UNIT–I:

Introduction to Computers: Computer hardware and software, System Software, Programming languages, Program Developing Steps, Algorithms, Flow charts.
Introduction to C: History of C**,** Structure of C Program**,** Keywords, Identifiers , Data Types, Constants, Variables, Operators, Expressions, Precedence and Order of Evaluation Type Conversion and type casting.

---

1. **List and explain the functions of various parts of computer hardware and software.**
   Ans: Computer System:

   A computer is a system made up of two major components:
   I. Computer Hardware.
   II. Computer Software.

   The following figure shows a computer system.

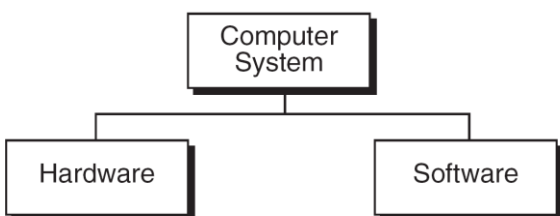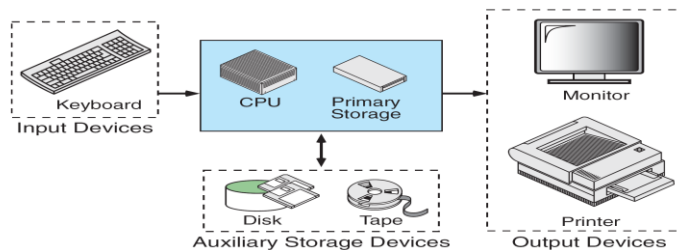
          Fig :   A Computer System
   I. Computer Hardware:
                  The computer hardware is the physical equipment. The   hardware component of computer system consists of   5 parts
                  A.   Input Devices.
                  B.   Central Processing Unit (CPU).
                  C.   Primary Storage.
                  D.   Output Devices.
                  E.   Auxiliary Storage Devices.
         The   following   figure   shows   the   *hardware*   components.

A. Input Devices: The   input   device   is   usually   a   keyboard   where   programs and data are   entered   into   the   computer. Other Input Devices : a touch screen , a mouse , a pen , an audio input unit.

B. Central Processing Unit (CPU):   It   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 or more CPU's

C. Primary Storage: It   is   also   known   as   main   memory. It   is   a   place where the programs   and data is   stored   temporarily   during   processing. The Data in primary storage   is   erased   when   we   turn   off   a   personal computer   or   when   we   log   off from a   time-sharing   computer (volatile).

D. Output Devices: The   output device   is usually   a   monitor or   a   printer to show output. If the output is   shown on the monitor, it is a soft copy and if the output is printed on the printer, it is a hard copy.

E. Auxiliary Storage Devices (secondary storage devices): It is used for both input and output. It is also known as secondary storage. It   is   a   place where   the   programs   and   data are   stored   permanently. When   we   turn   off   the   computer   the   programs   and   data remain in the secondary storage, ready for the next time when we need them.

II. Computer   Software: Software   is   the   collection   of   Programs (instructions)   that allow the hardware to do its job.
There are two types of Computer Software.
- A.   System   Software
- B.   Application   Software
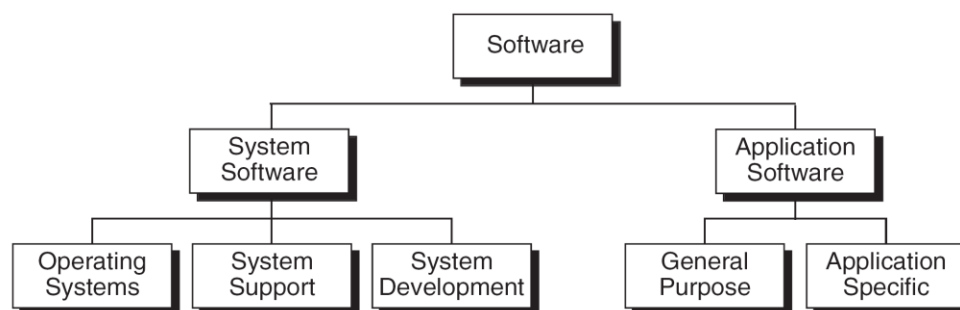
The following figure shows the Computer Software.

Fig: Computer Software

A. 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.
i.   Operating System Software.

ii.   System   Support   Software.
iii.   System   Development   Software.

 i.   Operating System Software: It provides services such as a user interface, files and data   base 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.

ii.   System Support Software: System Support Software provides system utilities and other operating services. Examples   of   system   utilities are sort   programs   and   disk   format programs. Operating services consist of programs that provide   performance   statistics for the operational staff and security monitors to protect the system and data.

iii. System Development Software: It includes language translators that convert programs in to machine language for execution , debugging tools to ensure that programs are error - free and computer -assisted software engineering (CASE) systems.

 B. Application Software: It is directly responsible for helping users to solve their problems. Application software is broken into two classes.
        i.   General - Purpose   Software
        ii.   Application - Specific   Software

 i. General Purpose Software: It is purchased from a software developer and can be used for more than one   application. Examples: word   processors, database management systems, computer – aided design systems. They   are   labeled   general   purpose   because they can   solve a variety of user computing problems.

 ii. Application Specific Software: It can be used only for its intended purpose. Example: A general ledger system used by accountants.
They can be used only for the task for which they were designed. They cannot be used for other generalized tasks.
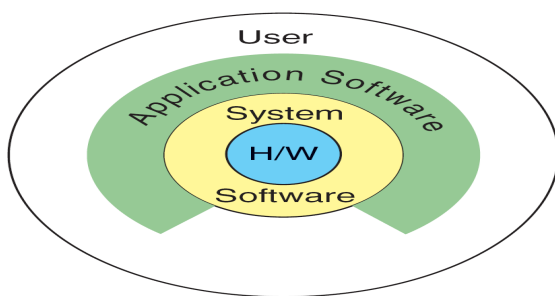Relationship between system and application software is shown in the figure:

Fig: Relationship between System and Application Software

 Each circle   is   an   interface point. The inner core is hardware. The user is represented by the outer layer. To work with the system, the typical user uses some form of application software. The application software inturn interacts with operating system (OS), which is part of system software layer. The System software   provides   the   direct interaction with the hardware. The opening   at the bottom of the figure is the path followed by the user   who interacts directly with   the Operating System when necessary.

2. Explain Creation and Running of programs ? (or)
  Describe how the Developers will write the programs ? (or)
   What are different steps followed in program development?

Ans: Creating and running programs:
 It is the job of programmer to write and test the program. The following
are four steps for creating and running programs:

> A. Writing and Editing the Program.
> B. Compiling the Program.
> C. Linking the Program with the required library modules.
> D. Executing the Program.

