UNIT–I:

Introduction to components of a computer system: disks, primary and secondary memory, processor, operating system, compilers, creating, compiling and executing a program etc., Number systems.

Introduction to Algorithms: steps to solve logical and numerical problems. Representation of Algorithm, Flowchart/Pseudo code with examples, Program design and structured programming.

Introduction to C Programming Language: variables (with data types and space requirements), Syntax and Logical Errors in compilation, object and executable code ,Operators, expressions and precedence, Expression evaluation, Storage classes (auto, extern, static and register), type conversion,

Bitwise operations: Bitwise AND, OR, XOR and NOT operators

Conditional Branching and Loops: Writing and evaluation of conditionals and consequent branching with if, if-else, switch-case, ternary operator, goto, Iteration with for, while, do while loops

I/O: Simple input and output with scanf and printf, formatted I/O, Introduction to stdin, stdout and stderr.

Command line arguments

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.

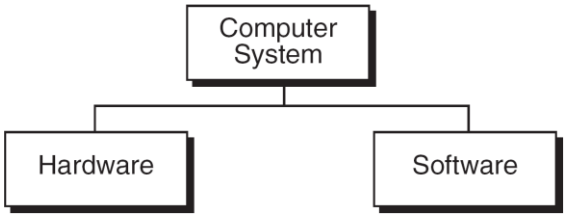


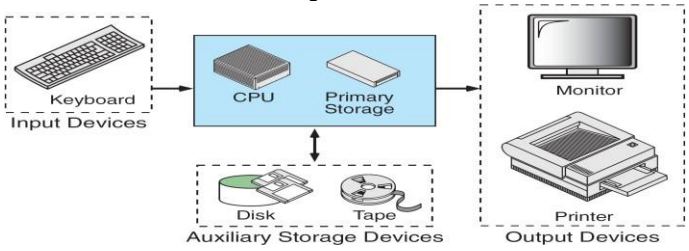
Fig : A Computer System

I.Computer Hardware:

The computer hardware is the physical equipment. The hardware component of computer system consists of 5 parts

- Input Devices.
- Central Processing Unit (CPU).
- Primary Storage.
- Output Devices.
- Auxiliary Storage Devices.

The following figure shows the *hardware* components.



- 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.

- 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

- 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).

- 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.

- 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.

- System Software
- Application Software

The following figure shows the Computer Software.

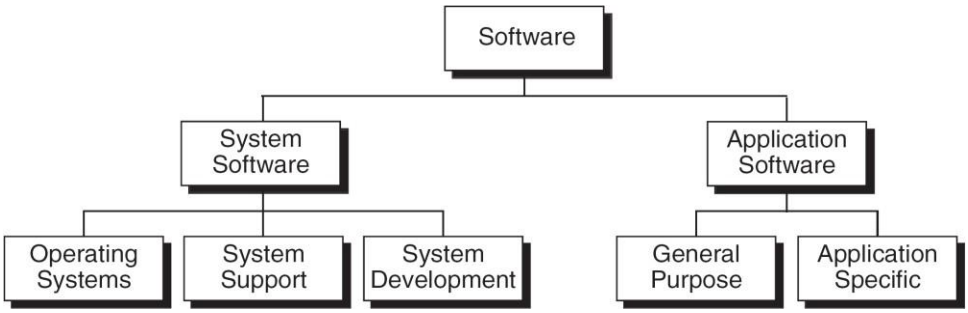


Fig: Computer Software

System Software: System Software consists of programs that manage the hardware resources of a computer and perform required information processing tasks.

These programs a r e divided into three classes.

- Operating System Software.
- System Support Software.
- 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.

Application Software: It is directly responsible for helping users to solve their problems. Application software is broken into two classes.

- General - Purpose Software
- 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:

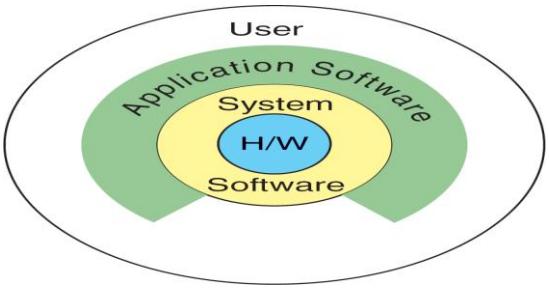


Fig: Relationship between System and Application Software

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:

- Writing and Editing the Program.
- Compiling the Program.
- Linking the Program with the required library modules.
- 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*

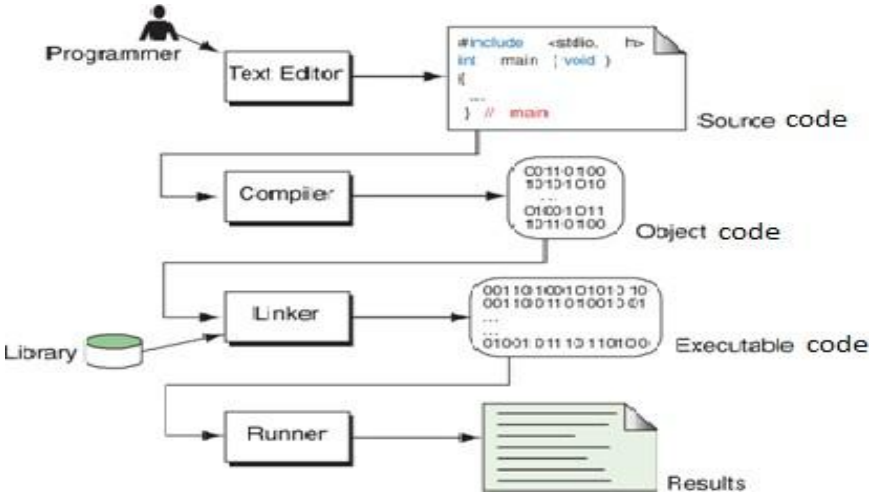


Fig: Building a C - program

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

- ***Input :*** Zero or more quantities are externally supplied.
- ***Output:*** At least one quantity is produced.
- ***Definiteness :*** Each instruction is clear and unambiguous. Ex: Add B or C to A
- ***Finiteness :*** Algorithm should terminate after finite number of steps when traced in all cases. Ex: Go on adding elements to an array
- ***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.

Step 1: Start
Step 2: Read the numbers a , b , c
Step 3: Compute the sum and divide by 3
Step 4: Store the result in variable d
Step 5: Print value of d
Step 6: End

Example3: Average of n inputted numbers.

Step 1: Start
Step 2: Read the number n
Step 3: Initialize i to zero
Step 4: Initialize sum to zero
Step 5: If i is greater than n
Step 6: Read a
Step 7: Add a to sum
Step 8: Go to step 5
Step 9: Divide sum by n & store the result in avg
Step 10: Print value of avg
Step 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?

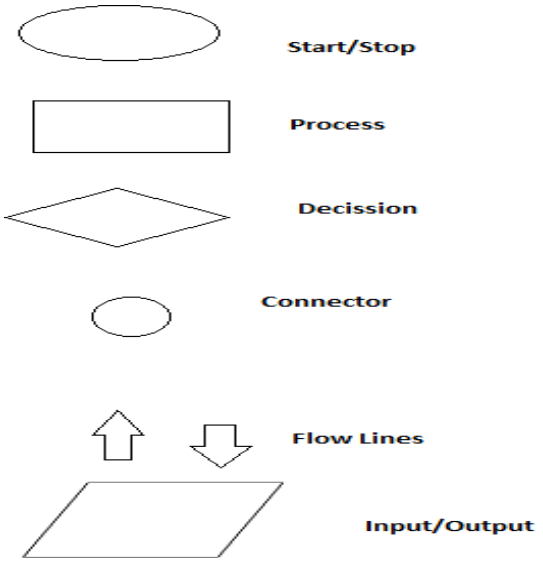
Ans:

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:

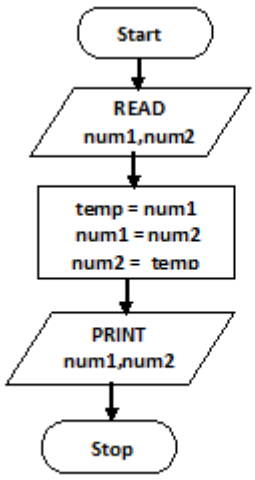


Fig a: With using third variable

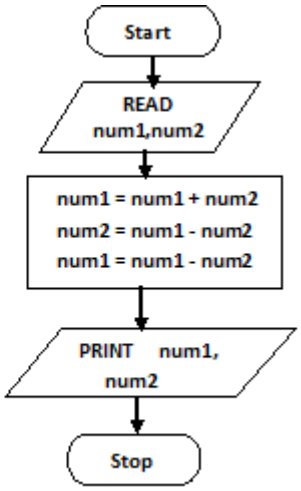


Fig b: Without using third variable

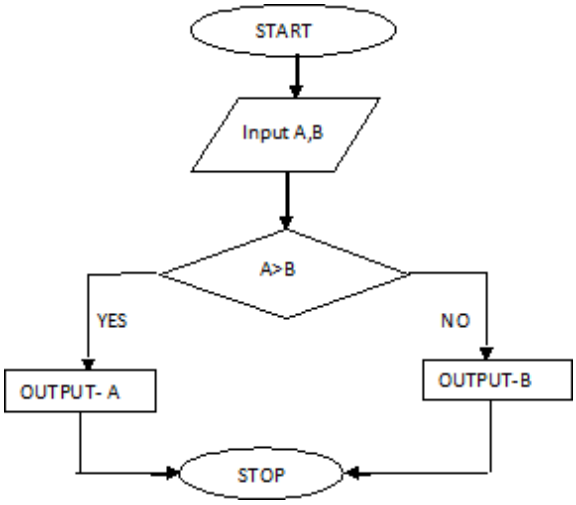
9. Write an algorithm and flowchart to find the largest among two numbers.

