Computer programming & Data Structures

<u>Unit I</u>

<u>Introduction to Computers</u>

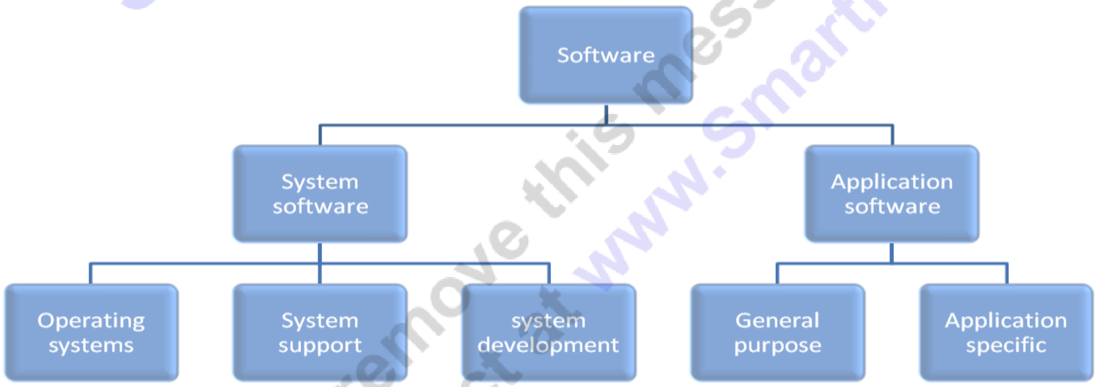
(1) What are system software and application software? What are the differences between them?

A piece of software is a program or a collection of programs plus associated data. It is created by a computer programmer who writes lines of code for the computer.

A computer program is a series of instructions telling the computer what to do. Computers can be programmed to perform many kinds of tasks. Such as computer games; each one is a program. Someone wrote the instructions to tell the computer how to create the sound and graphics that are part of the game. Eg. Word processing programs like Microsoft word; or a spreadsheet program like Microsoft Excel. In fact, working on a computer in any way, is using a program. These programs constitute the software used with a computer system.

An interpreter or compiler is a smaller program which changes the programmer's code into machine instructions for the central processing unit. After much testing and debugging, the programmer's code is finally "packaged" into executable files which make up the final "software" which can be purchased later, or might be "bundled" with the computer when you buy it.

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



System Software

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

The operating system provides services such as a user interface, file and database access, and interfaces to communication systems such as Internet protocols. The primary purpose of this software is to keep the system operating in an efficient manner while allowing the users access to the system.

System support software provides system utilities and other operating services. Examples of system utilities are sort programs and disk format programs. Operating services consists of programs that provide performance statistics for the operational staff and security monitors to protect the system and data. The system development software, includes the language translators that convert programs into machine language for execution, debugging tools to ensure that the programs are error free.

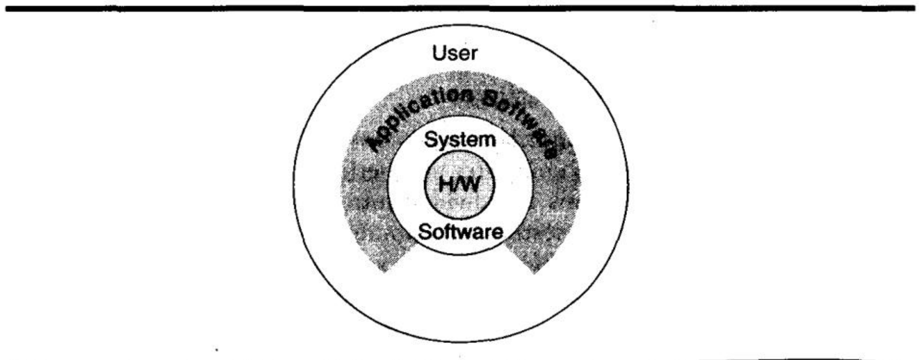
Application software

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

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

A general ledger system used by accountants and a material requirements planning system used by a manufacturing organization are examples of application-specific software. They can be used only for the task for which they were designed they cannot be used for other generalized tasks.

The relation ship between system and application software is shown below. In this figure, each circle represents an interface point .The inner core is hard ware. The user is represented by the out layer. To work with the system, the typical user uses some form of application software. The application software in turn interacts with the operating system, which is apart of the system software layer. The system software provides the direct interaction with the hard ware. The opening at the bottom of the figure is the path followed by the user who interacts directly with the operating system when necessary.



Relationship Between System and Application Software

(2) Explain in detail about System Software.

System software

- System software consists of a variety of programs that support the operation of a computer.
- It is a set of programs to perform a variety of system functions as resource management, I/O management and storage management.
- The characteristic in which system software differs from application software is machine dependency.
- An application program is primarily concerned with the solution of some problem, using the computer as a tool.
- System programs on the other hand are intended to support the operation and use of the computer itself, rather than any particular application.
- For this reason, they are usually related to the architecture of the machine on which they are run.
- Examples of system software are text-editors, compilers, loaders or linkers, debuggers, assemblers and operating systems.
- For example, an assembler is a system software. It translates mnemonic instructions into machine code. The instruction formats, addressing modes are of direct concern in assembler design.

- Similarly, compilers must generate machine language code, taking into account such hardware characteristics as the number and the types of registers and machine instructions available
- Operating systems are directly concerned with the management of nearly all of the resources of a computer system.
- There are some aspects of system software that do not directly depend upon the type of computing system being supported. These are known as machine-independent features.
- For example, the general design and logic of an assembler is basically the same on most computers.

Types of System Software: System software programs are divided into three classes: the operating system, system support, and system development. Among these, various types are as below.

