **ternary operator**.

Syntax : (Condition? true_value: false_value);

Example: (A > 100 ? 0 : 1);

In above example, if A is greater than 100, 0 is returned else 1 is returned. This is equal to if else conditional statements.

7. Increment/Decrement Operators

The symbols ++ and -- are called **Increment and Decrement operators**. These two operators are applied to variables only.

Increment operators:

Increment operators are used to increase the value of the variable by one. This operator is placed in two ways.

a) increment operator: After assigning to a variable, its value will be incremented by 1.

syntax `var ++ ; ==> var = var + 1;`

Eg. `x = 5`

`y = x ++, y = 5+1=6`

syntax. `-- var ; ==> var = var - 1;`

Eg. `x = 5`

`y = -- x`

decrement operator:

After assigning to a variable, its value will be decremented by 1.

syntax. `var --; ==> var = var - 1;`

Eg. `x = 5`

`y = x --`

`x = 4 and y = 5`

TYPE CONVERSIONS IN EXPRESSIONS:

When variables and constants of different types are combined in an expression then they are converted to same data type. The process of converting one predefined type into another is called **type conversion**. Type conversion in c can be classified into the following two types:

- a) Implicit type conversion
- b) Explicit type conversion.

a) Implicit type conversion:

An implicit conversion is performed automatically by the compiler when an expression needs to be converted into one of its compatible types. For example, any conversions between the primitive data types can be done implicitly.

`long a = 5; // int implicitly converted to long`
`double b = a; // long implicitly converted to double`

Program:

```
#include <stdio.h>
void main()
{
    int i;
    double d;
    i = 3;
    d = i;
    printf("d = %lf\n", d);
    d = 5.5;
    i = d;
    printf("i = %d\n", i);
    i = 6.6;
    printf("i = %d\n", i);
    getch();
}
```

b)Explicit type conversion:

The type conversion performed by the programmer by posing the data type of the expression of specific type is known as *explicit type conversion*. The explicit type conversion is also known as **type casting**. Type casting in c is done in the following form:

(data_type)expression;

where, ***data_type*** is any valid c data type, and expression may be constant, variable or expression.

For example, `x=(int)a+b*d;`

Program:

```
#include <stdio.h>
void main()
{
    double da,db,dc;
    int result;
    da = 3.3;
    db = 3.3;
    dc = 3.4;
    result = (int)da + (int)db + (int)dc;
    printf("result is =%d", result);
    getch();
}
```

C Expression Evaluation

In the C programming language, an expression is evaluated based on the operator precedence and associativity. When there are multiple operators in an expression, they are evaluated according to their precedence and associativity. The operator with higher precedence is evaluated first and the operator with the least precedence is evaluated last.

An expression is evaluated based on the precedence and associativity of the operators in that expression.

To understand expression evaluation in c, let us consider the following simple example expression...

10 + 4 * 3 / 2

In the above expression, there are three operators +, * **and** /. Among these three operators, both multiplication and division have the same higher precedence and addition has lower precedence. So, according to the operator precedence both multiplication and division are evaluated first and then the addition is evaluated. As multiplication and division have the same precedence they are evaluated based on the associativity. Here, the associativity of multiplication and division is **left to right**. So, multiplication is performed first, then division and finally addition. So, the above expression is evaluated in the order of * / **and** +. It is evaluated as follows...

4	*	3	====>	12
12	/	2	====>	6
10	+	6	====>	16

The expression is evaluated to **16**.

C Operator Precedence and Associativity

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, $x = 7 + 3 * 2$; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with $3*2$ and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] -> . ++ --	Left to right
Unary	+ - ! ~ ++ -- (type)* & sizeof()	Right to left
Multiplicative	* / %	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right

Category	Operator	Associativity
Bitwise OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right

Example :-

```
#include <stdio.h>
void main( )
{
    int a = 20;
    int b = 10;
    int c = 15;
    int d = 5;
    int e;
    e = (a + b) * c / d; // ( 30 * 15 ) / 5
    printf("Value of (a + b) * c / d is : %d\n", e );
    e = ((a + b) * c) / d; // (30 * 15) / 5
    printf("Value of ((a + b) * c) / d is : %d\n", e );
    e = (a + b) * (c / d); // (30) * (15/5)
    printf("Value of (a + b) * (c / d) is : %d\n", e );
    e = a + (b * c) / d; // 20 + (150/5)
    printf("Value of a + (b * c) / d is : %d\n", e );
}
```