# UNIT-I

**Overview of Computers and Programming : Electronic Computers Then and Now , Computer Hardware , Computer Software , Algorithm , Flowcharts , Software Development Method , Applying the Software Development Method.**

**Types, Operators and Expressions: Variable Names , Data Types and Sizes , Constants , Declarations , Arithmetic Operators, Relational and Logical Operators, Type Conversions ,Increment and Decrement Operators , Bitwise Operators , Assignment Operators and Expressions ,Conditional Expressions - Precedence and Order of Evaluation.**

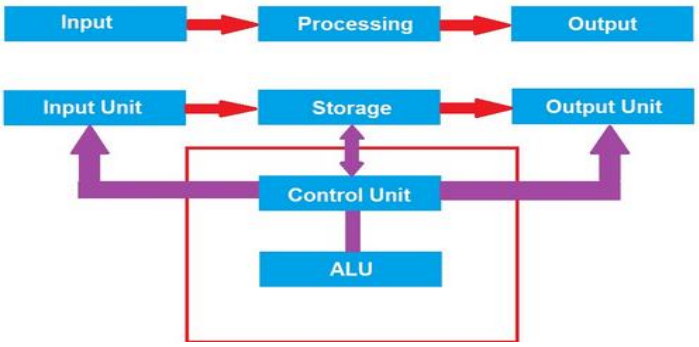
**What is a computer:**

- A **computer** is an electronic device that manipulates information, or data.
- It has the ability to **store**, **retrieve**, and **process** data.
- The term computer is derived from the Latin term '**computare**', this means to calculate or programmable machine.
- **Computer can not do anything without a Program**.

Four Functions about computer are:

| | |
|---|---|
| accepts data | Input |
| processes data | Processing |
| produces output | Output |
| stores results | Storage |

**Block diagram of a computer:**




**Input (Data):**
Input is the raw information entered into a computer from the input devices. It is the collection of letters, numbers, images etc.

**Process:**
Process is the operation of data as per given instruction. It is totally internal process of the computer system.

**Output:**
Output is the processed data given by computer after data processing. Output is also called as Result. We can save these results in the storage devices for the future use.

## Electronic Computers Then and Now:

**First Generation (1942-1955)** The computers of this generation used Electronic Valves (an array of Vacuum Tubes) as the basic component for memory and circuitry for central processing unit. These vacuum tubes were a fragile glass device like electric bulb, that could control and amplify electronic

signals. They produced a lot of heat and were prone to frequent fusing/ damaging of the installations. Therefore, they were used very expensive and could be afforded only by very large organizations.

***Advantages :***

i) these computers were the fastest calculating device of their time. They could perform computations in milliseconds.

ii) vacuum tube technology made possible the advent of electronic digital computers.

***Disadvantages :***

i) Too bulky in size

ii) Air conditioning required to control the temperature

iii) Prone to frequent hardware failure.

iv) Commercial production was difficult and costly.

v) Slow input and output operations.

**Second Generation (1955-1964)** The second generation computer used Semi-conductor transistor instead of vacuum tubes after the invention of transistor by a team led by William Shockley. This generation computers used transistors which were cheaper, consumed less power, more compact in size, more reliable and faster than the vacuum tubes used in first generation computers.

***Advantages :***

i) Smaller in size as compared to first generation computers ii) More reliable

iii) Less heat generated.

iv) Less prone to hardware failure

v) Wider commercial use

***Disadvantages :***

i) Air conditioning required

ii) Manual assembly of individual components into a functioning unit was a cumbersome task.

**Third Generation (1964-1975)** The third generation computers used Integrated Circuits (or IC Chips) in which many transistors, resistors, capacitors and other components (circuit elements) are fabricated or integrated and packaged together into a very small surface of silicon known as Chips. This new microelectronic technology was called Integrated Circuits. The IC was invented by Jack Kilby in 1958.

***Advantages :***

i) Smaller in size as compared to previous generation

ii) More reliable than second generation computers

iii) Low maintenance cost

iv) Easily portable

v) Commercial production was easier and cheaper.

***Disadvantages :***

i) Air conditioning required in many case

ii) Highly sophisticated technology required for the manufacture of IC chips

**Fourth Generation (1975 onwards)** Initially, the Integrated Circuits contained only about 10 to 20 components called Small Scale Integration (SSI). Now the manufacture of integrated circuits became so advanced as to incorporate hundreds of thousands of active components in volume of a fraction of inch, leading to Large Scale or Very Large Scale Integration (VLSI). Integrated Circuits which have the entire computer circuit on a single silicon chip are called Microprocessors. The development of microprocessors made it possible to place complete CPU of a computer on a single chip.

***Advantages :***

i) Smaller in size because of high component density
ii) Heat generated is negligible
iii) Much faster in computation then previous generations
iv) Less power consumption
v) No air conditioning is required in most cases

***Disadvantages :***

i) Highly sophisticated technology is required for the manufacturing of VLSI chips
ii) Highly skilled people are required in its manufacturing

**Fifth Generation (Future Generation: 1991 onwards )** : the fifth generation computers are under development stage. These computers will use Ultra Large Scale Integration (ULSI) chips instead of VLSI. These will be employing two or more processors which compute in parallel. The fifth generation machine are proposed to be based on Parallel Processing hardware and Artificial Intelligence software with genuine I.Q. (Intelligence Quotient). That provide the ability to reason logically and with real knowledge of the world like human do.

***Characteristics of future generation computers :***

i) Decreasing cost of hardware and software
ii) High speed processing
iii) Knowledge based processing system
iv) Development of natural language processing
v) Advancement in supercomputer technology
vi) Artificial Intelligence , think and behave like human (humanoid)

## Computer Hardware:

Computer hardware consists of 3 units

1.Input unit 2.output unit 3.storage unit

**1. Input Unit :**

The devices which are used to input the data and programs in the computer are known as "Input Devices". Input unit accepts instructions and data from the user and converts these instructions and data in computer acceptable format which are sent to computer system for processing.

**Keyboard**:

- Keyboard is most common input device. The data and instructions are input by typing on the keyboard.
- The message typed on the keyboard reaches the memory unit of a computer. It is connected to a computer via a cable.
- Apart from alphabet and numeral keys, it has other function keys for performing different functions.

**Mouse:**

- It is a pointing device.
- The mouse is rolled over the mouse pad, which in turn controls the movement of the cursor in the screen.
- You can click, double click or drag the mouse.

**Scanner:**

- Scanners are used to enter information directly in to the computer memory.

- This device works like a Xerox machine.
- The scanner converts any type of printed or written information including photographs into digital pulses, which can be manipulated by the computer.

**Bar Code Reader:**

- This device reads bar codes and coverts them into electric pulses to be processed by a computer.
- A bar code is nothing but data coded in form of light and dark bars.

**Voice Input Systems:**

- It converts spoken words to machine language form.
- A microphone is used to convert human speech into electric signals.
- The signal pattern is then transmitted to a computer when it is compared to a dictionary of patterns that have been previously placed in a storage unit of computer.

**Digital Camera:**

- It converts graphics directly into digital form.
- An electronic chip in used in camera, when light falls, on the chip though the lens, it converts light waves into electrical waves

**Bar Code Reader:**

- This device reads bar codes and coverts them into electric pulses to be processed by a computer.
- A bar code is nothing but data coded in form of light and dark bars.

**Voice Input Systems:**

- It converts spoken words to machine language form.
- A microphone is used to convert human speech into electric signals.
- The signal pattern is then transmitted to a computer when it is compared to a dictionary of patterns that have been previously placed in a storage unit of computer.