A. Writing and Editing Program: The Software to write programs is known
as text editor. A text editor helps us enter, change and store character data.
Depending on the editor on our system, it could be used to write letters, create
reports or write programs.
Example : word processor.
The text editor could be generalized word processor, but every compiler comes
with associated text editor. Some of the features of editors are
Search :   To locate and replace statements.
Copy , Paste :   To copy and move statements.
Format :   To set tabs to align text.
After the program is completed the program is saved in a file to disk. This file
will be input to the compiler, it is known as source file. The following figure
shows the various steps in building a *C – program*
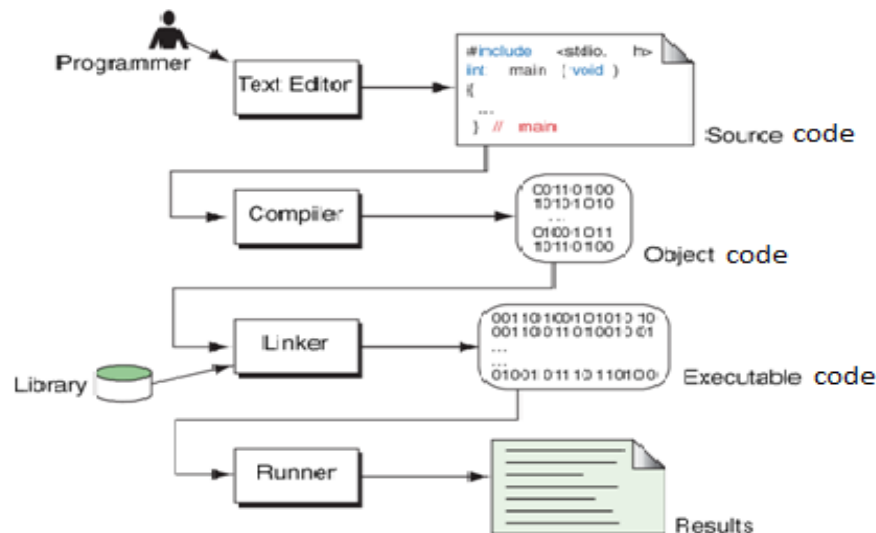

Fig: Building a C - program

B. Compiling Programs: The code in a source file on the disk must be
translated into machine language. This is the job of compiler which translates

code in source file stored on disk into machine language. The C compiler is actually two separate programs: the preprocessor and the translator. The preprocessor reads the source code and prepares it for the compiler. It scans special instructions known as preprocessor commands. These commands tell the preprocessor to take for special code libraries, make substitutions in the code. The result of preprocessing is called translation unit. The translator reads the translation unit and writes resulting object module to a file that can be combined with other precompiled units to form the final program. An object module is the code in machine language. This module is not ready for execution because it does not have the required C and other functions included.

C. Linking Programs: C programs are made up of many functions.
Example: printf( ) , cos( )… etc
Their codes exists elsewhere , they must be attached to our program. The *linker* assembles all these functions, ours and the system's, into a final executable program.

D. Executing Programs : Once our program has been linked, it is ready for execution. To execute a program, we use operating system command, such as *run* to load the program in to main memory and execute it. Getting program into memory is the function of an Operating System programs called loader. *Loader* locates the executable program and reads it into memory. In a typical program execution, the program reads data for processing, either from user or from file. After the program processes the data, it prepares output. Data output can be to user's monitor or to a file. When program has executed, *Operating System* removes the program from memory.

3. Explain about Algorithm ? (or)

Define an Algorithm and State Properties of it ?

Ans: Algorithm :
Algorithm is a finite set of instructions that , if followed accomplishes a particular task.
Algorithm should satisfy the following criteria
 1. *Input :* Zero or more quantities are externally supplied.
 2. *Output:* At least one quantity is produced.
 3. *Definiteness :* Each instruction is clear and unambiguous.
         Ex: Add B or C to A
 4. *Finiteness :* Algorithm should terminate after finite number of steps
         when traced in all cases. Ex: Go on adding elements to an array
 5. *Effectivenes:* Every instruction must be basic i.e., it can be carried out, by a person using pencil and paper.
Algorithm must also be general to deal with any situation.

4. List out the advantages and disadvantages of algorithm.
   Ans: **Advantages of Algorithms:**
- It provides the core solution to a given problem. the solution can be implemented on a computer system using any programming language of user's choice.
- It facilitates program development by acting as a design document or a blue print of a given problem solution.
- It ensures easy comprehension of a problem solution as compared to an equivalent computer program.
- It eases identification and removal of logical errors in a program.
- It facilitates algorithm analysis to find out the most efficient solution to a given problem.

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

5. Explain the Algorithm with Examples ?
   Ans: Algorithm:
The same problem can be solved with different methods. So, to solve a problem different algorithms, may be accomplished. Algorithm may vary in time, space utilized. User writes algorithm in his / her own language. So, it   cannot be executed on computer. Algorithm should be in sufficient detail that it   can be easily translated into any of the languages.

   Example1 : Add two numbers.
   Step 1:   Start
   Step 2:   Read 2 numbers as A and B
   Step 3:   Add numbers A and B and store result in C
   Step 4 :   Display C
   Step 5:   Stop
   Example2:   Average of 3 numbers.
          1.   Start
          2.   Read the numbers a , b , c
          3.   Compute the sum and divide by 3
          4.   Store the result in variable d
          5.   Print value of d
          6.   End

   Example3: Average of n inputted numbers.
          1.   Start
          2.   Read the number n
          3.   Initialize i   to   zero
          4.   Initialize sum to zero
          5.   If i is greater than n
          6.   Read a
          7.   Add a to sum

6

8.   Go to step 5
9.   Divide sum by n & store the result in avg
10.   Print value of avg
11   End

6. Explain about a Flowchart ?

Ans: *Flowchart :*
A *flowchart* is a visual representation of the sequence of steps for solving a problem . A *flowchart* is a set of symbols that indicate various operations in the *program.* For every process , there is a corresponding *symbol* in the *flowchart* . Once an *algorithm* is written , its pictorial representation can be done using *flowchart symbols*. In other words, a pictorial representation of a *textual algorithm* is done using a *flowchart.*

A *flowchart* gives a pictorial representation of an *algorithm.*

- The first *flowchart* is made by John Von Neumann in 1945.
- It is a symbolic diagram of operations sequence, dataflow, control flow and processing logic in information processing.
- The symbols used are simple and easy to learn.
- It is a very helpful tool for programmers and beginners.

Purpose of a Flowchart :
- Provides communication.
- Provides an overview.
- Shows all elements and their relationships.
- Quick method of showing program flow.
- Checks program logic.
- Facilitates coding.
- Provides program revision.
- Provides program documentation.

Advantages of a Flowchart :
- *Flowchart is* an important aid in the development of an algorithm itself.
- Easier to understand than a program itself.
- Independent of any particular programming language.
- Proper documentation.
- Proper debugging.
- Easy and clear presentation.

Limitations of a Flowchart :
- Complex logic.
- Drawing is time consuming.
- Difficult to draw and remember.
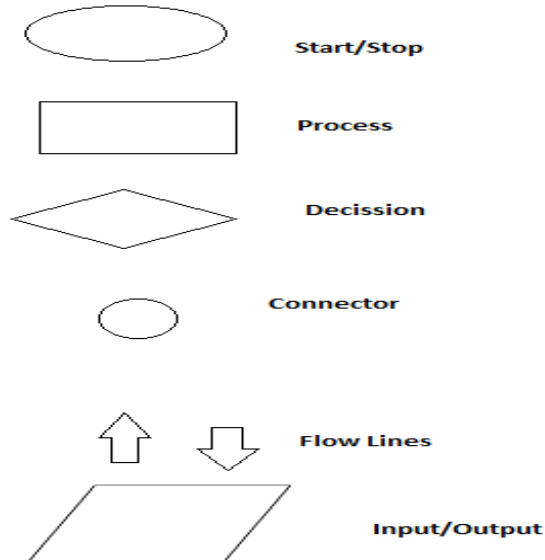- Technical detail.

7. **Explain the Symbols in a Flowchart** ?