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



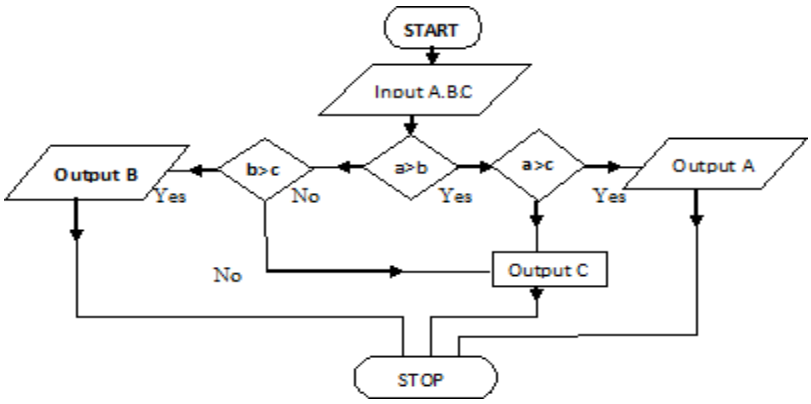
10. Write an algorithm and flowchart to find the largest among three numbers.

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

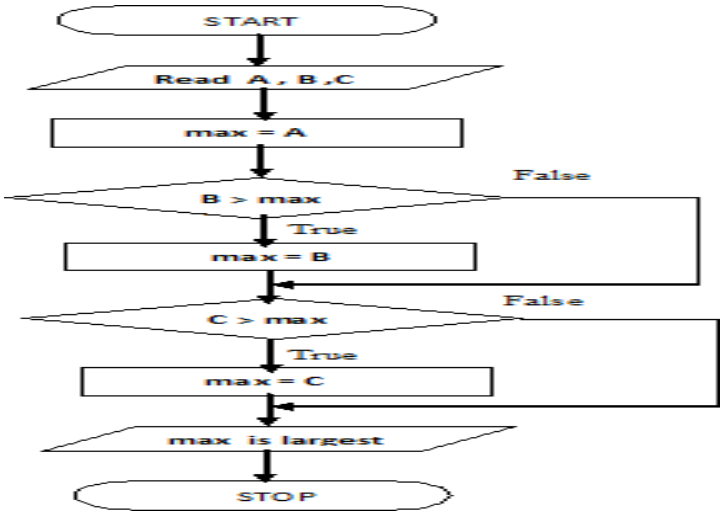
Flowchart:



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

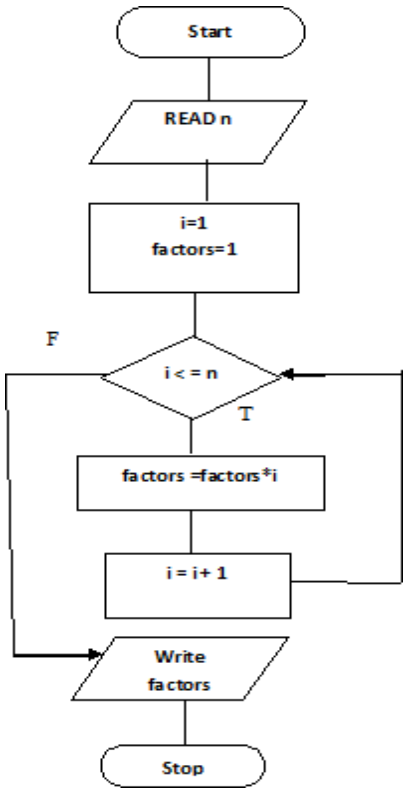
Flowchart:



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

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

Ans: Flowchart:



Algorithm:

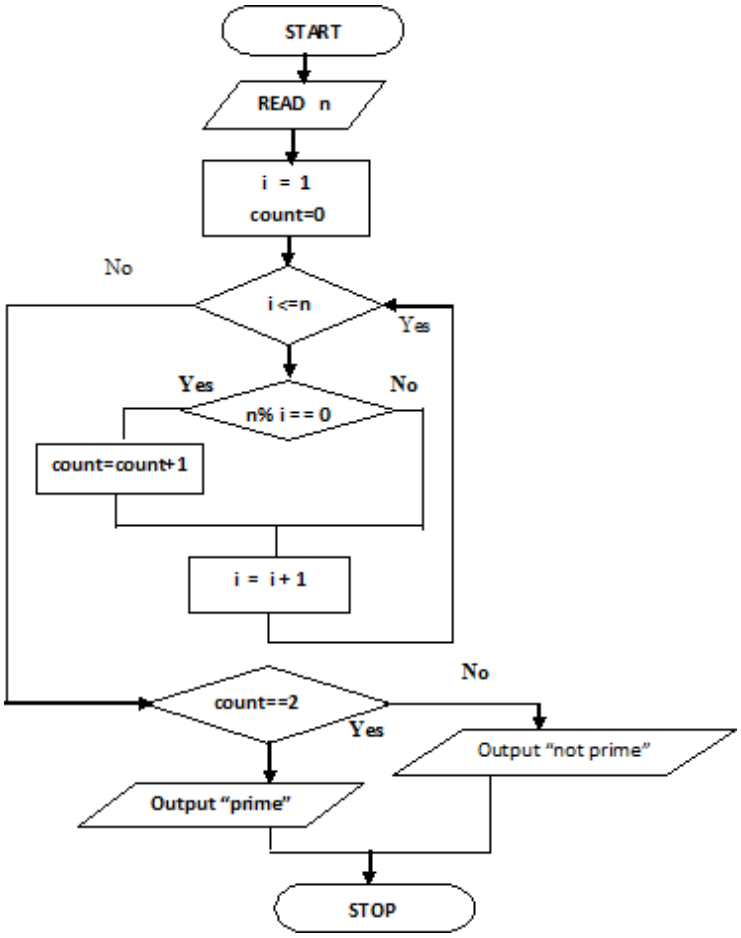
Step 1: Start
Step 2: Input n
Step 3: Initialize counter variable, i , to 1 and factors = 1
Step 4: if i <= n go to step 5 otherwise go to step 7
Step 5: calculate factors = factors * i
Step 6: increment counter variable, i, and go to step 4
Step 7: output factors.
Step 8: stop

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:

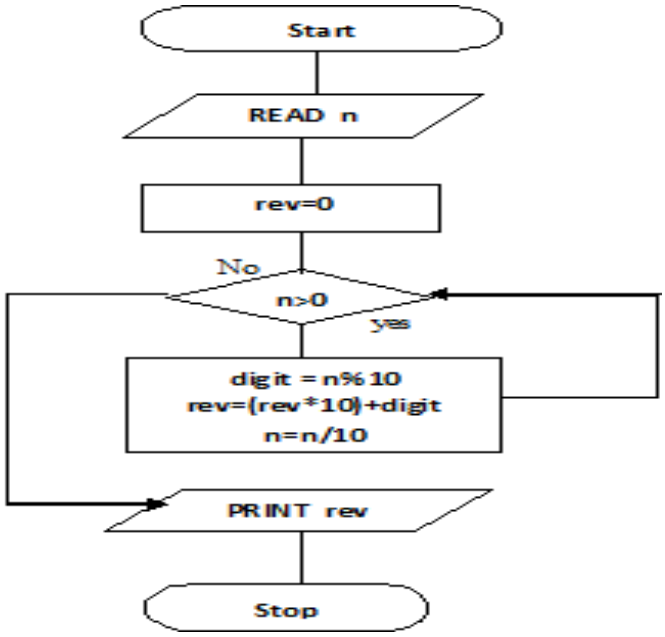
Step 1: Start
Step 2: Input n
Step 3: Let i = 1, count=0
Step 4: if i > n/2 go to step 7
Step 5: if (n % i = = 0) count = count + 1
Step 6: increment i and go to step 4
Step 7: if count=2 then print ”prime number” else print ”not prime number”
Step 8: Stop

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:

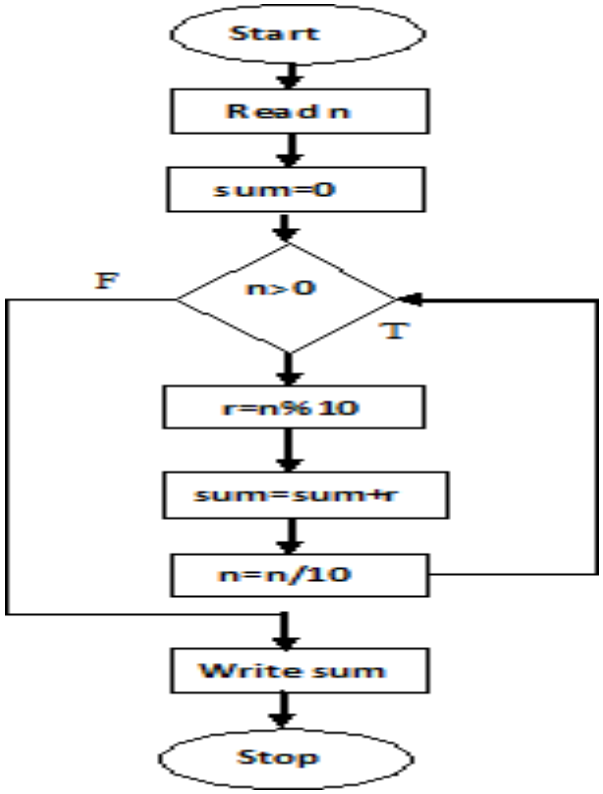


14. Write an algorithm and flowchart to find the Sum of individual digits if a given number

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)**
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 Telugu. 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.

- 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.
- 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:

- As we can locate and identify syntax errors in assembly language, it is easy to debug it.
- It is easier to develop a computer application using assembly language in comparison to machine language.
- 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:

- To initialize and test the system hardware prior to booting the operating system. This assembly language code is stored in ROM.
- To write patches for disassembling viruses, in anti-virus product development companies.
- To attain extreme optimization, for example, in an inner loop in a processor- intensive algorithm.
- For direct interaction with the hardware.
- In extremely high-security situations where complete control over the environment is required.
- 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:

- Procedure-oriented languages (third generation)
- Problem-oriented languages (fourth generation)
- Natural languages (fifth generation)

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.

