Unit - 1: Introduction to Programming

Introduction to components of a computer system: disks, primary and secondary memory, processor, operating system, types of computer languages, compilers, creating, compiling and executing a program etc., Introduction to Algorithms: steps to solve logical and numerical problems. Representation of Algorithm, Flowchart with examples.

Introduction to C Programming Language: History, Basic Structure of a C program, variables (with data types and space requirements), Syntax and Logical Errors in compilation, object and executable code, Operators, expressions and precedence, Expression evaluation, type conversion, Bitwise operations: Bitwise AND, OR, XOR and NOT operators.

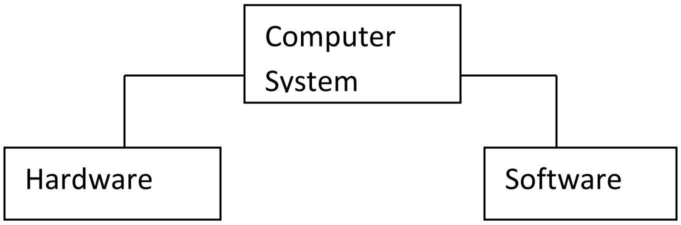
I/O: Simple input and output with scanf and printf.

UNIT - I

1. Introduction to Computers

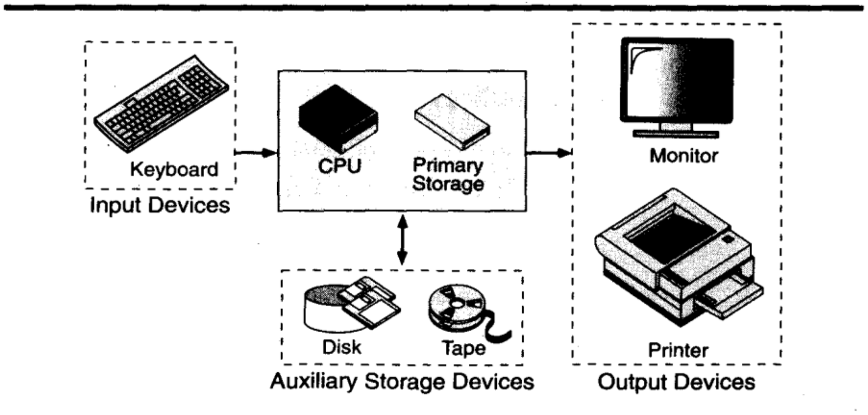
Computer Systems:

A computer is a system made of two major components: hardware and software. The computer hardware is the physical equipment. The software is the collection of programs (instructions) that allow the hardware to do its job.



Computer Hardware

The hardware component of the computer system consists of five parts: input devices, central processing unit (CPU) ,primary storage, output devices, and auxiliary storage devices.



Basic Hardware Components

Fig: **Computer Hardware**

1. **input device**
2. **central processing unit (CPU**)
3. **Primary storage**,
4. **secondary storage**,
5. **output device**

- **Input device** is usually a keyboard where programs and data are entered into the computers. Examples of other input devices include a mouse, a pen or stylus, a touch screen, or an audio input unit.
- **Central processing unit (CPU**) is responsible for executing instructions such as arithmetic calculations, comparisons among data, and movement of data inside the system. Today's computers may have one, two, or more CPUs .
- **Primary storage**, also known as **main memory,** is a place where the programs and data are stored temporarily during processing. The data in primary storage are erased when we turn off a personal computer or when we log off from a time-sharing system.
- **Auxiliary storage**, also known as **secondary storage**, is used for both input and output. It is the place where the programs and data are stored permanently. When we turn off the computer, or programs and data remain in the secondary storage, ready for the next time we need them.
- **Output device** is usually a monitor or a printer to show output. If the output is shown on the monitor, we say we have a **soft copy**. If it is printed on the printer, we say we have a hard copy.

Computer Software

Computer software is divided in to two broad categories**: system software and application software** .System software manages the computer resources .It provides the interface between the hardware and the users. Application software, on the other hand is directly responsible for helping users solve their problems.

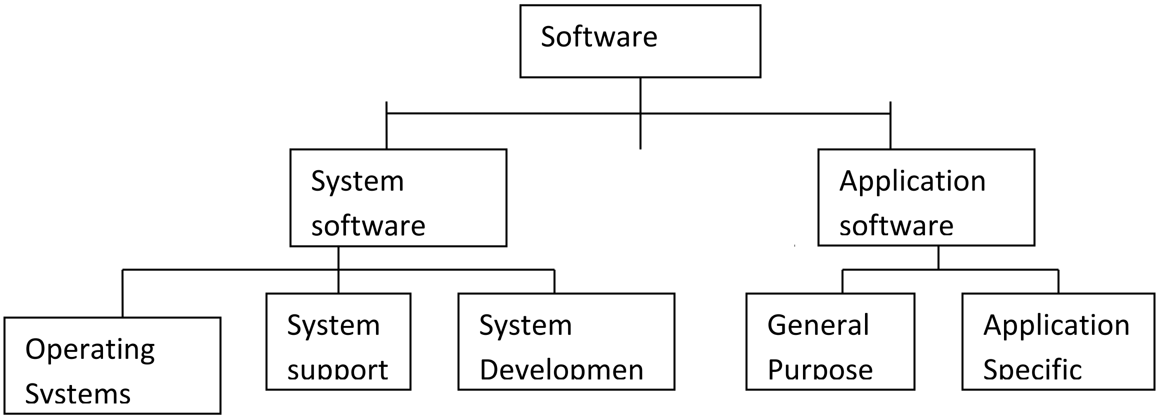


Fig: Types of software

System Software:

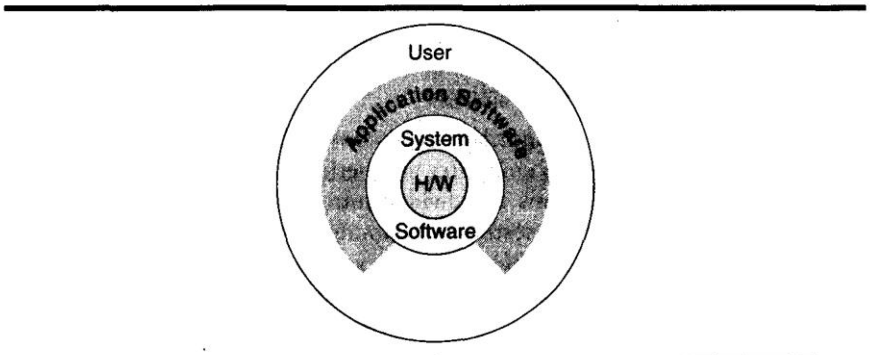
System software consists of programs that manage the hardware resources of a computer and perform required information processing tasks. These programs are divided into three classes: the operating system, system support, and system development.

- **Operating system** provides services such as a user interface, file and database access, and interfaces to communication systems such as Internet protocols. The primary purpose of this software is to keep the system operating in an efficient manner while allowing the users access to the system.
- **System support software** provides system utilities and other operating services. Examples of system utilities are sort programs and disk format programs. Operating services consists of programs that provide performance statistics for the operational staff and security monitors to protect the system and data.
- **System development software** includes the language translators that convert programs into machine language for execution, debugging tools to ensure that the programs are error free and computer –assisted software engineering (CASE) systems.

Application software

- **Application software** is broken in to two classes: general-purpose software and application –specific software.
- **General purpose software** is purchased from a software developer and can be used for more than one application. Examples of general purpose software include word processors ,database management systems ,and computer aided design systems. They are labeled general purpose because they can solve a variety of user computing problems.

Application –specific software can be used only for its intended purpose.



Relationship Between System and Application Software

Fig: Computer System

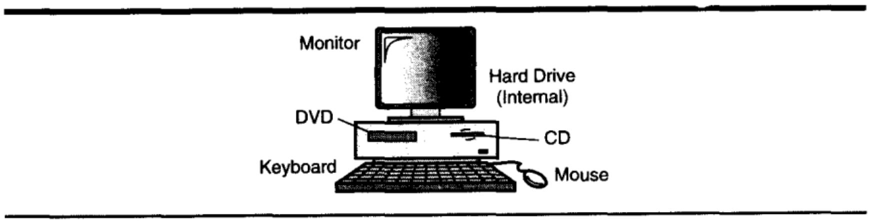
<u>Computing Environments:</u>

1. **Personal Computing Environment**
2. **Time-Sharing Environment**
3. **Client/Server Environment**
4. **Distributed Computing**

1. Personal Computing Environment

In 1971,Marcian E.Hoff, working for Intel, combined the basic elements of the central processing unit into the microprocessor. The first computer on a chip was the Intel 4004 and was the grandparent many times removed of Intel’s current system.

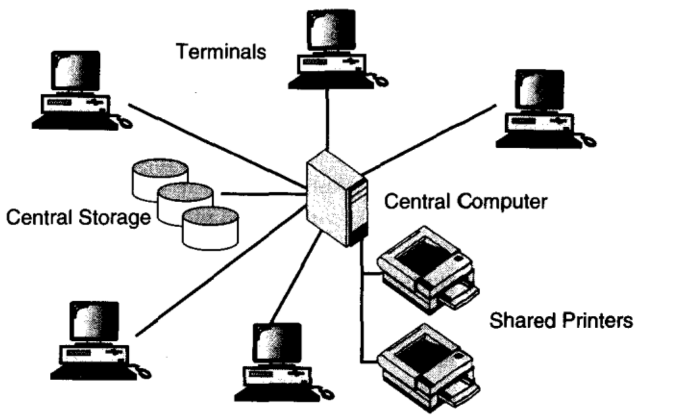
If we are using a personal computer, all of the computer hardware components are tied together in our personal computer(PC).



Personal Computing Environment

2. Time-Sharing Environment

Employees in large companies often work in what is known as a **time-sharing environment.** In the times-sharing environment, many users are connected to one or more computers. These computers may be minicomputers or central mainframes. The terminals they use are often nonprogrammable, although today we see more and more microcomputers being used to simulate terminals. Also, in the time-sharing environment, the output devices and auxiliary storage devices are shared by all of the users. A typical college lab in which a minicomputer is shared is shared by many students is shown in figure:

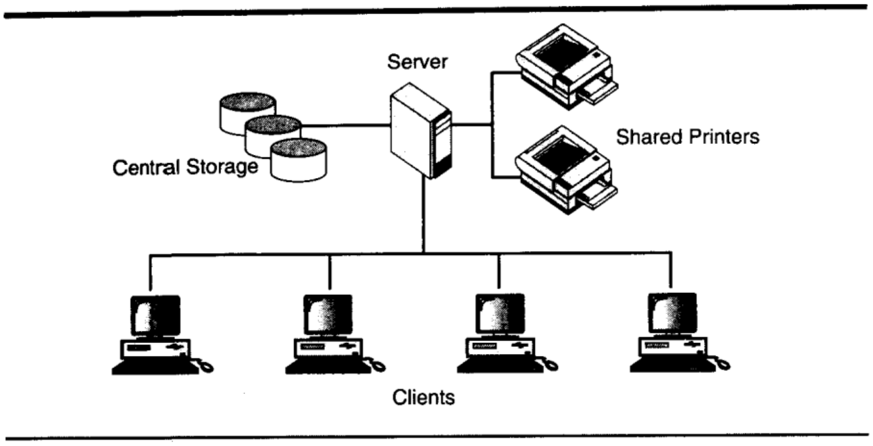


Time-sharing Environment

In the time-sharing environment, all computing must be done by the central computer. The central computer has many duties: It must control the shared resources; it must manage the shared data and printing and it must do the computing.