Ans: *Symbols : Symbols are divided in to the following two parts.*
   I. Auxiliary *Symbols.*
   II. *Primary Symbols.*

Some of the *common symbols used in flowcharts* are shown below :

8. **Write an algorithm and flow chart for swapping two numbers**

Ans: To Swap two integer numbers:

Algorithm : using third variable

            Step 1 :   Start
            Step 2 :   Input num1 , num2
            Step 3 :   temp = num1
            Step 4 :   num1 = num2
            Step 5 :   num2 = temp
            Step 6 :   Output num1 , num2
            Step 7 :   Stop

Algorithm : without using third variable

            Step 1 :   Start
            Step 2 :   Input num1 , num2
            Step 3 :   calculate num1 = num1 + num2
            Step 4 :   calculate num2 = num1 − num2
            Step 5 :   calculate num1 = num1 − num2
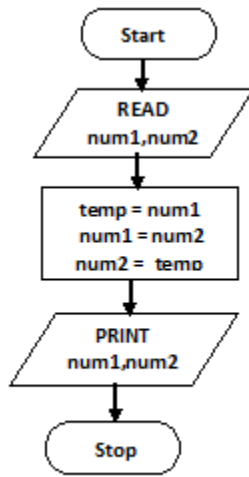   Step 6 :   Output num1 , num2
            Step 7 :   Stop

Flowcharts :

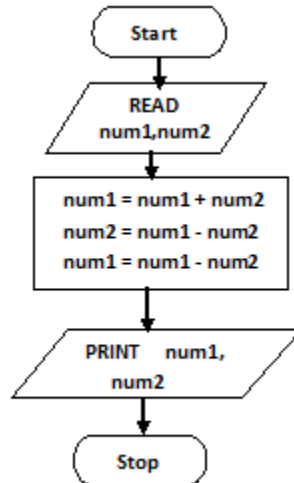Fig a: With using third variable   Fig b: Without using third variable

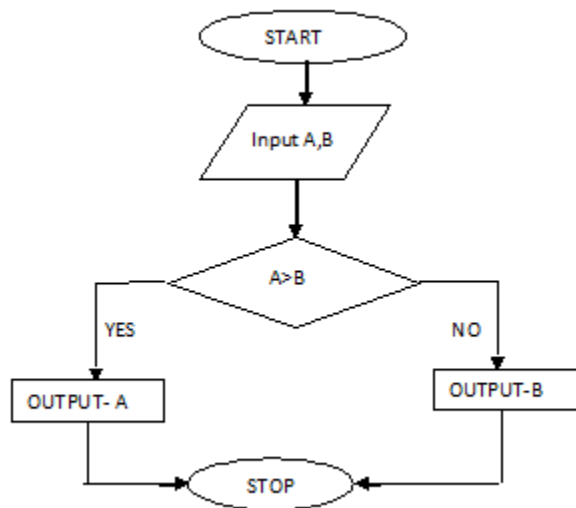**9. Write an algorithm and flowchart to find the largest among two numbers.**

Ans:

Algorithm:

Step 1 :   Start
Step 2 :   Input   A , B
Step 3 :   if A > B then output A
             else output B
Step 4 :   Stop

Flowchart :

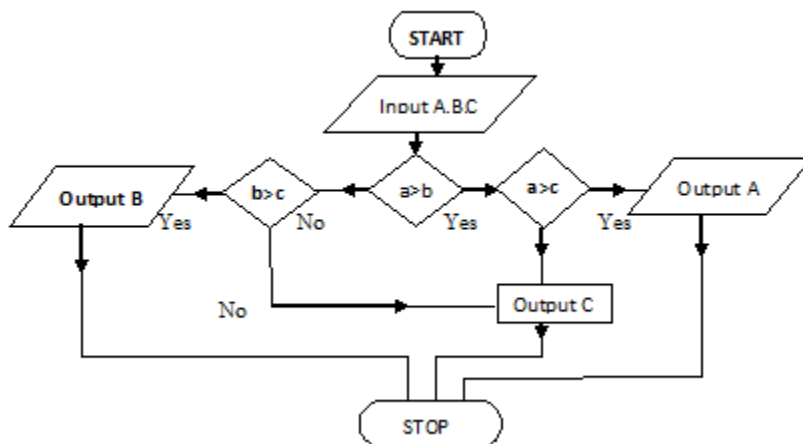**10. Write an algorithm and flowchart to find the largest among three numbers.**

Ans: Algorithm:
                Step 1 :   Start
                Step 2 :   Input A, B, C
                Step 3 :   if $A > B$ go to step 4 , otherwise go to step 5
                Step 4 :   if $A > C$ go to step 6 , otherwise go to step 8
                Step 5 :   if $B > C$ go to step 7, otherwise go to step 8
                Step 6 :   print " A is largest ", go to step 9
                Step 7 :   print " B is largest ", go to step 9
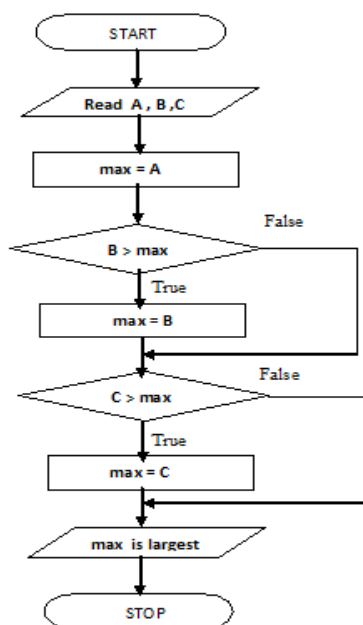   Step 8 :   print " C is largest ", go to step 9
                Step 9 :   Stop

Flowchart:

Algorithm:
                Step 1 :   Start
                Step 2 :   Input A, B, C
                Step 3 :   Let max = A
                Step 4 :   if B > max then max = B
                Step 5 :   if C > max then max = C
                Step 6 :   output max is largest
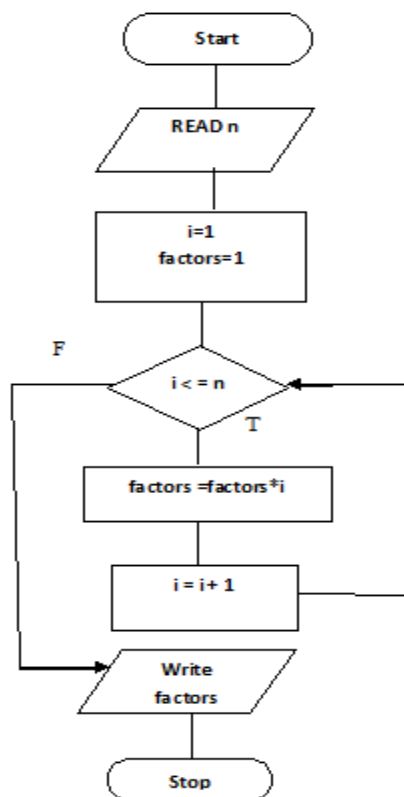   Step 7 :   Stop

Flowchart:

11. Write an algorithm and flowchart to find factorial of a number

Hint: fact (4) = 1 * 2* 3* 4

Ans: Flowchart:

Algorithm:

Step 1:   Start
Step 2:   Input n
Step 3:   Initialize counter variable, i , to 1 and factors = 1
Step 4:   if i <= n go to step 5 otherwise go to step 7
Step 5:   calculate factors = factors * i
Step 6:   increment counter variable, i, and go to step 4
Step 7:   output factors.
Step 8:   stop

12. Write an algorithm and flowchart to find whether a number is prime or not.

Hint: A number is said to be prime number for which the only factors are 1 and itself
Ans:

Flow chart:
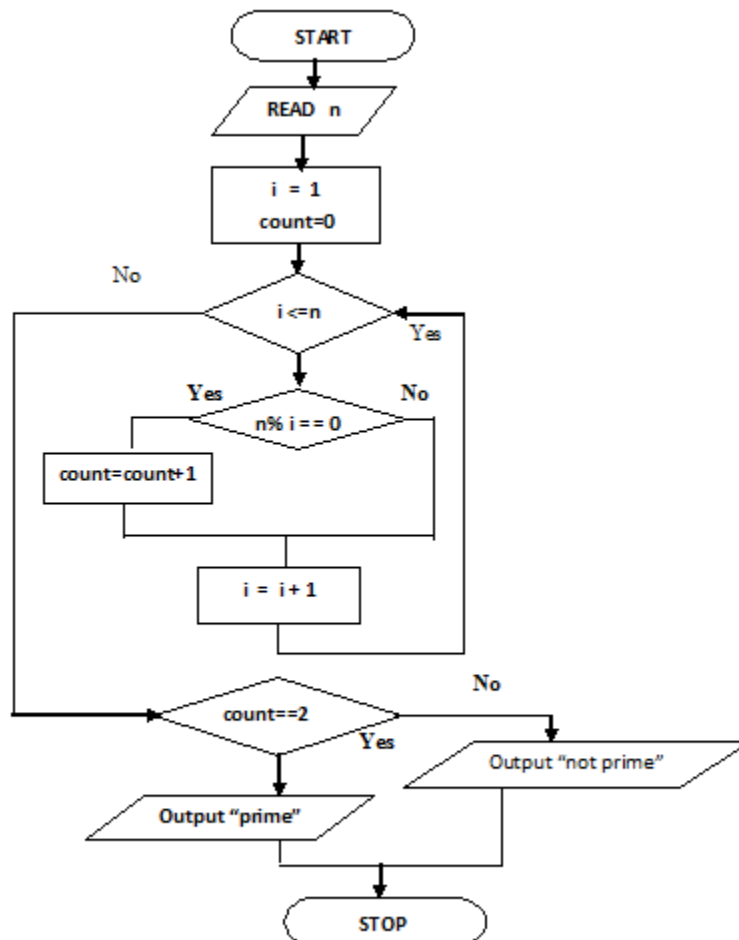
Algorithm:

	Step 1:	Start
	Step 2:	Input n
	Step 3:	Let i = 1, count=0
	Step 4:	if i > n/2 go to step 7
	Step 5:	if (n % i = = 0) count = count + 1
	Step 6:	increment i and go to step 4
	Step 7:	if count=2 then print "prime number" else print "not prime number"
	Step 8:	Stop

13. **Write an algorithm and flowchart to find reverse of a number**.

Ans: Algorithm:

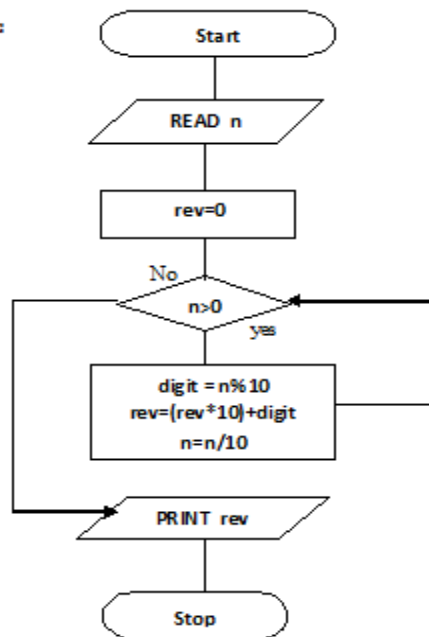|	Step 1:	Start
	Step 2:	Input n
	Step 3:	Let rev=0
	Step 4:	if n<=0 go to step 8
	Step 5:	digit = n % 10
	Step 6:	rev = (rev * 10) + digit
	Step 7:	n = n / 10 then go to step 4
	Step 8:	output rev
	Step 9:	Stop

Flow chart:
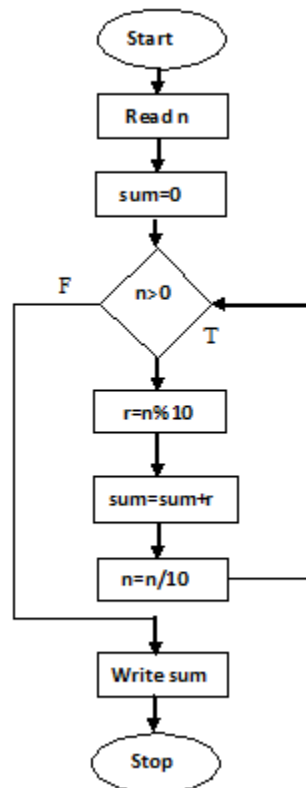
14. Write an algorithm and flowchart to find the Sum of individual digits if a given number

Ans: Algorithm :-

            Step 1 :   Start
            Step 2 :   read n
            Step 3 :   Sum = 0
            Step 4 :   While n > 0 do
            Step 4.1 : r = n % 10
            Step 4.2 : Sum = Sum + r
            Step 4.3 : n = n/10
            Step 5 :   End while
            Step 6 :   Write Sum
            Step 7 :   Stop

Flowchart :-

15. Explain the programming languages?

Ans: "The language used in the communication of computer instructions is known as the **programming language**". The computer has its own language and any communication with the computer must be in its language or translated into this language. Three levels of programming languages are available. They are:

1. Machine languages (low level languages)

14

2. Assembly (or symbolic) languages

3. Procedure-oriented languages (high level languages)

1. Machine language: Computers are made of two-state electronic devices they can understand only pulse and no-pulse (or '1' and '0') conditions. Therefore, all instructions and data should be written using binary codes 1 and 0. This binary code is called the **machine code or machine language**.

     Computers do not understand English, Hindi or Tamil. They respond only to machine language. Added to this, computers are not identical in design, therefore, each computer has its own machine language.(However the script 1 and 0 , is the same for all computers). This poses two problems for the user.

 1. It is difficult to understand and remember the various combinations of 1's and 0's representing numerous data and instructions. Also, writing error-free instructions is a slow process.

 2. As every machine has its own machine language, the user cannot communicate with other computers

.Machine languages are usually referred to as the first generation languages.

2. Assembly language: The Assembly language which is introduced in 1950s, reduced programming complexity and provided some standardization to build an application. The assembly language, also referred to as the **second-generation programming language**, is also a low-level language. In an assembly language, the 0s and 1s of machine language are replaced with **abbreviations o**r **mnemonic c**ode.

The main advantages of an assembly language over a machine language are:

1. As we can locate and identify syntax errors in assembly language, it is easy to debug it.

2. It is easier to develop a computer application using assembly language in comparision to machine language.

3. Assembly language operates very efficiently.

An assembly language program consists of a series of instructions and mnemonics that correspond to a stream of executable instructions. An assembly language instruction consists of a mnemonic code followed by zero or more operands. The mnemonic code is called the **operation code or opcode**, which specifies the operation to be performed on the given arguments.

   During the execution, the assembly language program is converted into the machine code with the help of an **assembler**. The simple assembly language statements had one-to-one correspondence with the machine language statements. This one-to-one correspondence still generated complex programs. Then, macroinstructions were devised so that multiple machine language statements could be represented using a single assembly language instruction. Even today programmers prefer to use an assembly language for performing certain tasks such as:

15

----

1. To initialize and test the system hardware prior to booting the operating system. This   assembly language code is stored in ROM.

2. To write patches for disassembling viruses, in anti-virus product development companies.

3. To attain extreme optimization, for example, in an inner loop in a processor-

    intensive algorithm.

4. For direct interaction with the hardware.

5. In extremely high-security situations where complete control over the environment is required.

6. To maximize the use of limited resources, in a system with serve resource constraints.

3. High-level languages: High level languages further simplified programming tasks by reducing the number of computer operation details that had to be specified. High level languages like COBOL, Pascal, FORTRAN, and C are more abstract, easier to use, and more portable across platforms, as compared to low level programming languages. Instead of dealing with registers, memory addresses and call stacks, a programmer can concentrate more on the logic to solve the problem with help of variables, arrays or Boolean expressions.