1. Operating system
2. Language translators
    a. Compilers
    b. Interpreters
    c. Assemblers
    d. Preprocessors
3. Loaders
4. Linkers

1. OPERATING SYSTEM

- It is the most important system program that act as an interface between the users and the system. It makes the computer easier to use.
- It provides an interface that is more user-friendly than the underlying hardware.
- The functions of OS are:
    1. Process management
    2. Memory management
    3. Resource management
    4. I/O operations
    5. Data management
    6. Providing security to user's job.

2. LANGUAGE TRANSLATORS

It is the program that takes an input program in one language and produces an output in another language.

$$\begin{tikzcd}\textbf{Source Program} \arrow[r] & \boxed{\textbf{Language Translator}} \arrow[r] & \textbf{Object Program}\end{tikzcd}$$

a. Compilers

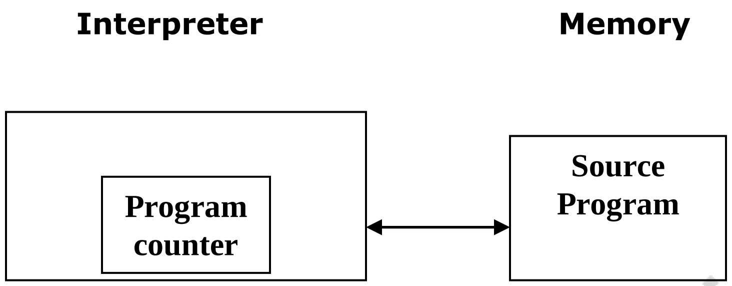
- A compiler is a language program that translates programs written in any high-level language into its equivalent machine language program.
- It bridges the semantic gap between a programming language domain and the execution domain.
- Two aspects of compilation are:
  - Generate code to increment meaning of a source program in the execution domain.
  - Provide diagnostics for violation of programming language, semantics in a source program.
- The program instructions are taken as a whole.

$$\begin{tikzcd}\textbf{High level language} \arrow[r] & \boxed{\textbf{Compiler}} \arrow[r] & \textbf{Machine level language}\end{tikzcd}$$

b.Interpreters

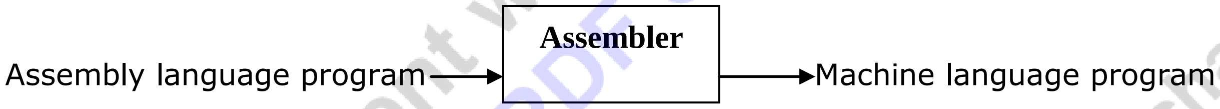
- It is a translator program that translates a statement of high-level language to machine language and executes it immediately. The program instructions are taken line by line.
- The interpreter reads the source program and stores it in memory.
- During interpretation, it takes a source statement, determines its meaning and performs actions which increments it. This includes computational and I/O actions.
- Program counter (PC) indicates which statement of the source program is to be interpreted next. This statement would be subjected to the interpretation cycle.
- The interpretation cycle consists of the following steps:
  - Fetch the statement.
  - Analyze the statement and determine its meaning.

    - Execute the meaning of the statement.
- The following are the characteristics of interpretation:
    - The source program is retained in the source form itself, no target program exists.
    - A statement is analyzed during the interpretation.



c.Assemblers

- Programmers found it difficult to write or red programs in machine language. In a quest for a convenient language, they began to use a mnemonic (symbol) for each machine instructions which would subsequently be translated into machine language.
- Such a mnemonic language is called Assembly language.
- Programs known as Assemblers are written to automate the translation of assembly language into machine language.

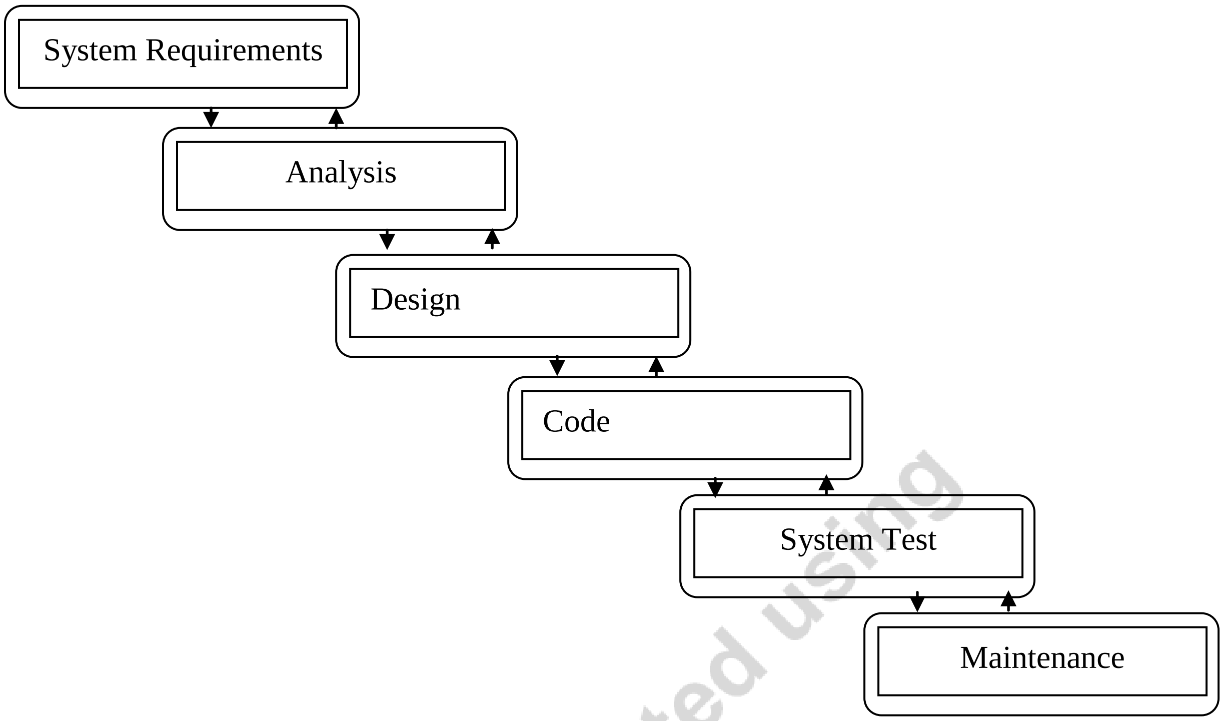


Fundamental functions:

- Translating mnemonic operation codes to their machine language equivalents.
- Assigning machine addresses to symbolic tables used by the programmers.