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

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.

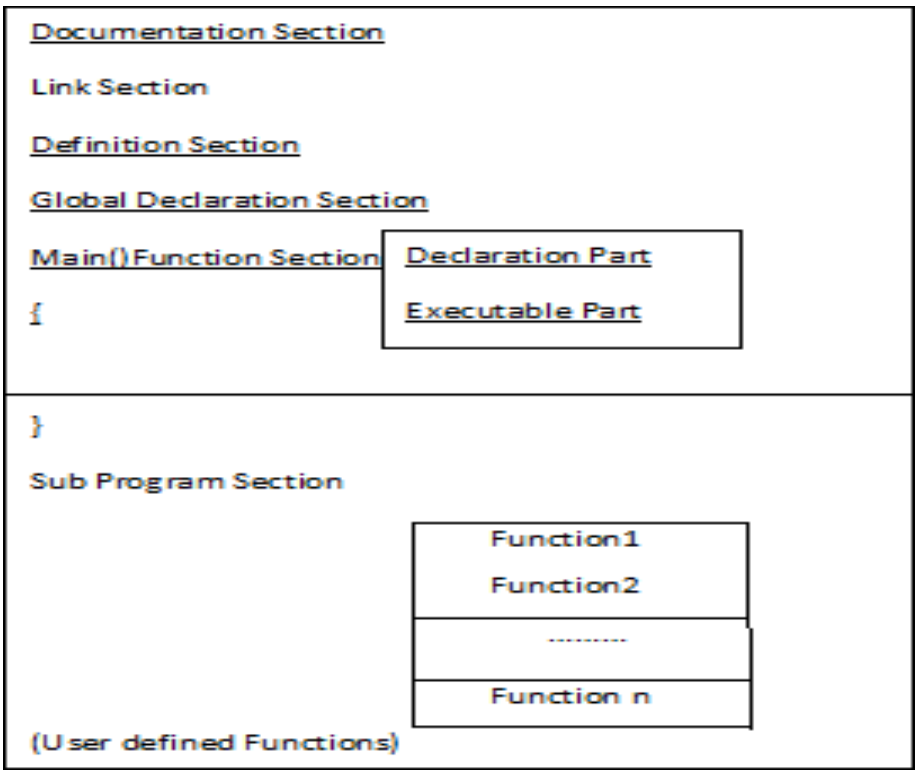
'C' was evolved from ALGOL,BCPL and B by Dennis Ritchie at the Bell laboratories in 1972.The language became more popular after the publication of the book **"The C Programming Language**" by **Brain Kerningham and Dennis Ritchie** in 1978.C uses many concepts from these languages and added the concept 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:

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 */

// Addition of two numbers

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

Ex:- # include<stdio.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:

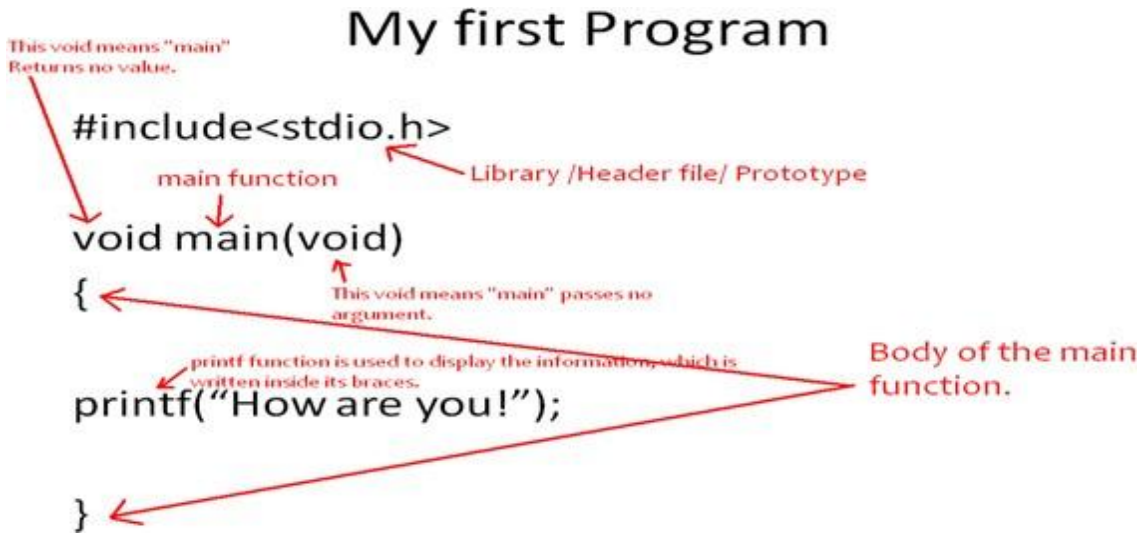
Every C program must have one main ( ) function section. This contains two parts.
Declaration part: This part declares all the variables used in the executable part.

Ex:- int a,b;

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 "_"(underscore)
- 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:

- It tells the compiler what the variable name is.
- 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;
        long int 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.

1byte = 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 represented in two parts. The first part present before 'e' is called **Mantissa** and the part following 'e'is called **Exponent**. **For ex.** 0.000342 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!”’ ”979” ”welcome”’ “ !1”’ “5+3” “X”

Rules for constructing integer constants:

- An integer constant must have at least one digit.
- It must not have a decimal point.
- It can be either positive or negative.

- 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 real constant must have at least one digit
- It must have a decimal point.
- It could be either positive or negative.
- 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 character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas.
- 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:

- What appears on the left-hand side need not be repeated and therefore it becomes easier to write
- The statement is more concise and easier to read
- 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:

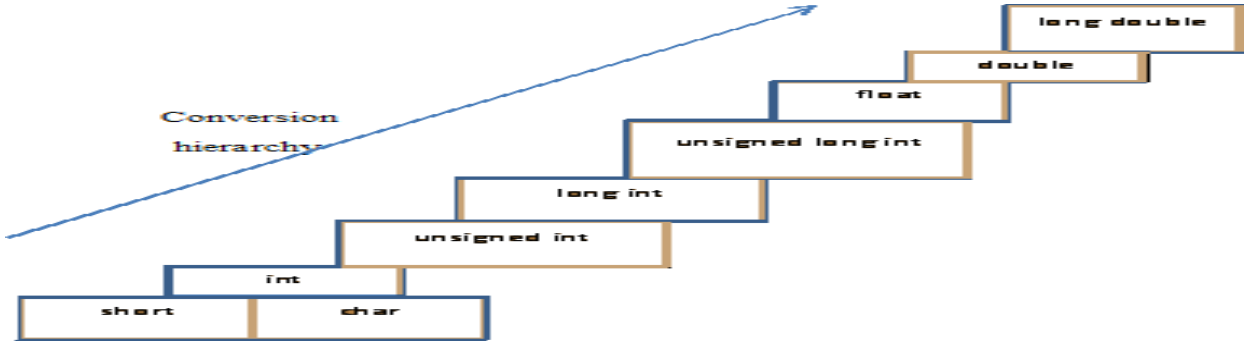


Fig: Conversion hierarchy

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

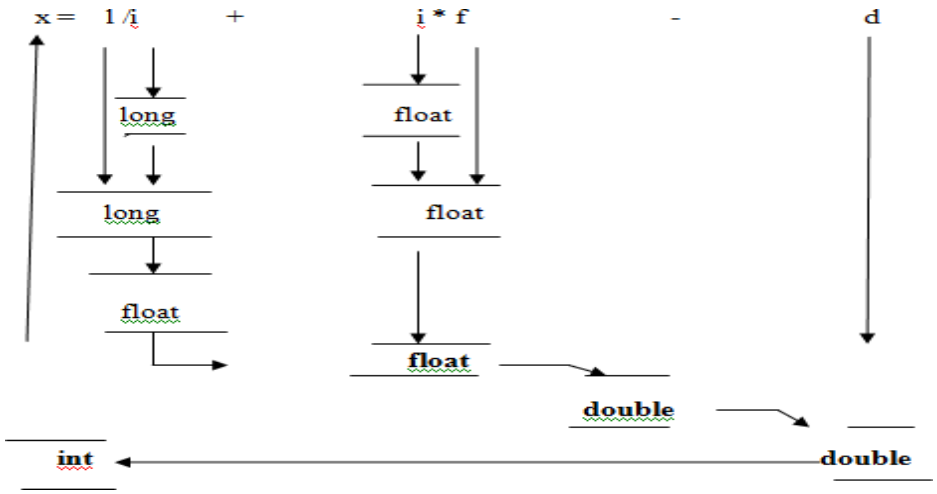
- All **short** and **char** are automatically converted into **int**
- if one of the operands is **long double** the other will be converted to **long double** and the result will be **long double**.
- if one of the operand is **double,**the other will be converted to **double** and the result will be **double**.
- if one of the operand is **float** ,the other will be converted to **float** and the result will be **float**.
- 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.**
- if one of the operand is **long int**, the other is **unsigned int** then **unsigned int** can be converted into **long int**, the **unsigned int** operand will be converted as such and the result will be **long int.**
- else both operands will be converted **to unsigned long int** and the result will be unsigned long int.
- else if one of the operand **is long int** ,the other will be converted into **long int** and the result will be **long int.**
- 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; long int 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,

- If an expression has parenthesis, sub expression within the parenthesis is evaluated first and arithmetic expression without parenthesis is evaluated first.
- The operators of high level precedence are evaluated first.
- 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:

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

Hierarchy of operators in C

There are some operators 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

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

26. What are different types of 'if' statements available in c? Explain them. (Or)

Describe all the variants of if-else statement with clear syntax and examples.