3. Client/Server Environment

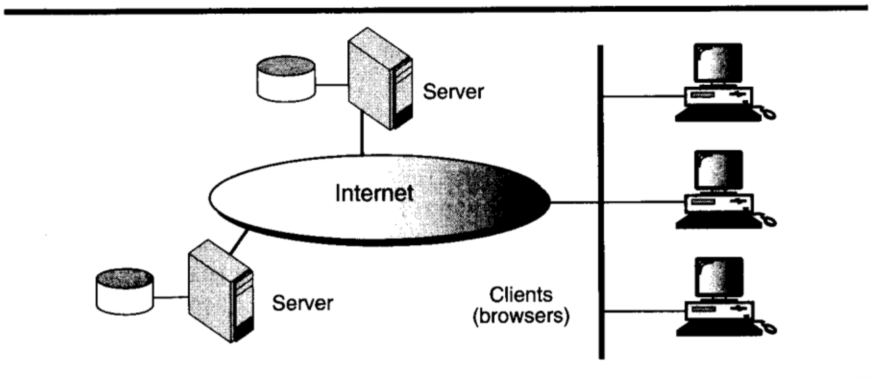
A client/server computing environment splits the computing function between a central computer and users' computers. The users are given personal computers or work stations so that some of the computation responsibility can be moved from the central computer and assigned to the workstations. In the client-server environment, the users' micro computers or workstations are called the **client**. The central computer, which may be a powerful microcomputer, minicomputer, or central mainframe system, is known as the **server.** Because the work is now shared between the users' computers and the central computer, response time and monitor display are faster and the users are more productive.



The Client/Server Environment

4. **Distributed Computing**

A **Distributed Computing** environment provides a seamless integration of computing functions between different servers and client's .The internet provides connectivity to different servers throughout the world. For example eBay uses several computers to provide its auction services. This environment provides a reliable, scalable, and highly available network.



Distributed Computing

Fig: Distributed Computing

Computer Languages:

To write a program for a computer, we must use a **computer language.** Over the years computer languages have evolved from machine languages to natural languages.

1. Machine level Languages
2. Symbolic Languages
3. High-Level Languages

1940’s	Machine level Languages
1950’s	Symbolic Languages
1960’s	High-Level Languages

1. Machine Languages

In 1940’s the earliest days of computers, the only programming languages available were machine languages. Each computer has its own machine language, which is made of streams of 0’s and 1’s.

Instructions in machine language must be in streams of 0’s and 1’s because the internal circuits of a computer are made of switches transistors and other electronic devices that can be in one of two states: off or on. The off state is represented by 0 , the on state is represented by 1.

The only language understood by computer hardware is machine language.

2. Symbolic Languages:

In early 1950’s Admiral Grace Hopper, A mathematician and naval officer developed the concept of a special computer program that would convert programs into machine language. The early programming languages simply mirror to the machine languages using symbols of mnemonics to represent the various machine language instructions because they used symbols, these languages were known as symbolic languages.

Computer does not understand symbolic language it must be translated to the machine language. A special program called assembler translates symbolic code into machine language. Because symbolic languages had to be assembled into machine language they soon became known as assembly languages.

Symbolic language uses symbols or mnemonics to represent the various ,machine language instructions.

3. High Level Languages:

Symbolic languages greatly improved programming effificiency; they still required programmers to concentrate on the hardware that they were using. Working with symbolic languages was also very tedious because each machine instruction has to be individually coded. The desire to improve programmer efficiency and to change the focus from the computer to the problem being solved led to the development of high-level language.

High level languages are portable to many different computers, allowing the programmer to concentrate on the application problem at hand rather than the intricacies of the computer. High-level languages are designed to relieve the programmer from the details of the assembly language. High level languages share one thing with symbolic languages, They must be converted into machine language. The process of converting them is known as compilation.

The first widely used high-level languages, FORTRAN (FORmula TRANslation)was created by John Backus and an IBM team in 1957;it is still widely used today in scientific and engineering applications. After FORTRAN was COBOL(Common Business-Oriented Language). Admiral Hopper was played a key role in the development of the COBOL Business language.

C is a high-level language used for system software and new application code.

Creating and Running Programs:

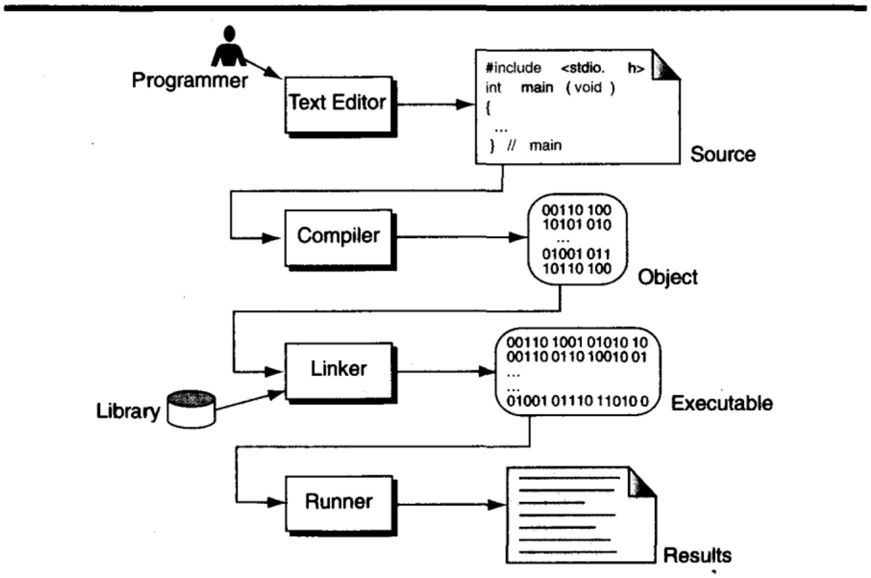
Computer hardware understands a program only if it is coded in its machine language. It is the job of the programmer to write and test the program .There are four steps in this process:

1. Writing and editing the program

2. Compiling the program

3. Linking the program with the required library modules

4. Executing the program.



```
#include   <stdio.   h>
int   main  ( void )
{
  ...
}  //   main
```

Building a C 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. The **preprocessor** and the **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.

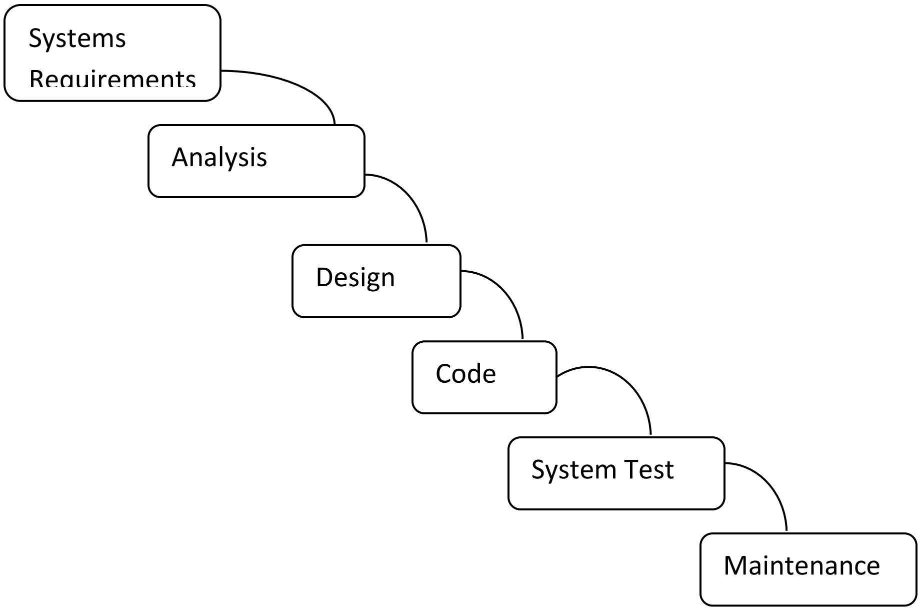
In a typical program execution, the reads data for processing ,either from the user or from a file. After the program processes the data, it prepares the output. at output can be to the user's monitor or to a file. When the program has finished its job, it tells the operating system ,which then removes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

System Development Method

A **software development process** is a structure imposed on the development of a software product. This critical process determines the overall quality and success of our program. If we carefully design each program using good structured development techniques, programs will be efficient, error-free, and easy to maintain.

System Development Life Cycle

Today's large-scale modern programming projects are built using a series of interrelates phases commonly referred to as the system development cycle. Although the exact number and names of the phases differ depending on the environment there is general agreement as to the steps that must be followed. One very popular development life cycle developed, this modal consists of between 5 and 7 phases.



The water fall modal starts with systems requirements in this phase the systems analyst defines requirements that specify what the proposed system is to accomplish. The requirements are usually stated in terms that the user understands. The analysis phase looks at different alternatives from a systems point of view while the design phase determined how the system will be built. In the design phase the functions of the individual programs that will make up the system are determined and the design of the files and / or the databases is completed. Finally in the 4th phase code, we write the programs. After the programs have been written and tested to the programmer's satisfaction, the project proceeds to the system test. All of the programs are tested together to make sure of the system works as a whole. The final phase maintenance keeps the system working once it has been put into production.

Although the implication of the water falls approach is that the phases flow in a continuous stream from the first to the last, this is not really the case. As each phase is developed, errors and omissions will often be found in the previous work. When this happens it is necessary to go back to the previous phase to rework it for consistency and to analyze the impact caused by the changes.

ALGORITHM /PSEUDOCODE:

Definitions of an algorithm.

1. Algorithm is a step-by-step procedure, to solve a given logical problem.
2. A method of representing the step – by – step logical procedure for solving a problem in natural language (like English, etc.) is called as algorithm.
3. 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.

Characteristics /properties of an Algorithm

1. **Finiteness** − Algorithms must terminate after a finite number of steps.
2. **Definiteness:** - Each step in the algorithm should be clear and unambiguous.
3. **Effectiveness** – All the operations used in the algorithm can be performed exactly in a fixed duration of time.
4. **Input** − An algorithm valid inputs.
5. **Output** − An algorithm should have well-defined outputs, and should match the desired output.
6. **Generality:** Algorithm suitable to solve the similar problems.

pseudocode.

- Algorithm may write any language but algorithm written in **English like** language is called pseudocode.
- pseudocode is similar to everyday English

Example: Algorithm/pseudo code to add two numbers

```
Step1:  start
Step2:  read a,b values
Step3:  add a and b and store in variable c
Step4:  display c

Step5:  stop
```

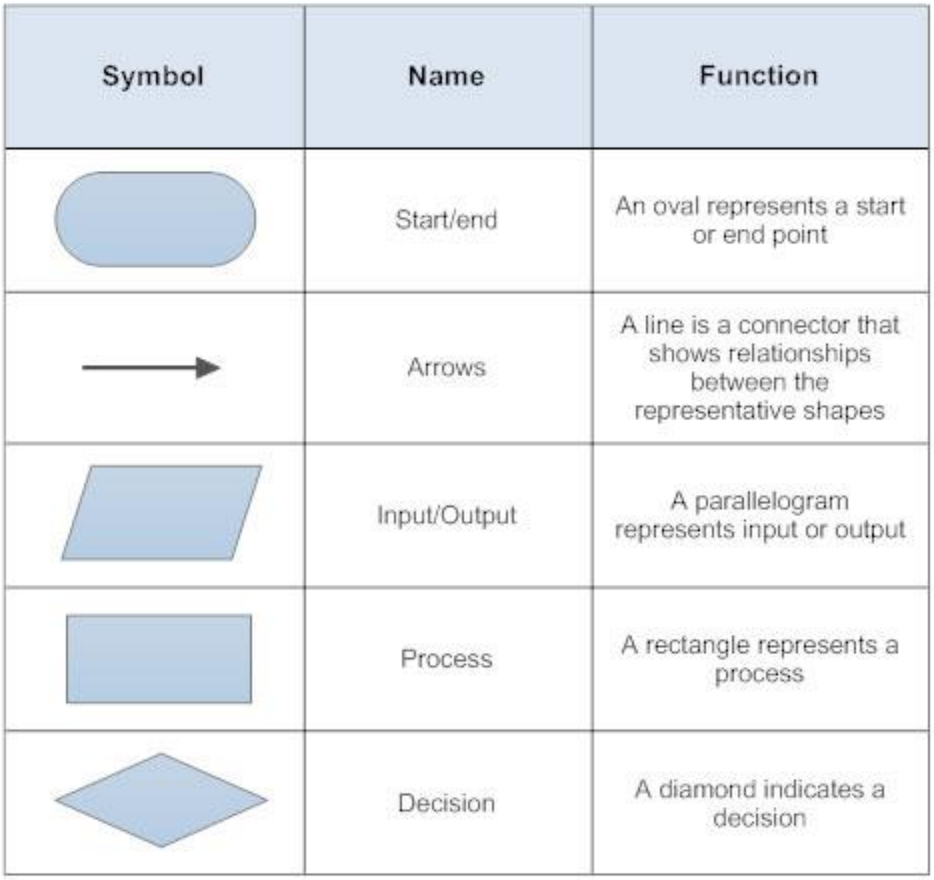
.