3. Explain in detail the "System Development Life Cycle"

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



This model widely known as **"water fall model"** starts with **systems requirements** in this phase the systems analyst defines requirements that specify what the proposed system is to accomplish. The requirements are usually stated in terms that the user understands.

The **analysis** phase looks at different alternatives from a systems point of view while the **design** phase determined how the system will be built. In the design phase the functions of the individual programs that will make up the system are determined and the design of the files and / or the databases is completed. In the next phase **code** programs are written. After the programs have been written and tested to the programmer's satisfaction, the project proceeds to the **system test**. All of the programs are tested together to make sure of the system works as a whole. The final phase **maintenance,** which keeps the system working once it has been put into production.
The entire procedure is iterative and any errors made in each phase have the possibility to be rectified by going back to the required stage before reaching the final stage.

(4) Describe various program developing steps.
Programming can be defined as the development of a solution to an identified problem. This critical process determines the overall quality and success of our program. If we carefully design each program using good structured development techniques, programs will be efficient, error-free, and easy to maintain.There are seven basic steps in the development of a program:

1. Define the problem

This step (often overlooked) involves the careful reading and re-reading of the problem until the programmer understands completely what is required.

2. Outline the solution (analysis)

Once the problem has been defined, the programmer may decide to break the problem up into smaller tasks or steps, and several solutions may be considered. The solution outline often takes the shape of a hierarchy or structure chart.

3. Develop the outline into an algorithm (design)

Using the solution outline developed in step 2, the programmer then expands this into a set of precise steps (algorithm) that describe exactly the tasks to be performed and the order in which they are to be carried out. This step can use both structured programming techniques and pseudo code.

4. Test the algorithm for correctness (desk check)

This step is one of the most important in the development of a program as is often forgotten. Test data needs to be walked though each step in the algorithm to check that the instructions described in the algorithm will actually do what they are supposed to. If logic errors are discovered then they can be easily corrected.

5. Code the algorithm into a specific programming language (coding)

It is only after all design considerations have been met that a programmer should actually start to code the program. In preceding analysis it may have been necessary to consider which language should be used, as each has its own peculiarities (advantages and disadvantages).

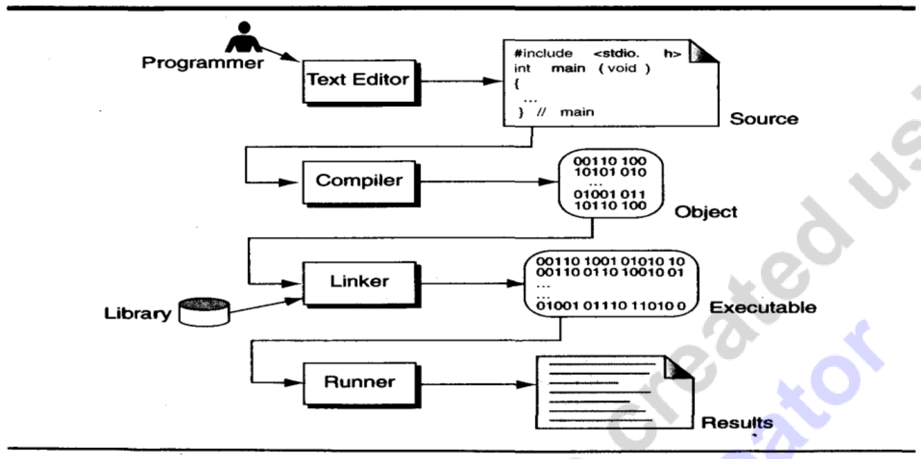
6. Run the program on the computer (testing) This step uses a program compiler and test data to test the code for both syntax and logic errors. If the program is well designed then the usual time-wasting frustration and despair often associated with program testing are reduced to minimum. This step will often need to be done several times until the programmer is satisfied that the program is running as required.

7. Document and maintain the program (documentation)

Program documentation should not be just listed as the last step in the development process, as it is an ongoing task from the initial definition of the problem to the final test results. Documentation also involves maintenance - the changes that are made to a program, often by another programmer, during the life of that program. The better a program has been documented and the logic understood, the easier it is for another to make changes. The whole process of defining problems to providing the coded solution are an ongoing process that is circular in nature and can be called the ***System Development Life Cycle*** (SDLC).

(5) Write a short notes on Creating and running the programs (which is a part of program developing steps)

Creating and Running Programs: Computer hardware understands a program only if it is coded in its machine language. It is the job of the programmer to write and test the program .There are four steps in this process:1.Writing and Editing the program2.Compiliing the program 3.Linking the program with the required library modules 4.Executing the program.



Building a C Program

Writing and Editing Programs

The software used to write programs is known as a **text editor.** A text editor helps us enter, change, and store character data. Depending on the editor on our system, we could use it to write letters, create reports, or write programs. The main difference between text processing and program writing is that programs are written using lines of code, while most text processing is done with character and lines.

