

## MODULE-I

### HISTORY OF COMPUTERS

A **Computer** is an advanced electronic device that takes raw data as an input from the user and processes it under the control of a set of instructions (called programs), produces a result (output), and saves it for future use.

#### Evolution of computers

The need for a device to do calculations along with the growth in commerce and other human activities leads to the evolution of computers.

**Sand Tables:** In sand table pebbles are used to represent numbers and it is known to be the earliest device of computations.

**Abacus:** An extensive modification of sand table results in a device called Abacus. An abacus consists of sliding beads arranged on a rack. The arithmetic operations such as addition and subtraction can be performed by positioning the beads appropriately.

**Naiper Bones:** This was a remarkable invention of John Naiper, a Scottish Mathematician, which enabled to perform the multiplication and division operations by converting into simple addition and subtraction operations.

**Slide Rule:** The invention of logarithms influenced the development of Slide rule. It was based on the principle that actual distances from the starting point of the rule is directly proportional to the logarithm of the numbers printed on the rule.

**Pascaline:** It was invented by Blaise Pascal, and was the first functional calculator. It had a complex arrangement of wheels, gears and windows for displaying numbers.

**Stepped Reckoner:** It was the first mass-produced calculating device, which was designed to perform multiplication by repeated additions. But it was not very reliable.

**Punch Card System:** In 1801, Joseph Jacquard, a French textile weaver, invented a power loom with an automatic card reader. The idea of using punched cards to communicate with machines was an important step in the development of computers. The presence or absence of a hole in a punched card represented the two digits of the binary system, which is the base for all modern digital computers.

**Difference Engine:** Charles Babbage in 1822 designed and built a model called Difference Engine. His invention could perform calculations without human intervention. Hence is popularly named as Father of Computers.

**Analytical Engine:** It was also designed by Babbage and is considered to be the first general purpose programmable calculator. It provided base to the technology of modern computers. It had an arithmetic unit to perform calculations and mechanism to store results and instructions.

#### Some early Computers

The below are some well-known computers of the past, which are considered to be predecessors of modern computers.

1. MARK-I Computer
2. ABC (Atanasoff Berry Computer)
3. Colossus
4. ENIAC (Electronic Numerical Integrator and Calculator)
5. EDVAC (Electronic Discrete Variable Automatic Computer)
6. EDSAC (Electronic Delay Storage Automatic Computer)
7. UNIVAC (UNiversal Automatic Computer)

These early computers used vacuum tubes as their basic electronic component. Today new generations of computers were considerably smaller, faster, and less expensive than previous ones.

Using today's technology, the entire circuitry of a computer processor can be packaged in a single electronic component called a **CPU** or **microprocessor chip**. These chips are installed in watches, PDS's (Personal Digital Assistance), GPS systems, cameras, home appliances, automobiles, and of course computers.

#### Generation of Computers

Evolution of Modern Computers from the olden days is known as "Generation of Computers." Generations of computers are broadly classified based on the following characteristics:

- Increasing in storage capacity.
- Increasing in processing speed.
- Increasing reliability.

There are totally 5 generations of computers till today.

#### First Generation Computers:

**Period:** (1945-1955)

**Technology:** Vacuum tubes

The ENIAC was the first computer developed in this generation. It was capable of performing 5000 additions (or) 350 multiplications per second. It uses about 18000 vacuum tubes and it consumes 150 kw/hr.

**Limitations:** The limitations of first-generation computers are as follows.

- Less operating capacity.
- High power consumption.
- Very large space requirement.
- Produce high temperature.
- Very costly.

**Second Generation Computers:****Period:** (1955-1965)**Technology:** Transistors

- The second-generation computers use transistors as the main component in CPU. They are very small in size when compared to vacuum tubes and produce less heat. They are fast and reliable.
- Due to this invention of Transistors the size, maintenance cost, power consumption has decreased.
- During this period magnetic storage devices have been started their development. Because of this, speed and storage capacity has been increased. They are capable to perform 20,000 to 50,000 additions per second.

**Third Generation Computers:****Period:** (1965-1975)**Technology:** Integrated Circuits (ICs)

- In this generation the computers used integrated circuits instead of Transistors
- Integrated circuit is a miniature form of an electronic circuit made of silicon and enclosed in a metal package.
- These IC's are called "chips." The cost and size of the computers were greatly reduced. The magnetic disk technology has improved rapidly. It is capable to perform 10 million additions per second.

**Fourth Generation Computers:****Period:** (1975-1990)**Technology:** Very Large-Scale Integrated Circuit (VLSI)

- IC's packing nearly 10,000 transistors are grouped in to a single silicon chip known as "microprocessor". The computers which use microprocessors are called "Micro Computers".
- Intel Corporation invented the microprocessor in the year 1980 with this development the cost of a computer has reduced a lot.
- The floppy disk technology was developed during this generation.

**Fifth Generation Computers:****Period:** (1990- till date)**Technology:** Artificial Intelligence

- Artificial Intelligence is a technique by which we make the computer to think and take decisions in its own.
- These computers are under research.
- Artificial Intelligence can be achieved by means of problem solving, Game playing, and Expert systems.

**Types of Computers**

Computers can be generally classified by size and power, although there can be considerable overlap. Following are descriptions of several different types of computers.

**Mainframe computers** are large-sized, powerful multi-user computers that can support concurrent programs. That means, they can perform different actions or 'processes' at the same time. Mainframe computers can be used by as many as hundreds or thousands of users at the same time. Large organizations may use a mainframe computer to execute large-scale processes such as processing the organization's payroll.

**Mini-computers** are mid-sized multi-processing computers. Again, they can perform several actions at the same time and can support from 4 to 200 users simultaneously. In recent years the distinction between mini-computers and small mainframes has become blurred. Often the distinction depends upon how the manufacturer wants to market its machines. Organizations may use a mini-computer for such tasks as managing the information in a small financial system or maintaining a small database of information about registrations or applications.

**Workstations** are powerful, single-user computers. They have the capacity to store and process large quantities of data, but they are only used by one person at a time. However, workstations are typically linked together to form a computer network called a local area network, which means that several people, such as staff in an office, can communicate with each other and share electronic files and data.

A workstation is similar to a personal computer but is more powerful and often comes with a higher-quality monitor. In terms of computing power, workstations lie in between personal computers and mini-computers. Workstations commonly support applications that require relatively high-quality graphics capabilities and a lot of memory, such as desktop publishing, software development and engineering applications.

**Personal computers (PCs)**, also called microcomputers, are the most popular type of computer in use today. The PC is a small-sized, relatively inexpensive computer designed for an individual user. Today, the world of PCs is basically divided between IBM-compatible and Macintosh-compatible machines, named after the two computer manufacturers. Computers may be called 'desktop' computers, which stay on the desk, or 'laptop' computers, which are lightweight and portable. Organizations and individuals use PCs for a wide range of tasks, including word processing, accounting, desktop publishing, preparation and delivery of presentations, organization of spreadsheets and database management. Entry-level PCs are much more powerful than a few years ago, and today there is little distinction between PCs and workstations.

**Supercomputers:** Largest capacity and fastest computers, used by research laboratories and computationally number crunching application such as weather forecasting.

