

# Programming for problem solving using C Notes

## Unit - I

Computer History, Hardware, Software, Programming Languages and Algorithms: Components and functions of a Computer System, Concept of Hardware and Software  
Programming Languages: Low-level and High-level Languages, Program Design Tools: Algorithm, Flowchart, Pseudo code. Introduction to C Programming: Introduction, Structure of a C Program, Comments, Keywords, Identifiers, Data Types, Variables, Constants, Input/Output Statements, Operators, Type Conversion

## INTRODUCTION TO PROGRAMMING

### 1.1 Introduction to Computer Software

When we talk about a computer, we actually mean two things:

- First is the computer hardware that does all the physical work computers.
- Second is the computer software that commands the hardware what to do and how to do it.

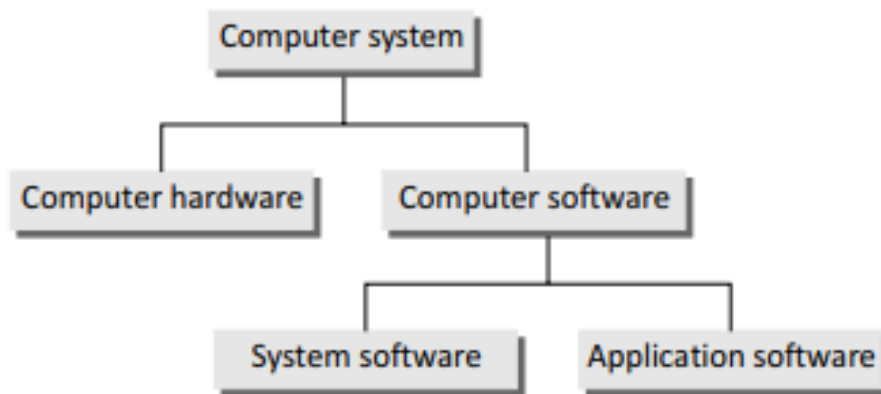


Figure 1.1 Parts of a computer system

### Computer Hardware

- Computer hardware is a digital machine (Physical components of a computer), it can only understand two basic states: on and off.
- The computer hardware cannot think and make decisions on its own. So, it cannot be used to analyse a given set of data and find a solution on its own.
- The hardware needs a software (a set of programs) to instruct what has to be done.

- Example: Mother Board, Processor, RAM, Monitor, Keyboard, Mouse, Hard Disk, DVD Drive, SMPS, etc.

## Computer Software

- Computer software is the set of instructions and associated data that direct the computer to do a task. Software is written by computer programmers using a programming language.
- The programmer writes a set of instructions (program) using a specific programming language. Such instructions are known as the source code.
- A program is a set of instructions that are arranged in a sequence to guide a computer to find a solution for a given problem. The process of writing a program is called programming.

### 1.2 Classification of Computer Software

Computer software can be broadly classified into two groups:

1. System Software
2. Application Software

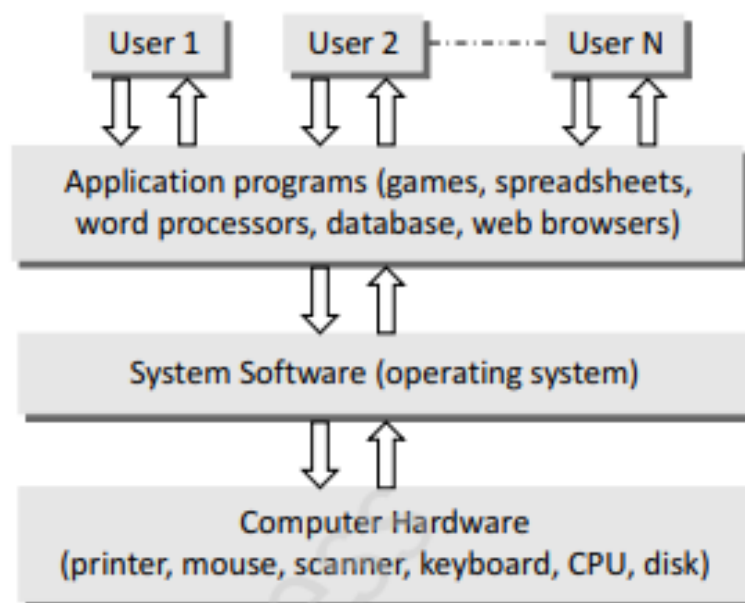


Figure 1.2 Relationship between hardware, system software, and application software

## System Software

System software represents programs that allow the hardware to run properly. System software is software designed to operate the computer hardware and to provide and maintain a platform for running application software. System software is transparent to the user and acts as an interface between the hardware of the computer and the application software that users need to run on the computer.

## Computer BIOS and Device Drivers

The computer BIOS and device drivers provide basic functionality to operate and control the hardware connected to or built into the computer.

BIOS or Basic Input/Output System is a de facto standard defining a firmware interface. BIOS is built into the computer and is the first code run by the computer when it is switched on. The key role of BIOS is to load and start the operating system.

BIOS is stored on a ROM chip built into the system and has a user interface like that of a menu (Below Figure) that can be accessed by pressing a certain key on the keyboard when the computer starts.

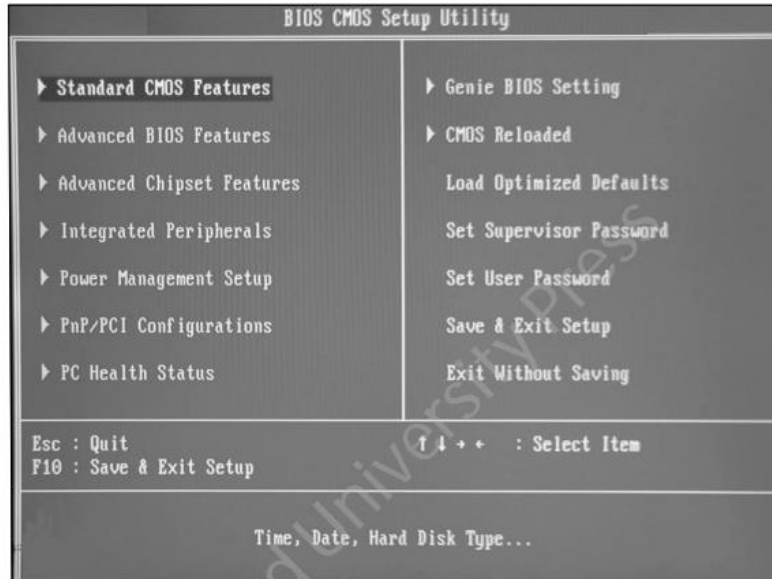


Figure 1.3 BIOS menu

To summarize, BIOS performs the following functions:

- Initializes the system hardware

- Initializes system registers
- Initializes power management system
- Tests RAM
- Tests all the serial and parallel ports
- Initializes CD/DVD drive and hard disk controllers

## **Operating System**

An Operating System (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

The primary goal of an operating system is to make the computer convenient and efficient to use. An operating system offers generic services to support user applications. Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc.

## **Utility Software**

Utility software is used to analyse, configure, optimize, and maintain the computer system. Utility programs may be requested by application programs during their execution for multiple purposes. It is used to support the computer infrastructure - in contrast to application software, which is aimed at directly performing tasks that benefit ordinary users.

Some of them are as follows: Disk checkers, Disk cleaners, File managers, System profiler, Anti-virus utilities, Data compression, Network utilities, Command line interface (CLI) and Graphical user interface (GUI).