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

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

Compiling Programs:

The code in a source file stored on the disk must be translated into machine language ,This is the job of the **compiler.** The c compiler is two separate programs. the **preprocessor** and the **translator.**

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

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

Linking Programs:

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

Executing Programs:

Once program has been linked, it is ready for execution. To execute a program we use an operating system command, such as run, to load the program into primary memory and execute it. Getting the program into memory is the function of an operating system program known as the loader. It locates the executable program and reads it into memory. When everything is loaded, the program takes control and it begins execution.

(6) Discuss and differentiate the machine-level, pneumonic and high-level languages.

The language that is used in the communication of computer instructions is called as programming language.
The programming languages are classified as 1) Machine language (Low-level language) 2) Assembly language (symbolic language) and 3) High level language (procedure oriented language).

Over the years computer languages have evolved from machine languages to natural languages.

1940's Machine level Languages

1950's Symbolic Languages

1960's High-Level Languages

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

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

The only language understood by computer hardware is machine language. It is very difficult to understand and also it is machine dependent that is code varies from one computer to another.

Assembly Language: An assembly language uses symbols or mnemonics to represent the various, machine language instructions. It is machine dependent that is code varies from one computer to another. Since computer does not understand symbolic language it must be translated to the machine language. A special program called assembler translates symbolic code into machine language. Because symbolic languages had to be assembled into machine language they soon became known as assembly languages.

Working with symbolic languages was also very tedious because each machine instruction has to be individually coded.

High Level Languages: It is a simple English language. By seeing that program, one should be able to understand what the program does and what data it uses. It is 'machine-independent'.

High-level languages are designed to relieve the programmer from the details of the assembly language. High level languages share one thing with symbolic languages, They must be converted into machine language. The process of converting them is known as compilation.

A program written in high level language is known as the source program and can be run on different machines using different translators. The translated program (from high-level language to machine-level language) is known as object code.

(7) What are algorithm, pseudo code and flowchart? Mention their advantages and disadvantages.

Pseudo code : Pseudo code is a programming analysis tool that is used for planning program logic. It is an imitation of actual computer instructions written in an ordinary natural language such as English. Pseudo code is also called as Program Design Language(PDL).
Pseudo code is made up of the following basic logic structures.

1. Sequence – The sequence logic is used for performing instructions one after another in sequence.
2. Selection (If…then…else (or) if..then) – The selection logic is also called as the decision making logic, as it is used for making decisions and selecting proper path out of the two or more alternative paths in the program logic.
3. Iteration (Do..while (or) Repeat.. Until) – The iteration logic is used to produce loops when one or more instructions may be executed several times depending on some condition.

Eg: Write a pseudo code to calculate sum of first 50 natural numbers.

```
1. Define sum and number.

2. Initialize sum and number to zero.

3. Increment N by 1.

4. Calculate sum
```

5. If number<50, repeat from step3.

6. Print sum.

Advantages of Pseudo code:

1. Converting a Pseudo code to a programming language is much easier as compared to converting a flowchart or decision table.
2. As compared to a flowchart, it is easier to modify the Pseudo code of program logic when program modifications are necessary.
3. Writing of pseudo code involves much less time and effort than drawing an equivalent flowchart.

Limitations of Pseudo code:

1. A graphic representation of a program is not available in pseudo code.
2. There are no standard rules to follow in pseudo code.

Algorithm: A logical list of procedures or steps for solving a given problem is called as Algorithm.

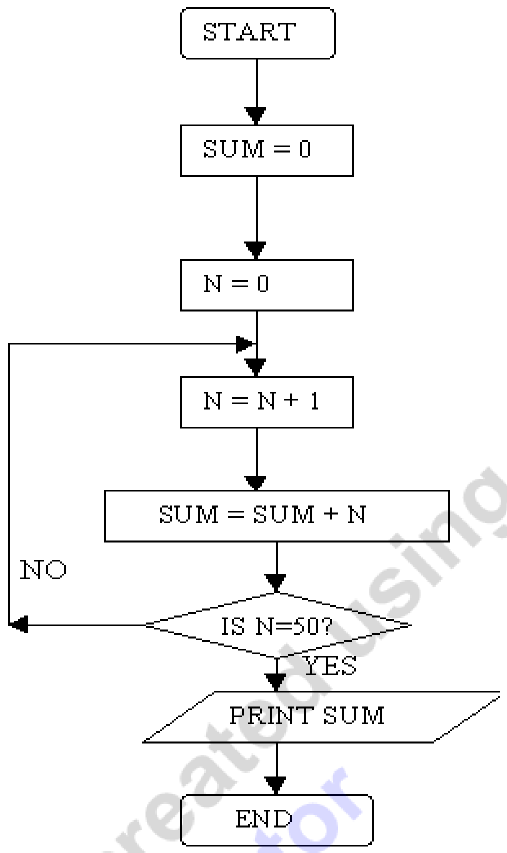
Eg: Write an algorithm to find the sum of first 50 numbers.

```
Step1: Begin
Step2: sum=0, N=0
Step3: N=N+1
Step4: sum=sum+N
Step5: if N<50, go to Step3
Step6: Print sum
Step7: End
```

Flow chart: Flow chart is a pictorial representation of instructions to be followed by the computer.

A flowchart consists of various types of boxes connected by arrows. Each box represents some operations which are described by the statements written within the box. Boxes of different shapes are used to indicate different types of operations. The arrow indicates the sequence in which operations are to be performed. The flowchart is drawn according to defined rules and using standard flowchart symbols prescribed by the American National Standard Institute, Inc.

Eg: Draw a flowchart to find the sum of first 50 numbers.



Advantages of flow charts:

1. It is machine-independent; hence we can use it for any computer system.
2. The logic of the program is easily understood through it.
3. Any logical errors can be easily checked and corrected through it.
4. It is especially used when repeated calculations are involved.
5. Program flowcharts serve as a good program documentation, which is needed for various purposes.
6. The flowchart helps in debugging process.
7. The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part.