**Digital Camera:**

- It converts graphics directly into digital form.
- An electronic chip in used in camera, when light falls, on the chip though the lens, it converts light waves into electrical waves

**2. Output Unit :**

- Output Device produces the final results of computer into human understandable form.
- Output unit accepts the results produced by the computer which are in coded form and it converts these coded results to human readable form.

**Monitor:**

- The monitor looks like a television screen.
- It is also called Visual Display Unit (VDU) and it is used to display information from the computer.
- There are coloured as well as black and white monitors. The monitor displays text and graphics.
- Based on the technology used, monitor is classified into two types. They are Cathode Ray Tube (CRT) monitor and Liquid Crystal Display (LCD) monitor.

**Printer:**

- A printer is used for transferring data from the computer to the paper.
- There are colour printers as well as black and white printers.
- The different types of printers are Dot Matrix Printers, Inkjet Printer and Laser Printers.

**3. Storage Unit**

- The data and instructions that are entered into the computer system through input units have to be stored inside the computer before the actual processing starts.

- Similarly, the results produced by the computer after processing must also be kept somewhere inside the computer system before being passed on to the output units.
- The Storage Unit or the primary / main storage of a computer system is designed to do all these things.
- It provides space for storing data and instructions, intermediate results and for the final results.

**CPU:**

- The main unit inside the computer is the CPU
- It consists of Control Unit and Arithmetic and Logic unit.
- The CPU is the brain of any computer system.
- Similarly, in a computer system, all major calculations and comparisons are made inside the CPU and it activates and controls the operations of other units like Hard disk, Printer etc. of a computer system.

**Arithmetic and Logic Unit (ALU):**

The arithmetic and logic unit (ALU) is the part where actual computations take place. It consists of circuits that perform arithmetic operations (e.g. addition, subtraction, multiplication, division over data received from memory and capable to compare numbers (less than, equal to, or greater than etc).

**Control Unit:**

The control unit acts as a central nervous system for the components of the computer.
The control unit directs and controls the activities of the internal and external devices.

**Computer Memory:**

A memory is just like a human brain. It is used to store data and instructions. Computer memory is the storage space in computer where data is to be processed and instructions required for processing are stored.

Memory is primarily of three types

- Cache Memory
- Primary Memory or Main Memory
- Secondary Memory

**Cache Memory:**

- Cache memory is a very high speed semiconductor memory which can speed up CPU.
- It acts as a buffer between the CPU and main memory.
- It is used to hold those parts of data and program which are most frequently used by CPU.
- The parts of data and programs are transferred from disk to cache memory by operating system from where CPU can access them.

**Advantages**

The advantages of cache memory are as follows:

• Cache memory is faster than main memory.
• It consumes less access time as compared to main memory.
• It stores the program that can be executed within a short period of time.
• It stores data for temporary use.

**Disadvantages**

The disadvantages of cache memory are as follows:

• Cache memory has limited capacity.

**Primary Memory (Main Memory):**

- Primary memory is also known as main memory.
- It holds only those data and instructions on which computer is currently working.
- It has limited capacity and data is lost when power is switched off.
- It is generally made up of semiconductor device.

**Characteristics of Main Memory**

- These are semiconductor memories
- It is known as main memory.
- Usually volatile memory.
- Data is lost in case power is switched off.
- Faster than secondary memories.

**Secondary Memory :**

- It is also known as external memory or non-volatile memory.
- It is slower than main memory.
- These are used for storing data or information permanently. For example: hard disk, CD-ROM, DVD etc.

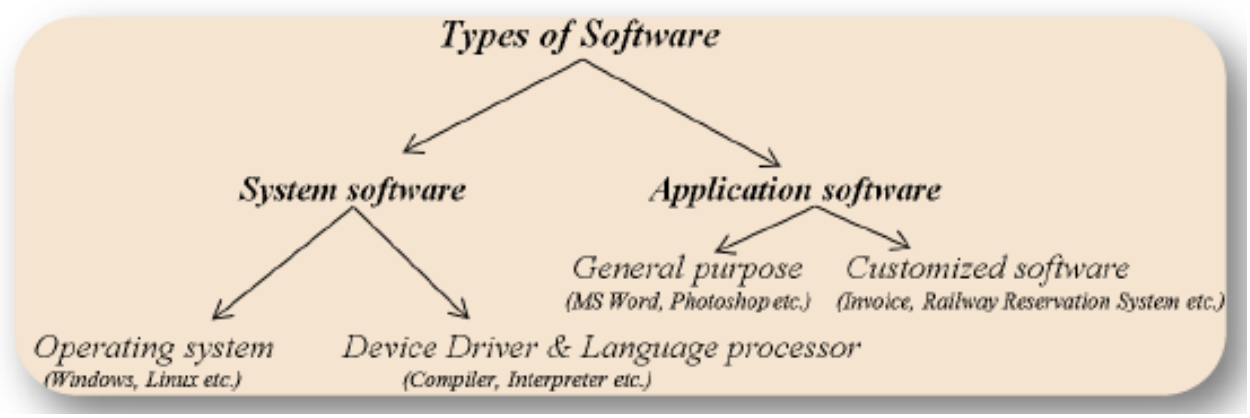
**Characteristic of Secondary memory**

- These are magnetic and optical memories
- It is non-volatile memory which is used for storage of data in a computer.
- Data is permanently stored even if power is switched off.
- Computer may run without secondary memory.
- Slower than primary memories.

## Computer Software:

Computer software also called program is a set of instructions that directs a computer to perform specific tasks or operations. Computer software consists of computer programs and libraries.

**1. System software**: Software that directly operates the computer hardware to provide basic




functionality needed by users and other software and to provide a platform for running application software. System software includes:

**Operating system (OS):** Operating system manages resources of computer system like memory, CPU, hard disk, printer etc. also provides an interface between user and computer system & provides various services to other software.

**Language Processor & Device drivers:** All the devices like mouse, keyboard, modem etc needs at least one corresponding device driver. A device driver is a program that controls a device. A language processor is a hardware device designed or used to perform tasks, such as processing program code to machine code. Language processors are found in languages such as Fortran and COBOL

**2. Application software:** Software that performs special functions or provides entertainment functions beyond the basic operation of the computer itself. There are many different types of application software.

**General purpose:** Microsoft Word, Microsoft Excel, MS PowerPoint, Photoshop etc.
**Customized:** Invoice Management System, Airline Reservation System etc.
**Utilities:** Antivirus, Memory tester, Disk partitioning and Disk defragmenter etc.

## Algorithm:

DEF: Step by step procedure for solving a problem is called an algorithm.

(or)

An algorithm is defined as a finite set of steps that provide a chain of actions for solving a problem.

**Procedure for writing an algorithm:-**

An Algorithm is a well organized and textual computational module that receives one or more input values and provides one or more output values.

1. These well defined steps are arranged in a sequence that processes given input into output.
2. The steps of the algorithm are written using English like statements which are easy to understand.
3. This will enable the reader to translate each step into a program.
4. Every step is known as instruction.
5. An algorithm is set to be accurate only when it provides the exact required output.
6. If the procedure is lengthy then sub divided the procedure into small parts as it make easy to solve the problem.

**Type of Algorithms**

An algorithm is classification into the three categories based on *control structures. They are:*

1. Sequence
2. Branching (Selection)
3. Loop (Repetition)

**1. Sequence**

- The steps described in an algorithm are performed successively one by one without skipping any step.
- The sequence of steps defined in an algorithm should be simple and easy to understand.