Example: Algorithm/pseudo code to add two numbers

```
Step 1: Start
Step 2:Read the two numbers in to a,b
Step 3: c=a+b
Step 4: write/print c
Step 5: Stop.
```

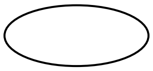
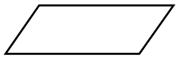
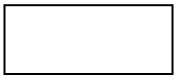
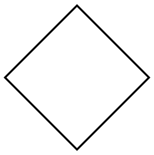
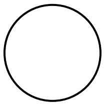
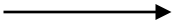
FLOW CHART :

A Flow chart is a **Graphical representation** of an Algorithm or a portion of an Algorithm. Flow charts are drawn using certain special purpose symbols such as Rectangles, Diamonds, Ovals and small circles. These symbols are connected by arrows called flow lines. (or) The diagrammatic representation of way to solve the given problem is called flow chart.



Symbol	Name	Function
⬭	Start/end	An oval represents a start or end point
→	Arrows	A line is a connector that shows relationships between the representative shapes
▱	Input/Output	A parallelogram represents input or output
▭	Process	A rectangle represents a process
◇	Decision	A diamond indicates a decision

The following are the most common symbols used in Drawing flowcharts:

Oval		Terminal	start/stop/begin/end.
Parallelogram		Input/output	Making data available For processing(input) or recording of the process information(output).
Rectangle		Process	Any processing to be Done .A process changes or moves data.An assignment operation.
Diamond		Decision	Decision or switching type of operations.
Circle		Connector	Used to connect Different parts of flowchart
Arrow		Flow	Joins two symbols and also represents flow of execution.

Test the Program

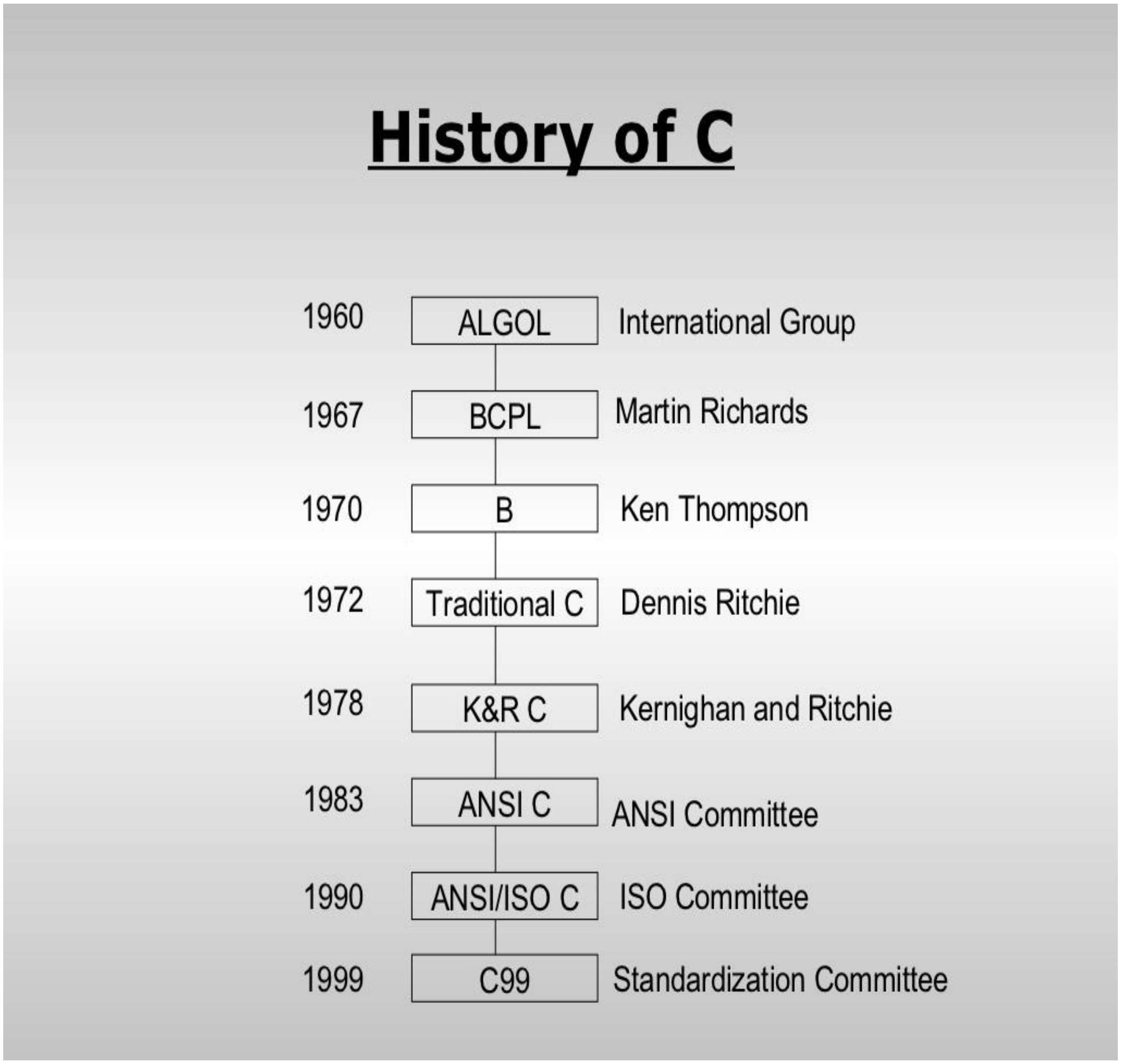
Program testing can be a very tedious and time- consuming part of program development. As the programmer we are responsible for completely testing our program. In large-development projects test engineers are responsible for testing to make sure all the programs work together.

There are 2 types of testing.

1. **Black box testing:** This is done by the system test engineer and the user. Black box testing is the programs are tested without knowing what is inside it, with out knowing how it works. Black box test plans are developed by looking only the requirements statement. The test engineer uses these requirements to develop test plans.
2. **White box testing:** This is the responsibility of the programmer. White box testing assumes that the tester knows everything about the program.

Part-2 Introduction of C Language (History)

- C Programming language was developed in 1972 by Dennis Ritchie at Bell laboratories of AT&T (American Telephone & Telegraph), located in the U.S.A.
- Dennis Ritchie is known as the founder/father of the c language.



History of C

1960	ALGOL	International Group
1967	BCPL	Martin Richards
1970	B	Ken Thompson
1972	Traditional C	Dennis Ritchie
1978	K&R C	Kernighan and Ritchie
1983	ANSI C	ANSI Committee
1990	ANSI/ISO C	ISO Committee
1999	C99	Standardization Committee

- The root of all modern languages is ALGOL(Introduces in 1960's
- ALGOL uses a structure programming

- ALGOL is popular in Europe
- in 1967 Martin Richards developed a language is called Basic Combined Programming Language (BCPL).
- Primarily BCPL is developed System Software.
- In 1970, Ken Thompson created a new language called B.
- B is created UNIX OS at Bell Laboratories.
- BCPL and B are "type less" languages.
- Using **BCPL** and **B** in 1972, Dennis Ritchie developed a language called C language.
- So, C language was invented for implementing UNIX operating system. Most of the UNIX components were rewritten in C.
- In 1978**, Dennis Ritchie and Brian Kernighan** published the first edition "The C Programming Language" and commonly known as K&R C.
- In 1983, the American National Standards Institute (ANSI) established a committee to provide a modern, comprehensive definition of C. The resulting definition, the ANSI standard, or "ANSI C", was completed late 1988.
- In 1990 **ISO** committee approved then "**ISO C"**
- C99 standard – Next revision was published in 1999 that introduced new futures like advanced data types and other changes.

Why was it named as C?

This programming language was named C because many of the ideas and principles were derived from previous programming language named B

Features/ Characteristics of C Language

C is the widely used language. It provides many features that are given below.

1. Simple and Efficient
2. Machine Independent or Portable
3. Modularity/Structured Programming Language
4. Existence of Libraries(Rich Library)
5. Dynamic Memory Management
6. Speed

7. Pointers
8. Recursion
9. Case Sensitive

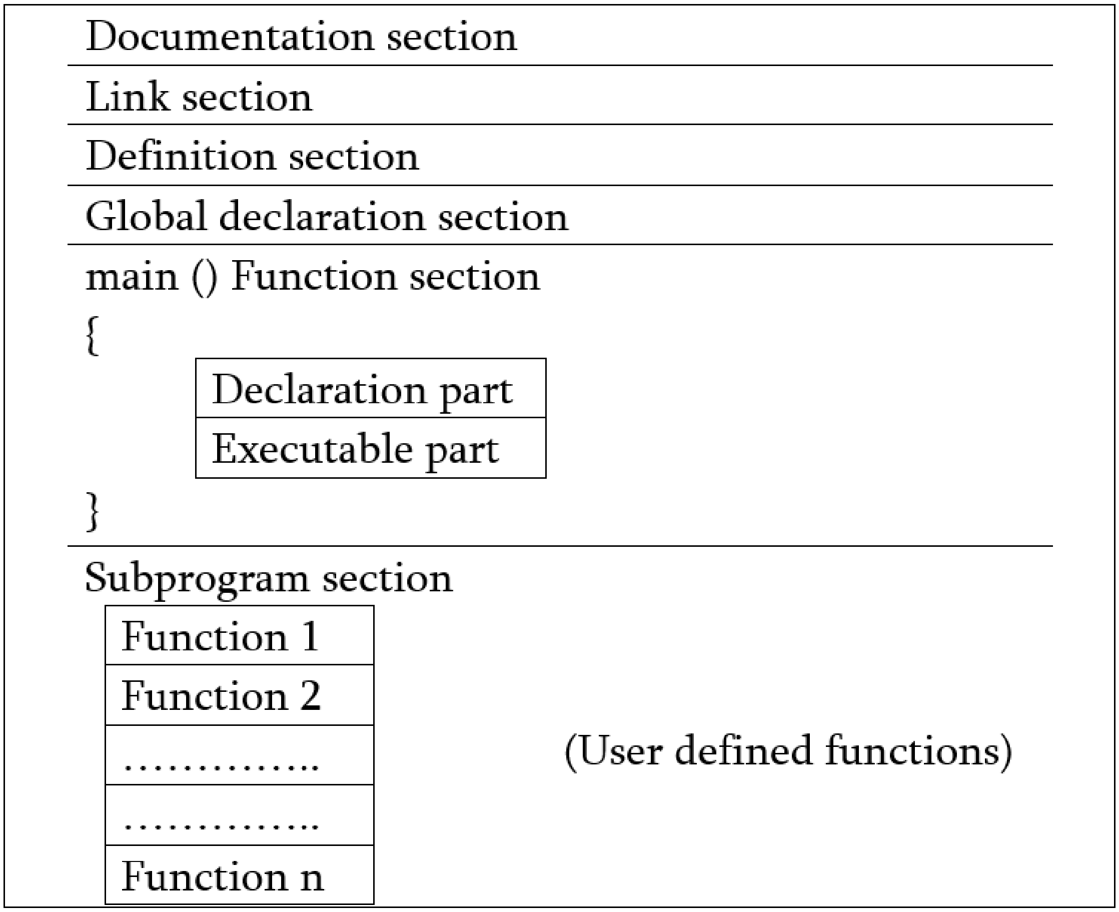
- C is structured language: Problem is solved using divide and conquers approach
- C implements Top-down approach.
- C is highly portable language: C language is machine independent. Source Code written using C can be compiled on any machine with little or no modification.
- C is a programmer's friendly language. Because the syntax of C can be learned very easily
- C is a robust language with a rich set of built-in functions and operators that can be used to write any complex program.
- Programs Written in C are efficient and fast.
- Using C language we can develop operating systems and Application software.
- There are 32 keywords.
- Another important feature of C program is its ability to extend itself.
- A C program is basically a collection of functions that are supported by C library. We can also create our own function and add it to C library.