Limitations of flow charts:

1. Flow charts are time consuming and difficult to draw with proper symbols and spacing, especially for large complex programs.

2. Owing to the symbol-string nature of flow charting, any changes or modifications in the program logic will usually require a completely new flow chart.

(8) Write about the history of C language.

- The C language is one of the powerful languages developed by Dennis Ritchie at the AT & T's Bell Laboratories in the year 1972.
- The root of all modern languages is ALGOL, introduced in early 1960's. It was the first computer language to use block structure.
- Further a language called BCPL (Basic Combined Programming Language) was invented by Martin Richards at the Cambridge University.
- After that based on BCPL, another language was developed by Ken Thompson at the AT & T's Bell Laboratories and was named as B, the first letter of BCPL.
- C was evolved from ALGOL, BCPL and B by Ritchie (name is given to C language from the next letter in BCPL).
- Since C was developed along with the UNIX operating system, it is strongly associated with UNIX.
- The C language was standardized by the American National Standard's Institute (ANSI) and referred as ANSI C.
- The standard was also adopted by ISO (International Standard's Organization). C is further referred as ANSI/ISO C standard.

(9) Explain various features of C language.

The C language has various features like:
a) C is portable – C language is machine independent i.e. if a program is written in the C language for one computer, it can be used on any other computer without making any changes. This makes the C language portable.
b) C is efficient and fast – The source code in the C language can be compiled and linked quickly. The executable programs obtained after compiling and linking, runs very fast. This is

because, the C language is capable of interacting directly with the hardware of the computer. Hence, a C program runs with a very high speed.

c) C is compact – The statements in the C language are generally short but are very powerful. Several operators can be combined together in just one statement. This makes the source code in the C language very compact.

d) C language is user-defined - User defined functions can be created and used in accordance to our need.

e) The C language has both the simplicity of the high level language and the advantage of the low level language. The source code in the C language can be written more easily as compared to the low level language such as assembly language. In this way, it has the advantage that it is similar to the high level language. But the C language can interact directly with the computer hardware. In this case, it has the advantage of a low level language.

(10) What are the characteristics of C language?

C is a structured language. The distinguishing feature of a structured language is compartmentalization of code and data. This is the ability of a language to section off and hide all information and instructions to perform a specific task from rest of the program.

a) C language is case-sensitive: It recognizes the difference between upper case and lower letters.

b) The C program can be written with a free format i.e. there is no rule that only instruction should be written in one line etc.

c) The C language as a structured language allows you a variety of programming possibilities. It directly supports several loop constructs such as 'while', 'do-while' and 'for'.

d) Every statement in the C program ends with a semi colon.

e) Every C program execution begins with main function.

```
main()
{
}
```

(f) Small in size.

(g) Extensive use of function calls.

Structure of C programs

A C program may contain one or more sections as shown in figure below.

Documentation Section – Comments	**/* program name */**
Link Section--instructions to the compiler to link functions from the system library (header files)	
Definition Section	symbolic constants or macros
Global Declaration	variable declaration, function declaration
Main()- Function Section	
{	
Declaration Part	Local variables declaration
Executable Part	
}	
Subprogram section	**Function 1**
	Function 2
	....
	Function N

1. The documentation section consists of a set of comment lines giving the name of the program, date and other details which the programmer would like to use later.
2. The link section provides instructions to the compiler to link functions from the system library.
3. The definition section defines all the symbolic constants.

4. There are some variables that are used in more than one function. Such variables are called Global variables and are declared in the global declaration section that is outside all the functions. This section also declares all the user-defined functions.
5. Every c program must have one main() function section. This section contains two parts – 1. Declaration part and 2. Execution part. The declaration declares all the variables used in the executable part. There is at least one statement in the executable part. 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 statements within the main section should end with a semicolon (;).
6. The subprogram 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.
7. All sections except the main section may be absent when they are not required.

C Tokens

The smallest element in the C language is the token. It may be a single character or a sequence of characters to form a single item . The tokens can be

- Keywords
- Identifiers
- Constants
- Operators
- Special symbols

Keywords and Identifiers

Identifiers:

Identifiers are names which are given to elements of a program such as variables , arrays & functions. Basically identifiers are the group of alphabets or digits.

Rules for making identifier name:

1. The first character of identifiers must be analphabet or an underscore.
2. All characters must be alphabets or digits.

3. There are no special characters allowed except the underscore "_".
4. Two underscores are not allowed.
5. Don't use key words as identifiers.

Keywords:

Keywords are the words whose meanings are already exist in the compiler (fixed meanings) and these meanings cannot be chaged.

1. Keywords cannot be used for variable names.
2. Keywords are also called "Reserved words".
3. There are only 32 keywords available in C.

"auto, double, int, struct, break, else, long, switch, case, enum, register, typedef, char, extern, return, union, const, float, short, unsigned, continue, for, signed, void, default, goto, sizeof, volatile, do, if, static, while"

Variable names

A variable is a data name that may be used to store the data value. Unlike constants that remain unchanged during the execution of a program, a variable may take different values at different times during the execution of the program. Every variable has a name and a value. The name identifies the variable, the value stores data.

There is a limitation on what these names can be.

- Every variable name in C must start with a letter; the rest of the name can consist of letters, numbers and underscore characters. (some systems permit underscore as the first character)
- C recognizes upper and lower case characters as being different i.e. tax is different from Tax.
- You cannot use any of C's keywords like main, while, switch etc as variable names.
- White space is not allowed in a variable name

Examples of legal variable names include

x	result	outfile	bestyet
x1	x2	out_file	best_yet
power	impetus	gamma	hi_score

It is conventional to avoid the use of capital letters in variable names. These are used for names of constants. Some old implementations of C only use the first 8 characters of a variable name. Most modern ones don't apply this limit though.