## BASIC ORGANIZATION OF A COMPUTER

A **Computer** is an advanced electronic device that takes raw data as an input from the user and processes it under the control of a set of instructions (called programs), produces a result (output), and saves it for future use.

### Working of a Computer

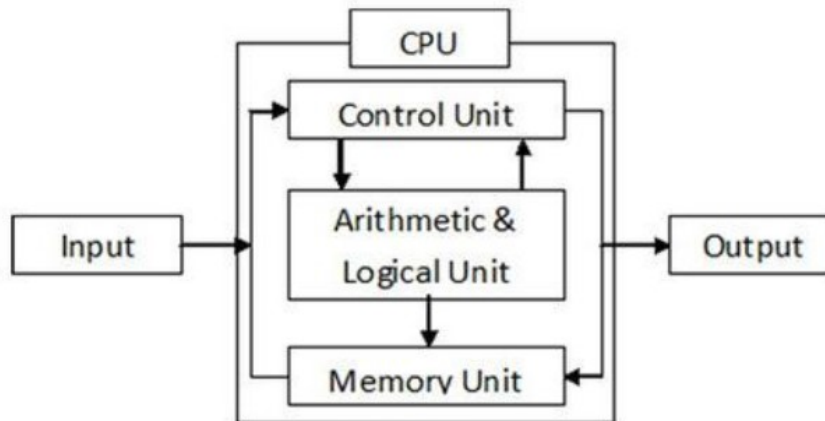
**Step 1 – Takes Input** → The process of entering data and instructions into the computer system.

**Step 2 – Store Data** → Saving data and instructions so that they are available for processing as and when required.

**Step 3 – Processing Data** → Performing arithmetic, and logical operations on data in order to convert them into useful information.

**Step 4 – Generates Output** → The process of producing useful information or results for the user, such as a printed report or visual display.

**Step 5 – Control the Flow** → Directs the manner and sequence in which all of the above operations are performed.



### Components of a Computer

In general, a computer system consists of the following components:

- Input Unit
- Central Processing Unit
- Output Unit

**Input Unit:** The process of sending the data and Instructions for the processing through some suitable devices such as Keyboard, Mouse etc. is called Input. The devices translate the data from human understandable form into electronic impulses which are understood by the computer.

**Central Processing Unit (CPU):-** Once the data accepted it fed in to Central Processing Unit before the output is generated as data has to be processed, which is done by CPU. This unit of the computer is the brain of computer system, which does all the processing, calculations, problem solving and controls all other functions of all other elements of the computer. The CPU consists of the following three distinct units namely.

1. Memory Unit (MU)
2. Control Unit (CU)
3. Arithmetic and Logic Unit (ALU)

1. **Memory Unit:** Which holds the data in in terms of Program and files. The data stored can be accessed and used whenever required by the CPU for necessary processing. This memory unit is usually referred as primary storage section. The units in which memory unit is measured are known as BYTES. BYTE is the space required to store 8 characters or alphabet or digits to any other special character.

- 1 Byte = 8 Bits.
- 1 Kilobyte = 1024 Bytes
- 1 Megabyte = 1024 Kilobytes
- 1 Gigabyte = 1024 Megabytes
- 1 Terabyte = 1024 Giga bytes

Where Bits are spaces required to store one Binary digit i.e. either 0 or 1.

2. **Control Unit:** This unit which coordinates all the activities of each and every element of computer. It decodes the instructions given by various users and it sends commands and signals that determine the sequence of various instructions. Through this unit does not process data but it acts as the central system for data manipulation, as it controls the flow of data to and from the main storage.

3. **Arithmetic and Logic Unit:** - This unit performs arithmetic operations such as addition, subtraction, multiplication and division. It also does Logical Operations such as comparison of numbers etc. Thus this unit helps by processing data and taking logical decisions.

**Output Unit:** The processing of extracting the data from CPU through some suitable devices is called Output. The commonly used output devices are VDU, Printers, Plotter, magnetic media like floppy, hard disks etc.

## WHAT IS A PROGRAM

A computer program is a collection of instructions that performs a specific task when executed by a computer. A computer program is usually written by a computer programmer in a programming language.

**Software** is a collection of computer programs and related data that provides the instructions for a computer what to do and how to do it.

There are four key types of software programs to consider, including:

**Application:** Application software carries out one specific task on a computer.

**System:** A system software design allows software applications to run on a computer's hardware. This software can include operating systems like Windows, macOS and Linux.

**Programming:** Programming software can allow the user to create an application. These programs provide individuals with the tools to develop, write, test and debug programming code.

**Driver:** Driver software is a software program that enables the communication between the operating system and a hardware device. Some hardware requires a driver installation before its usage, like printers, scanners and modems.

## WHAT IS PROGRAMMING

Computer Programming is a step-by-step process of designing and developing various sets of computer programs to accomplish a specific computing task. The purpose of computer programming is to find a sequence of instructions that solve a specific problem on a computer.

There is no. of programming languages, which can be used to write computer programs and following are a few of them – C, C++, Java, Python, PHP, Perl, Ruby etc.

Computer programs are being used in almost every field, household, agriculture, medical, entertainment, defense, communication, etc. Listed below are a few applications of computer programs –

- MS Word, MS Excel, Adobe Photoshop, Internet Explorer, Chrome, etc., are examples of computer programs.
- Computer programs are being used to develop graphics and special effects in movie making.
- Computer programs are being used to perform Ultrasounds, X-Rays, and other medical examinations.
- Computer programs are being used in our mobile phones for SMS, Chat, and voice communication.

## PROGRAMMING LANGUAGES

A programming language is a computer language that is used by programmers (developers) to communicate with computers. It is a set of instructions written in any specific language ( C, C++, Java, Python) to perform a specific task.

### Types of programming languages

#### 1. Low-level programming language

Low-level language is machine-dependent (0s and 1s) programming language. Low-level language is further divided into two parts -

*i. Machine Language:* Machine language is easier to read because it is normally displayed in binary or hexadecimal form (base 16) form. It does not require a translator to convert the programs because computers directly understand the machine language programs. The advantage of machine language is that it helps the programmer to execute the programs faster than the high-level programming language.

*ii. Assembly Language:* Assembly language (ASM) is also a type of low-level programming language that is designed for specific processors. It represents the set of instructions in a symbolic and human-understandable form. It uses an assembler to convert the assembly language to machine language. The advantage of assembly language is that it requires less memory and less execution time to execute a program.

#### 2. High-level programming language

High-level programming language (HLL) is designed for developing user-friendly software programs and websites. This programming language requires a compiler or interpreter to translate the program into machine language (execute the program). The main advantage of a high-level language is that it is easy to read, write, and maintain.

High-level programming language includes Python, Java, JavaScript, **PHP, C#, C++, Objective C, Cobol, Perl, Pascal, LISP, FORTRAN, and Swift programming language.**

## ALGORITHM

**Definition:** - An algorithm is defined as a finite set of steps that provide a chain of actions for solving a problem. Algorithm is a step-by-step method of solving a problem.

### Characteristics of an Algorithm:

- 1) Finiteness: - An algorithm terminates after a finite number of steps.
- 2) Definiteness: - Each step-in algorithm is unambiguous. This means that the action specified by the step cannot be interpreted (explain the meaning of) in multiple ways & can be performed without any confusion.
- 3) Input: - An algorithm accepts zero or more inputs
- 4) Output: - An algorithm should produce at least one output.