Ans: There are 4 if statements available in C:

1. Simple if statement.
2. if…else statement.
3. Nested if…else statement.
4. else if ladder

1. Simple if statement: Simple if statement is used to make a decision based on the available choice. It has the following form:

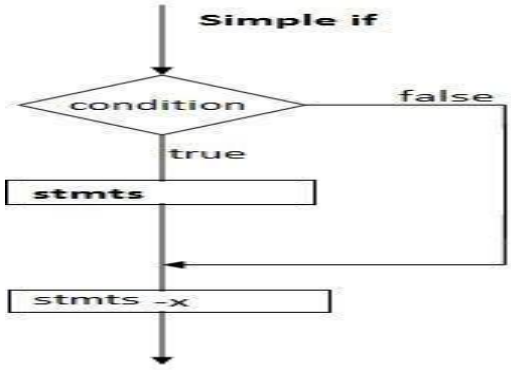
Syntax:

```
if ( condition )
{
        stmt block;
}
stmt-x;
```

In this syntax,

- if is the keyword. <*condition*>is a relational expression or logical expression or any expression that returns either true or false. It is important to note that the condition should be enclosed within parentheses '(' and ') '.
- The *stmt block* can be a simple statement or a compound statement or a null statement.
- *stmt-x* is any valid C statement.

The flow of control using simple if statement is determined as follows:



Whenever simple if statement is encountered, first the condition is tested. It returns either true or false. If the condition is false, the control transfers directly to stmt-x without considering the stmt block. If the condition is true, the control enters into the stmt block. Once, the end of stmt block is reached, the control transfers to stmt-x

Program for if statement:

```
#include<stdio.h>
int main()
{
    int age;
    printf("enter age\n");
    scanf("%d",&age);
    if(age>=55)
        printf("person is retired\n");
}
```

Output: enter age 57 person is retired

2. if—else statement

if…else statement is used to make a decision based on two choices.

It has the following form:

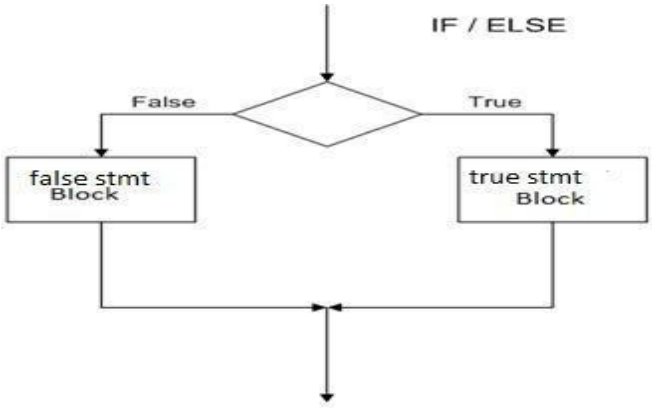
Syntax:

```
if(condition)
{
    true stmt block;
}
else
{
    false stmt block;
}
stmt-x;
```

In this syntax,

- if and else are the keywords.
- <*condition*>is a relational expression or logical expression or any expression that returns either true or false. It is important to note that the condition should be enclosed within parentheses (and).
- The *true stmt block* and false stmt block are simple statements or compound statements or null statements.
- *stmt-x* is any valid C statement.

The flow of control using if...else statement is determined as follows:



Whenever if...else statement is encountered, first the condition is tested. It returns either true or false. If the condition is true, the control enters into the true stmt block. Once, the end of true stmt block is reached, the control transfers to stmt-x without considering else-body.

If the condition is false, the control enters into the false stmt block by skipping true stmt block. Once, the end of false stmt block is reached, the control transfers to stmt-x.

Program for if else statement:

```
#include<stdio.h>
int main()
{
    int age;
    printf("enter age\n");
    scanf("%d",&age);
    if(age>=55)
        printf("person is retired\n");
    else
        printf("person is not retired\n");
}
```

Output: enter age 47 person is not retired

3. Nested if—else statement

Nested if…else statement is one of the conditional control-flow statements. If the body of if statement contains at least one if statement, then that if statement is called as ―Nested if…else statement‖. The nested if…else statement can be used in such a situation where at least two conditions should be satisfied in order to execute particular set of instructions. It can also be used to make a decision among multiple choices.

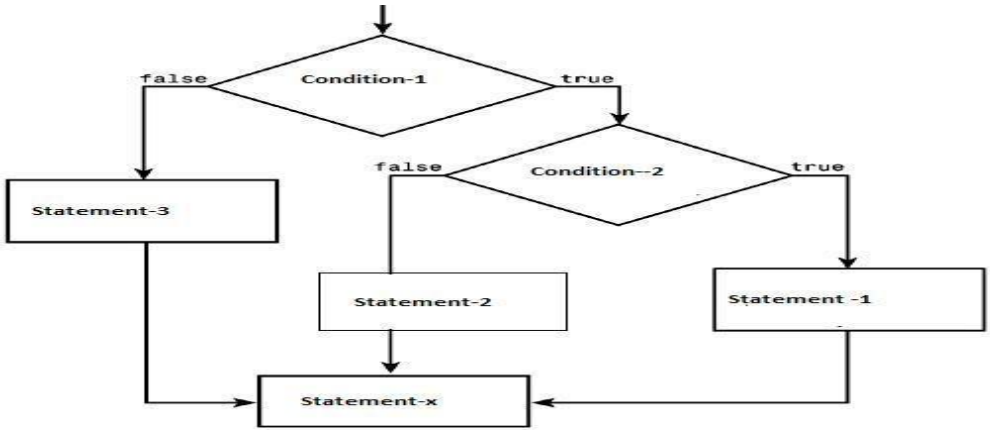
The nested if…else statement has the following form:

Syntax:

```
if(condition1)
{
        if(condition 2)
        {
                stmt1;
        }
        else
        {
                stmt2;
        }
}
else
{
        stmt 3;
}
stmt-x;
```

In this syntax,

- if and else are keywords.
- *<condition1>,<condition2> … <condition>*are relational expressions or logical expressions or any other expressions that return true or false. It is important to note that the condition should be enclosed within parentheses (and).
- *if-body* and *else-body* are simple statements or compound statements or empty statements.
- *stmt-x* is a valid C statement.



The flow of control using Nested if...else statement is determined as follows:

Whenever nested if…else statement is encountered, first <condition1> is tested. It returns either true or false.

If condition1 (or outer condition) is false, then the control transfers to else-body (if exists) by skipping if-body.

If condition1 (or outer condition) is true, then condition2 (or inner condition) is tested. If the condition2 is true, if-body gets executed. Otherwise, the else-body that is inside of if statement gets executed.

Program for nested if… else statement:

```
#include<stdio.h>
void main()
{
    int a ,b,c;
    printf("enter three values\n");
    scanf("%d%d%d",&a,&b,&c);
    if(a>b)
    {
        if(a>c)
            printf("%d\n",a);
        else
            printf("%d\n",c);
    }
    else
    {
```

```
        if(b>c)
                printf("%d\n",b);
        else
                printf("%d\n",b);
    }c
}
```

Output: enter three values 3 2 4

4

4. else—if Ladder

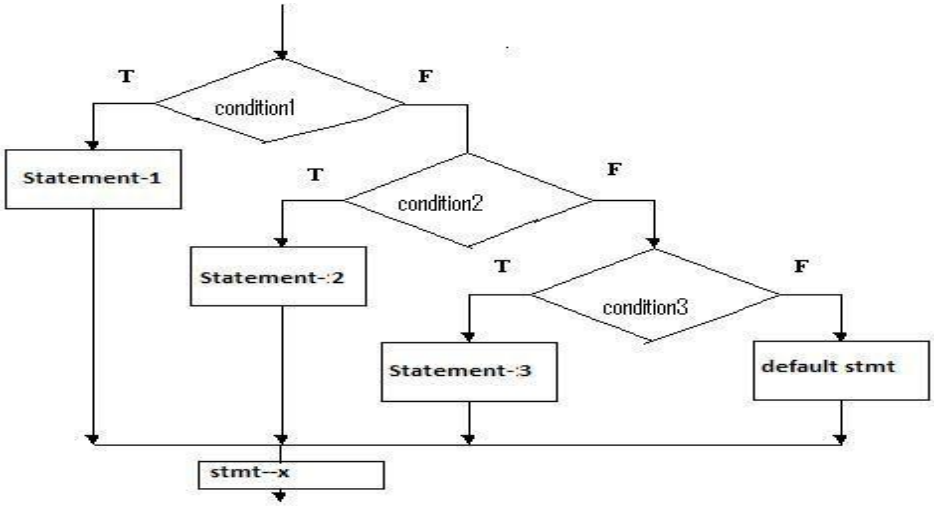
else-if ladder is one of the conditional control-flow statements. It is used to make a decision among multiple choices. It has the following form:

Syntax:

```
if(condition 1)
 {
      statement 1;
 }
 else if(condition 2)
 {
      statement 2;
 }
 else if(condition 3)
 {
        statement 3;
 }
 else if(condition n)
{
      statement n;
 }
 else
{
      default statement;
}
 stmt- x;
```

In this syntax,

- if and else are keywords. There should be a space between else and if, if they come together.
- *<condition1>,<condition2>....<condtionN>*are relational expressions or logical expressions or any other expressions that return either true or false. It is important to note that the condition should be enclosed within parentheses (and).
- *statement 1, statement 2,statement 3…….,statement n* and *default statement* are either simple statements or compound statements or null statements.
- *stmt-x* is a valid C statement.



The flow of control using else--- if ladder statement is determined as follows:

Whenever else if ladder is encountered, condition1 is tested first. If it is true, the statement 1 gets executed. After then the control transfers to *stmt-x*.

If *condition1* is false, then *condition2* is tested. If *condition2* is false, the other conditions are tested. If all are false, the default stmt at the end gets executed. After then the control transfers to *stmt-x*.

If any one of all conditions is true, then the body associated with it gets executed. After then the control transfers to *stmt-x*.