Variable declaration: The variable declaration begins with type of the variable and followed by name of the variable.

Syntax: data-type name;

Eg: int m;

```
    float x;
    char peak;
```

The keyword int tells the compiler that the variable contains an integer value. So the variable m contains an integer value.

More than one variable can be declared in the same statement as shown below. This is called multiple declarations.

Syntax: data-type var1,var2,var3….varn;

Eg: int m,n,o;

```
    float x,y,z;
```

The keyword float tells that the variables x, y and z contains a floating point value.

Assigning a value to a variable: (Initializing the variable)

A value can be assigned to a variable as shown below.

Eg: int m=9;

```
    float x=1.8;
    char peak='u';
    int m=1,n=2,o=3;
    float x=9.8,y=8.9,z=9;
```

We can also assign values to variable after declaration somewhere in the executable part such as,

int m;

float x;

```
-----
-----
m=10,x=1.9;
```

Note: Variable declarations appear as soon as the main() starts as a first line of code in declaration part of the main().

```
Void main()
{
  int x,y;
  float c;
----

}
```

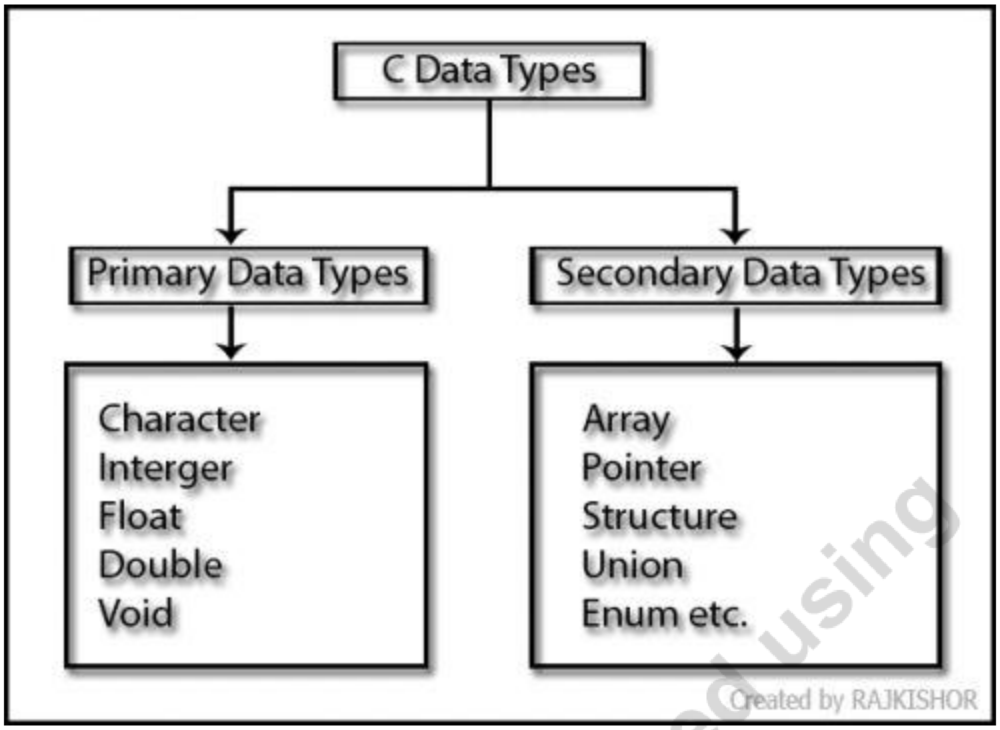
Data types

C language is rich of data types. A C programmer has to employ proper data type as per his requirement.

C has different data types for different types of data and can be broadly classified as :

1. Primary data types
2. Secondary data types

Data Types in C



Integer data types:

Integers are whole numbers with a range of values, range of values are machine dependent. Generally an integer occupies 2 bytes memory space and its value range limited to -32768 to +32767 (that is, -215 to +215-1). A signed integer use one bit for storing sign and rest 15 bits for number.

To control the range of numbers and storage space, C has three classes of integer storage namely short int, int and long int. All three data types have signed and unsigned forms. A short int requires half the amount of storage than normal integer. Unlike signed integer, unsigned integers are always positive and use all the bits for the magnitude of the number. Therefore the range of an unsigned integer will be from 0 to 65535. The long integers are used to declare a longer range of values and it occupies 4 bytes of storage space.

Syntax: int <variable name>;

Example:
```
int num1;
short int num2;
long int num3;
```

Integer Data Type Memory Allocation

Short Int	Int	Long int
1 Byte	2 Bytes	4 Bytes

Floating Point Data Types:

The float data type is used to store fractional numbers (real numbers) with 6 digits of precision. Floating point numbers are denoted by the keyword float. When the accuracy of the floating point number is insufficient, we can use the double to define the number. The double is same as float but with longer precision and takes double space (8 bytes) than float. To extend the precision further we can use long double which occupies 10 bytes of memory space.

Syntax: float <variable name>; like

Example: float num1;
double num2;
long double num3;

Floating Point Data Type Memory Allocation

Float	Double	Long double
4 Bytes	8 Bytes	10 Bytes

Character Data Type:

Character type variable can hold a single character. As there are singed and unsigned int (either short or long), in the same way there are signed and unsigned chars; both occupy 1 byte each, but having different ranges. Unsigned characters have values between 0 and 255 and signed characters have values from –128 to 127.

Syntax: char <variable name>; like

Example: char ch = ‘a’;

Void Type:

The void type has no values therefore we cannot declare it as variable as we did in case of integer and float.