**Example:**

// adding two timings
Step 1: start
Step 2: read h1, m1, h2, m2
Step 3: m=m1+m2
Step 4: h= h1 + h2 + m/60
Step 5: m=m mod 60
Step 6: write him
Step 7: stop

**2. Selection:**

The selection of statements can be shown as follows

```
if(condition)
        Statement-1;
else
        Statement-2;
```

- The above syntax specifies that if the condition is true , statement-1 will be executed , otherwise statement-2 will be executed.

**Example:**

// Person eligibility for vote
Step 1 : start
Step 2 : read age
Step 3 : if age > = 18 then step_4
else step_5
Step 4 : write "person is eligible for vote"
Step 5 : write " person is not eligible for vote"
Step 6 : stop

**3. Iteration**

- In a program, sometimes it is very necessary to perform the same action for a number of times.
- If the same statement is written repetitively, it will increase the program code.
- To avoid this problem, iteration mechanism is applied.
- The statement written in an iteration block is executed for a given number of times based on certain condition.

**Example:**

Step 1 : start
Step 2 : read n
Step 3 : repeat step 4 until n>0
Step 4 : (a) r=n mod 10
(b) s=s+r
(c) n=n/10
Step 5 : write s
Step 6 : stop

**Properties of algorithm :**

**1. Finiteness:** The algorithm must always terminate after a finite number of steps.

**2. Definiteness:** Each step must be precisely defined; the actions to be carried out must be rigorously and unambiguously specified for each case.

**3. Input:** An algorithm has zero or more inputs, taken from a specified set of objects.

**4. Output:** An algorithm has one or more outputs, which have a specified relation to the inputs.

**5. Effectiveness:** All operations to be performed must be sufficiently basic that they can be done exactly and in finite length.

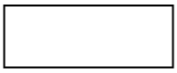
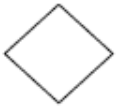
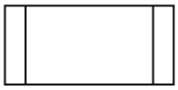
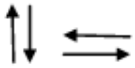
## FLOWCHART

Flowchart is an alternative technique for solving a problem. Instead of descriptive steps, we use pictorial representation for every step.

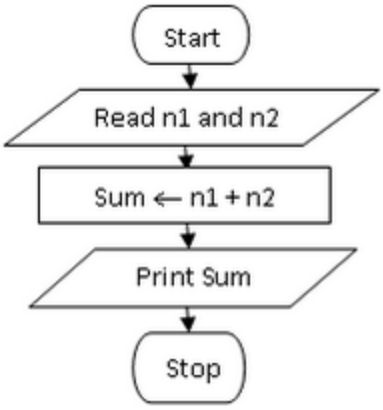
**Def:**

*Flowchart is a diagrammatic or pictorial representation of various steps involved in the Algorithm*

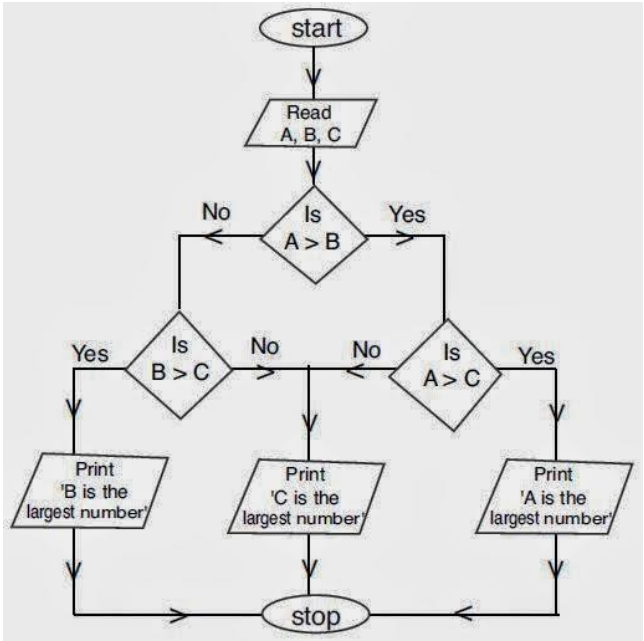
**SYMBOLS FOR FLOWCHART:**

| Symbol | Name | Function |
|---|---|---|
|  | Process | Indicates any type of internal operation inside the Processor or Memory |
|  | input/output | Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results |
|  | Decision | Used to ask a question that can be answered in a binary format (Yes/No, True/False) |
|  | Connector | Allows the flowchart to be drawn without intersecting lines or without a reverse flow. |
|  | Predefined Process | Used to invoke a subroutine or an Interrupt program. |
|  | Terminal | Indicates the starting or ending of the program, process, or interrupt program |
|  | Flow Lines | Shows direction of flow. |

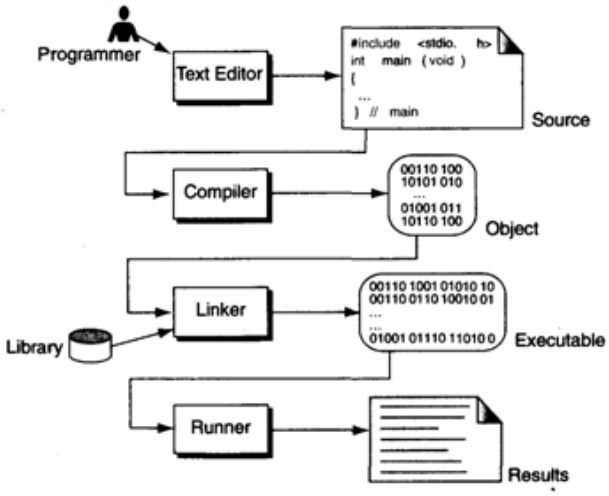
**Flowchart for adding 2 numbers:**




**Flowchart for finding maximum of 3 numbers:**



## Creating and Running Programs (or) how to build a C program :




Computer hardware understands a program only if it is coded in its machine language. It is the job of the programmer to write and test the program.

There are four steps in this process:

1. Writing and editing the program
2. Compiling the program
3. Linking the program with the required library modules
4. Executing the program.

**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. Depending on the editor on our system, we could use it to write letters, create reports, or write programs. The main difference between text processing and program writing is that programs are written using lines of code, while most text processing is done with character and lines.

Text editor is a generalized word processor, but it is more often a special editor included with the compiler. Some of the features of the editor are search commands to locate and replace statements, copy and paste commands to copy or move statements from one part of a program to another, and formatting commands that allow us to set tabs to align statements.

After completing a program, we save our file to disk. This file will be input to the compiler; it is known as a **source file.**

**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 c compiler is two separate programs. **Preprocessor** & **translator.**

The preprocessor reads the source code and prepares it for the translator. While preparing the code, it scans for special instructions known as preprocessor commands. These commands tell the preprocessor to look for special code libraries, make substitutions in the code, and in other ways prepare the code for translation into machine language. The result of preprocessing is called the translation unit.

After the preprocessor has prepared the code for compilation, the translator does the actual work of converting the program into machine language. The translator reads the translation unit and writes the resulting object module to a file that can then be combined with other precompiled units to form the final program. An object module is the code in machine language. The output of the compiler is machine language code, but it is not ready to run; that is, it is not executable because it does not have the required C and other functions included.

**3. Linking Programs:**

A C program is made up of many functions. We write some of these functions, and they are a part of our source program. There are other functions, such as input/output processes and, mathematical library functions that exist elsewhere and must be attached to our program. The linker assembles all of these functions, ours and systems into our final executable program.

**4. Executing Programs:**

Once program has been linked, it is ready for execution. 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.

In a typical program execution, the reads data for processing ,either from the user or from a file. After the program processes the data, it prepares the output. at output can be to the user's monitor or to a file. When the program has finished its job, it tells the operating system, which then removes the program from memory.

## System Development Method:

Programming is a problem-solving activity. In order to solve the problem, business students will use systems approach while engineering and science students use the engineering and scientific method. Programmers use the software development method.