High-level languages can be classified into the following three categories:

  1. Procedure-oriented languages (third generation)

  2. Problem-oriented languages (fourth generation)

  3. Natural languages (fifth generation)

1. Procedure-oriented languages

High level languages designed to solve general-purpose problems are called procedural languages or third generation languages. These include BASIC, COBOL, FORTRAN, C, C++, and JAVA, which are designed to express the logic and procedure of a problem.

2. Problem-oriented languages

Problem-oriented languages are used to solve specific problems and are known as the fourth-generation languages. These include query languages, report generators and application generators which have simple, english-like syntax rules.

3 . Natural languages

    Natural languages are designed to make a computer to behave like an expert and solve problems. The programmer just needs to specify the problem and the constraints for problem-solving. Natural languages such as LISP and PROLOG are mainly used to develop artificial intelligence and expert systems. These languages are widely known as fifth generation languages.

----

16. Explain the history of C ?

Ans: The history and development of C is shown as follows

Year	Language	Author
1960	ALGOL	International Group
1967	BCPL	Martin Richards
1970	B	Ken Thompson
1972	Traditional C	Dennis Ritchie
1978	K&R C	Kernighan and Ritchie
1989	ANSI C	ANSI Committee
1990	ANSI/ISO C	ISO Committee
1999	C99	Standardization Committee

fig: History of ansi c

The root of all modern languages is ALGOL which is introduced in early 1960s.ALGOL was the first computer language to use a block structure. ALGOL gave the concept of **structured programming** to computer science community.

In 1967,Martin Richards developed a language called BCPL(Basic Combined Programming Language)primarily for writing system software.

In 1970,Ken Thompson created a language using many features of BCPL and called it as "B"."B" was used to create early versions of UNIX operating system at BELL laboratories.

"C" was evolved from ALGOL,BCPL and B by Dennis Ritchie at the Bell laboratories in 1972.The language became more popular after the publication of the book **"The C Programming Language**" by **Brain Kerningham and Dennis Ritchie** in 1978.C uses many concepts from these languages and added the concept of datatypes and other powerful features. Since it was developed along with the UNIX operating system, it is strongly associated with UNIX.
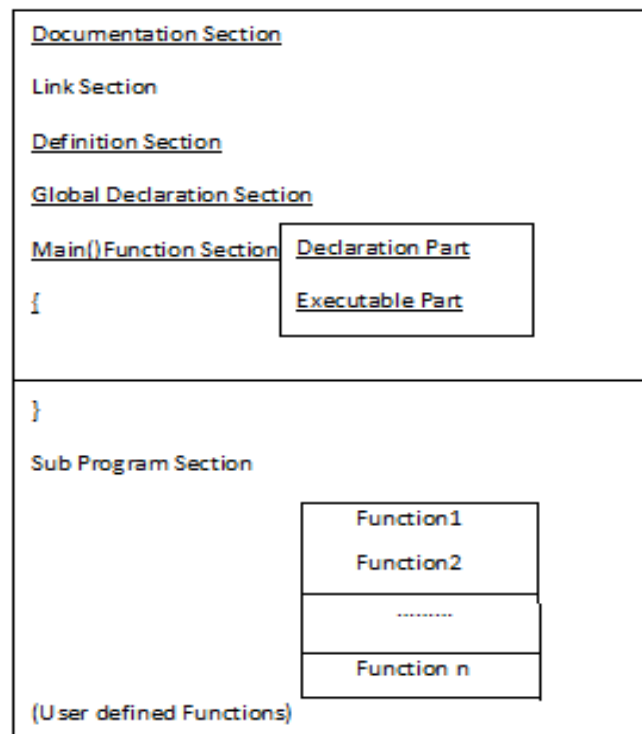
For many years C was used mainly in academic environments but eventually with the release of many C compilers for commercial use it increased its popularity. This rapid growth of C led to the development of different versions of the language that were similar but often

incompatible, this posed a serious problem for system developers. In order to assure that the C language remains standard in 1983 American National Standards Institute (ANSI) appointed a technical committee to define a standard for C. The committee approved a version of C in December 1989 which is now known as ANSI C which is approved by the International Standards Organization(ISO) in 1990.This version of C is referred to as C89.

During 1990's C++,a language entirely based on C , underwent a number of improvements and changes and became an ANSI/ISO approved language in November 1977.C++ added several new features to C to make it not only a true object –oriented language but also a more versatile language. Although C++ and Java were evolved out of C, the standardization committee of C felt that a few features of C++/Java , if added to C, would enhance the usefulness of the language. The result was the 1999 standard for C. This version is usually referred to as C99.

17. **What is the general structure of a `C' program and explain with example?**

Ans: Structure of C program:

```
Documentation Section

Link Section

Definition Section

Global Declaration Section

Main() Function Section  | Declaration Part
                         |
{                        | Executable Part
-----------------------------------------------
}

Sub Program Section

                         | Function1
                         | Function2
                         | ..........
                         | Function n

(User defined Functions)
```

Documentation section :

This section consists of a set of comment lines giving the name of the program, and other details. which the programmer would like to user later.

Ex:-   /*Addition of two numbers */

Link section: Link section provides instructions to the compiler to link functions from the system library.

Ex:-   # include<stdio.h>

           # include<conio.h>

Definition section: Definition section defines all symbolic constants.

Ex:-   # define A 10.

Global declaration section: Some of the variables that are used in more than one function throughout the program are called global variables and declared outside of all the functions. This section declares all the user-defined functions.

Main() function section:

   Every C program must have one main ( ) function section. This contains two parts.

i) Declaration part: This part declares all the variables used in the executable part.

Ex:-   int a,b;

ii) Executable part: This part contains at least one statement .These two parts must appear between the opening and closing braces. The program execution begins at the opening brace and ends at the closing brace. All the statements in the declaration and executable parts end with a semicolon (;).

Sub program section: This section contains all the user-defined functions, that are called in the main () function. User- defined functions are generally placed immediately after the main() function, although they may appear in any order.

Ex:

----
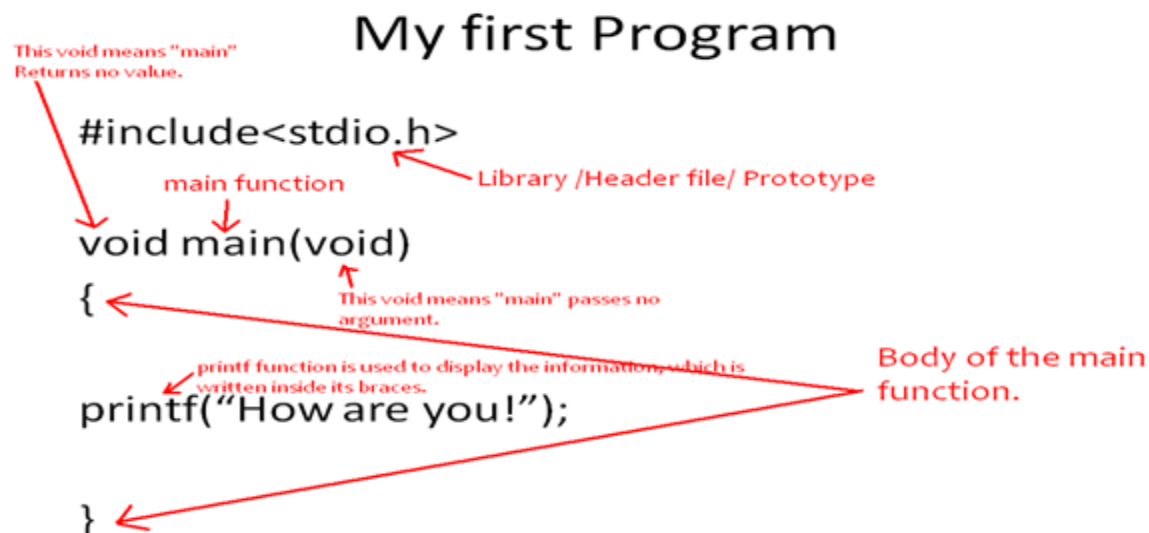
18. **What is a variable? Write the rules for constructing a variable?**

Ans: Variable: It is a data name that may be used to store a data value. It cannot be changed during the execution of a program. A variable may take different values at different times during execution. A variable name can be chosen by the programmer in a meaningful way so as to reflect its function or nature in the program.