5) Effectiveness: - It consists of basic instructions that are realizable. This means that the instructions can be performed by using the given inputs in a finite amount of time.

#### Advantages of Algorithms:

1. It is a stepwise representation of a solution to a given problem, which makes it easy to understand.
2. An algorithm uses a definite procedure.
3. It is not dependent on any programming language, so it is easy to understand for anyone even without programming knowledge.
4. Every step in an algorithm has its own logical sequence so it is easy to debug.
5. By using algorithm, the problem is broken down into smaller pieces or steps hence, it is easier for programmer to convert it into an actual program.

#### Disadvantages of Algorithms:

1. Algorithms is Time consuming.
2. Difficult to show Branching and Looping in Algorithms.
3. Big tasks are difficult to put in Algorithms.

#### Example:

**Problem** – Design an algorithm to add two numbers and display the result.

#### Algorithm: -

**Step 1** – START

**Step 2** – declare three integers **a**, **b** & **c**

**Step 3** – define values of **a** & **b**

**Step 4** – add values of **a** & **b**

**Step 5** – store output of step 4 to **c**

**Step 6** – print **c**

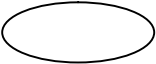


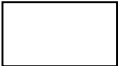
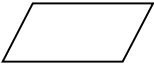

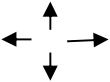
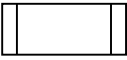

**Step 7** – STOP

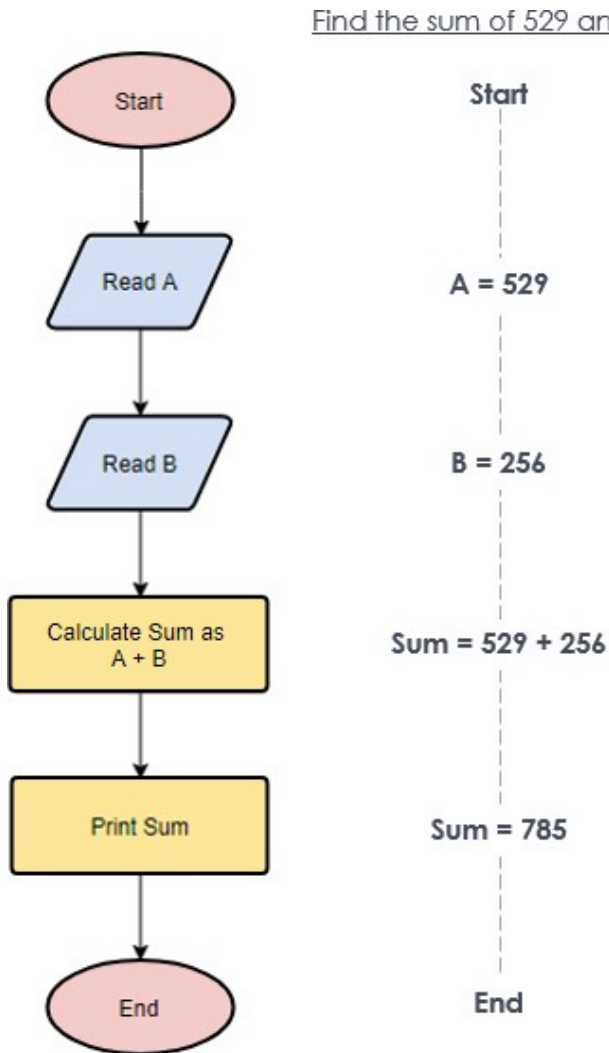
#### FLOWCHART

**Definition:** - Flowchart is a diagrammatic or graphical or pictorial representation of various steps involved in the solution of the problem.

Flowchart also represents the flow of data. It involves a set of symbols that indicates various operations or processes in solving a problem. For every process there is a corresponding symbol in the flowchart. Once the algorithm is written, its pictorial representation can be done using this flowchart symbols.

Some of the commonly used **Flowchart Symbols** are listed below:

Symbol	Description
	<b>Start/Stop</b> The terminator symbol represents the starting or ending point of the system.
	<b>Decision</b> A diamond represents a decision or branching point. Lines coming out from the diamond indicate different possible situations, leading to different sub-processes.
	<b>Connector</b> This symbol would contain a letter inside. It indicates that the flow continues on a matching symbol containing the same letter somewhere else.
	<b>Process</b> A box indicates some particular operation.
	<b>Input/output</b> It represents information reading (input) or writing (output) in problem solving.
	<b>Loop</b> It represents the repeated information in problem solving.
	<b>Arrows</b> Lines represent the flow of the sequence and direction of a process.
	<b>Predefined Process</b> It used to represent some predefined process like functions.
	<b>Document</b> This represents a printout, such as a document or a report.

**Example:****Problem** – Design a flowchart to add two numbers and display the result.**Flow Chart:** -**Design an algorithm to Exchange the values of Two Variables by using a third variable.****Algorithm:**

Step-1: START

Step-2: Read two Variables as A &amp; B

Step-3: Define a third Variable as T

Step-4: Print Values of A &amp; B before Exchange

Step-5: Copy Value of A to T ( $T \leftarrow A$ )Step-6: Copy Value of B to A ( $A \leftarrow B$ )Step-7: Copy Value of T to B ( $B \leftarrow T$ )

Step-8: Print Values of A &amp; B after Exchange

Step-9: STOP

**Design an algorithm to Exchange the values of Two Variables without using a third variable.****Algorithm:**

Step-1: START

Step-2: Read two different values for Variables X &amp; Y

Step-3: Print Values of X &amp; Y before Exchange

Step-4: Store the value of  $X+Y$  in X ( $X \leftarrow X+Y$ )Step-5: Store the value of  $X-Y$  in Y ( $Y \leftarrow X-Y$ )Step-6: Store the value of  $X-Y$  in X ( $X \leftarrow X-Y$ )

Step-7: Print Values of X &amp; Y after Exchange

Step-8: STOP

**Design Algorithm and Flowchart to check whether the given number is Even or Odd.****Algorithm:**

Step-1: START

Step-2: Read a number as N

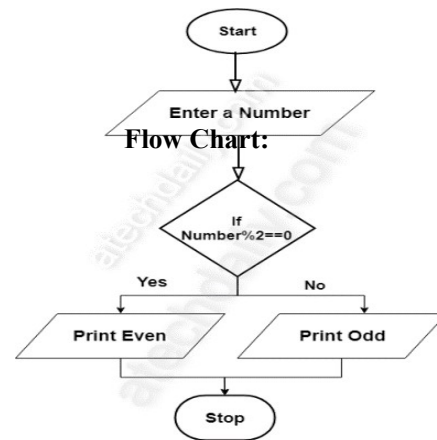
Step-3: Find remainder R of dividing N by 2

Step-4: If R is equals to 0 then Goto Step-6

Step-5: Print N is Odd then Goto Step-7

Step-6: Print N is Even

Step-7: STOP

**PSEUDO CODE**

Pseudo is a methodology that allows the programmer to represent the implementation of an algorithm that is independent of any programming language. It uses natural language like English, that is easily understood by humans.