Example Program:

Program for the else if ladder statement:

```
#include<stdio.h>
void main()
{
    int m1,m2,m3,avg,tot;
    printf("enter three subject marks");
    scanf("%d%d%d", &m1,&m2,&m3);
```

```
        tot=m1+m2+m3;
        avg=tot/3;
        if(avg>=75)
        {
                printf("distinction");
        }
        else if(avg>=60 &&avg<75)
        {
                printf("first class");
        }
        else if(avg>=50 &&avg<60)
        {
                printf("second class");
        }
        else if (avg<50)
        {
                printf("fail");
        }
}
```

Output: enter three subject marks85 80 81

Distinction

27. Explain the switch statement with Example program.

Ans: switch statement is one of decision-making control-flow statements. Just like else if ladder, it is also used to make a decision among multiple choices. switch statement has the following form:

Syntax:

```
        switch(<exp>)
        {
                case<exp-val-1>: statements block-1
                                        break;
                case<exp-val-2>: statements block-2
                                        break;
                case<exp-val-3>: statements block-3
                                        break;
```

case<exp-val-N>: statements block-N

break;

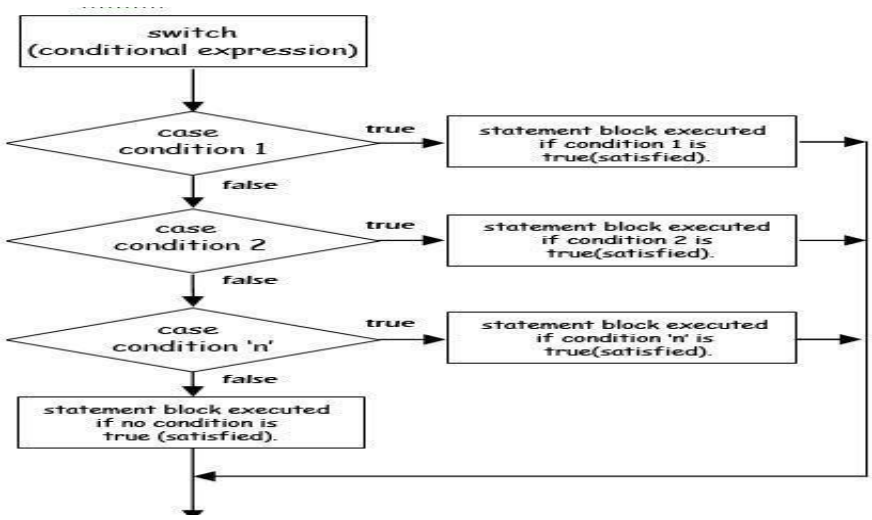
default: default statements block

}

Next-statement;

In this syntax,

- switch, case, default and break are keywords.
- <*exp*>is any expression that should give an integer value or character value. In other words, it should never return any floating-point value. It should always be enclosed with in parentheses (and). It should also be placed after the keyword switch.
- <*exp-val-1*>, <*exp-val-2*>, <*exp-val-3*>…. <*exp-val-N*> should always be integer constants or character constants or constant expressions. In other words, variables can never be used as <*exp- val*>. There should be a space between the keyword case and <*exp-val*>. The keyword case along with its <*exp-val*> is called as a case label. <*exp-val*> should always be unique; no duplications are allowed.
- *statements block-1, statements block-2, statements block-3… statements block-N* and *default statements block* are simple statements, compound statements or null statements. It is important to note that the statements blocks along with their own case labels should be separated with a colon (:)
- The break statement at the end of each statements block is an optional one. It is recommended that break statement always be placed at the end of each statements block. With its absence, all the statements blocks below the matched case label along with statements block of matched case get executed. Usually, the result is unwanted.
- The statement block and break statement can be enclosed with in a pair of curly braces { and }.
- The default along with its statements block is an optional one. The break statement can be placed at the end of default statements block. The default statements block can be placed at anywhere in the switch statement. If they are placed at any other place other than at end, it is compulsory to include a break statement at the end of default statements block.
- *Next-statement* is a valid C statement.



Whenever, switch statement is encountered, first the value of <*exp*> gets matched with case values. If suitable match is found, the statements block related to that matched case gets executed. The break statement at the end transfers the control to the *Next-statement*.

If suitable match is not found, the default statements block gets executed and then the control gets transferred to *Next-statement*.

Example Program:

```
#include<stdio.h>
void main()
{
    int a,b,c,ch;
    printf("\nEnter two numbers :");
    scanf("%d%d",&a,&b);
    printf("\nEnter the choice:");
    scanf("%d",&ch);
    switch(ch)
    {
        case 1: c=a+b;
                break;
        case 2: c=a-b;
                break;
        case 3: c=a*b;
```

```
                break;
        case 4: c=a/b;
                break;
        case 5:
                return;
    }
    printf("\nThe result is: %d",c);
}
```

Output: enter the choice 1 Enter two numbers 4 2 The Result is: 6

28. Write in detail about different types of loop statements in C.

Ans:

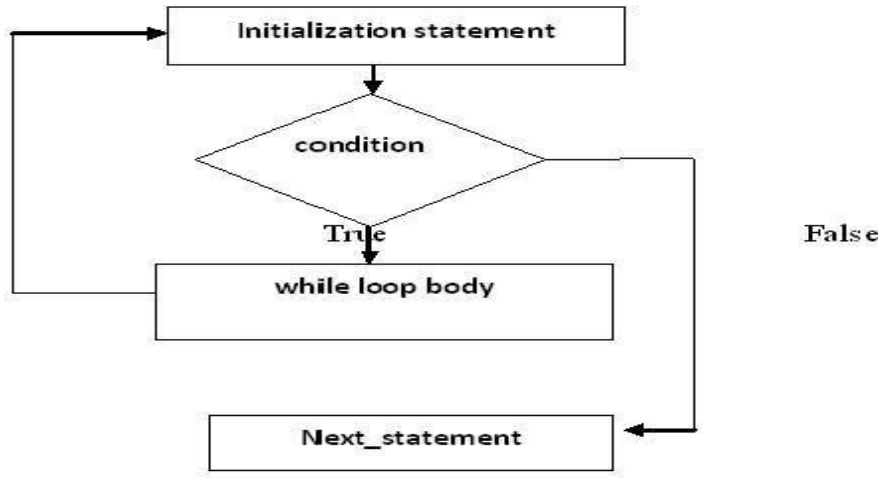
While: The simplest of all the looping structures in c is the while statement. The basic format of the while statement is:

Syntax:

```
Initialization statement;
while(condition)
{
    statements;
}
```

The while is an entry –controlled loop statement. The condition is evaluated and if the condition is true then the statements will be executed. After execution of the statements the condition will be evaluated and if it is true the statements will be executed once again. This process is repeated until the condition becomes false and the control is transferred out of the loop .On exit the program continues with the statement immediately after the body of the loop.

Flow chart:



Program to print n natural numbers using using while

```
#include<stdio.h>
 int main()
{
        int i,n;
        printf("enter the range\n");
        scanf("%d",&n);
        i=1;
        while(i<=n)
        {
                printf("%d ",i);
                i=i+1;
        }
}
```

Output: enter the range 10

1 2 3 4 5 6 7 8 9 10

do-while statement: It is one of the looping control statements. It is also called as *exit- controlled looping control statement*. i.e., it tests the condition after executing the do-while loop body.

The main difference between ―while‖ and ―do-while‖ is that in ―do-while‖ statement, the loop body gets executed at least once, though the condition returns the value false for the first time, which is not possible with while statement. In ―while‖ statement, the control enters into the loop body only when the condition returns true.

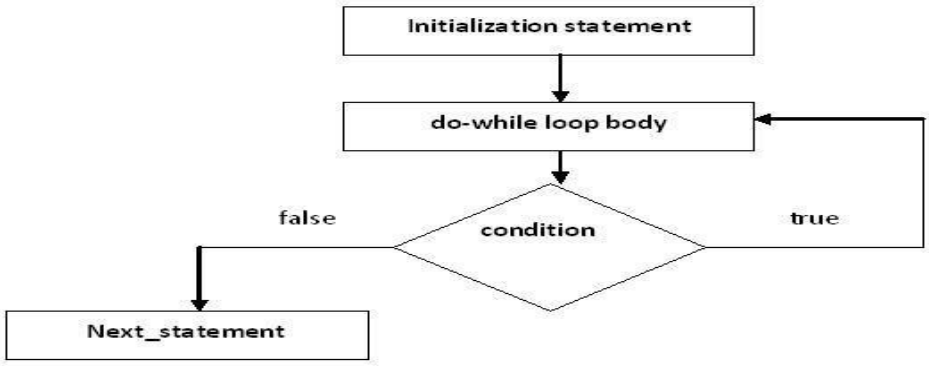
Syntax:

```
        Initialization statement;
        do
        {
                statement(s);
        } while(<condition>);
        next statement;
```

In this syntax:

while and **do** are the keywords. <condition> is a relational expression or a compound relational

expression or any expression that returns either true or false. initialization statement, statement(s) and next_statement are valid ‘c‘ statements. The statements within the curly braces are called as do-while loop body. The updating statement should be included with in the do-while loop body. There should be a semi-colon (;) at the end of while(<condition>).



Whenever “do-while” statement is encountered, the initialization statement gets executed first. After then, the control enters into do-while loop body and all the statements in that body will be executed. When the end of the body is reached, the condition is tested again with the updated loop counter value. If the condition returns the value false, the control transfers to next statement without executing do-while loop body. Hence, it states that, the do-while loop body gets executed for the first time, though the condition returns the value false.