Rules: Variable names may consist of letters, digits and under score(_) character.

> ➢ First char must be an alphabet or an '-'
> ➢ Length of the variable cannot exceed upto 8 characters, some C compilers can recognized upto 31 characters.
> ➢ White space is not allowed.
> ➢ Variables name should not be a keyword.
> ➢ Uppercase and lower case are significant.

     Ex:-   mark,sum1,tot_value,delhi→**valid**

                Prics\$, group one, char→ **invalid**

19. How to declare and initialize a variable?

Ans: Declaration does two things:

1. It tells the compiler what the variable name is.
2. It specifies what type of data the variable will hold.

The declaration of variables must be done before they are used in the program.

The syntax for declaring a variable is as follows:

data-type v1,v2,……,vn;

v1,v2,…,vn are the names of variables. Variables are separated by commas. A declaration statement must end with a semicolon. For example , valid declarations are:

int count;

int number, total;

double ratio;

The simplest declaration of a variable is shown in the following code fragment:

Ex:

/*…………….Program Name……………..*/

main()

{

/*……………….**Declaration**…………………………..*/

float x,y;

int code;

shortint count;

longint amount;

double deviation;

unsigned n;

char c;

/*………………………….**Computation**………………………….*/

  }/*………………………….**Program ends**…………………..*/

Initialization of variable : Initialize a variable in c is to assign it a starting value. Without this we can't get whatever happened to memory at that moment.

C does not initialize variables automatically. So if you do not initialize them properly, you can get unexpected results. Fortunately, C makes it easy to initialize variables when you declare them.

For Example :

int x=45;

int month_lengths[] ={23,34,43,56,32,12,24}**;**

struct role = **{** "Hamlet", 7, FALSE, "Prince of Denmark ", "Kenneth Branagh"};

Note : The initialization of variable is a good process in programming.

----

20. Explain data types in 'C'?

Ans: Data types

Data type is the type of the data that are going to access within the program. C supports different data types. Each data type may have pre-defined memory requirement and storage representation. C supports 4 classes of data types.

1. Primary or (fundamental) data type(int, char, float, double)
2. User-defined data type(typedef)
3. Derived data type(arrays, pointers, structures, unions)
4. Empty data type(void)- void type has no value.
1   byte = 8 bits (0's and 1's)

1. Primary or (fundamental) data type

All C compilers support 4 fundamentals data types, they are

   i)   Integer (int)
   ii)   Character(char)
   iii)   Floating (float)
   iv)   Double – precision floating point(double)

i). Integer types:

Integers are whole numbers with a range of values supported by a particular machine. Integers occupy one word of storage and since the word size of the machine vary. If we use 16 bit word length the size of an integer value is -32768 to +32767. In order to control over the range of numbers and storage space, C has 3 classes of integer storage, namely short, long, and unsigned.

DATA TYPES	RANGE	Size	Control string
Int	-2^{15} to 2^{15} -1 -32768 to +32767	2 bytes	%d (or) %i
signed short int(or) short int	-128 to +127	1 byte	%d (or) %i
unsigned short int	0 to 255	1 byte	%d (or) %i
unsigned int	0 to 65,535	2 bytes	%u
unsigned long int	0 to 4,294,967,295	4 bytes	%lu
long int (or)signed long int	-2147483648 to +2147483647	4 bytes	%ld

ii). Character type:-

Single character can be defined as a character (char) type data. Characters are usually stored in 8 bits of internal storage. Two classes of char types are there.

signed char, unsigned char.

signed char(or) char   → 1 byte-→ -128 to +127→%c

unsigned char→ 1 byte→ 0 to 255→%c

iii). Floating point types:

Floating point (real) numbers are stored in 32 bits, with 6 digits of precision when accuracy provided by a float number is not sufficient

float → 4 bytes→3.4e-38 to 3.4e+38→%f

iv). Double precision type:

double datatype number uses 64 bits giving a precision of 14 digits. These are known as double precision numbers. Double type represents the same data type that float represents, but with a greater precision. To extend the precision further, we may use long double which uses 80 bits.

double→ 8 bytes → 1.7e-308 to 1.7e+308→%lf

long double→ 10 bytes → 3.4e-4932 to 1.1e+4932→ %lf

2. User defined data types:

C –supports a feature known as "type definition" that allows users define an identifier that would represents an existing type.

Syntax: typedef data-type identifier;

Where data-type indicates the existing datatype

identifier indicates the new name that is given to the data type.

Ex:- typedef int marks;

marks m1, m2, m3;

→typedef cannot create a new data type ,it can rename the existing datatype. The main advantage of typedef   is that we can create meaningful datatype names for increasing the readability of the program.

Another user-defined datatype is "enumerated data type(enum)"

Syntax: enum identifier{value1, value2,……….valuen};

Where identifier is user-defined datatype which is used to declare variables that can have one of the values enclosed within the braces. value1 ,value2,……valuen all these are known as enumeration constants.

Ex:-   enum identifier v1, v2,……vn

v1=value3;
v2=value1;………
Ex:-enum day {Monday,Tuesday…………. sunday};
enum day week-f,week-end

Week-f = Monday
-
-
-
(or)
enum day{Monday…Sunday}week-f, week-end;

21. Describe the different types of constants in C with example?   or
    What are the rules for creating C constants explain with example?

Ans: Types of c constants
            1.Integer constants
            2.Real constants
            3.Character constants
            4. String constants

1. Integer constants: An integer constant refers to a sequence of digits. There are three types of integers, namely, decimal integer, octal integer and hexadecimal integer.

Examples of Integer Constants:
426 **,**+786 , -34(decimal integers)

037,   0345,   0661(octal integers)

0X2, 0X9F,   0X (hexadecimal integers)

2. Real constants: These quantities are represented by numbers containing fractional parts like 18.234. Such numbers are called real (or floating point) constants.

Examples of Real Constants:
+325.34
426.0
-32.67 etc.
The exponential form of representation of real constants is usually used if the value of the constant is either too small or too large. In exponential form of representation the real constant is

24

represented in two parts. The first part present before 'e' is called **Mantissa** and the part following 'e'is called **Exponent**.

For ex. .000342 can be written in Exponential form as 3.42e-4.

3. Single Character constants: Single character constant contains a single character enclosed within a pair of single quote marks**.**

        For ex. 'A',’5’,’;’,’ ‘

Note that the character constant’5’ is not same as the number 5. The last constant is a blank space.Character constant has integer values known as ASCII values. For example, the statement

Printf("%d",a); would print the number 97,the ASCII value of the letter a. Similarly, the statement **printf("%c",97);** would output the letter ‘a’

Backslash Character Constants: C supports some special backslash character constants that are used in output functions. Some of the back slash character constants are as follows:

Constant	Meaning
‘\0’	Null
‘\t’	Horizontal tab
‘\b’	Back space
‘\a’	Audible alert
‘\f’	Form feed
‘\n’	New line
‘\r’	Carriage return
‘\v’	Vertical tab
‘\’’	Single quote
‘\”’	Double quote
‘\?’	Question mark
‘\\’	backslash

These character combinations are called **escape sequences**.