The void data type is usually used with function to specify its type. Like in our first C program we declared "main()" as void type because it does not return any value. The concept of returning values will be discussed in detail in the C function hub.

Secondary Data Types

- **Arrays:**
  An array is a collection of similar data-type, means an array is a block of consecutive memory locations, all given one symbolic name. Each location in the block is called as an element of the array and each element is of the same type. Arrays can be created from any of the C data-types such as int, float etc.

- **Pointers**:
  A pointer is a variable that represents the address location of a variable or an array element.
- **Structures:**

  A structure is a collection of data items of different data types using a single name. Structure is used for packaging data of different types which are logically related to each other.

User defined type declaration

C language supports a feature where user can define an identifier that characterizes an existing data type. This user defined data type identifier can later be used to declare variables. In short its purpose is to redefine the name of an existing data type.

Syntax: typedef <type> <identifier>; like
Example: typedef int number;

Now for example: "int x1" or "number x1" both statements declaring an integer variable. We have just changed the default keyword "int" to declare integer variable to "number".

Data type modifiers

Data type modifiers are used to alter the meaning of basic data type to suit the need for the program. The data type modifiers are

1. signed
2. unsigned
3. short
4. long

The following table represents the basic data types along with the data type modifiers and range of values.

Character		
Type	**Size (byte)**	**Range**
char or signed char	1	-128 to 127
unsigned char	1	0 to 255
Integer		
Type	**Size (byte)**	**Range**
int or signed int	2	-32768 to 32767
unsigned int	2	0 to 65535
short int or signed short int	1	-128 to 127
unsigned short int	1	0 to 255
long int or signed long int	4	-2147483648 to 2147483647
unsigned long int	4	0 to 4294967295
Float and Double		
Type	**Size (byte)**	**Range**
float	4	3.4 e-38 to 3.4 e+38
double	8	1.7e-308 to 1.7e+308
long double	10	3.4 e-4932 to 3.4 e+4932

Constants

Constants in C refer to the fixed values that do not change during the execution of a program. C supports several types of constants as given below.

1. Integer constants
2. Real constants
3. Single character constants
4. String constants

Integer and real constants are called as numerical constants and single character constants and string constants are called as character constants.

1. Integer constants: An integer constant refers to a sequence of digits. (Any number without decimal point is known as an integer constant). There are three types of integer constants, namely, decimal constants, octal constants, hexadecimal constants.

   Decimal integers consists of digits from 0 to 9, preceded by an optional + or – sign.

   Eg: 123 -123 3456 -89

   Non-digit characters, commas, spaces are not allowed between the digits.

   Eg: Invalid integer constants are -12 3 -12,34 &100

   An octal integer consists of any combination of digits from the set of 0 to 7, with a leading 0.

   Eg: 037 023 0677

   Invalid octal integers are 0987 89 0x89

   A sequence of digits preceded by 0x or 0X is considered as hexadecimal integer constant. They may also include alphabets from A to F or a to f.

   Eg: 0x89 0xAd 0X1f 0x8e

   Invalid hexadecimal integers are 0x0p 0xcl 89 097

   The range of integer values is -32768 to 32767(on 16-bit machines). It is also possible to store larger integer constants by appending qualifiers such as U, L, UL to the constants.

   Eg: 57890U or 57890u (unsigned integer)
   987665L or 987665l (long integer)

2. Real constants: Numbers containing fractional parts like 17.58 are considered as real (or floating point) constants.

   Eg: 0.00083 -0.786 +24.899 **.**9 -**.**71

   A real number may also be expressed in exponential (or scientific) notation.Eg: 215.78 may be written as 2.15778e2 in exponential notation. e2 means multiply by 10^2. The general form is

   Mantissa e exponent

The mantissa is either a real number expressed in decimal notation or an integer. The exponent is an integer number with optional + or – sign. The letter **e** used as a separator may be **e** or **E**.

Eg: 0.65e4   12e-3   1.45E+3

Note: Spaces, commas, special symbols other than '**.**' is not allowed.

The range of the real integer can be increased by declaring it as Long float (l or L).

3. <u>Single character constants</u>: A single character constant consists of a single character enclosed within a pair of single quotes.

   Eg: 'r'   'O'   '9'   ' '

   Note: White space and special symbols are allowed here.

   Note: The character constant '9' is not same as the number 9.

   Character constants have 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 print the letter 'a'.

   Note: It is not possible to perform arithmetic operations on character constants.

4. <u>String constants</u>: A string constant is a sequence of characters enclosed within double quotes.

   Eg: "Hello!"   "How are you?"   "9876"

   Note: Constant 'x' is not equivalent to "x". Both are different.

5. <u>Backslash character constant</u>: (Escape sequences)

   C supports some special backslash constants that are used in output functions. Each of them represents one character although they consist of two characters. These character combinations are known as escape sequences.

\'	single quote
\"	double quote
\\	backslash character
\b	backspace
\a	audible alert (bell)
\f	next page

\n	start a new line
\nnn	treat nnn as an octal number
\r	carriage return
\t	horizontal tab
\v	vertical tab
\?	Question mark
\0	null character marking the end of a string

Declaring a variable as a constant:

Syntax: const data-type variable name;

Example: const int temp = 10;

const is new data type qualifier. This tells the compiler that the value of the variable temp must not be modified by the program.

Volatile

There is an another qualifier **volatile** that could be used to tell explicitly the compiler that a variable's value may be changed at any instance some of time by some external sources (from outside the program).

Syntax: volatile data-type variable_name;

Example: volatile int temp;

The value of the temp may be altered by some external forces even if it does not appear on the left-hand side of an assignment statement.

If the value must not be altered by the program while may be altered by some other process, then we may declare the variable as both const and volatile as shown below.

volatile const int temp=10;

Defining Symbolic constants (Preprocessor Directives)

Symbolic constants or symbolic names or constant identifiers is a piece of text which is used to expand the text before compilation.