**Software development Method:** The software development method is simply called as SDM. The Software Development Method will consist of the following steps.

1. Specify the problem requirements.
2. Analyze the problem.
3. Design the algorithm to solve the problem
4. Implement the algorithm
5. Test and verify the completed program.
6. Maintain and update the program.

1. ***Specifying the problem requirements*** forces you to state the problem clearly and unambiguously and to gain a clear understanding of what is required for its solution.
2. ***Analysis***: Analyzing the problem involves identifying the problem inputs, outputs, and any additional requirements which are needed for solving the particular problem.
   *Input:* Input is the data we have to work with.
   *Output:* Output is the desired results.
   *Additional requirements or constraints:* These are the necessary conditions to solve the problem.
3. **Design:** Designing the algorithm to solve the problem requires you to develop a list of steps called algorithm to solve the problem and to then verify that the algorithm solves the problem as intended. Writing the algorithm is often the most difficult part of the problem-solving process.
4. **Implementation:** Implementing the algorithm involves writing it as a program. We will convert each algorithm step into one or more statements in a program.
5. **Testing:** Testing and verifying the program requires testing the completed program to verify that it works as desired. Don't rely on just one test case. We have to check the program with various types of inputs and we have to verify the desired outputs manually.

6. **Maintaining and updating the program**: maintaining and updating the program involves modifying a program to remove previously undetected errors and to keep it up-to date ad government regulations or company policies change. Many organizations maintain a program for five years or more, often after the programmers who originally coded it have left or moved on to other positions.

## Applying Software Development Method:

For applying the software development method, we use the first five steps of the software development method to solve programming problems. These problems begin with ***problem Statement***. As part of the ***Problem Analysis***, we identify the data requirements for the problem, indicating the problem inputs and the desired outputs. Next, we design and refine the initial algorithm.

Finally, we ***implement*** the algorithm as a C program. We also provide sample run of the program and discuss how to ***test*** the program. Here, we will explain this concept with an example.

➔**PROBLEM:** Converting Miles to Kilometers

➔**ANALYSIS:** Analyzing the problem statement involves through understanding about the problem statement and identifying the ***problem inputs***, ***outputs***, and any ***additional requirements*** which are needed for solving the particular problem.

***DATA REQUIREMENTS***

Problem Input: ***miles***

Problem Output: ***kms***

Relevant Formula: ***1 mile = 1.609 Kilometers***

➔ **DESIGN:** Next, formulate the algorithm that solves the problem. Begin by listing the three major steps, or subproblems, of the algorithm.

**ALGORITHM:**

Step 1: Begin

Step 2: Get the distance in miles.

Step 3: Convert the distance to kilometers

Step 4: Display the distance in Kilometers

Step 5: End

Now we need to decide whether any steps of the algorithm need further refinement or whether they are perfectly clear as stated. Step 2 (getting the data) and Step 4 ( displaying a value) are basic steps and require no further refinement. Step 3 is fairly straightforward, but some detail might help:

**Step 2 Refinement**

Step 3.1 The distance in kilometers is 1.609 times the distance in miles.

We list the complete algorithm with refinements below to show you how it all fits together.

ALGORITHM WITH REFINEMENTS

Step 1: Begin

Step 2: Get the distance in miles.

Step 3: Convert the distance to kilometers

3.1 The distance in kilometers is 1.609 times the distance in miles.

Step 4: Display the distance in Kilometers

Step 5: End

**IMPLEMENTATION**

To implement the solution, you must write the algorithm as a C program. To do this, you must first tell the C compiler about the problem data requirements that is, what memory cell names you are using and what kind of data will be stored in each memory cell. Next, convert each algorithm step into one or more C statements. If an algorithm has been refined, you must convert the refinements, not the original step, into C statements.

```
/* C program to Convert Miles to Kilometers */
#include<stdio.h>                    / * printf, scanf definitions are present in stdio.h */
#define KMS_PER_MILE  1.609      /* conversion constant */
main ( )
{
double miles, kms;

/* Get the distance in miles */
printf("\n Enter the distance in miles \n");
scanf("%lf",&miles);

/* Convert the distance to kilometers */
kms =KMS_PER_MILE * miles ;

/* Display the distance in kilometers */
printf("\n The %lf miles is equals to %lf kilometers \n", miles, kms);
}
```

**TESTING:** after writing a program we must check it. How do you know that the sample run is correct? You should always examine program results carefully to make sure that they make sense. In this run, a distance of 10.0 miles is converted to 16.09 kilometers, as it should be. To verify that the program works properly, enter a few more test values of miles and check the system obtained results with manual calculations to verify the sample program is correct.

## Introduction to 'C' Language:

- 'C' is a Structured Programming Language
- It is considered as a middle level language as it supports some of the features of low level language also.
- 'C' is a case sensitive language i.e., upper case letters and lower case letters are distinct.
- It is well suitable for developing application software as well as system software.
- It supports the concept of "Modular Programming", which means that a larger task is divided into a number of sub tasks each subtask is referred to as a "module" or a "function". Due to this modular programming, it is easy to identify and rectifies the errors. Execution speed also increases.
- 'C' language is highly portable. i.e., a 'C' program written for one computer can be easily run in another computer under certain conditions.
- It supports data structures like arrays, pointers, structures and unions.
- The power of 'C' is increased due to the support of a number of library (or) inbuilt functions.
- 'C' has a special type of operators called "Bitwise" operators.

**History of 'C'**:

| YEAR | LANGUAGE | DEVELOPER |
|---|---|---|
| 1960 | ALGOL - 60 | INTERNATIONAL COMMITTEE |
| 1963 | CPL | CAMBRIDGE UNIVERSITY |
| 1969 | BCPL | MARTIN RICHARDS |
| 1970 | B | KEN THOMSON (AT & T BELL LABORATORIES) |
| 1972 | C | DENNIS RITCHIE (AT & T BELL LABORATORIES) |

- 'C' is a general purpose structured programming language.
- 'C' was originally developed in '1972' by "Dennis Ritchie" at "AT&T" Bell Laboratories.
- It is an outgrowth of two earlier languages called "BCPL" {Basic Common Programming Language} & "B" which were also developed at Bell laboratories

## BASIC STRUCTURE OF C LANGUAGE :

The program written in C language follows this basic structure. The sequence of sections should be as they are in the basic structure. A C program should have one or more sections but the sequence of sections is to be followed.

1. Documentation section
2. Linking section
3. Definition section
4. Global declaration section
5. Main function section
   {
   Declaration section
   Executable section
   }
6. Sub program or function section

**1. DOCUMENTATION SECTION :** comes first and is used to document the use of logic or reasons in your program. It can be used to write the program's objective, developer and logic details. The documentation is done in C language with /* and */ . Whatever is written between these two are called comments.

**2. LINKING SECTION :** This section tells the compiler to link the certain occurrences of keywords or functions in your program to the header files specified in this section.

e.g. #include <stdio.h>

**3. DEFINITION SECTION :** It is used to declare some constants and assign them some value.

e.g. #define MAX 25

Here #define is a compiler directive which tells the compiler whenever MAX is found in the program replace it with 25.

**4. GLOBAL DECLARATION SECTION :** Here the variables which are used through out the program (including main and other functions) are declared so as to make them global(i.e accessible to all parts of program)
e.g. int i; (before main())

**5. MAIN FUNCTION SECTION :** It tells the compiler where to start the execution from

```
main()
{
        point from execution starts
}
```

main function has two sections
1. declaration section : In this the variables and their data types are declared.
2. Executable section : This has the part of program which actually performs the task we need.