Applications of C Programming

- Operating Systems

- Embedded Systems/Application Software
- GUI(GUI stands for Graphical User Interface)
  Ex: Adobe Photoshop
- New Programming Platforms
- Compiler Design
- Gaming and Animation

STRUCTURE OF C PROGRAME



Documentation section

Link section

Definition section

Global declaration section

main () Function section

{

Declaration part

Executable part

}

Subprogram section

Function 1

Function 2

…………..

…………..

Function n

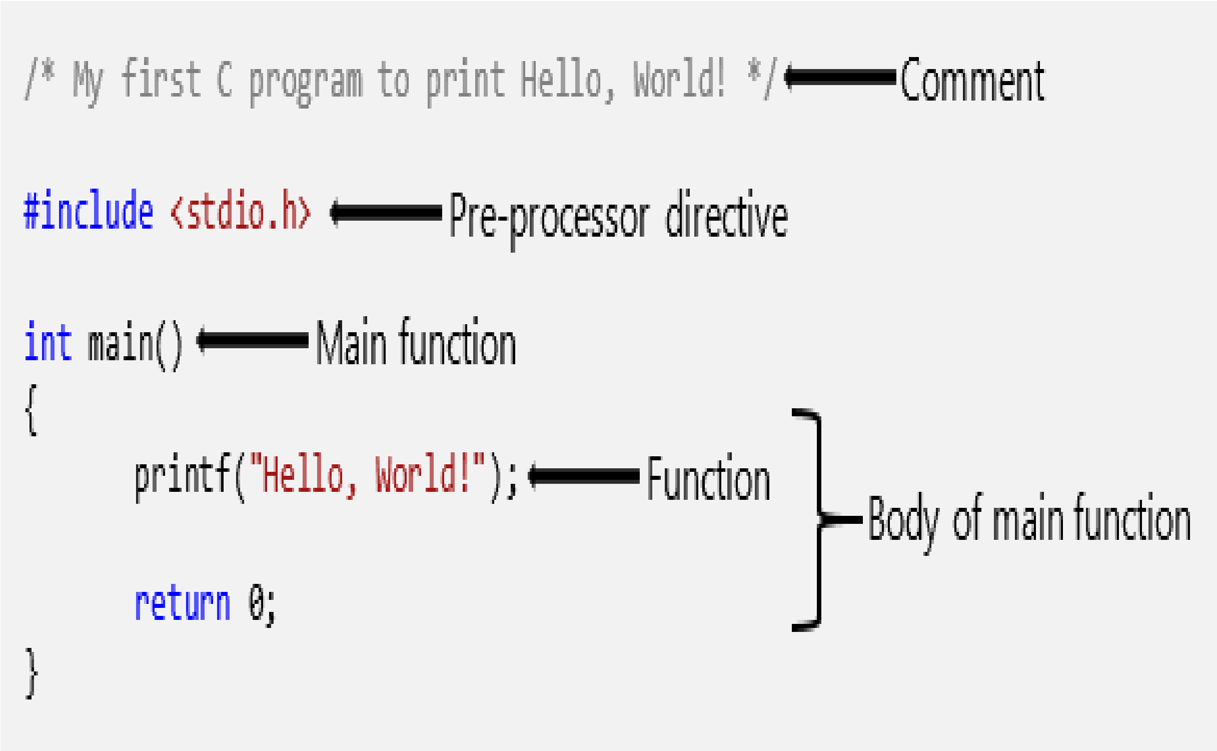
(User defined functions)

- **Documentations (Documentation Section)**
- **Preprocessor Statements (Link Section)**
- **Global Declarations (Definition Section)**
- **The main() function**
- **Local Declarations**
- **Program Statements & Expressions**
- **User Defined Functions**

```
/*Write c program to pint "Hello, World!" on the screen*/ (Documentation Section)
#include<stdio.h> (Preprocessor section)
void main() (main() function)
{
 printf(" Hello World \n"); //Local Declarations
```

```
                              // Program Statements
  }

void add()
{
----
----                          // Function
----
}
```



```
/* My first C program to print Hello, World! */

#include <stdio.h>

int main()
{
        printf("Hello, World!");

        return 0;
}
```

1. **Documentation Section(optional)**

- ➢ The documentation section gives the details associated with the program. He usually gives the name of the program.
- ➢ The details of the author and other details like the time of coding and description.
- ➢ **Comments**

We can add comments in our program to describe what we are doing in the program.

These comments are ignored by the compiler and are not executed.

1. **Single line comment:** start it by adding two forward slashses // followed by the comment.
2. **Multiline comment:** enclose it between /* .... */, just like in the program above.

Example:

// write a c program addition two numbers

// Author name: aaaaaa

OR

/* write a c program addition two numbers

Author name: aaaaaa */

2. **Pre-processor**

- #include is the first word of any C program. It is also known as a pre-processor.
- All Pre-processor Command start with # (pound) Symbol
- The task of a pre-processor to link the program with the header files required.
- So, when we say #include <stdio.h>, it is to inform the compiler to include the stdio.h header file to the program before executing it.

3. **Definition Section (optional)**

In this section, we define different constants. The keyword define is used in this part.

#define PI 3.14

4. **Global Declaration Section(optional)**

This part of the code is the part where the global variables are declared.

float area

int a=7;

5. **Main Function Section**

- Every C-programs need to have the main function.

- The execution star from main function

  Each main function contains 2 parts.

  1. Declaration part
  2. Execution part.

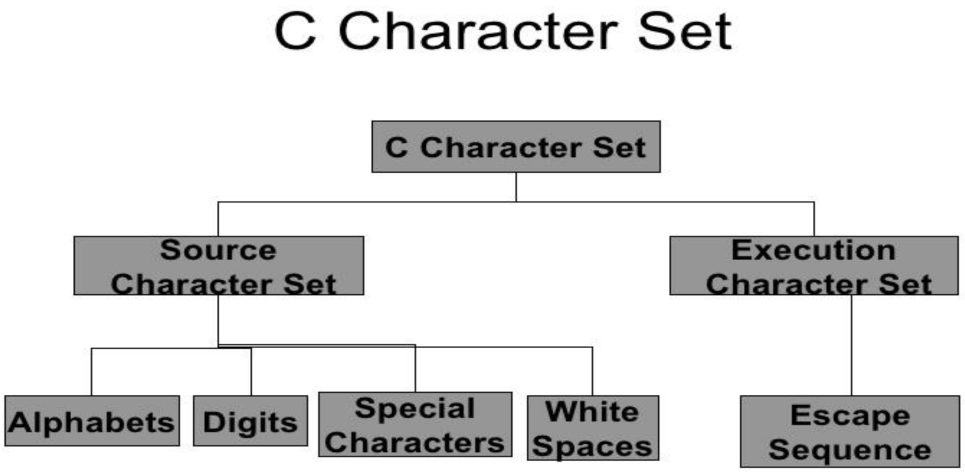
- The declaration part is the part where all the variables are declared.
- The execution part begins with the curly brackets and ends with the curly close bracket.
- Both the declaration and execution part are inside the curly braces.

6. Sub Program Section (optional)

- We write here user defined program

CHARACTER SET

- The character set is fundamental element of any language and they used to represent the information.
- The character set of c represent Alphabet, digits and any symbol to represent the information.



characters in C are grouped into the following two categories:

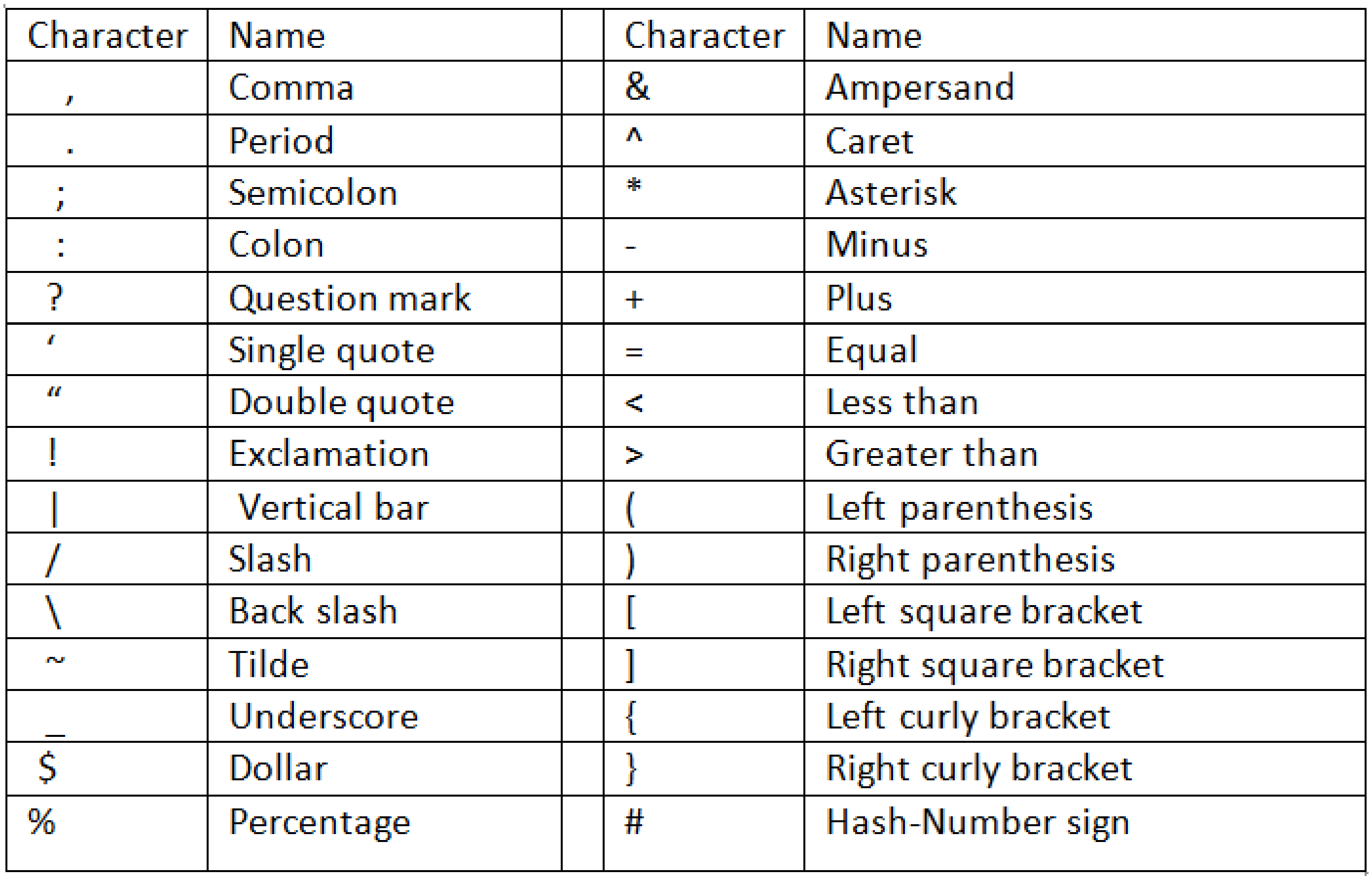
1. Source character set

1. Alphabets (a to z and A to Z)
2. Digits (0 to 9)
3. Special Characters
4. White Spaces

2. Execution character set

    1. Escape Sequence

SPECIAL CHARACTERS



Character	Name		Character	Name
,	Comma		&	Ampersand
.	Period		^	Caret
;	Semicolon		*	Asterisk
:	Colon		-	Minus
?	Question mark		+	Plus
‘	Single quote		=	Equal
“	Double quote		<	Less than
!	Exclamation		>	Greater than
|	Vertical bar		(	Left parenthesis
/	Slash		)	Right parenthesis
\	Back slash		[	Left square bracket
~	Tilde		]	Right square bracket
_	Underscore		{	Left curly bracket
$	Dollar		}	Right curly bracket
%	Percentage		#	Hash-Number sign

WHITESPACE CHARACTERS:

Blank space, Horizontal tab, vertical tab, newline, form feed

Escape Sequence in C

- Escape sequence used at the time of execution.
- This set of characters are also called as **non-graphic** character Because these characters are invisible and cannot be printed or Displayed directly (The meaning will printed).