Program to print n natural numbers using using do while

```
#include<stdio.h>
int main()
{
    int i,n;
    printf("enter the range\n");
    scanf("%d",&n);
    i=1;
    do
    {
        printf("%d\n",i);
        i=i+1;
    }while(i<=n);
}
```

Output: enter the range 8 1 2 3 4 5 6 7 8

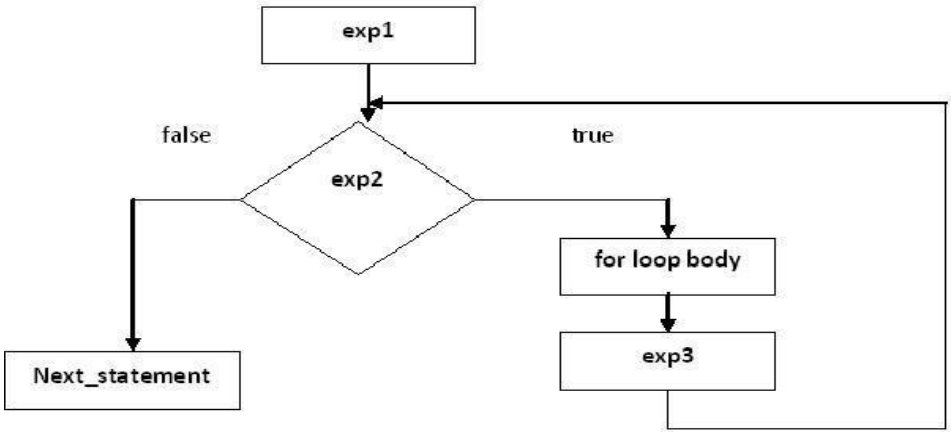
for statement:

It is one of the looping control statements. It is also called as *entry-controlled looping control statement*. i.e., it tests the condition before entering into the loop body. The syntax for ”for” statement is as follows:

Syntax:

```
for(exp1;exp2;exp3)
{
        for-body;
}
next_statement;
```

In this syntax, for is a keyword. exp1 is the initialization statement. If there is more than one statement, then the statements must be separated with commas. exp2 is the condition. It is a relational expression or a compound relational expression or any expression that returns either true or false. The exp3 is the updating statement. If there is more than one statement then, they must be separated with commas. exp1, exp2 and exp3 should be separated with two semi-colons. exp1, exp2, exp3, for- body and next_statement are valid “c‘ statements. for-body is a simple statement or compound statement or a null statement.



Whenever ”for” statement is encountered, first exp1 gets executed. After then, exp2 is tested. If exp2 is true then the body of the loop will be executed otherwise loop will be terminated.

When the body of the loop is executed the control is transferred back to the for statement after evaluating the last statement in the loop. Now exp3 will be evaluated and the new value is again tested .if it satisfies body of the loop is executed .This process continues till condition is false.

Program to print n natural numbers using for loop

```
#include<stdio.h>
 void main()
{
        int i,n;
        printf("enter the value");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                printf("%d ",i);
        }
}
```

Output: enter the value 5

1 2 3 4 5

29. Explain Jumping control-flow statements. (Or)

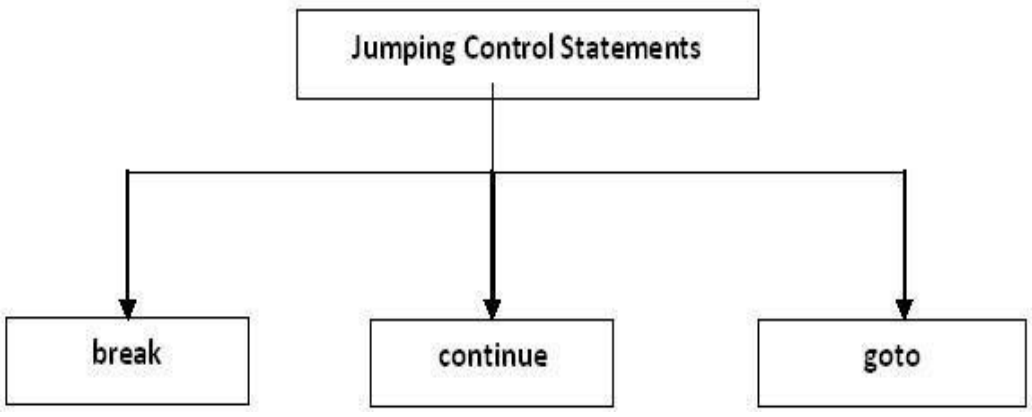
Differentiate between the following with example?

- **Break statement**
- **Continue statement**
- **Goto statement (Or)**

What are the differences between break statement and Continue statement?

Ans:

Jumping control-flow statements are the control-flow statements that transfer the control to the specified location or out of the loop or to the beginning of the loop. There are 3 jumping control statements:



1. break statement

The "break" statement is used within the looping control statements, switch statement and nested loops. When it is used with the for, while or do-while statements, the control comes out of the corresponding loop and continues with the next statement.

When it is used in the nested loop or switch statement, the control comes out of that loop / switch statement within which it is used. But, it does not come out of the complete nesting.

Syntax for the "break" statement is:

```
break;
```

In this syntax, break is the keyword.

The following representation shows the transfer of control when break statement is used:

```
Any loop
{
        statement_1;
        statement_2;
                break;
}
next_statement;
```

Program for break statement:

```
#include<stdio.h>
int main()
{
    int i;
    for(i=1; i<=10; i++)
    {
        if(i==6)
            break;
        printf("%d",i);
    }
}
```

Output: 12345

2. Continue statement

A continue statement is used within loops to end the execution of the current iteration and proceed to the next iteration. It provides a way of skipping the remaining statements in that iteration after the continue statement. It is important to note that a continue statement should be used only in loop constructs and not in selective control statements.

Syntax for continue statement is:

```
continue;
```

where continue is the keyword.

The following representation shows the transfer of control when continue statement is used:

```
Any loop
{
        statement_1;
        statement_2;
                continue;
}
next_statement;
```

Program for continue statement:

```
#include<stdio.h>
int main()
{
        int i, sum=0, n;
        for(i=1; i<=10; i++)
                {
                        printf("enter any no:");
                        scanf("%d",&n);
                        if(n<0)
                                continue;
                        else
                                sum=sum+n;
                        printf("%d\n",sum);
                }
```

} **Output:** enter any no:8 18

3. goto statement

The goto statement transfers the control to the specified location unconditionally. There are certain situations where goto statement makes the program simpler. For example, if a deeply nested loop is to be exited earlier, goto may be used for breaking more than one loop at a time. In this case, a break statement will not serve the purpose because it only exits a single loop.

Syntax for goto statement is:

```
label:
{
    statement_1;
    statement_2;
}
goto label;
```

In this syntax, goto is the keyword and label is any valid identifier and should be ended with a colon (:).

The identifier following goto is a statement label and need not be declared. The name of the statement or label can also be used as a variable name in the same program if it is declared appropriately. The compiler identifies the name as a label if it appears in a goto statement and as a variable if it appears in an expression.

If the block of statements that has label appears before the goto statement, then the control has to move to backward and that goto is called as backward goto. If the block of statements that has label appears after the goto statement, then the control has to move to forward and that goto is called as forward goto.

Program for goto statement:

```
#include<stdio.h>
void main()
{
    printf("www.");
    goto x;
    y:
        printf("expert");
```

```
    goto z;
    x:
    printf("c programming");
    goto y;
    z:
    printf(".com");
}
```

Output: www.c programming expert.com

30. What are the differences between while and do-while statements?

Ans:

The main difference between the while and do-while loop is in the place where the condition is to be tested.

In the while loops the condition is tested following the while statement then the body gets executed. Where as in do-while, the condition is checked at the end of the loop. The do-while loop will execute at least one time even if the condition is false initially. The do-while loop executes until the condition becomes false.

While	do-while
1) It is an entry-controlled control-flow statement. That is, it tests condition before the body gets executed.	1) It is an exit-controlled control-flow statement. That is, it tests condition after the body gets executed.
2) The body gets executed only when the condition is true.	2) The body gets executed at least once though the condition is false for the first time.

31. Explain the formatted I/O functions with example?

Explain unformatted (character oriented) I/O functions with example?

Ans: Managing input and output operations:

Reading, processing and writing of data are the three essential functions of a computer program. Most programs take some data as input and display the processed data. We have two methods of providing data to the program variables. One method is to assign values to variables through the assignment statements like x=5, a=0 and so on. Another method is to use the input function scanf, which can read data from a keyboard. We have used both the methods in programs. For outputting results, we have used extensively the function printf, which sends results out to a terminal.

Input – Output functions:-

The program takes some I/P- data, process it and gives the O/P.

- We have two methods for providing data to the program
  - Assigning the data to the variables in a program.
  - By using I/P-O/P statements.

C language has 2 types of I/O statements; all these operations are carried out through function calls.

- Unformatted I/O statements
- Formatted I/O statements

Unformatted I/O statements:-

getchar( ):- It reads single character from standard input device. This function don‘t require any arguments.

Syntax:- char variable_name = getchar( );

Ex:- char x;

x = getchar( );

putchar ( ):- This function is used to display one character at a time on the standard output device.

Syntax:- putchar(char_variable);

Ex:- char x;

putchar(x);

program:

```
main( )
{
    char ch;
```

```
        printf("enter a char");

        ch = getchar( );
        printf("entered char is");
        putchar(ch);
}
```

Output: enter a char a

entered char is a

getc() :- This function is used to accept single character from the file.

Syntax: char variable_name = getc();

Ex:-char c;

c = getc();

putc():- This function is used to display single character.

Syntax:- putc(char variable_name);

Ex:- char c;

putc(c);

These functions are used in file processing.

gets( ):- This function is used to read group of characters(string) from the standard I/P device.

Syntax:- gets(character array variable);

Ex:- gets(s);

puts( ):- This function is used to display string to the standard O/P device.

Syntax:- puts(character array variables);

Ex:- puts(s);

program:

```
main()
{
char s[10]; puts("enter name");
gets(s);
puts("print name");
puts(s);
}
```

Output: enter name ramu

print name ramu

getch():- This function reads a single character directly from the keyboard without displaying on the screen. This function is used at the end of the program for displaying the output (without pressing (Alt-F5).

Syntax: char variable_name = getch();

Ex:- char c

c = getch();

getche():- This function reads a single character from the keyboard and echoes(displays) it to the current text window.