**Example:** Let's consider an example of implementing a simple program that calculates the average of three numbers using pseudo code in C.

**Pseudo Code:**

1. Start
2. Input three numbers
3. Calculate the sum of the three numbers
4. Divide the sum by 3 to get the average
5. Display the average

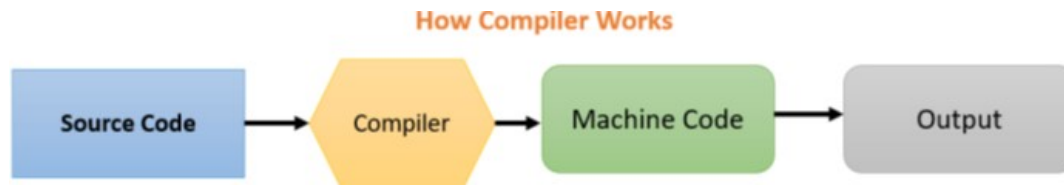
6. End

### ***Advantages of Pseudocode***

- It improves the readability of any approach.
- IT acts as a bridge between the program and the algorithm or flowchart.
- It can explain what exactly each line of a program should do, hence making the code construction phase easier for the programmer.

## **INTRODUCTION TO COMPILATION & EXECUTION**

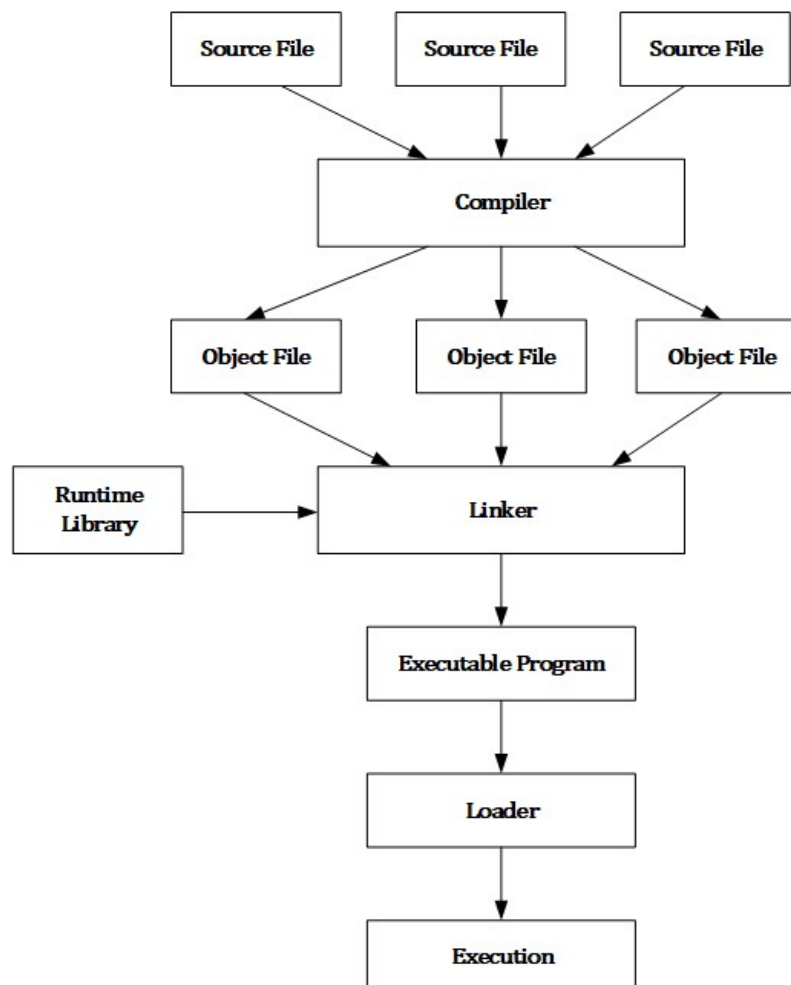
The compilation process in C is converting an understandable human code into a Machine understandable code and checking the syntax and semantics of the code to determine any syntax errors or warnings present in our C program. The compilation process done by a special program called COMPILER.



In Compilation process, the source code goes through several steps before it becomes an executable program.

- In the first step the source code is checked for any syntax errors.
- After the syntax errors are traced out the source file is passed through a compiler which first translates high level language into object code (A machine code not ready to be executed).
- A linker then links the object code with pre-compiled library functions, thus creating an executable program.
- This executable program is then loaded into the memory for execution.

General compilation process is shown in Figure below:





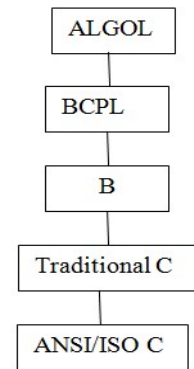
## INTRODUCTION TO C

C language facilitates a very efficient approach to the development and implementation of computer programs. The History of C started in 1972 at the Bell Laboratories, USA where Dennis M. Ritchie proposed this language. In 1983 the American National Standards Institute (ANSI) established committee whose goal was to produce “an unambiguous and machine independent definition of the language C “while still retaining its spirit.

C is the programming language most frequently associated with UNIX. Since the 1970s, the bulk of the UNIX operating system and its applications have been written in C. Because the C language does not directly rely on any specific hardware architecture, UNIX was one of the first portable operating systems. In other words, the majority of the code that makes up UNIX does not know and does not care which computer it is actually running on. Machine-specific features are isolated in a few modules within the UNIX kernel, which makes it easy for you to modify them when you are porting to different hardware architecture.

C was first designed by Dennis Ritchie for use with UNIX on DEC PDP-11 computers. The language evolved from Martin Richard's BCPL, and one of its earlier forms was the B language, which was written by Ken Thompson for the DEC PDP-7. The first book on C was *The C Programming Language* by Brian Kernighan and Dennis Ritchie, published in 1978.

In 1983, the American National Standards Institute (ANSI) established a committee to standardize the definition of C. The resulting standard is known as *ANSI C*, and it is the recognized standard for the language, grammar, and a core set of libraries. The syntax is slightly different from the original C language, which is frequently called K&R for Kernighan and Ritchie. There is also an ISO (International Standards Organization) standard that is very similar to the ANSI standard. It appears that there will be yet another ANSI C standard officially dated 1999 or in the early 2000 years; it is currently known as "C9X."

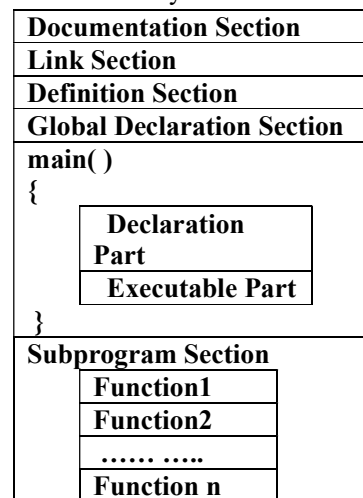


## BASIC STRUCTURE OF A C PROGRAM

Every C program consists of one or more modules called functions. One of the functions must be called **main**. The program will always begin by executing the **main** function, which may access other functions. Any other function definitions must be defined separately, either before or after **main**.

A C program may contain one or more sections as given below.