- All the escape sequences in C are represented by 2 characters,
    1. compulsorily being backslash (\)
    2. The other any character present in the C character set.
- Escape sequence used in **printf() function.**

Some of Escape Sequence in C (lists the escape sequences)

Escape Sequence	Description
\t	Inserts a tab in the text at this point.
\b	Inserts a backspace in the text at this point.
\n	Inserts a newline in the text at this point.
\r	Inserts a carriage return in the text at this point. (Carriage return means to return to the beginning of the current line without advancing downward.) (cursor to start of line)
\f	Inserts a form feed in the text at this point. (Form feed means advance downward to the next "page". It was commonly used as page separators)
\'	Inserts a single quote character in the text at this point.

\"	Inserts a double quote character in the text at this point.
\\	Inserts a backslash character in the text at this point.

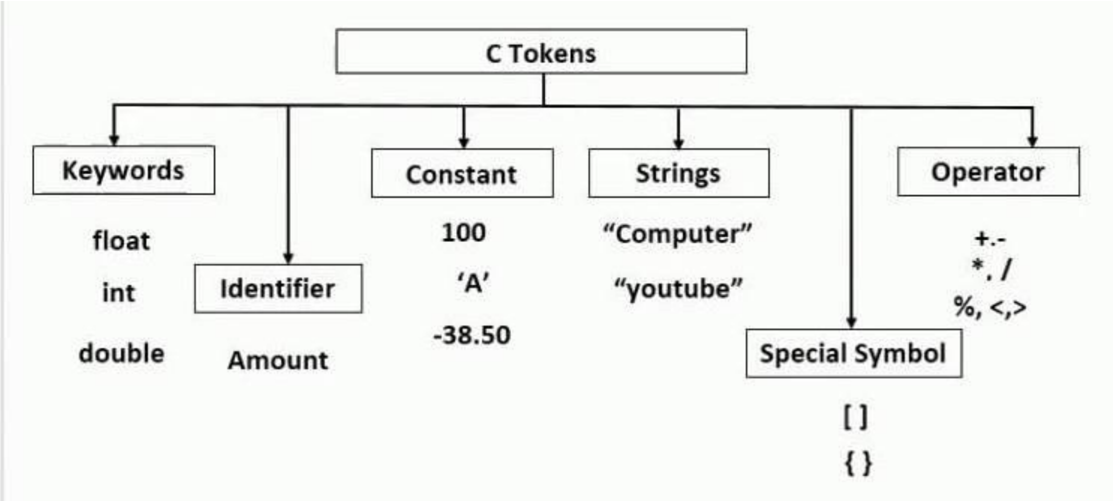
1. **\n** (New line) – We use it to shift the cursor control to the new line
2. **\t** (Horizontal tab) – We use it to shift the cursor to a couple of spaces to the right in the same line.
3. **\a** (Audible bell) – A beep is generated indicating the execution of the program to alert the user.
4. **\r** (Carriage Return) – We use it to position the cursor to the beginning of the current line.
5. \\ (Backslash) – We use it to display the backslash character.
6. \’ (Apostrophe or single quotation mark) – We use it to display the single-quotation mark.
7. \” (Double quotation mark)- We use it to display the double-quotation mark.
8. \0 (Null character) – We use it to represent the termination of the string.
9. \? (Question mark) – We use it to display the question mark. (?)
10. \nnn (Octal number)- We use it to represent an octal number.
11. \xhh (Hexadecimal number) – We use it to represent a hexadecimal number.
12. \v (Vertical tab)
13. \b (Backspace)
14. \e (Escape character)
15. \f (Form Feed page break)

C-TOKENS

- C programs, each word and punctuation is referred to as a token.
- C Tokens are the smallest building block or smallest unit of a C program.
- The compiler breaks a program into the smallest possible units and proceeds to the various stages of the compilation, which is called token.

C Supports Six Types of Tokens:

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



1. **Keywords**

   Definition:

   Keywords have fixed meanings, and the meaning cannot be changed while program execution

Some important points

- In 'C' every word can be either a keyword or an identifier.
- Keywords also called Reserved Words.
- They act as a building block of a 'C' program.
- There are total 32 keywords in 'C'. Keywords are written in lowercase letters.

Following table represents the keywords in 'C',

Auto	Double	int	struct
Break	Else	long	switch
Case	Enum	register	typedef
Char	Extern	return	union
Const	Short	float	unsigned
Continue	For	signed	void
Default	Goto	sizeof	volatile
Do	If	static	while

2. IDENTIFIER

Definition:

Identifiers refer to user-defined names of variables, functions and arrays in program.

Some important points

- Identifiers are the user-defined names consisting of letters and digits
- As the name says, identifiers are used to identify a particular element in a program.
- Both upper case and lower case letters are permitted but usually lower case letters are recommended.
- Each identifier must have a unique name.

- Underscore (_) character is also permitted in identifiers.
  It is usually as a link between two words in long Identifier.

  Example: stud_sum, emp_avg, matrix_add

Following rules must be followed for identifiers:

(Rules for Naming Identifiers)

- An identifier can only have alphanumeric characters (a-z , A-Z , 0-9) (I.e. letters & digits) and underscore (_) symbol.
- Identifier names must be unique
- The first character must be an alphabet or underscore.
- You cannot use a keyword as identifiers.
- Identifier cannot start with number or special Symbol.
- Only first thirty-one (31) characters are significant.
- Must not contain white spaces. Ex: sum std, emp sal
- Identifiers are case-sensitive.

Example of valid identifiers

total, sum, average, _m_, sum_1….ete

Example of invalid identifiers

- 2sum (starts with a numerical digit)
- int (reserved word)
- char (reserved word)
- m+n (special character, i.e., '+')

Differences between Keyword and Identifier

Keyword	Identifier
Keyword is a pre-defined word.	The identifier is a user-defined word
It must be written in a lowercase letter.	It can be written in both lowercase and uppercase letters.

Its meaning is pre-defined in the c compiler.	Its meaning is not defined in the c compiler.
It is a combination of alphabetical characters.	It is a combination of alphanumeric characters.
It does not contain the underscore character.	It can contain the underscore character.

CONSTANTS

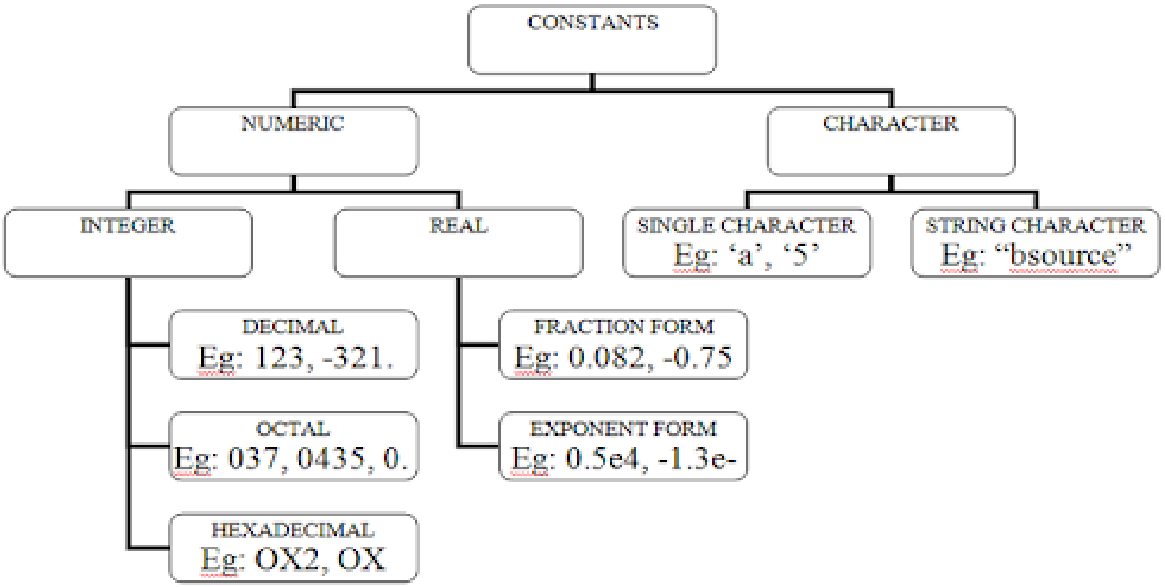
Definition:

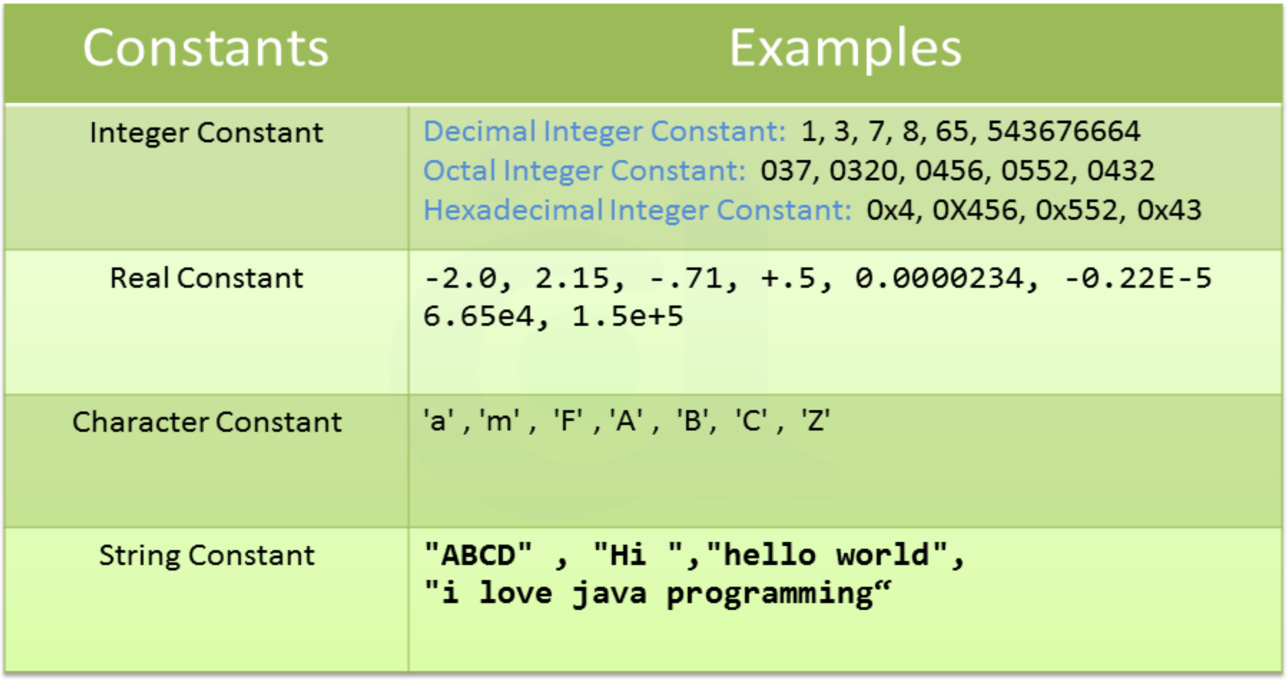
Constants in C are the fixed values that cannot change during the entire execution of the program.