Syntax:- char variable_name = getche();

Ex:- char c

c = getche();

program:

```
main()
{
char ch, c;
printf(“enter char”);
ch = getch();
printf(“%c”, ch);
printf(“enter char”);
c = getche();
printf(“%c”,c);
}
```

Output :- enter character a

enter character b

b

Character test functions:-

Function	Test
isalnum(c)	Is c an alphanumeric character?
isalpha(c)	Is c an alphabetic character
isdigit(c)	Is c a digit?
islower(c)	Is c alower case letter?
isprint(c)	Is c a character?
ispunct(c)	Is c a punctuation mark?
isspace(c)	Is c a white space character?
isupper(c)	Is c an upper case letter?
tolower(c)	Convert ch to lower case
toupper(c)	Convert ch to upper case

program:

```
main()
{
        char a;
        printf("enter char");
        a = getchar();
        if (isupper(a))
        {
                x= tolower(a);
                putchar(x);
        }
        else
                putchar(toupper(a));
}
```

Output:- enter char A

a

Formatted I/O Functions: Formatted I/O refers to input and output that has been arranged in a particular format.

Formatted I/P functions-→ scanf( ) , fscanf()

Formatted O/P functions-→ printf() , fprintf()

scanf( ) :-scanf() function is used to read information from the standard I/P device.

Syntax:-scanf("controlstring", &variable_name);

Control string (also known as format string) represents the type of data that the user is going to accept and gives the address of variable. (char-%c , int-%d , float - %f , double-%lf).Control string and variables are separated by commas. Control string and the variables going to I/P should match with each other.

Ex:- int n;

scanf("%d",&n);

Inputting Integer Numbers:

The field specification for reading an integer number is:

%w d

The percentage sign(%) indicates that a conversion specification follows. w is an integer number that specifies the field width of the number to be read and d known as data type character indicates that the number to be read is in integer mode.

Ex:- scanf("%2d,%5d",&a&b);

The following data are entered at the console:

50 31426

Here the value 50 is assigned to **a** and 31426 to **b Inputting Real Numbers:**

The field width of real numbers is not to be specified unlike integers. Therefore scanf reads real numbers using the simple specification **%f.**

Ex: scanf("%f %f",&x,&y);

Suppose the following data are entered as input:

23.45 34.5

The value 23.45 is assigned to **x** and 34.5 is assigned to **y**. **Inputting character strings:**

The field specification for reading character strings is

%ws or %wc

%c may be used to read a single character.

Ex: scanf("%s",name1);

Suppose the following data is entered as input:

Griet

Griet is assigned to name1.

Formatted Output:

printf( ): This function is used to output any combination of data. The outputs are produced in such a way that they are understandable and are in an easy to use form. It is necessary for the programmer to give clarity of the output produced by his program.

Syntax:-printf("control string", var1,var2……);

Control string consists of 3 types of items

- Characters that will be printed on the screen as they appear.
- Format specifications that define the O/P format for display of each item.
- Escape sequence chars such as \n , \t and \b…..

The control string indicates how many arguments follow and what their types are.

The var1, var2 etc.. are variables whose values are formatted and printed according to the specifications of the control string.

The arguments should match in number, order and type with the format specifications.

O/P of integer number: - Format specification :**%wd**

Format	**O/P**
	9 8 7 6
printf("%d", 9876)	
printf("%6d",9876)	9 8 7 6
printf("%2d",9876)	9 8
printf("%-6d",9876)	9 8 7 6
printf("%06d",9876)	0 0 9 8 7 6

O/P of real number: %w.pf

w ---------- -Indicates minimum number of positions that are to be used for display of the value.

p---------- Indicates number of digits to be displayed after the decimal point.

Format	**y=98.7654**	**O/P**
printf("%7.4f",y)		9 8 . 7 6 5 4
printf("%7.2f",y)		9 8 . 7 7
printf("%-7.2f",y)		9 8 . 7 7

printf("%10.2e",y)

		9	.	8	8	e	+	0	1

O/P of single characters and string:

%wc %ws

Format O/P

%s

R	a	J	U		R	a	j	e	s	h		R	a	N	i

%18s

	R	a	J	U		R	a	J	E	s	h		R	a	n	i

%18s

R	A	j	U		R	a	j	E	S	h		R	a	n	i	

32. Explain the command line arguments. What are the syntactic constructs followed in "C"?

Ans :

Command line argument is the parameter supplied to a program when the program is invoked. This parameter may represent a file name the program should process. For example, if we want to execute a program to copy the contents of a file named X_FILE to another one name Y_FILE then we may use a command line like

C:> program X_FILE Y_FILE Program is the file name where the executable code of the program is stored. This eliminates the need for the program to request the user to enter the file names during execution. The „main" function can take two arguments called argc, argv and information contained in the command line is passed on to the program to these arguments, when "main" is called up by the system. The variable **argv** is an argument vector and represents an array of characters pointers that point to the command line arguments. The **argc** is an argument counter that counts the number of arguments on the command line. The size of this array is equal to the value of argc. In order to access the command line arguments, we must declare the "main" function and its parameters as follows:

```
main(argc,argv)
int argc;
char *argv[ ];
{ }
```

Generally argv[0] represents the program name.

Example:-

A program to copy the contents of one file into another using command line arguments.

```
#include<stdio.h>
#include<stdlib.h>
void main(int argc,char* argv[]) /* command line arguments */
{
        FILE *ft,*fs; /* file pointers declaration*/
        int c=0;
        if(argc!=3)
        printf("\n insufficient arguments");
        fs=fopen(argv[1],"r");
        ft=fopen(argv[2],"w");
        if (fs==NULL)
        {
                printf("\n source file opening error");
                exit(1) ;
        }
        if (ft==NULL)
        {
                printf("\n target file opening error");
                exit(1) ;
        }
        while(!feof(fs))
        {
                fputc(fgetc(fs),ft);
                c++;
        }
        printf("\n bytes copied from %s file to %s file =%d", argv[1], argv[2], c);
        c=fcloseall(); /*closing all files*/
        printf("files closed=%d",c);
}
```

33. Write a c program to add two numbers using command line arguments?

Ans: Program:

```
#include<stdio.h>
int main(int argc, char *argv[])
{
        int x, sum=0;
        printf("\n Number of arguments are:%d", argc);
        printf("\n The agruments are:");
        for(x=0;x<argc; x++)
        {
                printf("\n agrv[%d]=%s", x, argv[x]);
                if(x<2)
                        continue;
                else
                        sum=sum+atoi(argv[x]);
        }
        printf("\n program name:%s",argv[0]);
        printf("\n name is:%s",argv[1]);
        printf("\n sum is:%d",sum);
        return(0);
}
```

34. What are different types of storage classes in 'C'? (or)

Explain briefly auto and static storage classes with examples?(or)

Explain extern and register storage classes with example programs.

Ans: Storage classes in 'C'

Variables in C differ in behavior. The behavior depends on the storage class a variable may assume. From C compiler"s point of view, a variable name identifies some physical location within the computer where the string of bits representing the variable"s value is stored. There are four storage classes in C:

(a) Automatic storage class
(b) Register storage class
(c) Static storage class
(d) External storage class

(a) Automatic Storage Class:-

The features of a variable defined to have an automatic storage class are as under:

Keyword	Auto
Storage	Memory.
Default initial value	An unpredictable value, which is often called a garbage value.
Scope	Local to the block in which the variable is defined.
Life	Till the control remains within the block in which the variable is defined

Following program shows how an automatic storage class variable is declared, and the fact that if the variable is not initialized it contains a garbage value.

```
main( )
{
   auto int i, j ;
   printf ( "\n%d %d", i, j ) ;
}
```

When you run this program you may get different values, since garbage values are unpredictable. So always make it a point that you initialize the automatic variables properly, otherwise you are likely to get unexpected results. Scope and life of an automatic variable is illustrated in the following program.

```
main( )
{
   auto int i = 1 ;
   {
         auto int i = 2 ;
         {
               auto int i = 3 ;
               printf ( "\n%d ", i ) ;
         }
         printf ( "%d ", i ) ;
   }
   printf ( "%d", i ) ;
}
```

Static Storage Class:-

The features of a variable defined to have a **static** storage class are as under:

Keyword	Static
Storage	Memory.
Default initial value	Zero.
Scope	Local to the block in which the variable is defined.
Life	Value of the variable persists between different function calls

The following program demonstrates the details of static storage class:

```
main( )
{
     increment( ) ;
     increment( ) ;
     increment( ) ;
}

increment( )
{
     auto int i = 1 ;
     printf ( "%d\n", i ) ;
     i = i + 1 ;
}
```

```
main( )
{
     increment( ) ;
     increment( ) ;
     increment( ) ;
}

increment( )
{
     static int i = 1 ;
     printf ( "%d\n", i ) ;
     i = i + 1 ;
}
```

The output of the above programs would be:

```
1
1
1
```

```
1
2
3
```

Extern Storage Class:-

The features of a variable whose storage class has been defined as external are as follows:

Keyword	Extern
Storage	Memory
default initial value	Zero
Scope	Global
Life	As long as the program execution does not come to end

External variables differ from those we have already discussed in that their scope is global, not local. External variables are declared outside all functions, yet are available to all functions that care to use them. Here is an example to illustrate this fact.

Ex:

```
#include<stdio.h
> extern int i;
void main()
{
   printf("i=%d",i);
}
```

Register Storage Class:-

The features of a variable defined to be of **register** storage class are as under:

Keyword	Register
Storage	CPU Registers
default initial value	An unpredictable value, which is often called a garbage value.
Scope	Local to the block in which the variable is defined.
Life	Till the control remains within the block in which the variable isdefined.