## **Compiler, Interpreter, Linker, and Loader**

**Compiler** reads source code written in a programming language (the source language (**high-level programming**)) into machine language or assembly language (**lower level language**) comprising just two digits, 1s and 0s (the target language). The resultant code in 1s

and 0s is known as the object code. It provides error not of one line, but errors of the entire program. It executes as a whole and it is fast.

**Interpreter** reads only one line of a source program at a time and convert it into an object code. In case of error or same will be indicated instantly and it executes line by line and it is slow.

**Linker** (link editor binder) It is a program that combines object modules to form an executable program. Generally, in case of a large program, the programmers prefer to break a code into smaller modules as this simplifies the programming task.

**Loader** It is a special type of program that copies programs from a storage device to main memory, where they can be executed.

## Application Software

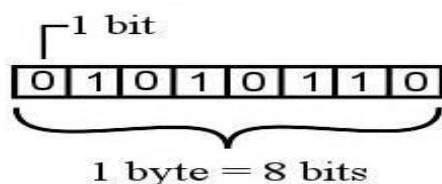
Application software (app for short) is a program or group of programs designed for end users. Application software is a type of computer software that employs the capabilities of a computer directly to perform a user-defined task.

Typical examples of software applications are word processors, spreadsheets, media players, education software, CAD, CAM, data communication software, and statistical and operational research software.

## 1.3 Representation of Data – Bits and Bytes

**Bits** – A bit is a smallest possible unit of data that a computer can recognize or use. Computer usually uses bits in groups.

**Bytes** – group of eight bits is called a byte. Half a byte is called a nibble.



The following table shows conversion of Bits and Bytes

- 1 Byte : 8 Bits

- 1024 Bytes : 1 Kilobyte
- 1024 Kilobytes : 1 Megabyte
- 1024 Megabytes : 1 Gigabyte
- 1024 Gigabytes : 1 Terabyte
- 1024 Terabytes : 1 Petabyte
- 1024 Petabytes : 1 Exabyte
- 1024 Exabytes : 1 Zettabyte
- 1024 Zettabytes : 1 Yottabyte
- 1024 Yottabytes : 1 Brontobyte
- 1024 Brontobyte : 1 Geopbytes

## **1.4 Programming Languages –High and Low Level Languages**

### **High Level Language**

High-level programming languages are easy for humans to read and understand, the computer understands the machine language that consists of numbers only. Each type of CPU has its own unique machine language.

High level languages is that they allow the programmer to write programs for all types of computers and systems. Every instruction in high level language is converted to machine language for the computer to comprehend.

### **Types of High-Level Language**

**Scripting languages** or scripts are essentially programming languages. These languages employ a high level construct which allows it to interpret and execute one command at a time. Scripting languages are easier to learn and execute than compiled languages. Some examples are AppleScript, JavaScript, Pearl etc.

**Object-Oriented Languages** These are high level languages that focus on the ‘objects’ rather than the ‘actions’. To accomplish this, the focus will be on data than logic. Some examples include Java, C+, C++, Python, Swift etc.

**Procedural Programming Language** This is a type of programming language that has well structured steps and complex procedures within its programming to compose a complete program. It has a systematic order functions and commands to complete a task or a program. FORTRAN, ALGOL, BASIC, COBOL are some examples.

### **Low Level Languages**

Low-level languages are the basic computer instructions or better known as machine codes. A computer cannot understand any instruction given to it by the user in English or any other high-level language. These low-level languages are very easily understandable by the machine.

The main function of low-level languages is to interact with the hardware of the computer. They help in operating, syncing and managing all the hardware and system components of the computer. They handle all the instructions which form the architecture of the hardware systems.

### **Types of Low-Level Language**

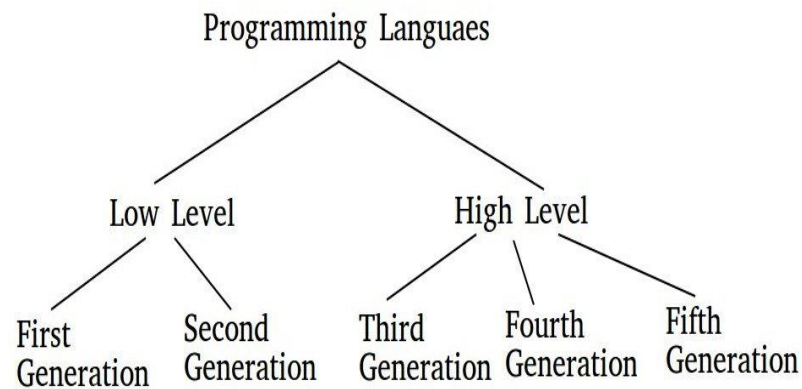
**Machine Language:** This is one of the most basic low-level languages. The language was first developed to interact with the first generation computers. It is written in binary code or machine code, which means it basically comprises of only two digits – 1 and 0.

**Assembly Language:** This is the second-generation programming language. It is a development on the machine language, where instead of using only numbers, we use English words, names, and symbols. It is the most basic computer language necessary for any processor.

### **1.5 Generation of Programming Languages**

1. First generation languages (1GL)
2. Second generation languages (2GL)

3. Third generation languages (3GL)
4. Fourth generation languages (4GL)
5. Fifth generation languages (5GL)



**First Generation (Machine Language):** Machine language is the lowest level of programming language that a computer understands. All the instructions and data values are expressed using 1s and 0s, corresponding to the 'on' and 'off' electrical states in a computer.

Advantages of first generation language

- They are translation free and can be directly executed by the computers.
- The programs written in these languages are executed very speedily and efficiently by the CPU of the computer system.
- The programs written in these languages utilize the memory in an efficient manner because it is possible to keep track of each bit of data.

**Second Generation (Assembly Language):** Assembly language is a low-level language that uses symbolic notation to represent machine language instructions. These languages are closely connected to machine language and the internal architecture of the computer system on which they are used.

The following instructions are a part of assembly language code to illustrate addition of two numbers:

MOV AX,4                      Stores value 4 in the AX register of CPU



MOV BX,6	Stores value 6 in the BX register of CPU
ADD AX,BX	Adds the contents of AX and BX registers. Stores the result in AX register

#### Advantages of second generation language

- It is easy to develop understand and modify the program developed in these languages are compared to those developed in the first generation programming language.
- The programs written in these languages are less prone to errors and therefore can be maintained with a great ease.

**Third Generation (High-Level Languages):** The third generation programming languages were designed to overcome the various limitations of the first and second generation programming languages. The languages of the third and later generation are considered as a high-level language because they enable the programmer to concentrate only on the logic of the programs without considering the internal architecture of the computer system.

#### Advantages of third generation programming language

- It is easy to develop, learn and understand the program.
- As the program written in these languages are less prone to errors they are easy to maintain.
- The program written in these languages can be developed in very less time as compared to the first and second generation language.

Examples: FORTRAN, ALGOL, COBOL, C++, C.

**Fourth generation language (Very High-level Languages):** The languages of this generation were considered as very high-level programming languages required a lot of time and effort that affected the productivity of a programmer. The fourth generation programming languages were designed and developed to reduce the time, cost and effort needed to develop different types of software applications.

#### Advantages of fourth generation languages

- These programming languages allow the efficient use of data by implementing the various database.
- They require less time, cost and effort to develop different types of software applications.
- The program developed in these languages are highly portable as compared to the programs developed in the languages of other generation.

Examples: SOL, CSS, coldfusion.