- Constants are also called literals.

Constants are categorized into two basic types,

1. Numeric Constants
   - Integer Constants (Ex: 10 , 20 ,30 ….etc)
   - Real Constants(Ex: 10.2 , 20.666,7E-5)
2. Character Constants
   - Single Character Constants (Ex: ‘a’, ‘c’, ‘1’,’}’…..)
   - String Constants(Ex: “hello”, “1234”





Constants	Examples
Integer Constant	Decimal Integer Constant: 1, 3, 7, 8, 65, 543676664 Octal Integer Constant: 037, 0320, 0456, 0552, 0432 Hexadecimal Integer Constant: 0x4, 0X456, 0x552, 0x43
Real Constant	-2.0, 2.15, -.71, +.5, 0.0000234, -0.22E-5 6.65e4, 1.5e+5
Character Constant	'a' , 'm' , 'F' , 'A' , 'B', 'C' , 'Z'
String Constant	**"ABCD" , "Hi ","hello world",** **"i love java programming“**

<u>Integer Constants:</u>

It's referring to a sequence of digits. Integers are of three types viz:

1. Decimal Integer (0 to 9)
2. Octal Integer (o to 8)

3. Hexadecimal Integer (0 to 9 ,A,B,C,D,E,F)

<u>Examples:</u>

15, -265, 0, 99818, +25, 045, 0X6

Here X- Hexa_Decimal

<u>Real Constants (Floating-point):</u>

The real constant made up of sequence of numeric digits with presence of a decimal point

Ex: 2.0

0.0000234

-0.22E-5

Note: E-5 = 10-5

<u>Characters</u>

A **<u>Single character</u>** literal enclosed within a pair of single quotation marks (i.e ‘ and’)

For example: 'a', 'm', 'F', '2', '}' etc.

- The character '8' is not the same as 8.
- Character constants have a specific set of integer values known as ASCII values (American Standard Code for Information Interchange).

<u>STRING LITERALS</u>

A string literal is a sequence of characters enclosed in double-quote marks.

For example:

```
"good"              //string constant

""                  //null string constant

"      "            //string constant of six white space

"x"                 //string constant having a single character.
```

```
"Earth is round\n"        //prints string with a newline
```

At the end of string '\0' (NULL) is automatically placed

Note: String size= string length +1;

1 bit for Null character

Const Keyword

If you want to define a variable whose value cannot be changed, you can use the **const** keyword. This will create a constant.

For example,

```
const double PI = 3.14;
```

Notice, we have added keyword const.

Here, PI is a symbolic constant; its value cannot be changed.

```
const double PI = 3.14;
PI = 2.9; //Error
```

DATA TYPES IN C LANGUAGE

Link: https://www.externcode.com/data-types-in-c-language/

Definition:

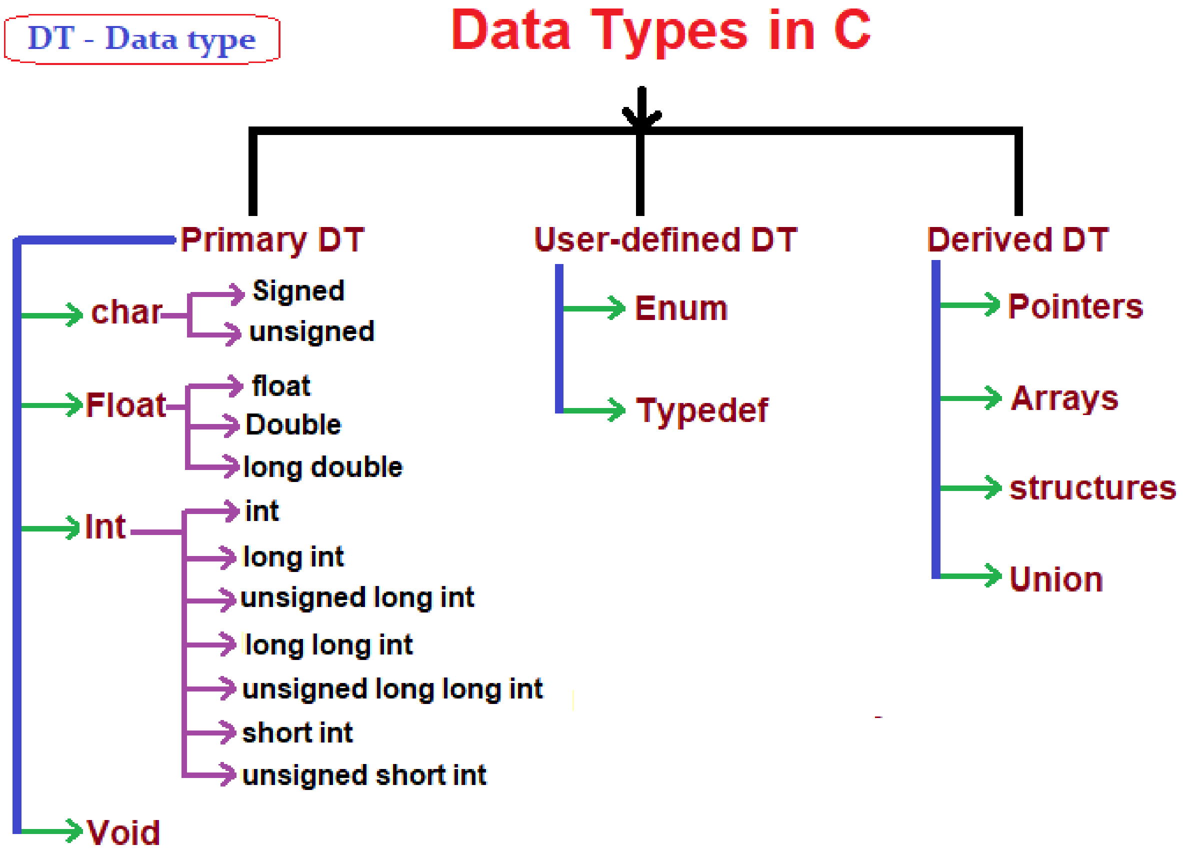
Data types specify what type of data we enter and how the enter data stored into our programs.

Data type is the type of the data that are going to access within the program.

- This determines the type and size of data associated with **variables.**

There are three classes of data types:

1. **Primary Data types.** Ex: char, int, float, double and void
2. **Derived Data types.** Ex: array, pointer, function, string….
3. **User Defined Data types**. using typedef keyword



1. Primary Data Types (in-built or Fundamental Data Type)

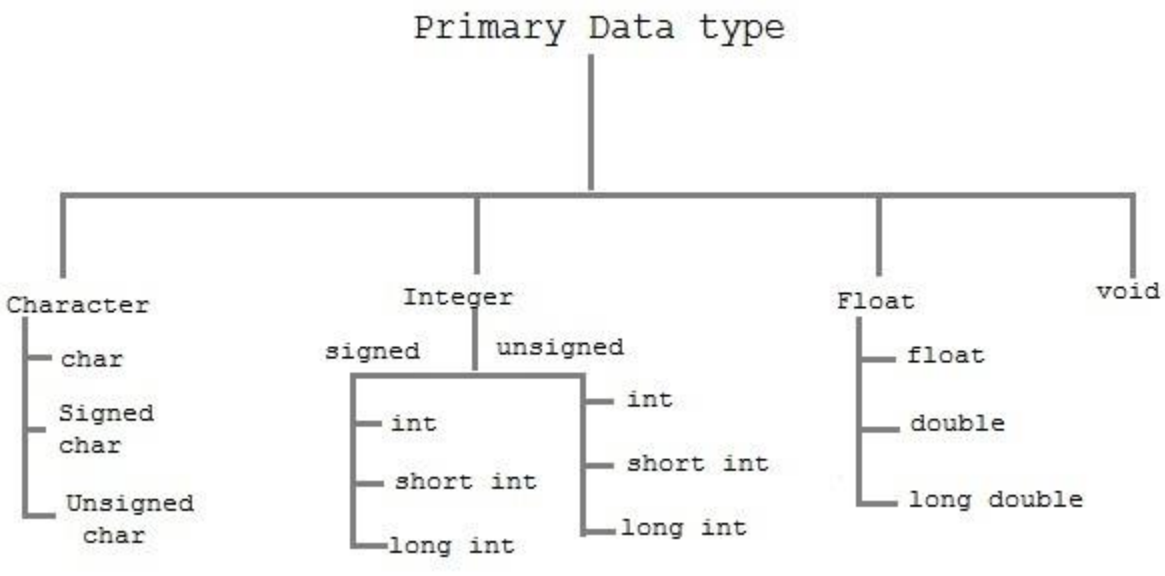
Primary data types are those which are already defined in programming languages also known in-built data types.

These data types are basic building blocks of any programming language

Primary data Types include:

1. Characters
2. Integers
3. Floating point
4. double Floating point
5. Void

Type	Size (bytes)	Format Specifier
Char	1	%c
Int	at least 2, usually 4	%d
Float	4	%f
Double	8	%lf
Void	-	-



Character Data Types:

A single character in C is of “char” type data. Characters stored in 8 bits or 1 byte of memory and the character can also be signed and unsigned.

Type	Size (bytes)	Range	Format Specifier
Char	1 Byte	-128 to 127 (-2^7 to +2^7-1)	%c
signed Char	1 Byte	-128 to 127 (-2^7 to +2^7-1)	%c
unsigned char	1 Byte	0 to 255 (0 to +2^8-1)	%c

INTEGER TYPES:

- It used to hold only integer values.
- An integer stores values in range limited from -2147483648 to 2147483647**(i.e. - 2^{32} to +2^{32} -1)**
- **A signed integer uses one bit reserve for sign bit**
- **The unsigned integer all bits used data.**

Size and range of Integer type on 16-bit machine:

Type	Size (bytes)	Range	Format Specifier
int or signed int	2	-32,768 to 32767 (-2^{15} to 2^{15}-1)	%d
unsigned int	2	0 to 65535 (0 to 2^{16}-1)	%u
short int or signed short int	1	-128 to 127 (-2^{7} to +2^{7}-1)	%hd
unsigned short int	1	0 to 255 (0 to +2^{8}-1)	%hu
long int or signed long int	4	-2,147,483,648 to 2,147,483,647 (-2^{31} to +2^{31}-1)	%ld
unsigned long int	4	0 to 4,294,967,295 (0 to +2^{32}-1)	%lu
long long int or signed long int	8	-(2^63) to (2^63)-1	%lld
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu

Note: In unsigned data type all the bits are used to represent value whereas signed the left most bit is reserved for the sign. Therefore unsigned size is more than (double) size of signed data types.

FLOATING POINT DATA TYPES:

Floating types are used to store real numbers.

Size and range of Integer type on 16-bit machine

Type	**Size(bytes)**	**Range**	Format Specifier
Float	4	3.4E-38 to 3.4E+38	%f
Double	8	1.7E-308 to 1.7E+308	%lf
long double	10	3.4E-4932 to 1.1E+4932	%Lf

Void DataTypes

void type means no value.

This is usually used to specify the type of functions which returns nothing.

Example:

void function(void);

DERIVED DATA TYPE:

- Using primitive data types we are derived another data type is derived data type.
- Derived data types to add some functionality to the basic data types as per program requirement.

Derived Data Types are formed by a grouping of two or more primary types.
Following are the Derived data types.

1. Pointers
2. Arrays
3. Union
4. Structures.

USER-DEFINED DATA TYPES:

- Data type that derived from an existing data type is called user-defined data type (UDT).
- We can create own (customized) data type using primary data type.

typedef keyword

Using typedef keyword we can create own data type

Syntax: typedef basicDT newDT;

Ex:

```
void main()
{
typedef  int hello;
hello a1 = 1, a2 = 2; // a1 and a2 are declared as ‘int’
printf("%d %d",a1,a2);
}
```

Output:

```
1 2
```

VARIABLES IN C

- A variable is a name of the memory location. It is used to store data. Its value can be changed, and it can be reused many times.
- In programming a variable is a container (storage area) to hold data.