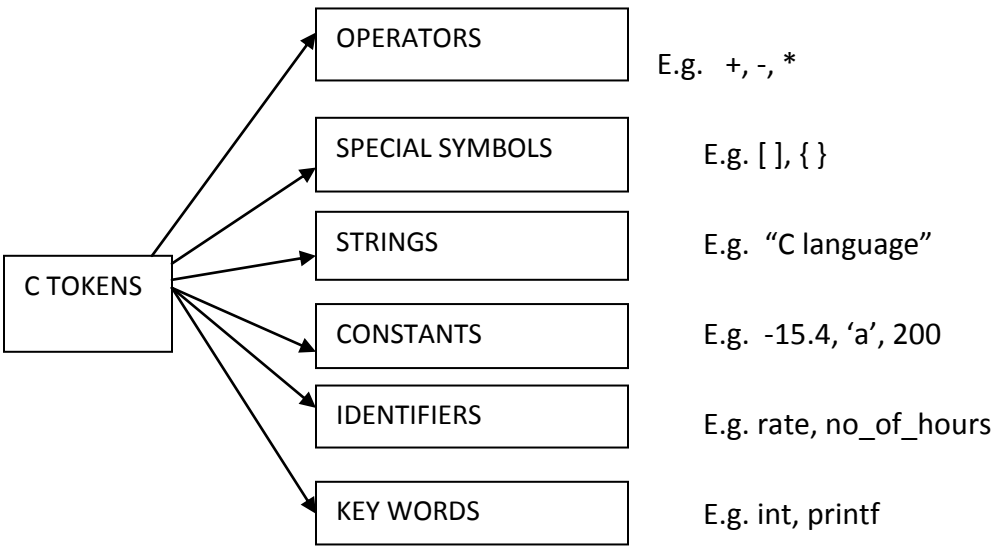
**6. SUB PROGRAM OR FUNCTION SECTION :** This has all the sub programs or the functions which our program needs.

**SIMPLE 'C' PROGRAM:**

```
/* simple program in c */
#include<stdio.h>
main()
{
printf("welcome to c programming");
} /*  End of main */
```

## C-TOKENS :

Tokens are individual words and punctuations marks in English language sentence. The smallest individual units are known as C tokens.



A C program can be divided into these tokens. A C program contains minimum 3 C tokens no matter what the size of the program is.

## DATA TYPES :

To represent different types of data in C program we need different data types. A data type is essential to identify the storage representation and the type of operations that can be performed on that data. C supports four different classes of data types namely

1. Basic data types (or) fundamental data types (or) primary data types
2. Derives data types
3. User defined data types
4. Pointer data types and
5. Enumerated data types

**1. BASIC DATA TYPES:**

A data type is a set of values and set of operations on those values. A standard data type in C is a data type that is predefined, such as *int, char, float and double.*

***Data type int:*** in mathematics, integers are whole numbers. The int data type is used to represent integers in C. Because of the finite size of a memory cell, not all integers cn be represented by type int. ANSI C specifies that the range of data type int must include at least the values -32768 through + 32767. Every integer variable occupies two bytes of memory to store a single integer variable.

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

| TYPE | STORAGE SIZE | VALUE RANGE |
|---|---|---|
| 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 | 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 |

To get the exact size of a type or a variable on a particular platform, you can use the **sizeof** operator. The expressions *sizeof(type)* yields the storage size of the object or type in bytes.

***Data type char:*** The char data type occupies one byte of memory to store for every single character variable. The range of character is -128 to +127. Char is supposed to store characters but not numbers, so why this range? The answer is that, in memory characters are stored in their ASCII codes.

For example, the character 'A' has the ASCII code 65. In memory we will not store 'A' but 65 (in binary number format)

Character variables can include any letter from the alphabet or from the ASCII chart and numbers 0-9 that are given within single quotes. In C, a number that is given in single quotes is not the same as a number without them. This is because 2 is treated as an integral value but '2' is considered as a character not an integer.

| TYPE | STORAGE SIZE | VALUE RANGE |
|---|---|---|
| CHAR | 1 BYTE | -128 to 127 or 0 to |
| UNSIGNED CHAR | 1 BYTE | 0 to 255 |
| SIGNED CHAR | 1 BYTE | -128 to 127 |

***Data types float and double:*** in computer memory, float and double values are stored in mantissa and exponent forms where the exponent represents power of 2 (not 10). The number of bytes used to represent a floating point number generally depends on the precision of the value. While float is used to declare single-precision values, double is used to represent double precision values.

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 BYTES | 3.4E-38 to 3.4E+38 | 6 DECIMAL PLACES |
| DOUBLE | 8 BYTES | 1.7E-308 to 1.7E+308 | 15 DECIMAL PLACES |
| LONG DOUBLE | 10 BYTES | 3.4E-4932 TO 1.1E+4932 | 19 DECIMAL PLACES |

```
/* PROGRAM FOR DATA TYPES*/
#include<stdio.h>
main()
{
        int a=10;
        float b=3.6;
        char c='a';
        printf(" value of a=%d",a);
        printf(" value of b=%f",b);
        printf(" value of c=%c",c);
        printf("size of integer variable a is %d",sizeof(a));
        printf("size of float variable b is %d",sizeof(b));
        printf("size of character variable c is %d",sizeof(c));
}
```

When you compile and execute the above program, it produces the following result on Linux:

```
value of a=10
value of b=3.600000
value of c=a
size of integer variable a is 2
size of float variable b is 4
size of character variable c is 1
```

**DERIVED DATA TYPES:**

Derived datatypes are used in 'C' to store a set of data values. Arrays and Structures are examples for derived data types.

Ex: int a[10];
char name[20];

**USER DEFINED DATATYPES:**

C Provides a facility called typedef for creating new data type names defined by the user. For Example,the declaration ,

**typedef int Integer;**

makes the name Integer a synonym of int.Now the type Integer can be used in declarations , casts, etc, like

**Integer num1, num2;**

Which will be treated by the C compiler as the declaration of num1 ,num2as int variables.
"typedef" ia more useful with structures and pointers.

**POINTER DATA TYPES:**

A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before you can use it to store any variable address. The general form of a pointer variable declaration is:

```
data type *var-name;
```