**Fifth generation language (Artificial Intelligence Language):** Fifth generation programming languages are centred on solving problems using constraints given to the program, rather than using an algorithm written by a programmer. Most constraint-based and logic programming languages and some declarative languages form a part of the fifth-generation languages.

Advantages of fifth generation languages

- These languages can be used to query the database in a fast and efficient manner.
- In this generation of language, the user can communicate with the computer system in a simple and an easy manner.

Examples: mercury, prolog, OPS5.

## **1.6 Program Design Tools**

### **Algorithms**

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

An algorithm should have the following characteristics –

- **Unambiguous** – Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.
- **Input** – An algorithm should have 0 or more well-defined inputs.
- **Output** – An algorithm should have 1 or more well-defined outputs, and should match the desired output.
- **Finiteness** – Algorithms must terminate after a finite number of steps.
- **Feasibility** – Should be feasible with the available resources.
- **Independent** – An algorithm should have step-by-step directions, which should be independent of any programming code.

#### **Advantages of Algorithms:**

1. It is a step-wise 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.

#### **Algorithm writing by using an example.**

**a) Write an algorithm to add two numbers entered by the user.**

Step 1: Start

Step 2: Accept the first integer as input from the user (num1)

Step 3: Accept the second integer as input from the user (num2)

Step 4: Calculate the sum of the two integers ( $\text{sum} = \text{num1} + \text{num2}$ )

Step 5: Display sum as the result

Step 6: End

**b) Write an algorithm to test the equality of two numbers.**

Step 1: Start

Step 2: Input first number as A

Step 3: Input second number as B

Step 4: if  $A = B$

    print " Equal"

    else

    print "Not Equal"




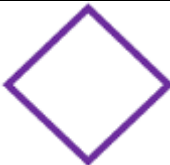

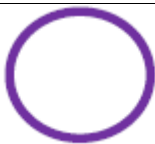

Step 5: End

**Flowcharts**

Flowchart is a diagrammatic representation of sequence of logical steps of a program. Flowcharts use simple geometric shapes to depict processes and arrows to show relationships and process/data flow.

## Flowchart Symbols

Here is a chart for some of the common symbols used in drawing flowcharts.

Symbol	Symbol Name	Purpose
	Start/Stop(Terminal)	Used at the beginning and end of the algorithm to show start and end of the program.
	Process	Indicates processes like mathematical operations.
	Input/ Output	Used for denoting program inputs and outputs.
	Decision	Stands for decision statements in a program, where answer is usually Yes or No.
	Arrow	Shows relationships between different shapes.
	On-page Connector	Connects two or more parts of a flowchart, which are on the same page.
	Off-page Connector	Connects two parts of a flowchart which are spread over different pages.

## Guidelines for Developing Flowcharts

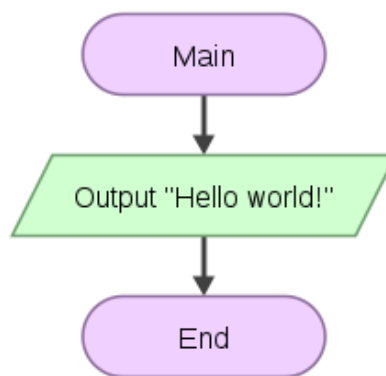
These are some points to keep in mind while developing a flowchart –

- Flowchart can have only one start and one stop symbol

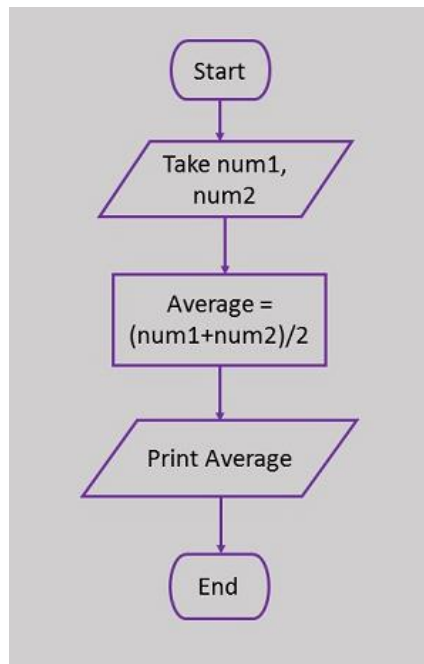
- On-page connectors are referenced using numbers
- Off-page connectors are referenced using alphabets
- General flow of processes is top to bottom or left to right
- Arrows should not cross each other

### Example Flowcharts

#### a) Flowchart displaying "Hello world!"



#### b) flowchart to calculate the average of two numbers.



## **Pseudo code**

**Pseudo code** is a term which is often used in programming and algorithm based fields. It is a methodology that allows the programmer to represent the implementation of an algorithm. Algorithms are represented with the help of pseudo codes as they can be interpreted by programmers no matter what their programming background or knowledge is.

Algorithm **is** an organized logical sequence of the actions or the approach towards a particular problem. A programmer implements an algorithm to solve a problem.

### **Advantages of Pseudocode**

- Improves the readability of any approach. It's one of the best approaches to start implementation of an algorithm.
- Acts as a bridge between the program and the algorithm or flowchart. Also works as a rough documentation, so the program of one developer can be understood easily when a pseudo code is written out. In industries, the approach of documentation is essential. And that is where a pseudo-code proves vital.
- The main goal of a pseudo code is to explain what exactly each line of a program should do, hence making the code construction phase easier for the programmer.

### **Examples:**

1. If student's grade is greater than or equal to 60

    Print "passed"

else

    Print "failed"

2. Set total to zero

Set grade counter to one

While grade counter is less than or equal to ten

    Input the next grade

Add the grade into the total

Set the class average to the total divided by ten

Print the class average.

## Types of Errors

Errors are the problems or the faults that occur in the program, which makes the behavior of the program abnormal, and experienced developers can also make these faults. Programming errors are also known as the bugs or faults, and the process of removing these bugs is known as debugging.

- Syntax error
- Run-time error
- Linker error
- Logical error
- Semantic error

### Syntax error

Syntax errors are also known as the compilation errors as they occurred at the compilation time, or we can say that the syntax errors are thrown by the compilers. These errors are mainly occurred due to the mistakes while typing or do not follow the syntax of the specified programming language.

For example:

If we want to declare the variable of type integer,

```
int a; // this is the correct form
```

```
Int a; // this is an incorrect form.
```

Commonly occurred syntax errors are:

- If we miss the parenthesis (}) while writing the code.
- Displaying the value of a variable without its declaration.
- If we miss the semicolon (;) at the end of the statement.



## Run-time error

Sometimes the errors exist during the execution-time even after the successful compilation known as run-time errors. When the program is running, and it is not able to perform the operation is the main cause of the run-time error. The division by zero is the common example of the run-time error. These errors are very difficult to find, as the compiler does not point to these errors.

Error:

warning: division by zero [-Wdiv-by-zero]

```
div = n/0;
```

## Linker Errors

These error occurs when after compilation we link the different object files with main's object using *Ctrl+F9* key(RUN). These are errors generated when the executable of the program cannot be generated. This may be due to wrong function prototyping, incorrect header files. One of the most common linker error is writing **Main()** instead of **main()**.

Error:

(.text+0x20): undefined reference to `main'

## Logical Errors

On compilation and execution of a program, desired output is not obtained when certain input values are given. These types of errors which provide incorrect output but appears to be error free are called logical errors. These errors solely depend on the logical thinking of the programmer and are easy to detect if we follow the line of execution and determine why the program takes that path of execution.