  Let's see the syntax to declare a variable:

  Syntax:

  data_type variable name;
- Here data type any primary data type

  i.e char,int,float, double and void
- Variable name is user defined name.

The example of declaring the variable is given below:

```
int a;

float b;

char c;
```

DECLARING & INITIALIZING C VARIABLE:

- Variables should be declared in the C program before to use.
- Memory space is not allocated for a variable while declaration. It happens only on variable definition.
- Variable initialization means assigning a value to the variable, using Assignment operator (=).

Type	Syntax

Variable declaration
Or
Multiple variables
Declaration
	data_type variable_name;
	Example: int x, y, z; char flat, ch;
	
	data_type variable1_name, variable2_name, variable3_name;
	Example:int x,y,z;

Variable initialization	data_type variable_name = value; Example: int x = 50, y = 30; char flag = ‘x’, ch=’l’;

A variable name can consist of alphabets (both upper and lower case), numbers and the underscore ‘_’ character. However, the name must not start with a number.

NOTE: A declaration statement must end with semicolon (;)

Input and output statements in c

Input and output operations are performed using pre-defined Library functions.

These are classified into two types

1. Formatted IO
2. Unformatted IO

Formatted Input and output

These are two types

1. pritnf ( ) 2. scanf( )

printf() function

This function is used for displaying the output on the screen

Syntax:

1. printf(“arguments”);

   Ex: printf(“\n hello world”);

2. printf(“format string”, arg1, arg2, …..);

Ex:printf("%d%c",a,b);

<u>scanf() Fuction</u>

- scanf() is one of the commonly used function to take input from the user.
- The scanf() function reads formatted input from the standard input such as keyboards.

<u>Syntax:</u>

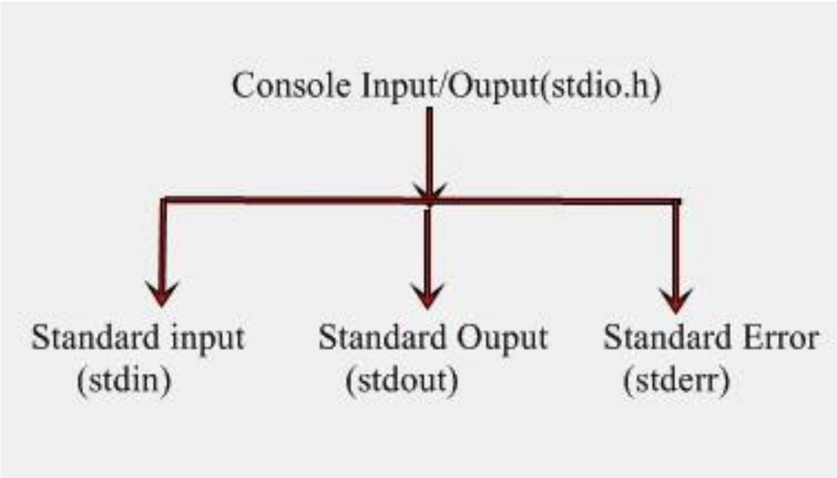
scanf ("format string", &arg1, &arg2, …..);

Ex: scanf("%d%d",&a,&b);

These I/O functions are categorized into three prcessing functions.

1. Console input/output function (deals with keyborad and monitor).
2. disk input/output function (deals with floppy or hard disk)
3. port input/output function (deals with serial or parallet port).

As all the input/output statements deals with the console, so these are also Console Input/Output functions. Console Input/Output function accesses the three major files before the execution of a C Program. These are as:



stdin: This file is used to receive the input (usually is keyborad file, but can also take input from the disk file).

stdout: This file is used to send or direct the output (usually is a monitor file, but can also send the output to a disk file or any other device).

stderr: This file is used to display or store error messages.

Pointer	Stream
Stdin	Standard input
Stdout	Standard output
Stderr	Standard error

The Standard Files

C programming treats all the devices as files. So devices such as the display are addressed in the same way as files and the following three files are automatically opened when a program executes to provide access to the keyboard and screen.

Standard File	File Pointer	Device
Standard input	stdin	Keyboard
Standard output	stdout	Screen
Standard error	stderr	Your screen

UNFORMATTED I/O FUNCTIONS

- Unformatted input and output functions are only work with character data type.
- Unformatted input and output functions do not require any format specifiers. Because they only work with character data type.

There are mainly six unformatted I/O functions discussed as follows:

1. getchar()
2. putchar()

3. gets()
4. puts()
5. getch()
6. putch()

Character **IO Functions**

1. getchar() Function

- The getchar() function reads a single character from the keyboard.
- The getchar() function reads one character at a time till the user presses the enter key.

syntax

v = getchar();

where v is the variable of character type. For example:

Example”

```
char n;
n = getchar();
```

getchar() Program

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
char c;
printf("Enter a character : ");
c = getchar();
printf("\nEntered character : %c ", c);
return 0;
}
```

OUTPUT

```
Enter a character: y
Entered character: y
```

Note:

Here, getchar() reads the input from the user and display back to the user.

putchar() Function

- This function is an output function.
- It is used to display a single character on the screen.

syntax

```
putchar(v);
```

where v is the variable of character type. For example:

```
char n;
putchar(n);
```

putchar() C Program

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
char c = 'K';
putchar(c);
return 0;
}
```

output

```
K
```

getch() Function

The getch() function reads the alphanumeric character from keyboard and immediately returns that character to program.

Syntax: getch();

getch() C Program

```
#include <stdio.h> //header file section
```

```
void main()
{
printf("\nHello, press any alphanumeric character to exit ");
getch();
}
```

OUTPUT:

```
Hello, press any alphanumeric character to exit
```

Note: The above program will run until you press one of many alphanumeric characters. The key pressed by you will not be displayed.

getche() Function

- All are same as getch() function execpt it is an echoed function.
- getche() function reads the alphanumeric character from the user input.
- It means when you type the character data from the keyboard it will visible on the screen.

getche()

syntax is as: getche();

C Program

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
printf("\nHello, press any alphanumeric character or symbol to exit \n ");
getche();
return 0;
}
```

OUTPUT:

```
Hello, press any alphanumeric character or symbol to exit
k
```

Note: The above program will run until you press one of many alphanumeric characters. The key pressed by you will be echoed.

ch = getch(); /*Typed character will not be displayed on the screen*/

ch = getche(); /*Typed character will be displayed on the screen*/

putchar() Function

putchar() function prints only one character at a time on an output device

syntax : putchar(v) where v is single character

putchar() C Program

```
//putchar.c
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
char c = 'K';
putchar(c);
return 0;
}
```

OUTPUT:

```
K
```

Note: Here, variable c is assigned to a character 'K'. The variable c is displayed by the putchar(). Use Single quotation mark ' ' for a character.

putch() Function

The putch() function prints any alphanumeric character.

syntax : putch(v) where v is single character

putch() C Program

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
char c;
printf("Press any key to continue\n ");
c = getch();
printf("input : ");
putch(c);
return 0;
}
```

OUTPUT:

```
Press any key to continue
input : d
```

Note:The getch() function will not echo a character. The putch() function displays the input you pressed.

String IO Functions

gets() Function

- The gets() function can read a full string even blank spaces presents in a string.
- But, the scanf() function leave a string after blank space space is detected.
- The gets() function is used to get any string from the user.

gets() C Program

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
char c[25];
printf("Enter a string : ");
gets(c);
printf("\n%s is awesome ",c);
return 0;
}
```

Output:

```
Enter a string: pps lab
pps lab is awesome
```

Note:The gets() function reads a string from through keyboard and stores it in character array c[25]. The printf() function displays a string on the console.

puts() Function

- The puts() function prints the charater array or string on the console.
- The puts() function is similar to printf() function, but we cannot print other than characters using puts() function.

puts() C Program

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
char c[25];
printf("Enter your Name : ");
gets(c);
puts(c);
return 0;
}
```

Output:

```
Enter your Name: john
```

John

Functions	Description
getch()	Reads a *single* character from the user at the console, ***without*** *echoing it*.
getche()	Reads a *single* character from the user at the console, *and echoing it*.
getchar()	Reads a *single* character from the user at the console, *and echoing it*, but needs an **Enter** key to be pressed at the end.
gets()	Reads a *single* string entered by the user at the console.
puts()	Displays a *single* string's value at the console.
putch()	Displays a *single* character value at the console.
putchar()	Displays a *single* character value at the console.

OPERATORS AND EXPRESSIONS IN C

OPERATORS:

- An operator is a symbol that tells the compiler to perform specific manipulation (mathematical or logical functions) on operands.
- The data on which the operators act are called operands.

Example: a+b Where a, b are operands

+ is a operator

An **expression** is a combination operands of and operators that reduce to single value.

Ex: a+b, x=a+b

C language is rich in built-in operators and provides the following types of operators −

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Bit wise operators
5. Conditional operators (ternary operators)
6. Increment/decrement operators
7. Assignment operators
8. Special operators

1. Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable A holds 10 and variable B holds 20 then –

Operator	Description	Example

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

2. Relational Operators

A relational operator checks the relationship between two operands.

- If the relation is true, it returns 1;
- if the relation is false, it returns value 0.

Relational operators are used in decision making and loops.

Operator	Meaning of Operator	Example	Return value
==	Equal to	5 == 3	0
>	Greater than	5 > 3	1
<	Less than	5 < 3	0

Operator	Meaning of Operator	Example	Return value
!=	Not equal to	5 != 3	1
>=	Greater than or equal to	5 >= 3	1
<=	Less than or equal to	5 <= 3	0

Logical Operators

- These operators are used to perform logical operations on the given expressions.
- An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false.
- There are 3 logical operators in C language.
    1. logical AND (&&)
    2. logical OR (||)
    3. logical NOT (!)

TRUTH TABLE

x	y	X&&Y	X||Y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

BIT-WISE OPERATORS

- Bitwise operators perform manipulations of data at bit level
- These operators are used to perform bit operations.
- Decimal values are converted into binary values which are the sequence of bits and bit wise operators work on these bits.

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR (Odd number of input is one)
<<	left shift
>>	right shift

X	Y	X & Y	X | Y	X ^ Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

CONDITIONAL OR TERNARY OPERATORS(?:) IN C:

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

Syntax:

expression 1 ? expression 2: expression 3;

Expression1 is true then expression2 is executed

Expression1 is false then expression3 is executed

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

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

INCREMENT/DECREMENT OPERATORS (++ AND - -)

Increment operators are used to increase the value of the variable by one and decrement operators are used to decrease the value of the variable by one in C programs.

Syntax:

Increment operator: ++var_name; (or) var_name++;

Decrement operator: – -var_name; (or) var_name – -;

Example:

Increment operator: ++ i ; i ++ ;

Decrement operator: – – i ; i – – ;

ASSIGNMENT OPERATORS IN C:

In C programs, values for the variables are assigned using assignment operators.

For example, if the value “10” is to be assigned for the variable “sum”, it can be assigned as “sum = 10;”

There are 2 categories of assignment operators in C language. They are,

1. Simple assignment operator (Example: =)
2. Compound assignment operators

(Example: +=, -=, *=, /=, %=, &=, ^=)

Operators	Example/Description
=	sum = 10; 10 is assigned to variable sum
+=	sum += 10; This is same as sum = sum + 10
-=	sum -= 10; This is same as sum = sum – 10
*=	sum *= 10; This is same as sum = sum * 10
/=	sum /= 10; This is same as sum = sum / 10
%=	sum %= 10; This is same as sum = sum % 10

&=	sum&=10; This is same as sum = sum & 10
^=	sum ^= 10; This is same as sum = sum ^ 10

<u>SPECIAL OPERATORS IN C:</u>

Operators	Description
&	This is used to get the address of the variable. Example : &a will give address of a.
*	This is used as pointer to a variable. Example : * a where, * is pointer to the variable a.
Sizeof ()	This gives the size of the variable. Example : size of (char) will give us 1.