String constants: A string constant is a sequence of character enclosed in double quotes. The characters may be letters, numbers, special characters and blank space.

Examples are: "HELLO!"   "1979"   "welcome"   "?.......!"

        "5+3"   "X"

Rules for constructing integer constants:

a. An integer constant must have at least one digit.
b. It must not have a decimal point.
c. It can be either positive or negative.
d. The allowable range for constants is -32768 to 32767
In fact the range of integer constants depends upon compiler.
For ex. 435
+786
-7000

Rules for constructing real constants:

a. A real constant must have at least one digit
b. It must have a decimal point.
c. It could be either positive or negative.
d. Default sign is positive.
For ex. +325.34
426.0
In exponential form of representation, the real constants is represented in two parts. The part appearing before 'e' is called mantissa where as the part following 'e' is called exponent.
Range of real constants expressed in exponential form is -3.4e38 to 3.4e38.
Ex. +3.2e-5

Rules for constructing character constants:

a. A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas.
b. The maximum length of character constant can be one character.
Ex `A`

22. What is an operator and List different categories of C operators based on their functionality? Give examples?

Ans: Operators:

An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables.

C operators can be classified into a number of categories, they are

1. Arithmetic Operators

2. Relational Operators

3. Logical Operators

4. Assignment Operators

5. Increment and decrement operators

----

6. Conditional operators

7. Bitwise Operators

8. Special operators

1. Arithmetic Operators: C provides all the basic arithmetic operators **.** These can operate on any built –in data type allowed in C.
The arithmetic operators are   listed as follows:

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo division

Integer division truncates any fractional part. The modulo division operation produces the

remainder of an integer division.

Ex: Here a and b are operands, let a=14 and b=4 we have the following results

a-b = 10

a+b = 18

a*b = 56

a/b = 3(coefficient)

a%b = 2(remainder)

2. Relational Operators:

Relational operators are used for comparing two quantities, and take certain decisions. For example we may compare the age of two persons or the price of two items….these comparisons can be done with the help of relational operators.

An expression containing a relational operator is termed as a relational expression. The value of a relational expression is either one or zero. It is one if the specified relation is true and zero if the relation is false.

Ex:-   13<34 (true)   23>35(false)

C supports 6 relational operators

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

Ex:-   4.5<=10(true)

6.5<-10(false)

10<4+12(true)

When arithmetic expression are used on either side of a relational operator, the arithmetic expression will be evaluated first and then the results compared, that means arithmetic operators have a higher priority over relational operators.

3. Logical Operator:

C has 3 logical operators. The logical operators are used when we want to test more than one condition and make decisions. The operators are as follows:

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

The logical operators && and || are used when we test more than one condition and make decisions.

Ex:- a>b && x==45

This expression combines two or more relational expressions, is termed as a logical expression or a compound relational expression. The logical expression given above is true only if a>b is true and x==10 is true. If both of them are false the expression is false. The truth table for logical and   and   logical or operators are:

OP1	OP2	OP1&&OP2	OP1||OP2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Some examples of logical expression

1. if (age >55 && salary<1000)
2. if(number<0 || number>100)

4. Assignment operator:

These operators are used to assign the result of an expression to a variable. The usual assignment operator is **'='.** In addition ,C has a set of shorthand assignment operators of the form:

$$v \ op = exp;$$

Where v is a variable, exp is an expression and op is a C binary arithmetic operator. The operator op=is known as the shorthand assignment operator.

The assignment statement v op=exp; is equivalent to

 v =v op (exp);

 Ex:- x+=y+1; is equivalent to   x= x+(y+1);

a*=a; is equivalent to   a=a*a;

The use of short hand assignment operators has the following advantages:

1) What appears on the left- hand side need not be repeated and therefore it becomes easier to write

2) The statement is more concise and easier to read

3)The statement is more efficient

5. Increment and Decrement operators:

++ and - - are increment and decrement operators in C. The operator ++ adds 1 to the operand, while - - subtracts 1.Both are unary operators.

++m; or m++; is equal to m=m+1(m+=1;)

--m; or m- - is equal to m=m-1(m- = 1;)

We use the increment and decrement statements in for and while loops extensively.

Ex:-   m=5;

y=++m;   the value of y=6 and m =6.

Suppose if we write the above statement as

     m=5;   y= m++; the value of y=5 and m=6.

A prefix operator first adds 1 to the operand and then the result is assigned to the variable on left. On the other hand, a postfix operator first assigns the value to the variable on left and then increments the operand.

6. Conditional operator:

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

exp1 ?exp2 : exp3

Where exp1,exp2 and exp3 are expressions,

The operator?: works as follows: exp1 is evaluated first. If it is non-zero (true), then the expression exp 2 is evaluated and becomes the value of the expression. If exp1 is false, exp3 is evaluated and its value becomes the value of the expression.

Ex:-   a=10; b=45;

    X = (a>b) ?a:b;

o/p:- X will be assigned the value of b (45).

7. Bitwise Operators:

C supports a special operator knows as bitwise operators for manipulation of data at bit level. These operators are used for testing the bits, or shifting them right to left. Bitwise operators may not be applied to float or double.

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	Shift left
>>	Shift right

8. Special operators:

C supports some special operators such as comma operator, size of operator, pointer operators (& and *) and member selection operators (. and ->).

----

comma operator: The comma operator is used to link the related expressions together. A comma- linked list of expressions is evaluated left to right and the value of right- most expression is the value of the combined expression. For example, the statement

value = (x=10, y=5, x+y);

This statement first assigns the value 10 to x, then assigns 5 to y and finally assigns 15.

 In for loops: for (n=1 , m=10, n<=m; n++, m++);

 sizeof operator: The sizeof is a compile time operator and when used with an operand, it returns the number of bytes the operand occupies. The operand may be variable, a constant or a data type qualifier.

    m = sizeof (sum);

    n = sizeof (long int);

The sizeof operator is normally used to determine the lengths of arrays and structures when their sizes are not known to the programmer. It is also used to allocate memory space dynamically to variables during execution of a program.

23. Explain the types of type conversions in C with example?

Ans: Type conversions: converting a variable value or a constant value temporarily from one data type to other data type for the purpose of calculation is known as type conversion.

There are two types of conversions

1. Automatic type conversion (or) implicit
2. Casting a value (or) explicit

1. **Implicit:** In this higher data type can be converted into lower data type automatically. The figure below shows the **C** conversion hierarchy for implicit –type conversion in an expression:
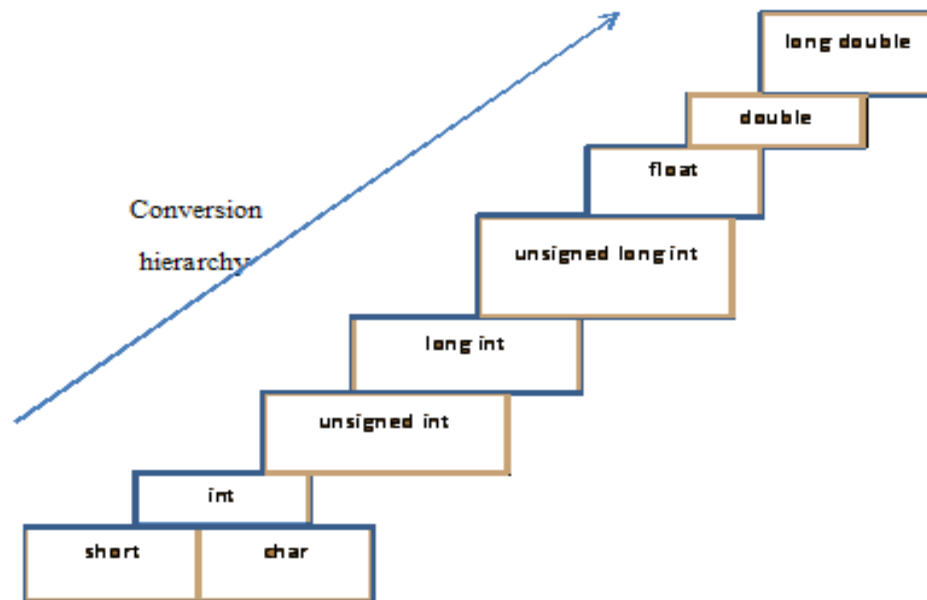
fig: Conversion hierarchy