Error:

No Output

## Semantic Errors

This error occurs when the statements written in the program are not meaningful to the compiler.

## Error

error: lvalue required as left operand of assignment

a + b = c; //semantic error

## Testing & Debugging Approaches

### Testing

Testing is the process of verifying and validating that a software or application is bug free, meets the technical requirements as guided by its design and development and meets the user requirements effectively and efficiently with handling all the exceptional and boundary cases.

### Debugging

Debugging is the process of fixing a bug in the software. It can be defined as the identifying, analyzing and removing errors. This activity begins after the software fails to execute properly and concludes by solving the problem and successfully testing the software. It is an extremely complex and tedious task because errors need to be resolved at all stages of debugging.

TESTING	DEBUGGING
Testing is the process to find bugs and errors.	Debugging is the process to correct the bugs found during testing.
It is the process to identify the failure of implemented code.	It is the process to give the absolution to code failure.
Testing is the display of errors.	Debugging is a deductive process.
Testing is done by the tester.	Debugging is done by either programmer or developer.
There is no need of design knowledge in the testing process.	Debugging can't be done without proper design knowledge.
Testing can be done by insider as well as outsider.	Debugging is done only by insider. Outsider can't do debugging.

Testing can be manual or automated.	Debugging is always manual. Debugging can't be automated.
It is based on different testing levels i.e. unit testing, integration testing, system testing etc.	Debugging is based on different types of bugs.
Testing is a stage of software development life cycle (SDLC).	Debugging is not an aspect of software development life cycle, it occurs as a consequence of testing.
Testing is composed of validation and verification of software.	While debugging process seeks to match symptom with cause, by that it leads to the error correction.
Testing is initiated after the code is written.	Debugging commences with the execution of a test case.

## **INTRODUCTION TO C**

### **Introduction**

- C is mother language of all programming language.
- It is a popular computer programming language.
- It is procedure-oriented programming language.
- It is also called middle level programming language.

### **History of C Language**

“C” is a programming language developed at AT & T Bell Laboratories of USA in 1972. It was developed Dennis Ritchie in late 1970’s. It began to replace the more familiar languages of that time like PL/1, ALGOL etc.

1. “C” became popular because of its reliability, simple and easy to use
2. It was friendly, capable and reliable
3. ALGOL 60 was developed and did not become popular because it was too general and too abstract.
4. They developed “CPL” (Combined Programming Language)
5. Next as it could not come up to make ALGOL 60 better one they moved to “BCPL” (Basic Combined Programming Language. Developed by Martin Richards Cambridge university)

6. At the same time a language called “B” written by ken Thompson at AT & T’S. Bell laboratories as a further simplification of BCPL.
7. “C” s compactness and coherence is mainly due to it’s one man language. Ex- LISP, AASCA.

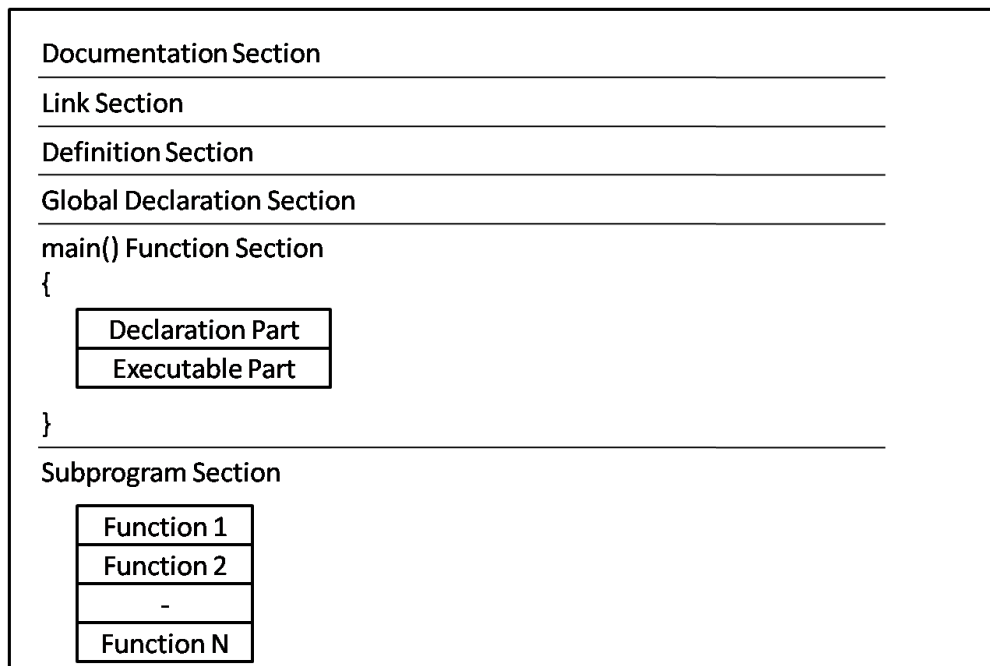
Language	Year	Developed By
AIGOL	1960	International Group
BCPL	1967	Martin Richard
B	1970	Ken Thompson
Traditional C	1972	Dennis Ritchie
K & R C	1978	Kernighan & Dennis Ritchie
ANSI C	1989	ANSI Committee
ANSI/ISO C	1990	ISO Committee
C99	1999	Standardization Committee

#### Features of “C” Language:

1. It is **robust** language because of rich set of binary in function
2. It is **efficient and fast** because of its variant datatypes and powerful operation.
3. It is highly **Portable** i.e., programs written in one computer can be run on another
4. It is well suited for structure program, thus allows the user to think about the problem in the terms of functional blocks.
5. Debugging, testing and maintenance is **easy**
6. **ability to extend** itself, we can continuously add our own functions to the program.

#### Structure of a C Program

A C program is a set of functions, data type definitions and variable declarations contained in a set of files. A C program always start its execution by the function with name main. Any function can invoke any other function and the variables declared outside the function are either global or local to the current file (if they are declared with the static prefix). The following figure shows the structure of a C program contained in several files.



**Documentation Section:** The documentation section is the part of the program where the programmer gives the details associated with the program. documentation section Consists of comments, some description of the program, programmer name and any other useful points that can be referenced later.

Example

```
/* Write a C Program to print HelloWorld */
```

**Link Section:** This part of the code is used to declare all the header files that will be used in the program. This leads to the compiler being told to link the header files to the system libraries.

Example

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

**Definition Section:** In this section, we define different constants. The keyword define is used in this part.

Example

```
#define PI=3.14
```

**Global Declaration Section:** This part of the code is the part where the global variables are declared. All the global variable used are declared in this part. The user-defined functions are also declared in this part of the code.

### Example

```
float area(float r);  
  
int a=7;
```

**Main Function Section:** Every C-programs needs to have the main function. Each main function contains 2 parts. A declaration part and an Execution part. The declaration part is the part where all the variables are declared. The execution part begins with the curly brackets and ends with the curly close bracket.

### Example

```
int main(void)  
  
{  
  
    int a=10;  
  
    printf(" %d", a);  
  
    return 0;  
  
}
```

**Sub Program Section:** All the user-defined functions are defined in this section of the program.

### Example

```
int add(int a, int b)  
  
{  
  
    return a+b;  
  
}
```

## Writing the First C Program

```
#include <stdio.h>

#include <conio.h>

void main()

{

    printf("Hello World!");

    getch();

}
```

### Describe the C Program :-

- #include <conio.h> includes the console input output library functions. The getch() function is defined in conio.h file.
- #include <stdio.h> includes the standard input output library functions. The printf() function is defined in stdio.h .
- void main() The main() function is the entry point of every program in c language. The void keyword specifies that it returns no value.
- printf() The printf() function is used to print data on the console.
- getch() The getch() function asks for a single character. Until you press any key, it blocks the screen.