Here, **data type** is the pointer's base type; it must be a valid C data type and **var-name** is the name of the pointer variable. The asterisk * you used to declare a pointer is the same asterisk that you use for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Following are the valid pointer declaration:

```
int   *ip;   /* pointer to an integer */
```

```
double *dp;   /* pointer to a double */
float  *fp;   /* pointer to a float */
char   *ch    /* pointer to a character */
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

**ENUEMRATED DATA TYPE:**

The enumerated data type is a user-defined data type based on the standard integer type. An enumeration consists of a set of named integer constants. In other words, in an enumerated type, each integer value is assigned an identifier. This identifier (which is also known an enumeration constant) can be used as a symbolic name to make the program more readable.

To define enumerated data types, we use the keyword enum, which is the abbreviation for ENUMERATE. Enumerations create new data types to contain values that are not limited t the values that fundamental data types may take. The syntax of creating an enumerated data type can be given as follows.

***enum enumeration_name { identifier1, identifier2, ……. Identifier n};***

The ***enum*** keyword is basically used to declare and initialize a sequence of integer constants. Here ***enumeration_name*** is optional. Consider the following example, which creates a new type of variable called COLORS to store colors constants.

***enum COLORS { RED, BLUE, GREEN, BLACK, YELLOW, PURPLE, WHITE}***

Note that no fundamental data type is used in the declaration of COLORS. After this statement , COLORS is the name given to the set of constants. Here, COLORS is the name given to the set of constants. In case you do not assign any value to a constant, the default value for the first one in the list- *RED* (in our example), has the value of 0. The rest of the undefined constants have a value 1 more than its previous one. That is, if you do not initialize the constants, then each one would have a unique value. The firs would be zero and the rest would count upwards. So, in our example,

***RED=0, BLUE=1, GREEN=2, BLACK=3, YELLOW=4, PURPLE=5, WHITE=6***

## VARIABLES:

A *variable* is defined as a meaningful name given to the data storage location in computer memory. A quantity that can vary during the execution of a program is known as a variable. To identify a quantity we name the variable for example if we are calculating a sum of two numbers we will name the variable that will hold the value of sum of two numbers as 'sum'.

C language supports two basic kinds of variables – numeric and character

**IDENTIFIERS :**

Identifiers are the names of the variables and other program elements such as functions, arrays etc are known as identifiers.

There are few rules that govern the way variable are named (identifiers).

1. Identifiers or variables should start with a character or underscore (_) but not with an number (digit).
2. Variable name can be the combination of A-Z, a-z, 0-9, _(Underscore).
3. Variable name should not exceed 32 characters. If a variable name is having more that 8 characters the first 8 characters are treated as important one and will be identified by first 8 characters only.
4. It should not contain the spaces in between the characters of the variable name.
5. Keywords are not used as variable names.
6. No special character is allowed as part of variable name except underscore (_).

After naming a variable we need to declare it to compiler of what data type it is.
The format of declaring a variable is

**data-type id1, id2,.....idn;**

where **data type** could be float, int, char or any of the data types.

**id1, id2, id3** are the names of variable we use. In case of single variable no commas are required.

**EX:** float a, b, c;
int e, f, grand total;
char present_or_absent;

**ASSIGNING VALUES :**

When we name and declare variables we need to assign value to the variable. In some cases we assign value to the variable directly like

a=10;

in our program.

In some cases we need to assign values to variable after the user has given input for that.

eg we ask user to enter any no and input it

**/* write a program to show assigning of values to variables */**

```
#include<stdio.h>
main()
{
        int a;
        float b;
        printf("Enter any number\n");
        b=190.5;
        scanf("%d",&a);
```

```
        printf("user entered %d", a);
        printf("B's values is %f", b);
}
```

## CONSTANTS:

The constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **literals**.

Constants can be of any of the basic data types like *an integer constant, a floating constant, a character constant, or a string literal*. There are also enumeration constants as well. The **constants** are treated just like regular variables except that their values cannot be modified after their definition.

### Integer literals

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.

Here are some examples of integer literals:

```
212       /* Legal */
215u      /* Legal */
0xFeeL    /* Legal */
078       /* Illegal: 8 is not an octal digit */
032UU     /* Illegal: cannot repeat a suffix */
```

Following are other examples of various type of Integer literals:

```
85       /* decimal */
0213     /* octal */
0x4b     /* hexadecimal */
30       /* int */
30u      /* unsigned int */
30l      /* long */
30ul     /* unsigned long */
```

### Floating-point literals

A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form.

While representing using decimal form, you must include the decimal point, the exponent, or both and while representing using exponential form, you must include the integer part, the fractional part, or both. The signed exponent is introduced by ***e*** or ***E***.

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 */
```

**Character constants**

Character literals are enclosed in single quotes, e.g., ***'x'*** and can be stored in a simple variable of **char** type.

A character literal can be a plain character (e.g., 'x'), an escape sequence (e.g., '\t'), or a universal character (e.g., '\u02C0').

There are certain characters in C when they are preceded by a backslash they will have special meaning and they are used to represent like newline (\n) or tab (\t). Here, you have a list of some of such escape sequence codes:

| 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 |

Following is the example to show few escape sequence characters:

```
#include <stdio.h>
int main()
{
  printf("Hello\tWorld\n\n");

  return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Hello   World
```

## String literals

String literals or constants are enclosed in double quotes "". A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters.
You can break a long line into multiple lines using string literals and separating those using whitespaces.
Here are some examples of string literals. All the three forms are identical strings.

```
"hello, dear"

"hello, \

dear"

"hello, " "d" "ear"
```

## Defining Constants

There are two simple ways in C to define constants:

1. Using **#define** preprocessor.
2. Using **const** keyword.

### The #define Preprocessor

Following is the form to use #define preprocessor to define a constant:

```
#define identifier value
```

Following example explains it in detail:

```
#include <stdio.h>

#define LENGTH 10
#define WIDTH  5
#define NEWLINE '\n'

int main()
{
  int area;

  area = LENGTH * WIDTH;
  printf("value of area : %d", area);
  printf("%c", NEWLINE);

  return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
value of area : 50
```

The const Keyword

You can use **const** prefix to declare constants with a specific type as follows:

```
const type variable = value;
```

Following example explains it in detail:

```
#include <stdio.h>
main()
{
  const int  LENGTH = 10;
  const int  WIDTH  = 5;
  const char NEWLINE = '\n';
  int area;
    area = LENGTH * WIDTH;
  printf("value of area : %d", area);
  printf("%c", NEWLINE);
}
```

When the above code is compiled and executed, it produces the following result:

```
value of area : 50
```

Note that it is a good programming practice to define constants in CAPITALS.

There is one another kind of constant i.e **Enumeration constant**, it is a list of constant integer values.

**Ex.: enum color { RED, Green, BLUE }**

The first name in the enum has the value 0 and the next 1 and so on unless explicit values are specified.If not all values specified , unspecified values continue the progression from the last specified value. For example

**enum months { JAN=1, FEB,MAR, …, DEC }**

Where the value of FEB is 2 and MAR is 3 and so on.

Enumerations provide a convenient way to associate constant values with names.

**KEYWORDS :**There are certain words, called keywords (reserved words) that have a predefined meaning in 'C' language. These keywords are only to be used for their intended purpose and not as identifiers. The following table shows the standard 'C' keywords.

| | | | | | |
|---|---|---|---|---|---|
| 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 | | | | |

## OPERATORS:

An operator is a symbol that tells the compiler to perform certain mathematical or logical manipulations. They form expressions.

C operators can be classified as

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment or Decrement operators
6. Conditional operator
7. Bit wise operators
8. Special operators

### 1. Arithmetic Operators

All basic arithmetic operators are present in C.

| operator | meaning |
|---|---|
| + | addition |
| - | subtraction |
| * | Multiplication |
| / | Division |
| % | modulo division (remainder) |

An arithmetic operation involving only real operands (or integer operands) is called real arithmetic (or integer arithmetic). If a combination of arithmetic and real is called mixed mode arithmetic.