Types of Operators	Description

Arithmetic_operators	These are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus
Assignment_operators	These are used to assign the values for the variables in C programs.
Relational operators	These operators are used to compare the value of two variables.
Logical operators	These operators are used to perform logical operations on the given two variables.
Bit wise operators	These operators are used to perform bit operations on given two variables.
Conditional (ternary) operators	Conditional operators return one value if condition is true and returns another value is condition is false.
Increment/decrement operators	These operators are used to either increase or decrease the value of the variable by one.
Special operators	&, *, sizeof( ) and ternary operators.

Expression evaluation in C

- In c language expression evaluation is mainly depends on priority and associativity.

- An expression is a sequence of operands and operators that reduces to a single value.

For example, the expression, 10+15 reduces to the value of 25.

An expression is a combination of variables constants and operators written according to the syntax of C language.

- Every expression results in some value of a certain type that can be assigned to a variable.

Priority

This represents the evaluation of expression starts from "what" operator.

Operators Precedence And Associativity

Precedence of operators

- If more than one operators are involved in an expression, C language has a predefined rule of priority for the operators.
- This rule of priority of operators is called operator precedence.

Associativity of operators

- It represents which operator should be evaluated first if an expression is containing more than one operator with same priority.

- If two operators of same precedence (priority) is present in an expression, Associativity of operators indicate the order in which they execute.

C operators with precedence and associativity

Operator	Priority	Associativity
{}, ( ), [ ]	1	Left to right
++, --, !	2	Right to left
*, /, %	3	Left to right
+, -	4	Left to right
<, <=, >, >=, ==, !=	5	Left to right
&&	6	Left to right
||	7	Left to right
?:	8	Right to left
=, +=, -=, *=, /=, %=	9	Right to left

Example1:

$$10 - \underbrace{3 \% 8} + 6 / 4$$

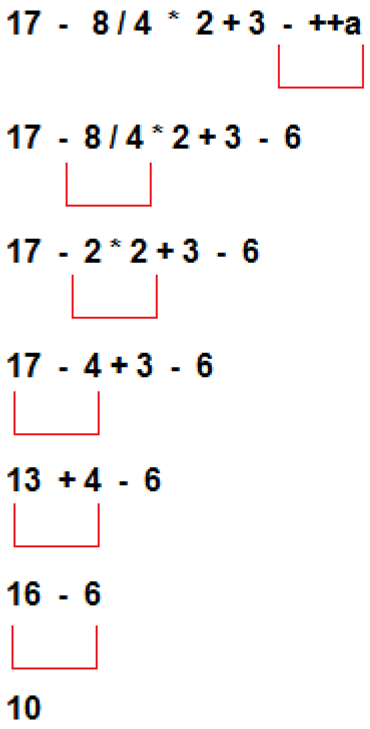
$$10 - 3 + \underbrace{6 / 4}$$

$$\underbrace{10 - 3} + 1$$

$$\underbrace{7 + 1}$$

$$8$$

Example2:



Expression evolution

Examples:

Ex1: 4*3/2+ (3+4/2)*2 **=4*3/2+ (3+4/2)*2**

=4*3/2+ (3+2)*2

= 4*3/2+ (5)*2

=4*3/2+10

=12/2+10

=6+10

=16

Ex2: 9-((12/3) +3*2)-1 =9-((12/3) +3*2)-1

=9-((4)+3*2)-1

=9-(4+6)-1

=9-10-1

= -1-1

= -2

Type Conversion/Type casting

Converting from one data type in to another data type is called type casting.

Two types of type conversions are

1. Implicit type conversion
2. Explicit type conversion.

Implicit type conversion

- Implicit type of conversion is done by the compiler automatically.
- When the operands are of different types, the lower type is converted to higher type automatically and the resultant value is of higher type

Example:

```
 int i=3;
float f;
 f=i;(i is lower data type (int) is automatically converted into float data type)
```

Explicit conversion (Type casting)

- When the higher type of data is converted into lower data type by forcefully then some value may get truncated
- A user can convert data explicitly by type casting.

Syntax:

The general form is

(data type) expression

Example: float a=6.5;

```
float b=7.5
int result= (int)a + (int)b;
```

I.e result would be equal to 13 instead of 14.

Example: float f;

```
int a = 20, b = 3;

f = (float)a/b;
```

Example: Explicit conversion

```
#include<stdio.h>
void  main()
{
   int a = 25, b = 13;
   float result;

   result = a/b; // display only 2 digits after decimal point

   printf("(Without typecasting) 25/13 = %.2f\n", result );

 result = (float)a/b; // display only 2 digits after decimal point

   printf("(With typecasting) 25/13 = %.2f\n", result );
 // signal to operating system everything works fine

}
```

Output:

```
(Without typecasting) 25/13 = 1.00
(With typecasting) 25/13 = 1.92
```

ERRORS IN C

- Error is an illegal operation performed by the user which results in abnormal working of the program.
- Programming errors often remain undetected until the program is compiled or executed. Some of the errors inhibit the program from getting compiled or executed. Thus errors should be removed before compiling and executing.

The most common errors can be broadly classified as follows.

1. **Syntax errors(compile time error):**
2. **Logical Errors:**
3. **Semantic errors:**
4. **Run-time Errors:**

1. Syntax errors(compile time error):

- Errors that occur when you violate the rules of writing C syntax are known as syntax errors.
- This compiler error indicates something that must be fixed before the code can be compiled.
- If any error is generated at the time of compilation is known as compile time error

Most frequent syntax errors are:

- Missing Parenthesis (})
- Printing the value of variable without declaring it
- Missing semicolon …..etc

Example 1: Missing Semicolon

```
int a=5 //semicolon is missing
```

Example 2: Errors in expression

```
x=(3+5; //Missing closing parenthesis’)’
y=3+*5; //Missing argument between ‘+’ and ‘*’
```

Example 3:

```
#include<stdio.h>
void main()
{
   int x = 10;
   int y = 15;
   printf("%d", (x, y)) // semicolon missed
}
error:
error: expected ';' before '}' token
```

2. **Logical Errors:**

- Logical error cannot detect by compiler. When you should you divide but you are multiplying.

- 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 appear to be error free are called logical errors.
- These are one of the most common errors done by beginners of programming.

Example 1:

```
int i=1;
while(i<=1) //Infinite loop does not terminate
printf("\n %d",i);
```

Example 2:

```
#include<stdio.h>
void main()
{
  int i;
  for(i = 0; i<5; i++);  //for loop ending with semicolon
{
    printf("Hello World");
  }
}
```

Error:

Here we want the line will be printed five times. But only one time it will be printed for the block of code.

3. **Semantic errors:**

- This error occurs when the statements written in the program are not meaningful to the compiler.
- If some expression is given at the left side of assignment operator, this may generate semantic error.

Example 1:

```
#include<stdio.h>
void main()
{
  int x, y, z;
  x = 10;
  y = 20;
  x + y = z; // semantic error
}
```

<u>Output</u>

```
[Error] lvalue required as left operand of assignment
```

<u>Example 2:</u> Type incompatibility

```
int a=”hello”; // the datatype int and string are not compatible
```

<u>Example 3:</u> Errors in expression

```
string a=”hello”;

int b=5-s; // the operator - does not support arguments of type string
```

4. **<u>Run-time Errors:</u>**

- Errors which occur during program execution (run-time) after successful compilation are called run-time errors.
- One of the most common run-time errors is division by zero also known as Division error.
- These types of error are hard to find as the compiler doesn’t point to the line at which the error occurs.

<u>For more understanding run the example given below.</u>

<u>/ C program to illustrate</u>

```
// run-time error
#include<stdio.h>
```

```
void main()
{
    int n = 9, div = 0;

    // wrong logic
    // number is divided by 0,
    // so this program abnormally terminates
    div = n/0;
    printf("resut = %d", div);
}
```

Error:

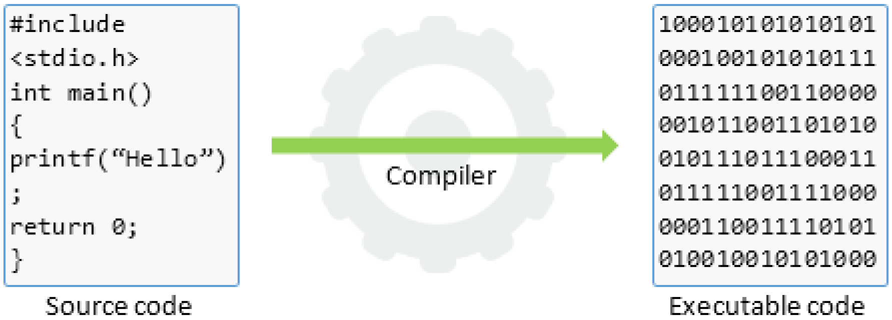
```
warning: division by zero [-Wdiv-by-zero]
     div = n/0;
```

Warning

Warning is also an abnormal condition but whenever it occurred execution of program will never be stopped.

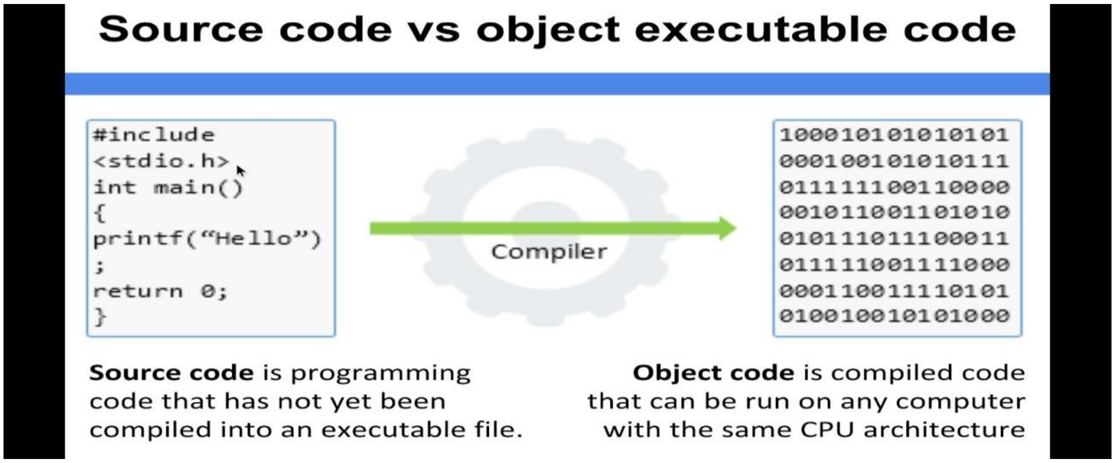
Note: In C language warning can be neglected but error can not be neglected.

Object and executable code in c



```
#include
<stdio.h>
int main()
{
printf(“Hello”)
;
return 0;
}
```

```
100010101010101
000100101010111
011111100110000
001011001101010
010111011100011
011111001111000
000110011110101
010010010101000
```



Source code vs object executable code

```
#include
<stdio.h>
int main()
{
printf(“Hello”)
;
return 0;
}
```

```
100010101010101
000100101010111
011111100110000
001011001101010
010111011100011
011111001111000
000110011110101
010010010101000
```

Source code is programming code that has not yet been compiled into an executable file.

Object code is compiled code that can be run on any computer with the same CPU architecture

<u>Source code</u> is the C program that you write in your editor and save with a ‘ .C ‘ extension.

<u>Object code</u> is the output of a compiler after it processes the source code. The object code is usually a machine code, also called a machine language,

<u>An executable file</u> is a complete program that can be run directly by an operating system (in conjunction with shared libraries and system calls)

Executable (also called the Binary) is the output of a linker after it processes the object code.

<u>Source Code vs Object Code – Difference between Source Code and Object Code</u>

Source Code	Object Code
Created by the programmer.	Created by the Compiler.
Text rich document.	Binary digits make up the Object Code.

Human Readable.	Machine Readable.
Serves as input to the compiler.	It is the output of the compiler.
Instructions written using English words and according to syntax of the language.	Instructions encoded in Binary digits.