### Output of Program is:-

Hello World!

### Header Files used in C Program

Header files offer these features by importing them into your program with the help of a preprocessor directive called **#include**. These preprocessor directives are responsible for instructing the C compiler that these files need to be processed before compilation.



Every C program should necessarily contain the header file **<stdio.h>** which stands for standard input and output used to take input with the help of **scanf() function** and display the output using **printf() function**.

**Basically, header files are of 2 types:**

1. Standard library header files: These are the pre-existing header files already available in the C compiler.
2. User-defined header files: Header files starting #define can be designed by the user.

### Syntax of Header File

```
#include<filename.h>
```

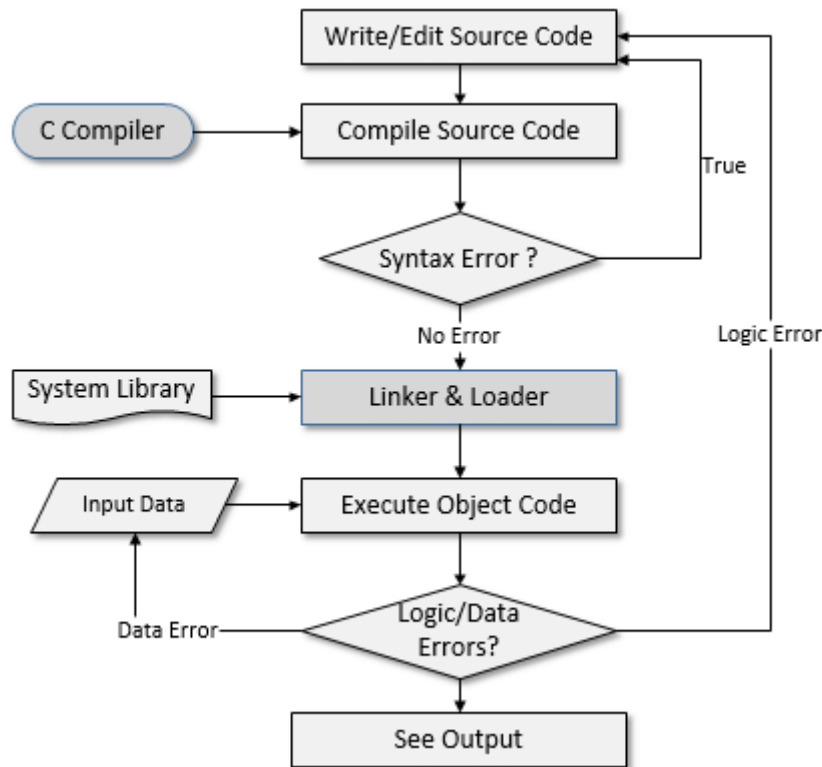
Here is the table that displays some of the header files in C language

Sr.No.	Header Files & Description
1	<b>stdio.h</b> Input/Output functions
2	<b>conio.h</b> Console Input/Output functions
3	<b>stdlib.h</b> General utility functions
4	<b>math.h</b> Mathematics functions

### Compiling and Executing C Programs

C program file is compiled and executed the compiler generates some files with the same name as that of the C program file but with different extensions.

Below image shows the compilation process with the files created at each step of the compilation process:



Process of compiling and running a C program

Every file that contains a C program must be saved with ‘.c’ extension. This is necessary for the compiler to understand that this is a C program file. Suppose a program file is named, first.c. The file first.c is called the source file which keeps the code of the program. Now, when we compile the file, the C compiler looks for errors. If the C compiler reports no error, then it stores the file as a .obj file of the same name, called the object file. So, here it will create the first.obj. This .obj file is not executable. The process is continued by the Linker which finally gives a .exe file which is executable.

### Important Points

- C program file (Source file) must save with .c extension.
- The compiler converts complete program at a time from high-level language to low-level language.
- Input to the compiler is .c file and output from the compiler is .exe file, but it also generates .obj file in this process.
- The compiler converts the file only if there are no errors in the source code.

- CPU places the result in User Screen window.

## TOKENS IN C

A token is the smallest element of a program that is meaningful to the compiler (or) In a “C” program the smallest individual units are known as “C” tokens. Tokens can be classified as follows:

1. Keywords
2. Identifiers
3. Constants
4. Strings
5. Special Symbols
6. Operators

**1. Keywords:** Keywords are reserved words by compiler. Keywords are assigned with fixed meaning and they cannot be used as variable name. No header file is needed to include the keywords.

C language supports **32** keywords which are given 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

## 2. Identifiers:

- Identifier refers to the name of variables, functions and arrays. These are user defined names and consists of a sequence of letters and digits.
- Both uppercase and lowercase letters can be used, and C language is case sensitive. A special symbol underscore ( \_ ) is also permitted.
- Rules for identifiers
  - First character must be an alphabet or underscore.
  - Must consist of only letters, digits or underscore.
  - Should not be a keyword and should not have any blank space.
  - C is a case – sensitive language i.e, AREA and area both are different identifiers.
- **Example:** -int num;

Char name;

Where num and name are identifier names.

**3. Constants:** constants in “C” are applicable to the values which not change during the execution of a program.

**Integer Constants:** Sequence of number 0-9 without decimal points, fractional part or any other symbols. It requires two or four bytes, can be +ve, -ve or Zero the number without a sign is as positive.

Eg: -10, +20, 40

**Real Constants:** Real constants are often known as floating constants.

Eg: 2.5, 5.521, 3.14 etc.

**Character Constants:** Single character constant: A single character constants are given within a pair of single quote mark.

Eg : ‘a’, ‘8’, etc.

**String Constant:** These are the sequence of character within double quote marks

Eg : “Straight” “India”, “4”

**4. Strings:** String in C are always represented as an array of characters having null character '\0' at the end of the string. This null character denotes the end of the string. Strings in C are enclosed within double quotes, while characters are enclosed within single characters. The size of a string is a number of characters that the string contains.

Now, we describe the strings in different ways:

```
char a[10] = "pragati"; // The compiler allocates the 10 bytes to the 'a' array.
```

```
char a[] = "pragati"; // The compiler allocates the memory at the run time.
```

```
char a[10] = {'p','r','a','g','t','i','\0'}; // String is represented in the form of characters.
```

**5. Special Symbols:** The following special symbols are used in C having some special meaning and thus, cannot be used for some other purpose.[] () {}, ; \* = #

- **Brackets []:** Opening and closing brackets are used as array element reference. These indicate single and multidimensional subscripts.
- **Parentheses ():** These special symbols are used to indicate function calls and function parameters.
- **Braces {}:** These opening and ending curly braces marks the start and end of a block of code containing more than one executable statement.
- **comma (,):** It is used to separate more than one statements like for separating parameters in function calls.
- **Semi colon :** It is an operator that essentially invokes something called an initialization list.
- **asterisk (\*):** It is used to create pointer variable.
- **assignment operator:** It is used to assign values.
- **pre processor(#):** The preprocessor is a macro processor that is used automatically by the compiler to transform your program before actual compilation.

**6. Operators:** Operators are symbols that triggers an action when applied to C variables and other objects. The data items on which operators act upon are called operands. Depending on the number of operands that an operator can act upon, operators can be classified as follows:

- **Unary Operators:** Those operators that require only single operand to act upon are known as unary operators.