- The **Documentation section** consists a set of comment lines that gives the name of the program, author and other details. The comments must be enclosed within the pair `/* ..... */`
- The **Link Section** provides instructions to the compiler to link functions from the system library. This is performed by pre-processor directive such as `"#include"`.
- The **Definition section** defines all symbolic constants by using `#define` directive.
- The **Global Declaration** section contains the variables that are global, to be used in more than one function. Global variables are known throughout the program. The variables hold their values throughout the program's execution.
- Every program have one **main( ) section**, it contains two parts, Declaration part and Executable part. The declaration part declares all the variables used in the executable part. There is at least one executable part in the program. These two parts must appear between the opening `{` and closing `}` braces.
- The **subprogram section** contains all the user-defined functions that are called in the main function.



## KEYWORDS IN C

Keywords are predefined, reserved words used in programming that have special meanings to the compiler. Keywords are also known as ‘Reserved Words’. These words cannot be used as variable names, function names or as a constant. There are 32 keywords available in C. The list of key word is listed below:

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

All the reserved words appear in lower case, they have special meaning in C and cannot be used for other purpose.



## IDENTIFIERS IN C

Identifiers are the user defined names used to identify the variables, symbolic constants, functions, arrays etc. There are certain rules regarding identifier names, as mentioned below:

- Identifier must be a sequence of alphabets, digits and under score
- The first character must be an alphabet or Under score.
- Identifier names should not be the same as keywords.
- C is case sensitive (ie. Upper- and lower-case letters are treated differently). Thus, the names **rate**, **Rate** and **RATE** denote different identifiers.
- No commas or blanks are allowed within an identifier.
- No special symbol other than an underscore can be used in an identifier.

## VARIABLES IN C

C variable is a named location in a memory where a program can manipulate the data. This location is used to hold the value of the variable. The value of the C variable may get change in the program. C variable might be belonging to any of the data type like int, float, char etc.

### Rules for Naming Variables in C are:

- Variable must be a sequence of alphabets, digits and under score
- The first character must be an alphabet or Under score.
- Variable names should not be the same as keywords.
- C is case sensitive (ie. Upper- and lower-case letters are treated differently). Thus, the names **rate**, **Rate** and **RATE** denote different variables.
- No commas or blanks are allowed within a variable name.
- No special symbol other than an underscore can be used in a variable name.

## VARIABLE DECLARATION

In C, it is required declare a variable before using it. When a variable is declared, the compiler allocates some memory based on the type declared and assign some random garbage value till it is not initialized.

The syntax for declaring a variable is as follows:

**Data-type var1, var2, ....., varn;**

**var1, var2....., varn** are the names of variables. Variable are separated by commas. A declaration statement must end with a semicolon.

For **example**, valid declarations are:

*int rno;*

*float avg, total;*

*char st\_name;*

## VARIABLE INITIALIZATION

When declaring a variable, the compiler by default assigns some random garbage value to it. The process of assigning some desired value at the time of declaring a variable is known as Variable Initialization.

The syntax for initializing a variable is as follows:

**Data-type variable-name = value;**

It is also possible to declare and initialize multiple variables at the same time as given below:

**Data-type var1 = val1, var2 = val2, ....., varn = valn;**

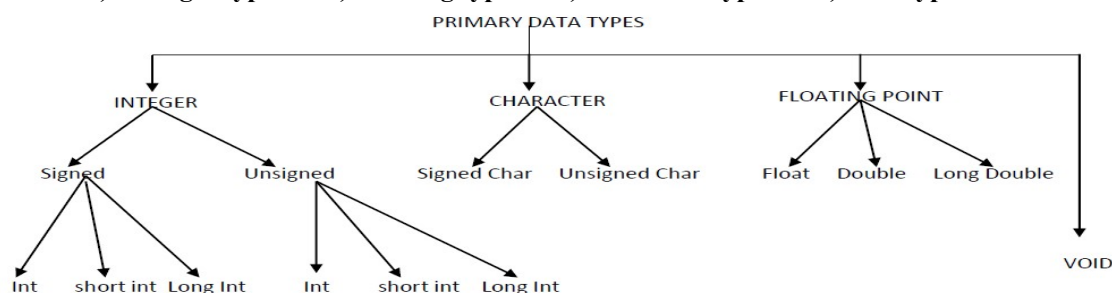
**Example:** *int a = 10, b = 20, c;*

## DATA TYPES IN C

In the C programming language, data types refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted. C data types are defined as the data storage format that a variable can store a data to perform a specific operation. Data types are used to define a variable before to use in a program.

Fundamental Data Types in C:

- 1) Integer types
- 2) Floating types
- 3) Character types
- 4) Void type



## Integer & Character Types

Following table give details about standard integer types with its storage sizes and value ranges:

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

## Floating-Point Types

Following table gives you details about standard floating-point types with storage sizes and value ranges and their precision:

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

## Void Type

Void is an empty data type that has no value. This data type is typically used in the definition and declaration of functions to indicate that either nothing is being passed in and/or nothing is being returned.

## CONSTANTS IN C

Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **Literals**.

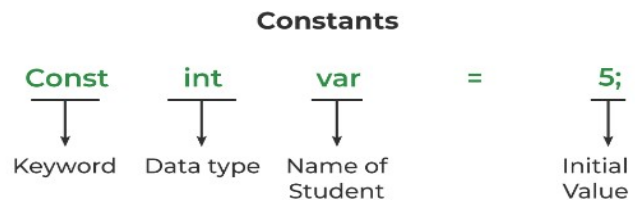
The **const** keyword can be used in variable declaration to define a C Constant. Once defined, it will be a read-only variable whose values cannot be modified in the C program.

### Syntax to Define Constant

***const data\_type var\_name = value;***

Constants can be of any of the basic data types like—

- 1) an integer constant (literal),
- 2) a floating constant (literal),
- 3) a character constant (literal), or
- 4) a string literal.



### 1) Integer Constants

An integer literal can be a decimal, octal, or hexadecimal constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and nothing for decimal.

An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order.

<i>Here are some examples of integer literals –</i>	<i>Following are other examples of various types of integer literals –</i>
212      /* Legal */	85      /* decimal */
215u    /* Legal */	0213   /* octal */
0xFeeL /* Legal */	0x4b   /* hexadecimal */
078     /* Illegal: 8 is not an octal digit */	30      /* int */
032UU   /* Illegal: cannot repeat a suffix */	30u     /* unsigned int */
	30l      /* long */
	30ul    /* unsigned long */

### 2) Floating-point Constants

These are used to represent and store real numbers. The real number has an integer part, real part, fractional part, and exponential part. The floating-point literals can be stored either in decimal form or exponential form.

**Here are some examples of floating-point literals –**

```
3.14159    /* Legal */
314159E-5L /* Legal */
510E       /* Illegal: incomplete exponent */
210f       /* Illegal: no decimal or exponent */
.e55       /* Illegal: missing integer or fraction */
```

### 3) Character Constants

Character Constants are enclosed in single quotes, e.g., 'x' can be stored in a simple variable of char type. A character literal can be a plain character (e.g., 'x'), an escape sequence (Backslash Character Constant) (e.g., '\t').

The Backslash Character Constants have some special meaning in the C language.