The sequence of rules that are applied while implicit type conversion is as follows:

All **short** and **char** are automatically converted into **int** then

1. if one of the operands is **long double** the other will be converted to **long double** and the result will be **long double**.
2. else, if one of the operand is **double,**the other will be converted to **double** and the result will be **double**.
3. else, if one of the operand is **float** ,the other will be converted to **float** and the result will be **float**.
4. else, if one of the operand is **unsigned long int**, the other will be converted to **unsigned long int** and the result will be **unsigned long int.**
5. else, if one of the operand is **long int**, the other is **unsigned int** then

    a) if **unsigned int** can be converted into **long int**, the **unsigned int** operand will be converted as such and the result will be **long int.**

    b) else both operands will be converted **to unsigned long int** and the result will be **unsigned long int.**

6. else if one of the operand **is long int** ,the other will be converted into **long int** and the result   will be **long int.**

----

7. else if one of the operand **unsigned int** ,the other will be converted into **unsigned int** and the result will be **unsigned int.**

The final result of an expression is converted to the type of the variable on the left of the assignment sign before assigning the value to it. The following changes are introduced during the final assignment.

float to int causes truncation of the fractional part

double to float causes rounding of digits

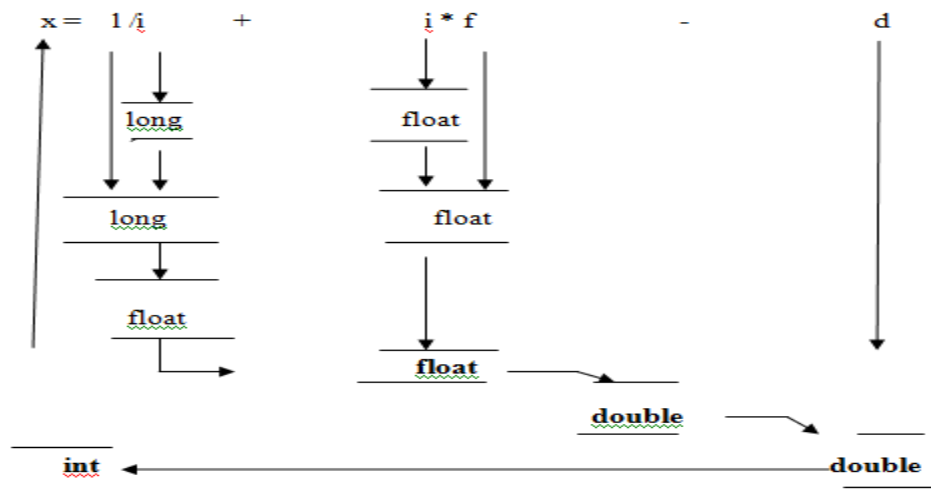long int to int causes dropping of the excess higher order bits

Ex: Consider the following variables along with their datatypes:

int i,x;

float f;

double d;

longint l;

2. Explicit: In this type of conversion, the programmer can convert one data type to other data type explicitly.

Syntax: (datatype) (expression)

Expression can be a constant or a variable

Ex: y = (int) (a+b)

        y= cos(double(x))

        double   a = 6.5

        double   b = 6.5

        int result = (int) (a) + (int) (b)

        result = 12 instead of 13.

        int a=10

        float(a)->10.00000

24. Define an expression? How can you evaluate an expression?

Ans: Expressions:

An expression in C is some combination of constants, variables, operators

and function calls.

Sample expressions are:

        a + b

        tan(angle)

• Most expressions have a value based on their contents.

• A statement in C is just an expression terminated with a semicolon.

For example:

        sum = x + y + z;

        printf("Go Buckeyes!");

The rules given below are used to evaluate an expression,

        1) If an expression has parenthesis, sub expression within the parenthesis is evaluated
        first and arithmetic expression without parenthesis is evaluated first.

        2) The operators of high level precedence are evaluated first.

----

3)The operators at the same precedence are evaluated from left to right or right to

left depending on the associativity of operators.

Expressions are evaluated using an assignment statement of the form:

variable = expression;

variable is any valid C variable name. When the statement is encountered, the expression is evaluated first and the result then replaces the previous value of the variable on the left-hand side. All variables used in the expression must be assigned values before evaluation is attempted.

Ex:- x = a*b-c;

y = b/c*a;

z = a-b / c+d;

Ex:- x= a-b/3+c*2-1   when a=9, b=12, and c=3 the expression becomes.

x = 9-12/3 +3*2-1

Step1: x = 9-4+3*2-1

Step2: x = 9-4+6-1

Step3:   x = 5+6-1

Step4: x = 11-1

Step5:   x = 10

25. Define precedence and associativity? Give an example? or

Explain the hierarchy (priority) and associativity(clubbing)of operators in 'C' with example?

Ans: Operator precedence:

Various relational operators have different priorities or precedence. If an arithmetic expression contains more operators then the execution will be performed according to their properties. The precedence is set for different operators in C.

Type of operator	Operators	Associativity
Unary operators	+,-,!,++,--,type,¯,size of	Right to left
Arithmetic operators	*, /,%,+,-	Left to right
Bit – manipulation operators	<<,>>	Left to right
Relational operators	>,<,>=,<=,==,!=	Left to right
Logical operators	&&,||	Left to right

Conditional operators	?,:	Left to right
Assignment operators	=,+=,-=,*=,/=,%=	Right to left

Important note:

1. Precedence rules decide the order in which different operators are applied
2. Associativity rule decides the order in which multiple occurrences of the same level operators are applied.

Hierarchy of operators in C
There are some operator which are given bellow with their meaning. The higher the position of an operator is, higher is its priority.

Operator	Type
!	Logical
*/ %	Arithmetic
+ -	Arithmetic
<><=>=	Relational
==!=	Relational
&&	Logical
||	Logical OR
=	Assignment

Associativity of operators

When an expression contains two operators of equal priority the tie between them is settled using the associativity of the operators.

Associativity can be of two types—Left to Right or Right to Left.

Left to Right associativity means that the left operand must be unambiguous. Unambiguous in what sense? It must not be involved in evaluation of any other sub-expression. Similarly, in case of Right to Left associativity the right operand must be unambiguous. Let us understand this with an example.

Consider expression a=3/2*5 Here there is a tie between operators of same priority, that is between / and *. This tie is settled using the associativity of / and *. But both enjoy Left to Right associativity.

While executing an arithmetic statement, which has two or more operators, we may have some problem as to how exactly does it get executed.

Priority	Operators	Description
1st	*, / , %	multiplication, division, modular division
2nd	+, -	addition, subtraction
3rd	=	Assignment

For Example :

i= 2*3/4+4/4+8-2+5/8
i=6/4+4+8-2+5/8
i=1+4/4+8-2+5/8
i=1+1+8-2+5/8
i=1+1+8-2+0
i=2+8-2+0
i=10-2+0
i=8+0
i=8