**For Example** increment and decrement operators

- **Binary Operators:** Those operators that require two operands to act upon are called binary operators.

Binary operators are classified into :

1. Arithmetic operators
  2. Relational Operators
  3. Logical Operators
  4. Assignment Operators
  5. Conditional Operators
  6. Bitwise Operators
- **Ternary Operators:** These operators requires three operands to act upon. For Example Conditional operator(?:).

## **Basic Data Types**

C language is rich in data types

### **Syntax**

`datatype name variablename=value;`

ANSI – American National Standard Institute

ANSI C Supports Four classes of data types.

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

## Fundamental data types

1. Character (char)
2. Integer (int)
3. floating point (float)
4. double-precision (double)
5. void

### Range of data types:

Data type	Bytes in Ram	Range of data type
char	1 bytes	-128 to 127
int	2 bytes	-32, 768 to 32,767
float	4 bytes	$3.4 \times 10^{-38}$ to $3.4 \times 10^{38}$
double	8 bytes	$1.7 \times 10^{-308}$ to $1.7 \times 10^{308}$

**1. Character:** The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.

**Ex:** `char ch='h';`

**2. Integer:** Integers are whole numbers with a range of variables supported by a particular machine.

**Ex:** `int a=10,b=20,sum;`

- C has three classes of integer storage
  - short int
  - int
  - long int
- It has a set of qualifiers i.e.,
  - sign qualifier

unsigned qualifier

- short int uses half the range of storage amount of data
- signed integer uses one bit for sign and 15 bits for magnitude
- unsigned int use all the bits for the magnitude of the number and are positive.

**3. Floating:** It is used to store decimal numbers (numbers with floating point value) with single precision.

Ex: float a=2.5,b=10.5,div;

**4. Double:** It is used to store decimal numbers (numbers with floating point value) with double precision.

Ex: double a=1250.0023,b=2558.043;

**5.Void:** A void type has no value this is usually used to specify the return type of function , this function does not return any value to calling function

**Details list of data type:**

DATA TYPE	MEMORY (BYTES)	RANGE	FORMAT SPECIFIER
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
<b>int</b>	<b>4</b>	<b>-2,147,483,648 to 2,147,483,647</b>	<b>%d</b>
long int	8	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	8	0 to 4,294,967,295	%lu
long long int	8	-(2 <sup>63</sup> ) to (2 <sup>63</sup> )-1	%lld
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
<b>Char</b>	<b>1</b>	<b>-128 to 127 or 0 to 255</b>	<b>%c</b>
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c



<b>float</b>	<b>4</b>	<b>1.2E-38 to 3.4E+38</b>	<b>%f</b>
double	8	2.3E-308 to 1.7E+308	%lf
long double	16	3.4E-4932 to 1.1E+4932	%Lf

### **Variables**

This is a data name used for storing a data, its value may be changed during the execution. The variables value keep"s changing during the execution of the program

Eg : height, average, sum, etc.

**Declaration of Variable:** It tells the compiler what the variable name is used, what type of data is held by the variable.

**Syn:**

datatype v1,v2,...vn;

**Eg :**

int a, b; float sum; double ratio;

Representation of Constant

**Eg:**

const int r = 10;

Assigning values to variables

Eg :

int x,y; x= 10; y=5;

**Programs:** Program for variable declaration

main( )

{

float x,p;

x=10.1;

p=5.2;

```
printf ("x = %f", x);  
printf ("p = %f", p);  
}
```

**Output:**

x= 10.10000

p = 5.2

**Comments**

- Comments in C language are used to provide information about lines of code. It is widely used for documenting code.
- There are 2 types of comments in the C language.

1. Single Line Comments

2. Multi-Line Comments

**1. Single Line Comments**

- Single line comments are represented by double slash \\  
• **Example:**

```
#include<stdio.h>  
  
int main()  
{  
    //printing information  
  
    printf("Hello C");  
  
    return 0;  
}
```

**2. Mult Line Comments**

- Multi-Line comments are represented by slash asterisk \\* ... \*. It can occupy many lines of code.
- **Example:**

```
#include<stdio.h>

int main()

{

    /*printing information

    Multi-Line Comment*/

    printf("Hello C");

    return 0;

}
```

### **Input / Output Statements In C**

C has many input output functions in order to read data from input devices and display the results on the screen.

Classification of Input / Output statements in C

1. Formatted Functions
  - Input (scanf())
  - Output (printf())
2. Unformatted Functions
  - Input (getch(), getche(), gethar(), gets())
  - Output (putch(), puts())

### **Formatted Functions**

These functions read and write all types of data values. They require a conversion symbol to indent the data type using these functions the O/P can be presented in an aligned manner.

### **Data Types with conversion symbol (format string)**

	<b>Data Type</b>	<b>Conversion symbol</b>
Integer	Short integer	%d or %i
	Short unsigned	%u
	long signed	%ld
	Long unsigned	%lu
	Unsigned hexadecimal	%x
	Unsigned octal	%o
Real	float	%f or %g
	double	%lf
	signed character	%c
	unsigned char	%c
	string	%s

### **Escape Sequences with their ASCII values**

<b>Escape Sequence</b>	<b>Use</b>	<b>ASCII Value</b>
\n	New line	10
\b	Backspace	8
\f	Form feed	12
\'	Single Quote	39
\\	Back slash	92
\0	Null	0
\t	Horizontal tab	9
\r	Carriage return	13
\a	Alert	7

?	Question marks	63
\“	Double Quote	34

**scanf()** function is used to read values using key board. It is used for runtime assignment of variables.

**The general form of scanf() is**

```
scanf(“format String “ , list_of_addresses_of_Variables );
```

The format string contains - Conversion specifications that begin with % sign

Eg:

```
scanf(“ %d %f %c”, &a &b, &c);
```

&” is called the “address” operator. In scanf() the „&” operator indicates the memory location of the variable. So that the Value read would be placed at that location.

**printf()** function is used to Print / display values of variables using monitor.

The general form of printf() is

**Syntax:**

```
printf(“control String “ , list_of_Variables );
```

- Characters that are simply printed as they are
- Conversion specifications that begin with a % sign
- Escape sequences that begin with a „\” sign.

**Eg:**

```
main ( )
{
    int avg = 346;
    float per = 69.2;
```

```

        printf(" Average = %d \n percentage = %f", avg, per);
    }

```

### Output:

Average = 346

Percentage = 69.200000

printf( ) examines the format string from left to right and prints all the characters until it encounter a „%“ or „\“ on the screen. When it finds % (Conversion Specifier) it picks up the first value. when it finds „\“ (escape sequence) it takes appropriate action (\n-new line). This process continues till the end of format string is reached.

### Example: Program ( )

```

main ( )
{
    Float avg,per;
    printf("Enter values for avg & per");
    scanf(" %d %f", & avg, & per);
    printf( " Average = %d \n Percentage = %f", avg, per);
}

```

### O/P:

Enter values for avg & per 346 69.2

Average = 346

Percentage = 69.200000

### Unformatted Functions

**getchar ( )** function is used to read one character at a time from the key board

### Syntax

ch = getchar ( ); where ch is a char Var.

**Eg:**

```
main ( )  
{  
    char ch;  
  
    printf("Enter a char");  
    ch = getchar ( );  
    printf("ch =%c", ch);  
}
```

**O/P**

Enter a char M

M

ch = M

When this function is executed, the computer will wait for a key to be pressed and assigns the value to the variable when the “enter” key pressed.