The list of Backslash Character Constants is shown below --

Escape sequence	Meaning
\\	\ character
'	' character
"	" character
\?	? character
\a	Alert or bell
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab

### 4) String Literals

A string literal in C is a sequence of chars, enclosed with in double quoted and terminated by a Null Character (\0). String literals are not modifiable and attempting to alter their values results in undefined behavior. String literals, same as character constants, support different character sets.

**Examples:** "ABCD" , "Hai" , "hello, world" , "12323loki\_\*&t" etc.

### BASIC INPUT & OUTPUT

The input and output operations in C are carried out using formatted and unformatted I/O Library Functions.

#### Formatted I/O Functions

Formatted I/O functions are used to take various inputs from the user and display multiple outputs to the user. These types of I/O functions can help to display the output to the user in different formats using the format specifiers. These I/O supports all data types like int, float, char, and many more.

These functions are called formatted I/O functions because format specifiers are used in these functions and hence, one can format these functions according to needs.

List of some format specifiers-

Format Specifier	Type	Description
%d	int/signed int	used for I/O signed integer value
%c	char	Used for I/O character value
%f	float	Used for I/O decimal floating-point value
Format Specifier	Type	Description
%s	string	Used for I/O string/group of characters
%ld	long int	Used for I/O long signed integer value
%u	unsigned int	Used for I/O unsigned integer value
%i	unsigned int	used for the I/O integer value
%lf	double	Used for I/O fractional or floating data
%n	prints	prints nothing

The following formatted I/O functions will be discussed in this section-

1. printf()
2. scanf()

**printf():** printf() function is used in a C program to display any value like float, integer, character, string, etc on the console screen. It is a pre-defined function that is already declared in the stdio.h(header file).

**Syntax 1:**

To display any variable value.

```
printf("Format Specifier", var1, var2, ..., varn);
```

**Syntax 2:**

To display any string or a message

```
printf("Enter the text which you want to display");
```

**scanf():** scanf() function is used in the C program for reading or taking any value from the keyboard by the user, these values can be of any data type like integer, float, character, string, and many more. This function is declared in stdio.h(header file), that's why it is also a pre-defined function. In scanf() function we use &(address-of operator) which is used to store the variable value on the memory location of that variable.

Syntax:

```
scanf("Format Specifier", &var1, &var2, ..., &varn);
```

***Unformatted Input/Output functions***

Unformatted I/O functions are used only for character data type or character array/string and cannot be used for any other datatype. These functions are used to read single input from the user at the console and it allows to display the value at the console.

The following unformatted I/O functions will be discussed in this section-

1.     getch()
2.     getche()
3.     getchar()
4.     putchar()
5.     gets()
6.     puts()
7.     putch()

**getch()** function reads a single character from the keyboard by the user but doesn't display that character on the console screen and immediately returned without pressing enter key. This function is declared in conio.h(header file). getch() is also used for hold the screen.

Syntax:

```
getch();
```

or

```
variable-name = getch();
```

**getche()** function reads a single character from the keyboard by the user and displays it on the console screen and immediately returns without pressing the enter key. This function is declared in conio.h(header file).

Syntax:

```
getche();
```

or

```
variable_name = getche();
```

**getchar()** function is used to read only a first single character from the keyboard whether multiple characters is typed by the user and this function reads one character at one time until and unless the enter key is pressed. This function is declared in stdio.h(header file)

Syntax: Variable-name = getchar();

**putchar()** function is used to display a single character at a time by passing that character directly to it or by passing a variable that has already stored a character. This function is declared in stdio.h(header file)

Syntax: putchar(variable\_name);

**gets()** function reads a group of characters or strings from the keyboard by the user and these characters get stored in a character array. This function allows us to write space-separated texts or strings. This function is declared in stdio.h(header file).

Syntax:

```
char str[length of string in number]; //Declare a char type variable of any length
```

```
gets(str);
```

**puts()** function is used to display a group of characters or strings which is already stored in a character array. This function is declared in stdio.h(header file).

Syntax: puts(identifier\_name );

**putch()** function is used to display a single character which is given by the user and that character prints at the current cursor location. This function is declared in conio.h(header file)

Syntax: `putch(variable_name);`

### Formatted I/O Vs Unformatted I/O

Formatted I/O functions	Unformatted I/O functions
These functions allow us to take input or display output in the user's desired format.	These functions do not allow to take input or display output in user desired format.
These functions support format specifiers.	These functions do not support format specifiers.
These are used for storing data more user friendly	These functions are not more user-friendly.
Here, we can use all data types.	Here, we can use only character and string data types.
printf(), scanf, sprintf() and sscanf() are examples of these functions.	getch(), getche(), gets() and puts(), are some examples of these functions.

## OPERATORS IN C

Operators are the foundation of any programming language. We can define operators as symbols that help us to perform specific mathematical and logical computations on operands. In other words, we can say that an operator operates the operands. For example, '+' is an operator used for addition, as shown below:

`c = a + b;`

Here, '+' is the operator known as the addition operator and 'a' and 'b' are operands. The addition operator tells the compiler to add both operands 'a' and 'b'.

The functionality of the C programming language is incomplete without the use of operators.

C has many built-in operators and can be classified into 6 types:

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

**Operators in C**

Operators	Type
++, --	Unary operator
+, -, *, /, %	Arithmetic operator
<, <=, >, >=, ==, !=	Relational operator
&&,   , !	Logical operator
&,  , <<, >>, ~, ^	Bitwise operator
=, +=, -=, *=, %=	Assignment operator
?:	Ternary or conditional operator

Unary operator ←

Binary operator ←

Ternary operator ←

**1. Arithmetic Operators:** These operators are used to perform arithmetic/mathematical operations on operands.

Examples: (+, -, \*, /, %, ++, --).

Operators that operate or work with two operands are binary operators. For example: Addition(+), Subtraction(-), multiplication(\*), Division(/) operators.

S.no	Arithmetic Operators	Operation	Example
1	+	Addition	A+B
2	-	Subtraction	A-B
3	*	multiplication	A*B
4	/	Division	A/B
5	%	Modulus	A%B

**2. Relational Operators:** These are used for the comparison of the values of two operands. For example, checking if one operand is equal to the other operand or not, whether an operand is greater than the other operand or not, etc.

Operator	Meaning of Operator	Example
==	Equal to	5 == 3 is evaluated to 0
>	Greater than	5 > 3 is evaluated to 1
<	Less than	5 < 3 is evaluated to 0
!=	Not equal to	5 != 3 is evaluated to 1
>=	Greater than or equal to	5 >= 3 is evaluated to 1
<=	Less than or equal to	5 <= 3 is evaluated to 0

**3. Logical Operators:** Logical Operators are used to combine two or more conditions/constraints or to complement the evaluation of the original condition in consideration. The result of the operation of a logical operator is a Boolean value either true or false.

Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression ((c==5) && (d>5)) equals to 0.
	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c==5)    (d>5)) equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression !(c==5) equals to 0.

**4. Bitwise Operators:** The Bitwise operators are used to perform bit-level operations on the operands. The operators are first converted to bit-level and then the calculation is performed on the operands. Mathematical operations such as addition, subtraction, multiplication, etc. can be performed at the bit level for faster processing. For example, the bitwise AND operator represented as '&' in C takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1(True).