Following table shows all the arithmetic operators supported by C language. Assume variable **A** holds 10 and variable **B** holds 20 then:

**Show Examples**

| Operator | Description | Example |
|---|---|---|
| + | Adds two operands | A + B will give 30 |
| - | Subtracts second operand from the first | A - B will give -10 |
| * | Multiplies both operands | A * B will give 200 |
| / | Divides numerator by de-numerator | B / A will give 2 |
| % | Modulus Operator and remainder of after an integer division | B % A will give 0 |

**/* programs to exhibit the use of Arithmetic operators */**

```
#include<stdio.h>
main()
{
  int sum, mul, modu;
  float sub, divi;
  int i,j;
  float l, m;
 printf("Enter two integers ");
 scanf("%d%d",&i,&j);
 printf("Enter two real numbers");
 scanf("%f%f",&l,&m);
 sum=i+j;
 mul=i*j;
 modu=i%j;
 sub=l-m;
 divi=l/m;
 printf("sum is %d", sum);
 printf("mul is %d", mul);
 printf("Remainder is %d", modu);
 printf("subtraction of float is %f", sub);
 printf("division of float is %f", divi);
}
```

**2. RELATIONAL OPERATORS :** We often compare two quantities and depending on their relation take certain decisions for that comparison we use relational operators.

| operator | meaning |
|---|---|
| < | is less than |
| > | is greater than |
| <= | is less than or equal to |
| >= | is greater than or equal to |
| == | is equal to |
| != | is not equal to |

Following table shows all the relational operators supported by C language. Assume variable **A** holds 10 and variable **B** holds 20, then:

**Show Examples**

| Operator | Description | Example |
|---|---|---|
| == | Checks if the values of two operands are equal or not, if yes then condition becomes true. | (A == B) is not true. |
| != | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. | (A != B) is true. |
| > | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (A > B) is not true. |
| < | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | (A < B) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (A >= B) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (A <= B) is true. |

**/* PROGRAM FOR RELATIONAL OPERATORS*/**

```
#include <stdio.h>
 main()
{
   int m=40,n=20;
   if (m == n)
   {
      printf("m and n are equal");
   }
   else
   {
      printf("m and n are not equal");
   }
}
```

**3. LOGICAL OPERATORS :** An expression of this kind which combines two or more relational expressions is termed as a logical expressions or a compound relational expression. The operators and truth values are

| Operator 1 | operator 2 | op-1 && op-2 | op-1 \|\| op-2 |
|---|---|---|---|
| non-zero | non-zero | 1 | 1 |
| non-zero | 0 | 0 | 1 |
| 0 | non-zero | 0 | 1 |
| 0 | 0 | 0 | 0 |

| Operator 1 | !operator 1 |
|---|---|
| non-zero | zero |
| zero | non-zero |

Following table shows all the logical operators supported by C language. Assume variable **A**holds 1 and variable **B** holds 0, then:

**Show Examples**

| Operator | Description | Example |
|---|---|---|
| && | Called Logical AND operator. If both the operands are non-zero, then condition becomes true. | (A && B) is false. |
| \|\| | Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true | (A \|\| B) is true. |
| ! | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. | !(A && B) is true. |

**/* PROGRAM FOR LOGICAL OPERATORS*/**

```
#include <stdio.h>
 int main()
{
   int m=40,n=20;
   int o=20,p=30;
   if (m>n && m !=0)
   {
      printf("&& Operator : Both conditions are true\n");
   }
   if (o>p || p!=20)
   {
      printf("|| Operator : Only one condition is true\n");
   }
  }
```

**4. ASSIGNMENT OPERATORS :** They are used to assign the result of an expression to a variable. The assignment operator is '='.

Syntax:

**v op= exp**

where **v** is variable

**op** binary operator

**exp** expression

**op=** short hand assignment operator

short hand assignment operators

| use of simple assignment operators | use of short hand assignment operators |
|---|---|
| a=a+1 | a+=1 |
| a=a-1 | a-=1 |
| a=a%b | a%=b |

There are following assignment operators supported by C language:

**Show Examples**

| Operator | Description | Example |
|---|---|---|
| = | Simple assignment operator, Assigns values from right side operands to left side operand | C = A + B will assign value of A + B into C |
| += | Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand | C -= A is equivalent to C = C - A |
| *= | Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand | C *= A is equivalent to C = C * A |
| /= | Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand | C /= A is equivalent to C = C / A |
| %= | Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand | C %= A is equivalent to C = C % A |
| <<= | Left shift AND assignment operator | C <<= 2 is same as C = C << 2 |
| >>= | Right shift AND assignment operator | C >>= 2 is same as C |

| | | |
|---|---|---|
| | | = C >> 2 |
| &= | Bitwise AND assignment operator | C &= 2 is same as C = C & 2 |
| ^= | bitwise exclusive OR and assignment operator | C ^= 2 is same as C = C ^ 2 |
| \|= | bitwise inclusive OR and assignment operator | C \|= 2 is same as C = C \| 2 |

**/* PROGRAM FOR ASSIGNMENT OPERATORS*/**

```
# include <stdio.h>
 main()
{
        int Total=0,i;
        for(i=0;i<10;i++)
        {
                Total+=i; // This is same as Total = Total+i
        }
        printf("Total = %d", Total);
}
```

**5. INCREMENT AND DECREMENT OPERATORS :**

++ and -- are called increment and decrement operators. The increment and decrement operators are used to increment the variable by 1, decrement operator is used to decrement by 1. The increment and decrement operators are having two different forms that is pre increment and post increment, similarly pre-decrement and post decrement.

Both increment and decrement operators are unary operators and are shown below.
Pre-increment example is **++i** and post increment example is **i++**
Pre-decrement example is **--i** and post decrement example is **i--**
Where **i** is a variable name.

The difference between ++m and m++ is, where **m** is a variable name here

if m=5; y=++m then it is equal to m=5;m++;y=m;

if m=5; y=m++ then it is equal to m=5;y=m;m++;

Following table shows all the arithmetic operators supported by C language. Assume variable **A** holds 10 and variable **B** holds 20 then:

**Show Examples**

| Operator | Description | Example |
|---|---|---|
| ++ | Increments operator increases integer value by one | A++ will give 11 |
| -- | Decrements operator decreases integer value by one | A-- will give 9 |

**/* PROGRAM FOR INCREMENT AND DECREMENT OPERATORS*/**

```
#include<stdio.h>
main()
{
        int i;
        printf("Enter a number");
        scanf("%d", &i);
        i++;
        printf("after incrementing  %d ", i);
        i--;
        printf("after decrement %d", i);
}
```

**6. CONDITIONAL OPERATOR :** A ternary operator pair "?:" is available in C to construct conditional expressions of the form

```
exp1 ? exp2 : exp3;
```

It work as

if exp1 is true then exp2 will be executed as answer else exp3 will be executed.

**/* program using ternary operator and assignment  */**

```
#include<stdio.h>
main()
{
        int i,j,large;
        printf("Enter two numbers ");
        scanf("%d%d",&i,&j);
        large=(i>j)?i:j;
        printf("largest of two is  %d",large);
}
```

**7. BIT WISE OPERATORS :** C supports special operators known as bit wise operators for manipulation of data at bit level. They are not applied to float or double.

| operator | meaning |
|---|---|
| & | Bitwise AND |
| \| | Bitwise OR |
| ^ | Bitwise exclusive OR |
| << | left shift |
| >> | right shift |
| ~ | one's complement |

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ are as follows:

| p | q | p & q | p \| q | p ^ q |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 |

Assume if A = 60; and B = 13; now in binary format they will be as follows:

A = 0011 1100

B = 0000 1101

---

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A  = 1100 0011

The Bitwise operators supported by C language are listed in the following table. Assume variable A holds 60 and variable B holds 13, then:

**Show Examples**

| Operator | Description | Example |
|---|---|---|
| & | Binary AND Operator copies a bit to the result if it exists in both operands. | (A & B) will give 12, which is 0000 1100 |
| \| | Binary OR Operator copies a bit if it exists in either operand. | (A \| B) will give 61, which is 0011 1101 |
| ^ | Binary XOR Operator copies the bit if it is set in one operand but not both. | (A ^ B) will give 49, which is 0011 0001 |
| ~ | Binary Ones Complement Operator is unary and has | (~A ) will give -61, |

| | | |
|---|---|---|
| | the effect of 'flipping' bits. | which is 1100 0011 in 2's complement form. |
| << | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. | A << 2 will give 240 which is 1111 0000 |
| >> | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | A >> 2 will give 15 which is 0000 1111 |

**9. SPECIAL OPERATORS :** These operators which do not fit in any of the above classification are ,(comma), sizeof, Pointer operators(& and *) and member selection operators (. and ->). The comma operator is used to link related expressions together.
sizeof operator is used to know the number of bytes occupied by the given operand operand.