**putchar ( )**: function is used to display one character at a time on the monitor.

**Syntax:**

```
putchar (ch);
```

**Eg:**

```
char ch = „M“  
  
putchar (ch);
```

The Computer display the value char of variable „ch“ i.e M on the Screen.

**getch ( )**: function is used to read a char from a key board and does not expect the “enter” key press.

**Syntax:**

```
ch = getch ( );
```

When this function is executed ,computer waits for a key to be pressed from the key board.

**getche ( )**: function is used to read a char from the key board without expecting the enter key to be pressed. The char read will be displayed on the monitor.

**Syntax:**

```
ch = getche ( );
```

Note that getche ( ) is similar to getch ( ) except that getche ( ) displays the key pressed from the key board on the monitor. In getch ( ) „e“ stands for echo.

**String I/O functions**

**gets ( )** function is used to read a string of characters including white spaces. Note that white spaces in a string cannot be read using scanf( ) with %s format specifier.

**Syntax:**

```
gets (S); where „S“ is a char string variable
```

**Ex:**

```
char S[ 20 ];
```

```
gets (S);
```

When this function is executed the computer waits for the string to be entered.

**Header Files used in C Program**

Header files offer these features by importing them into your program with the help of a preprocessor directive called **#include**. These preprocessor directives are responsible for instructing the C compiler that these files need to be processed before compilation.

Every C program should necessarily contain the header file **<stdio.h>** which stands for standard input and output used to take input with the help of **scanf() function** and display the output using **printf() function**.



**Basically, header files are of 2 types:**

3. Standard library header files: These are the pre-existing header files already available in the C compiler.
4. User-defined header files: Header files starting #define can be designed by the user.

### **Syntax of Header File**

```
#include<filename.h>
```

Here is the table that displays some of the header files in C language

<b>Sr.No.</b>	<b>Header Files &amp; Description</b>
1	<b>stdio.h</b> Input/Output functions
2	<b>conio.h</b> Console Input/Output functions
3	<b>stdlib.h</b> General utility functions
4	<b>math.h</b> Mathematics functions

## **Operators In C**

**Operator:** An operator is a symbol that tells the Computer to perform certain mathematical or logical manipulations.

**Expression:** An expression is a sequence of operands and operators that reduces to single value

**Eg:** 10+25 is an expression whose value is 35

C Operators can be classified into a no. of categories.

1. Arithmetic
2. Relational
3. Logical
4. Assignment
5. Increment and Decrement
6. Conditional
7. Bitwise
8. Special

**Arithmetic Operators:** C provides all the basic arithmetic operators, they are +, -, \*, /, % Integer division truncates any fractional part. The modulo division produces the remainder of an integer division.

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication

Operator	Meaning of Operator
/	division
%	remainder after division (modulo division)

**Integer Arithmetic:** When the operands in an expression are integers then the expression is an integer expression and the operation is called integer arithmetic. This always yields an integer value.

**Eg.** a = 14 and n = 4 then

a - b = 10      Note : During modulo division, the

a + b = 18      sign of the result is always the sign

a \* b = 56      of the first operand (the dividend )

a / b = 3      - 14 % 3 = -2

a % b = 2      -14 % - 3 = 2

14 % -3 = 2

**Write a program to illustrate the use of all Arithmetic operator**

```
main ( )
{
    int sum, prod , sub, div, mod, a, b ;

    printf("Enter values of a, b :");

    scanf("/.d %d", & a, & b) ;

    sum = a+b ;

    printf("sum = %d", sum);
```

```

    sub = a-b; printf("sub = %d", sub);

    prod = a * b ;

    printf("prod = %d", a* b);

    div = a/b;

    printf(" Div = %d", div);

    mod = a % b ;

    printf(" mod = %d",a % b);

}

```

**Real Arithmetic / Floating Pont Arithmetic:** Floating Point Arithmetic involves only real operands of decimal or exponential notation. If x, y & z are floats, then

$$x = 6.0/7.0 = 0.857143$$

$$y = -1.0/3.0 = 0.333333$$

$$z = 3.0/2.0 = 1.500000$$

% cannot be used with real operands

**Mixed mode Arithmetic:** When one of the operands is real and the other is integer the expression is a mixed mode arithmetic expression.

Eg:

$$15/10.0 = 1.500000$$

$$15/10 = 1$$

$$10/15 = 0$$

$$-10.0/15 = -0.666667$$

**Relational Operator:** These are the operators used to Compare arithmetic, logical and character expressions. The value of a relational express is either one or zero .it is 1 if one is the specified relation is true and zero if it is false.

The relational operators in C are

Operator	Meaning
<	is less than
< =	is less than or equal to
>	is greater than or equal to
> =	is greater than or equal to
==	is equal to
!=	is not equal to

**Example:**

```
#include <stdio.h>

int main()

{

    int a = 5, b = 5;

    printf("%d == %d is %d \n", a, b, a == b);

    printf("%d > %d is %d \n", a, b, a > b);

    printf("%d < %d is %d \n", a, b, a < b);

    printf("%d != %d is %d \n", a, b, a != b);

    printf("%d >= %d is %d \n", a, b, a >= b);

    printf("%d <= %d is %d \n", a, b, a <= b);
```

```
        return 0;

    }
```

## Output

5 == 5 is 1

5 > 5 is 0

5 < 5 is 0

5 != 5 is 0

5 >= 5 is 1

5 <= 5 is 1

**Logical operator:** Logical Operators are used when we want to test more than one condition and make decisions. here the operands can be constants, variables and expressions Logical operators are &&, ||, !

&&     Logical AND. True only if all operands are true

||       Logical OR. True only if either one operand is true

!        Logical NOT. True only if the operand is 0

## Example:

```
#include <stdio.h>

int main()

{

    int a = 5, b = 5, c = 10, result;

    result = (a == b) && (c > b);

    printf("(a == b) && (c > b) is %d \n", result);
```

```

result = (a == b) && (c < b);

printf("(a == b) && (c < b) is %d \n", result);

result = (a == b) || (c < b);

printf("(a == b) || (c < b) is %d \n", result);

result = (a != b) || (c < b);

printf("(a != b) || (c < b) is %d \n", result);

result = !(a != b);

printf("!(a == b) is %d \n", result);

result = !(a == b);

printf("!(a == b) is %d \n", result);

return 0;

}

```

## Output

(a == b) && (c > b) is 1

(a == b) && (c < b) is 0

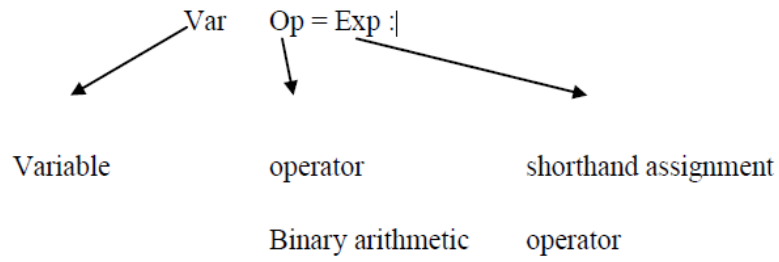
(a == b) || (c < b) is 1

(a != b) || (c < b) is 0

!(a != b) is 1

!(a == b) is 0

**Assignment Operator:** Used to assign the result of an expression to a variable. “=” is the assignment operator. In addition C has a set of „short hand“ assignment operators of the form



### Example:

// Working of assignment operators

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

```
int main()
```

```
{
```

```
    int a = 5, c;
```

```
    c = a;    // c is 5
```

```
    printf("c = %d\n", c);
```

```
    c += a;   // c is 10
```

```
    printf("c = %d\n", c);
```

```
    c -= a;   // c is 5
```

```
    printf("c = %d\n", c);
```

```
    c *= a;   // c is 25
```

```
    printf("c = %d\n", c);
```

```
    c /= a;   // c is 5
```

```
    printf("c = %d\n", c);
```

```
    c %= a;   // c = 0
```

```
    printf("c = %d\n", c);
```



```
    return 0;  
}
```

### Output

```
c = 5  
  
c = 10  
  
c = 5  
  
c = 25  
  
c = 5  
  
c = 0
```

### Shorthand operator

```
a += 1  
  
a -= 1  
  
a *= n+1  
  
a /= n+1  
  
a %= b
```

### Assignment operator

```
a = a+1  
  
a=a-1  
  
a = a* (n + 1)  
  
a = a/(n+1)  
  
a = a % b
```

**Increment and Decrement Operators:** C programming has two operators increment ++ and decrement -- to change the value of an operand (constant or variable) by 1.

Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.

++ and --

### Example:

```
++x or x ++    ==> x+=1 ==> x=x+1
```

-- x or x- -      == > x-=1 == > x=x-1

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

```
int main()
```

```
{
```

```
    int a = 10, b = 100;
```

```
    float c = 10.5, d = 100.5;
```

```
    printf("++a = %d \n", ++a);
```

```
    printf("--b = %d \n", --b);
```

```
    printf("++c = %f \n", ++c);
```

```
    printf("--d = %f \n", --d);
```

```
    return 0;
```

```
}
```

## Output

++a = 11

--b = 99

++c = 11.500000

++d = 99.500000

**Conditional operator:** is used to check a condition and Select a Value depending on the Value of the condition.

Variable = (condition)? Value 1 : Value 2:

If the Value of the condition is true then Value 1 is e valued assigned to the variable, otherwise Value2.

**Example:** `big = (a>b)? a:b;`

This exxp is equal to

```
if (a>b)
    big = a;
else
    big = b;
```

**Bitwise operator:** are used to perform operations at binary level i. e. bitwise. these operators are used for testing the bits, or Shifting them right or left . These operators are not applicable to float or double. Following are the Bitwise operators with their meanings.

Operators	Meaning of operators
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

**Example:** consider `a = 13` & `b = 6` as 8 bit short int (1byte)

<< Left Shift

`a = 13`      Binary 00001101

`b = 6`              00000110

Consider `a << 2` which Shifts two bits to left , that is 2 zeros are inserted at the right and two bits at the left are moved out.

00001101

Moved

00110100

Finally the result is 00110100 . Deci 52 (13x4)

**sizeof operator:** is used to find the on. of bytes occupied by a variable / data type in computer memory.

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

```
int main()
```

```
{
```

```
    int a;
```

```
    float b;
```

```
    double c;
```

```
    char d;
```

```
    printf("Size of int=%lu bytes\n",sizeof(a));
```

```
    printf("Size of float=%lu bytes\n",sizeof(b));
```

```
    printf("Size of double=%lu bytes\n",sizeof(c));
```

```
    printf("Size of char=%lu byte\n",sizeof(d));
```

```
    return 0;
```

```
}
```

### **Output**

Size of int = 4 bytes

Size of float = 4 bytes

Size of double = 8 bytes

Size of char = 1 byte

**comma operator:** can be used to link the related expressions together. A comma- linked: list of expressions are evaluated left to right and the value of right-most exp is the value of combined expression.

**Example:** value = ( x = 10, y = 5, x = y)

### **Precedence and Associativity Rules**

**Precedence:** precedence is nothing but priority that indicates which operator has to be evaluated first when there are more than one operator.

**Example:**

int x = 5 - 17\* 6;

**Associativity:** when there are more than one operator with same precedence [ priority ] then we consider associativity , which indicated the order in which the expression has to be evaluated. It may be either from Left to Right or Right to Left.

**Example:**

5 \* 4 + 10 / 2

= 20 + 5

=25

### **Operators Precedence & Associativity Table**

Operator	Meaning of operator	Associativity
()	Functional call	Left to right
[]	Array element reference	
->	Indirect member selection	
.	Direct member selection	
!	Logical negation	Right to left

~	Bitwise(1 's) complement	
+	Unary plus	
-	Unary minus	
++	Increment	
--	Decrement	
&	Dereference (Address)	
*	Pointer reference	
sizeof	Returns the size of an object	
(type)	Typecast (conversion)	
*	Multiply	Left to right
/	Divide	
%	Remainder	
+	Binary plus(Addition)	Left to right
-	Binary minus(subtraction)	
<<	Left shift	Left to right
>>	Right shift	
<	Less than	Left to right
<=	Less than or equal	
>	Greater than	
>=	Greater than or equal	
==	Equal to	Left to right
!=	Not equal to	
&	Bitwise AND	Left to right
^	Bitwise exclusive OR	Left to right
	Bitwise OR	Left to right
&&	Logical AND	Left to right

	Logical OR	Left to right
?:	Conditional Operator	Right to left
=	Simple assignment	Right to left
*=	Assign product	
/=	Assign quotient	
%=	Assign remainder	
+=	Assign sum	
-=	Assign difference	
&=	Assign bitwise AND	
^=	Assign bitwise XOR	
=	Assign bitwise OR	
<<=	Assign left shift	
>>=	Assign right shift	
,	Separator of expressions	Left to right

## Type Casting Types

Normally before an operation takes place both the operands must have the same type. C converts One or both the operands to the appropriate data types by “Type conversion”. This can be achieved in 3 ways.

**Implicit Type conversion:** In this the data type /Variable of lower type (which holds lower range of values or has lower precision ) is converted to a higher type (which holds higher range of values or has high precision). This type of conversion is also called “promotion”.

If a „char“ is converted into „int“ it is called as Internal promotion.

### Example:

```
int I;
```

```
char C;
```

C = „A“;

I = C;

Now the int Variable I holds the ASCII code of the char „A“

a) An arithmetic operation between an integer and integer yields an integer result.

b) Operation b/w a real yields a real

c) Operation b/w a real & an integer always yields a real result

**Example:**

$$5/2 = 2$$

$$2/5 = 0$$

$$5.0/2 = 2.5$$

$$2.0/5.0 = 0.4$$

$$5/2.0 = 2.5$$

$$2/5.0 = 0.4$$

$$5.0/2.0 = 2.5$$

$$2.0/5.0 = 0.4$$

**Assignment Type Conversion:** If the two Operands in an Assignment operation are of different data types the right side Operand is automatically converted to the data type of the left side.

**Example:**

Let „k“ is an int var & „a“ is a float var

int k;	yes	float a;		yes
k= 5/2	2	k=2/5	0	a = 5/2 2.0
k=5.0/2	2	k=2.0/5	0	a = 5.0/2 2.5
k=55.0/2	2	k=2/5.0	0	a = 5/2.0 2.5
k=5.0/2.0	2	k=2.0/5.0	0	a = 2/5 0.0
				a = 2.0/5 0.4



$$a = 2.0/0.5 \ 0.4$$

**Explicit Type Conversion:** When we want to convert a type forcibly in a way that is different from automatic type conversion, we need to go for explicit type conversion.

### Syntax

(type name) expression;

Type name is one of the standard data type. Expression may be a constant variable or an expression this process of conversion is called as casting a value.

### Example:

x = (int) 7.5

A = (int) 21.3/(int) 4.5

Y = (int) (a + b)

P = (double)sum/n