Operator_symbol	Operator_name
&	Bitwise_AND
	Bitwise_OR
~	Bitwise_NOT
^	XOR
<<	Left Shift
>>	Right Shift

x	y	x y	x & y	x ^ y
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

#### Bitwise Left shift << operator

Bitwise Left shift operator is used to shift the binary sequence to the left side by specified position.

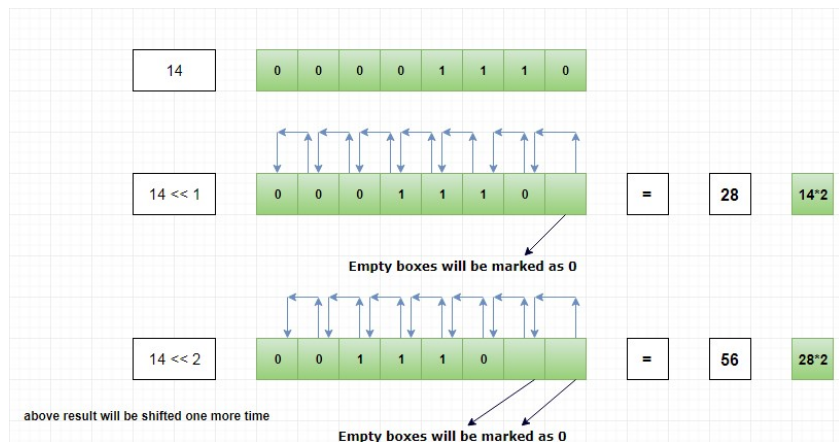
#### Example

Let's take a number 14.

Binary representation of 14 is 00001110 (for the sake of clarity let's write it using 8 bit)

14 = (00001110) 2

Then 14 << 1 will shift the binary sequence 1 position to the left side.



**5. Assignment Operators:** Assignment operators are used to assign value to a variable. The left side operand of the assignment operator is a variable and the right-side operand of the assignment operator is a value. The value on the right side must be of the same data type as the variable on the left side otherwise the compiler will raise an error.

Different types of assignment operators are shown below:

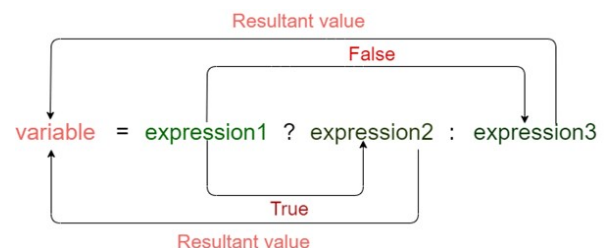
Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

#### 6. Conditional Operator (Ternary Operator)

The conditional operator is also known as a **ternary operator**. The conditional statements are the decision-making statements which depends upon the output of the expression. It is represented by two symbols, i.e., '?' and '!'.

#### Syntax of a conditional operator

expression1? expression2: expression3;



**Meaning of the above syntax.**

In the above syntax, the expression1 is a Boolean condition that can be either true or false value.

If the expression1 results into a true value, then the expression2 will execute.

If the expression1 returns false value, then the expression3 will execute.

**7. Increment & Decrement Operators**

Increment Operators are the unary operators used to increment or add 1 to the operand value. Decrement Operators are the unary operators used to decrement or subtract 1 to the operand value. The Increment operator is denoted by the double plus symbol (++) and Decrement operator is denoted by the double minus symbol (--). Increment or Decrement has two types, Pre-Increment (Pre-Decrement) and Post Increment (Post Decrement) Operators.

Operator	Description	Example	Explanation
Pre-Increment	It is used to increase the original value of the operand by 1 before assigning it to the expression.	$X = ++A$	The value of operand 'A' is increased by 1, and then a new value is assigned to the variable 'X'.
Post-Increment	It is used to increase the original value of the operand by 1 after assigning the present value of it to the expression.	$X = A++$	the value of operand 'A' is assigned to the variable 'X'. After that, the value of variable 'A' is increased by 1.
Pre-Decrement	It is used to decrease the original value of the operand by 1 before assigning it to the expression.	$X = --A$	The value of operand 'A' is decreased by 1, and then a new value is assigned to the variable 'X'.
Post-Decrement	It is used to decrease the original value of the operand by 1 after assigning the present value of it to the expression.	$X = A--$	the value of operand 'A' is assigned to the variable 'X'. After that, the value of variable 'A' is decreased by 1.

**8. Special Operators:** Below are some of the special operators in C.

Operators	Description
&	This is used to get the address of the variable.
	Example: <b>&amp;a</b> will give address of a.
*	This is used as pointer to a variable.
	Example: <b>* a</b> where, * is pointer to the variable a.
sizeof ()	This gives the size of the variable.
	Example: <b>sizeof(char)</b> will give us 1.

**Operator precedence and Associativity:** Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. 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.

Operator	Description	Associativity
( ) [ ] . -> ++ --	Parentheses (function call) Brackets (array subscript) Member selection via object name Member selection via pointer Postfix increment/decrement	left-to-right
++ -- + - ! ~ (type) * & sizeof	Prefix increment/decrement Unary plus/minus Logical negation/bitwise complement Cast (convert value to temporary value of type) Dereference Address (of operand) Determine size in bytes on this implementation	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right



Operator	Description	Associativity
< <=	Relational less than/less than or equal to	left-to-right
> >=	Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
= += -= *= /= %= &= ^=  = <<= >>=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	right-to-left
,	Comma (separate expressions)	left-to-right

**Example::** a = 20; b = 10; c = 15; d = 5;

e = (a + b) \* c / d; // (30 \* 15) / 5      90

e = ((a + b) \* c) / d; // (30 \* 15) / 5      90

e = (a + b) \* (c / d); // (30) \* (15/5)      90

e = a + (b \* c) / d; // 20 + (150/5)      50

### TYPE CONVERSION & CASTING

Converting one data type into another data type is called type conversions. C allows

- Implicit type conversion
- Explicit type conversion

#### Implicit Type Conversion

The compiler provides implicit type conversions when operands are of different data types. It is automatically done by the compiler by converting smaller data type into a larger data type.

#### Example

int i,x;

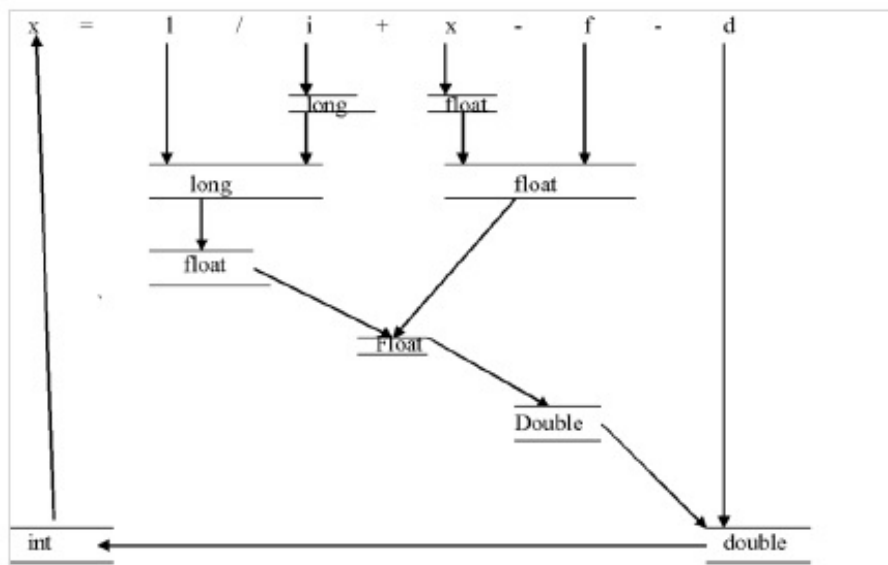
float f;

double d;

long int l;

Assume the Expression:  $x = l / i + x * f - d$

Implicit Conversion Process for evaluating the expression is shown below.