There are few other important operators including **sizeof** and **? :** supported by C Language.

**Show Examples**

| Operator | Description | Example |
|---|---|---|
| sizeof() | Returns the size of an variable. | sizeof(a), where a is integer, will return 2. |
| & | Returns the address of an variable. | &a; will give actual address of the variable. |
| * | Pointer to a variable. | *a; will pointer to a variable. |
| ? : | Conditional Expression | If Condition is true ? Then value X : Otherwise value Y |
| , | Comma operator used to separate any two values | int a, b, c, d; |

## EXPRESSIONS:

An expression is a sequence of operands and operators that reduces to a single value. Expression can be simple or complex. An **operator** is a syntactical token that requires an action be taken. An **operand** is an object on which an operation is performed.

A simple expression contains only one operator. E.g: 2 + 3 is a simple expression whose value is 5.A complex expression contains more that one operator. E.g: 2 + 3 * 2.

To evaluate a complex expression we reduce it to a series of simple expressions. In this first we will evaluate the simple expression 3 * 2 (6)and then the expression 2 + 6, giving a result of 8.

An expression can be divided into six categories based on the number of operators, positions of the operands and the precedence of operators. C supports the following six types of operators.

1. Primary expression
2. Post fix expression
3. Prefix expression
4. Unary expression
5. Binary expression
6. Ternary expression

**Primary Expression:** in C, the operand in the primary expression can be a Name, a Constant or a parenthesized expression.

Name is any identifier for a variable. A constant is the one whose value cannot be changed during program execution. Any value enclosed within parenthesis must be reduced to single value. A complex expression can be converted into primary expression by enclosing it with parenthesis. The following is an example

( 3 * 5 + 8 ); (or) ( c = a = 5 * c );

**Postfix Expression:** The postfix expression consists of one operand and one operator.
For example ***a function call***
The function name is operand and the parenthesis is the operator.

The other examples are post increment and post decrement. In post increment the variable is increased by 1, a++ results in the variable increment by 1. Similarly, in post decrement, the variable is decreased by 1, a—results in the variable decreased by 1.

**Prefix Expression:** Prefix expression consists of one operand and one operator, the operand come after the operator. Examples of prefix expressions are prefix increment and pre fix decrement i.e ++a, --a. The only difference between postfix and prefix operators is, in the prefix operator, the effect take place before the expression that contains operators is evaluated. It is other way in case of post-fix operations.

**Unary Expressions :** An unary expression is like a prefix expression consists of one operand and one operator and the operand comes after the operator.

Ex: ++A; -b; -c; +d;

**Binary Expression:** Binary Expressions are the Expressions which consists of two operands and an operator. Any two variables are added, subtracted, multiplied and divided in a binary expression.

**Ex: a+b; a-b; a*b; a/b; etc**

**Ternary Expression**: Ternary Expressions are also called Conditional Expressions. Ternary Expression is an expression which consists of a ternary operator pair ***"? :"***

**Ex: exp 1? exp 2: exp 3;**

In the above example, if exp1 is true exp2 is executed else exp3 is executed.

## Precedence and Associativity:

Every operator has a precedence value. An expression containing more than one operator is known as complex expression. Complex expressions are executed according to the precedence of operators. The order in which the operators in a complex expression are evaluated is determined by a set of priorities known as **precedence,** the higher the precedence, the earlier the expression containing the operator is evaluated. If two operators with the same precedence occur in a complex expression ,another attribute of an operator ,its associativety, takes control. **Associativity** is the parsing direction used to evaluate the expression. It can be either left-to-right or right-to-left. When two operators with the same precedence occur in an expression and their associativity is left-to-right the the left operator is evaluated first and proceeds to right side. If it is having right-to-left associativity, then right operator is evaluated first and proceeds to left.

For example, in the expression 3*4/6, there are two operators multiplication and division, with the same precedence and left-to-right associativity. Therefore the multiplication is evaluated before the division.

For example x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has 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.
**The following table shows the precedence and associativity of operators:**

| Precedence | Operator | Description | Associativity |
|---|---|---|---|
| 1 | ++ -- | Suffix/postfix increment and decrement | Left-to-right |
| | () | Function call | |

| | | | |
|---|---|---|---|
| | [] | Array subscripting | Left-to-right |
| | . | Structure and union member access | |
| | -> | Structure and union member access through pointer | |
| | (type){list} | Compound literal) | |
| 2 | ++ -- | Prefix increment and decrement | Right-to-left |
| | + - | Unary plus and minus | |
| | ! ~ | Logical NOT and bitwise NOT | |
| | (type) | Type cast | |
| | * | Indirection (dereference) | |
| | & | Address-of | |
| | sizeof | Size-of | |
| | _Alignof | Alignment requirement) | |
| 3 | * / % | Multiplication, division, and remainder | Left-to-right |
| 4 | + - | Addition and subtraction | |
| 5 | << >> | Bitwise left shift and right shift | |
| 6 | < <= | For relational operators < and ≤ respectively | |
| | > >= | For relational operators > and ≥ respectively | |
| 7 | == != | For relational = and ≠ respectively | |
| 8 | & | Bitwise AND | |
| 9 | ^ | Bitwise XOR (exclusive or) | |
| 10 | \| | Bitwise OR (inclusive or) | |
| 11 | && | Logical AND | |
| 12 | \|\| | Logical OR | |
| 13 | ?: | Ternary conditional | |
| 14 | = | Simple assignment | |

| | | | |
|---|---|---|---|
| | += -= | Assignment by sum and difference | Right-to-Left |
| | *= /= %= | Assignment by product, quotient, and remainder | |
| | <<= >>= | Assignment by bitwise left shift and right shift | |
| | &= ^= \|= | Assignment by bitwise AND, XOR, and OR | |
| 15 | , | Comma | Left-to-right |

**Example:**Try the following example to understand the operator precedence available in C programming language:

```
#include <stdio.h>
main()
{
   int a = 20,b = 10,c = 15,d = 5,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 );
  }
```

When you compile and execute the above program it produces the following result:

```
Value of (a + b) * c / d is : 90
Value of ((a + b) * c) / d is  : 90
Value of a + (b * c) / d is  : 50
```

## TYPE CONVERSION:

Type casting (or) Type conversion is a way to convert a variable from one data type to another data type. For example, if you want to store a long value into a simple integer then you can type cast long to int. You can convert values from one type to another explicitly using the **cast operator** as follows:

```
(type_name) expression
```

In an expression that involves two different data types, such as multiplying an integer and a floating point number to perform these evaluations, one of the types must be converted. We have two types of conversions

1. Implicit Type Conversion

2. Explicit Type Conversion

**IMPLICIT TYPE CONVERSION:**

When the types of the two operands in a binary expression are different automatically compiler itself converts one type to another. This is known as implicit type conversion.

**EXPLICIT TYPE CONVERSION :**

Explicit type conversion uses the unary cast operator, to convert data from one type to another. To cast data from one type to another, we specify the new type in parentheses before the value we want converted.

For example, to convert an integer, **a** to **float**, we code the expression like

(float) a

```
/*PROGRAM FOR TYPE CASTING*/
#include <stdio.h>
main()
{
  int sum = 17, count = 5;
  double mean;
  mean = (double) sum / count;
  printf("Value of mean : %f\n", mean );
}
```

When the above code is compiled and executed, it produces the following result:

```
Value of mean: 3.400000
```

It should be noted here that the cast operator has precedence over division, so the value of **sum** is first converted to type **double** and finally it gets divided by count yielding a double value.