A value stored in a CPU register can always be accessed faster than the one that is stored in memory. Therefore, if a variable is used at many places in a program it is better to declare its storage class as register. A good example of frequently used variables is loop counters. We can name their storage class as register.

```
main( )
{
   register int i ;
   for ( i = 1 ; i <= 10 ; i++ )
   printf ( "\n%d", i ) ;
}
```

35. Enumerate the scope rules in C

Ans:

Scope: Scope of a variable is the part of program in which it can be used

Scope rules

The rules are as under:

- Use **static** storage class only if you want the value of a variable to persist between different function calls.
- Use **register** storage class for only those variables that are being used very often in a program. Reason is, there are very few CPU registers at our disposal and many of them might be busy doing something else. Make careful utilization of the scarce resources. A typical application of **register** storage class is loop counters, which get used a number of times in a program.
- Use **extern** storage class for only those variables that are being used by almost all the functions in the program. This would avoid unnecessary passing of these variables as arguments when making a function call. Declaring all the variables as **extern** would amount to a lot of wastage of memory space because these variables would remain active throughout the life of the program.
- If we don‟t have any of the express needs mentioned above, then use the **auto** storage class. In fact most of the times we end up using the **auto** variables, because often it so happens that once we have used the variables in a function we don’t mind losing them.

36. Explain the computer number system in detail.

Ans:

When we type some letters or words, the computer translates them in numbers as computers can understand only numbers. A computer can understand the positional number system where there are only a few symbols called digits and these symbols represent different values depending on the position they occupy in the number.

The value of each digit in a number can be determined using −

- The digit
- The position of the digit in the number
- The base of the number system (where the base is defined as the total number of digits available in the number system)

Decimal Number System

The number system that we use in our day-to-day life is the decimal number system. Decimal number system has base 10 as it uses 10 digits from 0 to 9. In decimal number system, the successive positions to the left of the decimal point represent units, tens, hundreds, thousands, and so on.

Each position represents a specific power of the base (10). For example, the decimal number 1234 consists of the digit 4 in the units position, 3 in the tens position, 2 in the hundreds position, and 1 in the thousands position. Its value can be written as

(1 x 1000)+ (2 x 100)+ (3 x 10)+ (4 x l)
$(1 \times 10^3)+ (2 \times 10^2)+ (3 \times 10^1)+ (4 \times l0^0)$
1000 + 200 + 30 + 4
1234

As a computer programmer or an IT professional, you should understand the following number systems which are frequently used in computers.

S.No.	Number System and Description
1	**Binary Number System** Base 2. Digits used : 0, 1
2	**Octal Number System** Base 8. Digits used : 0 to 7
3	**Hexa Decimal Number System** Base 16. Digits used: 0 to 9, Letters used : A- F

Binary Number System

Characteristics of the binary number system are as follows −

- Uses two digits, 0 and 1
- Also called as base 2 number system
- Each position in a binary number represents a **0** power of the base (2). Example 2^0
- Last position in a binary number represents a **x** power of the base (2). Example 2^x where **x** represents the last position - 1.

Example

Binary Number: 10101_2

Calculating Decimal Equivalent −

Step	Binary Number	Decimal Number
Step 1	10101_2	$((1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2	10101_2	$(16 + 0 + 4 + 0 + 1)_{10}$
Step 3	10101_2	21_{10}

Note − 10101_2 is normally written as 10101.

Octal Number System

Characteristics of the octal number system are as follows −

- Uses eight digits, 0,1,2,3,4,5,6,7
- Also called as base 8 number system
- Each position in an octal number represents a **0** power of the base (8). Example 8^0
- Last position in an octal number represents a **x** power of the base (8).
    - Example 8^x where **x** represents the last position - 1

Example

Octal Number: 12570_8

Calculating Decimal Equivalent −

Step	Octal Number	Decimal Number
Step 1	12570_8	$((1 \times 8^4) + (2 \times 8^3) + (5 \times 8^2) + (7 \times 8^1) + (0 \times 8^0))_{10}$
Step 2	12570_8	$(4096 + 1024 + 320 + 56 + 0)_{10}$
Step 3	12570_8	5496_{10}

Note − 12570_8 is normally written as 12570.

Hexadecimal Number System

Characteristics of hexadecimal number system are as follows −

- Uses 10 digits and 6 letters, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Letters represent the numbers starting from 10. A = 10. B = 11, C = 12, D = 13, E = 14, F = 15
- Also called as base 16 number system
- Each position in a hexadecimal number represents a **0** power of the base (16). Example, 16^0
- Last position in a hexadecimal number represents a **x** power of the base (16). Example 16^x where **x** represents the last position - 1

Example

Hexadecimal Number: $19FDE_{16}$

Calculating Decimal Equivalent −

Step	Binary Number	Decimal Number
Step 1	$19FDE_{16}$	$((1 \times 16^4) + (9 \times 16^3) + (F \times 16^2) + (D \times 16^1) + (E \times 16^0))_{10}$
Step 2	$19FDE_{16}$	$((1 \times 16^4) + (9 \times 16^3) + (15 \times 16^2) + (13 \times 16^1) + (14 \times 16^0))_{10}$
Step 3	$19FDE_{16}$	$(65536+ 36864 + 3840 + 208 + 14)_{10}$
Step 4	$19FDE_{16}$	106462_{10}

Note − $19FDE_{16}$ is normally written as 19FDE.

37. Explain the process of converting from one number system to another in detail.

Ans:

There are many methods or techniques which can be used to convert numbers from one base to another. In this chapter, we'll demonstrate the following −

- Decimal to Other Base System
- Other Base System to Decimal
- Other Base System to Non-Decimal
- Shortcut method - Binary to Octal
- Shortcut method - Octal to Binary
- Shortcut method - Binary to Hexadecimal
- Shortcut method - Hexadecimal to Binary

Decimal to Other Base System

Step 1 − Divide the decimal number to be converted by the value of the new base.

Step 2 − Get the remainder from Step 1 as the rightmost digit (least significant digit) of the new base number.

Step 3 − Divide the quotient of the previous divide by the new base.

Step 4 − Record the remainder from Step 3 as the next digit (to the left) of the new base number.

Repeat Steps 3 and 4, getting remainders from right to left, until the quotient becomes zero in Step 3.

The last remainder thus obtained will be the Most Significant Digit (MSD) of the new base number.

Example

Decimal Number: 29_{10}

Calculating Binary Equivalent −

Step	Operation	Result	Remainder
Step 1	29 / 2	14	1
Step 2	14 / 2	7	0
Step 3	7 / 2	3	1
Step 4	3 / 2	1	1
Step 5	1 / 2	0	1

As mentioned in Steps 2 and 4, the remainders have to be arranged in the reverse order so that the first remainder becomes the Least Significant Digit (LSD) and the last remainder becomes the Most Significant Digit (MSD).

Decimal Number : 29_{10} = Binary Number : $11101_{2.}$

Other Base System to Decimal System

Step 1 − Determine the column (positional) value of each digit (this depends on the position of the digit and the base of the number system).

Step 2 − Multiply the obtained column values (in Step 1) by the digits in the corresponding columns.

Step 3 − Sum the products calculated in Step 2. The total is the equivalent value in decimal.

Example

Binary Number: 11101_2

Calculating Decimal Equivalent −

Step	Binary Number	Decimal Number
Step 1	11101_2	$((1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2	11101_2	$(16 + 8 + 4 + 0 + 1)_{10}$
Step 3	11101_2	29_{10}

Binary Number : 11101_2 = Decimal Number : 29_{10}

Other Base System to Non-Decimal System

Step 1 − Convert the original number to a decimal number (base 10).

Step 2 − Convert the decimal number so obtained to the new base number.

Example

Octal Number : 25_8

Calculating Binary Equivalent −

Step 1 - Convert to Decimal

Step	Octal Number	Decimal Number
Step 1	25_8	$((2 \times 8^1) + (5 \times 8^0))_{10}$
Step 2	25_8	$(16 + 5)_{10}$
Step 3	25_8	21_{10}

Octal Number : 25_8 = Decimal Number : 21_{10}

Step 2 - Convert Decimal to Binary

Step	Operation	Result	Remainder
Step 1	21 / 2	10	1
Step 2	10 / 2	5	0
Step 3	5 / 2	2	1
Step 4	2 / 2	1	0
Step 5	1 / 2	0	1

Decimal Number : 21_{10} = Binary Number : 10101_2

Octal Number : 25_8 = Binary Number : 10101_2

Shortcut Method − Binary to Octal

Step 1 − Divide the binary digits into groups of three (starting from the right).

Step 2 − Convert each group of three binary digits to one octal digit.

Example

Binary Number : 10101_2

Calculating Octal Equivalent −

Step	Binary Number	Octal Number
Step 1	10101_2	010 101
Step 2	10101_2	2_8 5_8
Step 3	10101_2	25_8

Binary Number : 10101_2 = Octal Number : 25_8

Shortcut Method ─ Octal to Binary

Step 1 − Convert each octal digit to a 3-digit binary number (the octal digits may be treated as decimal for this conversion).

Step 2 − Combine all the resulting binary groups (of 3 digits each) into a single binary number.

Example

Octal Number : 25_8

Calculating Binary Equivalent −

Step	Octal Number	Binary Number
Step 1	25_8	2_{10} 5_{10}
Step 2	25_8	010_2 101_2
Step 3	25_8	010101_2

Octal Number : 25_8 = Binary Number : 10101_2

Shortcut Method ─ Binary to Hexadecimal

Step 1 − Divide the binary digits into groups of four (starting from the right).

Step 2 − Convert each group of four binary digits to one hexadecimal symbol.

Example

Binary Number : 10101_2

Calculating hexadecimal Equivalent −

Step	Binary Number	Hexadecimal Number
Step 1	10101_2	0001 0101
Step 2	10101_2	1_{10} 5_{10}
Step 3	10101_2	15_{16}

Binary Number: 10101_2 = Hexadecimal Number: 15_{16}

Shortcut Method - Hexadecimal to Binary

Step 1 − Convert each hexadecimal digit to a 4-digit binary number (the hexadecimal digits may be treated as decimal for this conversion).

Step 2 − Combine all the resulting binary groups (of 4 digits each) into a single binary number.

Example

Hexadecimal Number : 15_{16}

Calculating Binary Equivalent −

Step	Hexadecimal Number	Binary Number
Step 1	15_{16}	1_{10} 5_{10}
Step 2	15_{16}	0001_2 0101_2
Step 3	15_{16}	00010101_2

Hexadecimal Number: 15_{16} = Binary Number: 10101_2