Here, the above expression finally evaluates to a 'double' value is type casted to int before assigning to integer variable x.

### Explicit Type Conversion

Explicit type conversion is done by the user by using (type) operator. It is actually called as **Type Casting**. Before the conversion is performed, a runtime check is done to see if the destination type can hold the source value.

### Example

```
int a,c;
float b;
c = (int) a + b
```

Here, the resultant of 'a+b' is converted into 'int' explicitly and then assigned to 'c'.

## PROBLEM SOLVING TECHNIQUES

### Algorithmic Approach

An algorithm is a process or set of rules which must be followed to complete a particular task. This is basically the step-by-step procedure to complete any task.

The algorithm is used for,

- To develop a framework for instructing computers.
- Introduced notation of basic functions to perform basic tasks.
- For defining and describing a big problem in small parts, so that it is very easy to execute.

### Characteristics of Algorithm

- An algorithm should be defined clearly.
- An algorithm should produce at least one output.
- An algorithm should have zero or more inputs.
- An algorithm should be executed and finished in finite number of steps.
- An algorithm should be basic and easy to perform.

## PROBLEM SOLVING STRATEGIES

Problem Solving strategy is a systematic approach to find and implement the solution to a problem. The Steps involved in Problem Solving are –

- Problem Definition
- Problem Analysis
- Design
- Coding
- Testing
- Maintenance

**Problem Definition** -- To solve a problem, the first step is to identify and define the problem. The problem must be stated clearly, accurately, and precisely.

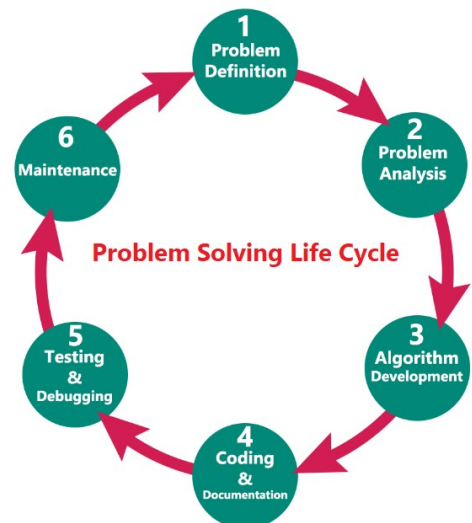
**Problem Analysis** -- All the factors like Input/output, processing requirement, memory requirements, error handling, interfacing with other programs must be taken into consideration in this stage. It also determines the required format in which results should be displayed.

**Design** -- The software developer makes use of tools like algorithms and flowcharts to develop the design of the program.

**Coding** -- Once the design process is complete, converting algorithm to a program by selecting any one of the high – level languages that is suitable for the problem.

**Testing** -- After writing a program, programmer needs to test the program for completeness, correctness, reliability, and maintainability.

**Maintenance** -- It means periodic review of the programs and modifications based on user requirements.



### Modular Approach for Programming:

The process of breaking a large problem into subproblems and then treating these individual parts as different functions is called modular programming. Each function behaves independent of another and there is minimal inter-functional communication. There are two methods to implement modular programming:

**Top-Down Approach:** In this method, the original problem is divided into subparts. These subparts are further divided. The chain continues until the very fundamental subpart of the problem which can't be further divided is achieved. Then solution is drawn for each of these fundamental parts.

**Bottom-Up Approach:** In this method, the smaller problems are solved, and then these solved problems are integrated to find the solution to a bigger problem.

### **Difference between Top-Down Approach and Bottom-Up Approach**

The major differences between top-down approach and bottom-up approach are detailed below –

<b>Top-Down Approach</b>	<b>Bottom-Up Approach</b>
In this approach, the problem is broken down into smaller parts.	In this approach, the smaller problems are solved.
It is generally used by structured programming languages such as C, COBOL, FORTRAN, etc.	It is generally used with object-oriented programming paradigm such as C++, Java, Python, etc.
It does not require communication between modules.	It requires relatively more communication between modules.
It contains redundant information.	It does not contain redundant information.
Decomposition approach is used here.	Composition approach is used here.

### **TIME & SPACE COMPLEXITIES OF ALGORITHMS**

There is more than one way to solve a problem. Hence it is required to compare the performance of different solutions (algorithms) and choose the best one to solve a particular problem. While analyzing an algorithm, a good algorithm is one that takes less time in execution and saves space during the process.

- **Time complexity** is the time taken by the algorithm to execute each set of instructions.
- **Space complexity** is usually referred to as the amount of memory consumed by the algorithm.

The factor of time is usually more important than that of space. The complexities are mostly related to the input size. As the size of the input increases, the run time also increases.

### **Asymptotic Notations**

Asymptotic analysis is used to compare space and time complexity. It analyzes two algorithms based on changes in their performance concerning the increment or decrement in the input size.

Primarily there are three types of Asymptotic notations:

- Big-Oh (O) notation.
- Big Omega ( $\Omega$ ) notation.
- Big Theta ( $\Theta$ ) notation.

#### **Big-Oh (O) notation**

Definition:  $O(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0 \}$

It is the most used notation for the Asymptotic analysis. It defines the upper bound of a function i.e., the maximum time taken by an algorithm or the worst-case time complexity of an algorithm.

#### **Big Omega ( $\Omega$ ) notation**

Definition:  $f(n)$  is  $\Omega(g(n))$  if there exist positive numbers  $c$  and  $N$ , such that  $f(n) \geq c \cdot g(n)$  for all  $n \geq N$ .

It defines the lower bound of a function — i.e., the minimum time taken by an algorithm. It gives the minimum output value (big- $\Omega$ ) for a respective input.

#### **Big Theta ( $\Theta$ ) notation**

Definition:  $f(n)$  is  $\Theta(g(n))$  if there exist positive numbers  $c_1$ ,  $c_2$ , and  $N$  such that  $c_1 g(n) \leq f(n) \leq c_2 g(n)$  for all  $n \geq N$

It defines the lower bound and upper bound of a function i.e., it exists as at both, most and at least boundaries for a given input value.

### **Best case, Worst case, and Average case in Asymptotic Analysis:**

**Best Case:** It defines as the condition that allows an algorithm to complete the execution of statements in the minimum amount of time. In this case, the execution time acts as a lower bound on the algorithm's time complexity.

**Average Case:** In the average case, we get the sum of running times on every possible input combination and then take the average. In this case, the execution time acts as both the lower bound and upper bound on the algorithm's time complexity.

**Worst Case:** It defines as the condition that allows an algorithm to complete the execution of statements in the maximum amount of time. In this case, the execution time acts as an upper bound on the algorithm's time complexity.