Use of preprocessor is advantageous because

1. Programs easier to develop

2. Easier to read
3. Easier to modify
4. The preprocessor also lets us to customize the language

Example: #define PI 3.14

#define STRENGTH 100

Note: Symbolic constant statements do not end with a semi colon. They are terminated by end of the line.

OPERATORS AND EXPRESSIONS

Operators form **expressions** by joining individual constants, variables. C supports a large set of operators which fall into different categories.

- Arithmetic operators,
- Relational operators,
- Logical operators,
- Assignment operators,
- Conditional operators,
- Increment and decrement operators,
- Bitwise operators and
- Special operators.

The data items on which operators act upon are called **operands**. Some operators require two operands while others require only one operand. Most operators allow the individual operands to be expressions. A few operators permit only single variable as operand.

Arithmetic Operators

There are five main arithmetic operators in 'C'. They are

1. '+' for additions,
2. '-' for subtraction,
3. '*' for multiplication,
4. '/' for division and
5. '%' for remainder after integer division. ('%' operator is also known as modulus operator)

Operands can be integer quantities, floating-point quantities or characters. The modulus operator requires that both operands be integers & the second operand be nonzero. Similarly, the division operator (/) requires that the second operand be nonzero, though the operands need not be integers. Division of one integer quantity by another is referred to as integer division. With this division the decimal portion of the quotient will be dropped. If division operation is carried out with two floating- point numbers, or with one floating-point number. & one integer, the result will be a floating-point quotient.

Example:

If we take two variables say x and y and their values are 20 and 10 respectively, and apply operators like addition, subtraction, division, multiplication and modulus on them then their resulted values will be as follows:

x=20, y=10

x+y	**30**
x-y	**10**
x*y	**200**
x/y	**2**
x %y	**0**

If one or both operands represent negative values, then the addition, subtraction, multiplication and division operations will result in values whose signs are determined by the usual rules of algebra.

The interpretation of remainder operation is unclear when one of the operands is negative. Operands that differ in type may undergo type conversion before the expression takes on its final value. The final result has highest precision possible, consistent with data types of the operands, which will be seen in the operator precedence and type conversion respectively.

For exponentiation there is no specific operator in 'C' instead there is one library function known as "**pow"** to carry out exponentiation.

Relational operators

Relational operators are symbols that are used to test the relationship between two variables, or between a variable and a constant. The test for equality, is made by means of two adjacent equal signs with no space separating them. 'C' has six relational operators as follows:

1. > greater than

2. < less than
3. >= greater than or equal to
4. >= less than or equal to
5. != not equal to
6. == equal to

The first four operators fall within the same precedence group, which is lower than the arithmetic operators. The associativity of these operators is left-to-right, which will be clearly demonstrated in the form of a table later in this section.

The equality operators "==" and "!=" fall into a separate precedence group, beneath the relational operators. Their associativity is also from left to right.

These relational operators are used to form logical expression representing condition that is either true or false. The resulting expression will be of type integer, since true is represented by the integer value and false is represented by the value 0.

Example:

x=2, y=3, z=4.

Expression	Status	Value
x<y	**true**	**1**
(x+y) >=z	**true**	**1**
(y+z)>(x+7)	**false**	**0**
z!=4	**false**	**0**
y ==3	**true**	**1**
(z/x)==x	**true**	**1**

Logical Operators

There are two logical operators in C language, they are **and** and **or**. They are represented by **"&&"** and **"||"** referred to as logical and, logical or, respectively. The result of a logical and operation will be true only if both operands are true, whereas the result of a logical or operation will be true if either operand is true or if both operands are true. The logical operators act upon operands that are themselves logical expressions.

Example:

Suppose x=7, y=5.5, z= 'w'

Logical expressions using these variables are as follows:

Expression	Interpretation	Value
(x>=6) && (z= ='w')	true	1
(x>=6) ||(y = =119)	true	1
(x<=6) && (z=='w')	false	0

Each of the logical operators falls into its own precedence group. Logical "and" has a higher precedence than logical "or". Both precedence groups are lower than the group containing the equality operators. The associativity is left to right.

Unary Operator:

'C' also includes the unary operator "!", that negates the value of a logical expression. This is known as logical negation or logical NOT operator. The associativity of negation operator is right to left.

Precedence of operators in decreasing order is as follows:

1. -, ++ - - Operators ! size of (type)
2. * / %
3. + -
4. < <= > > =
5. == !=
6. &&
7. ||

Please Note:*If you have trouble remembering all these rules of precedence you can always resort to parentheses in order to express your order explicitly.*

Assignment Operators

There are several different assignment operators in C. All of them are used to form assignment expression, which assigns the value of an expression to an identifier. The most commonly used assignment operator is **"="**. The assignment expressions that make use of this operator are written in the form:

Identifier = expression

where identifier generally represents a variable and expression represents a constant, a variable or a more complex expression.

Assignment operator “=” and the equality operator “==” are distinctly different. The assignment operator is used to assign a value to an identifier, whereas the equality operator is used to determine if two expressions have the same value. These two operators cannot be used in place of one another.

If the two operands in an assignment expression are of different data types, then the value of the expression on the right will automatically be converted to the type of the identifier on the left. The entire assignment expression will then be of this same data type.

A floating point value may be truncated if assigned to an integer identifier, a double-precision value may be rounded if assigned to a floating-point identifier, and an integer quantity may be altered if assigned to a shorter integer identifier or to a character identifier.

Multiple assignments of the form
Identifier 1= identifier 2 = - - - -= expression are allowed in ‘C’.
In such situations, the assignments are carried out from right to left.
Example
x=y=10 (x and y are integer variables)
will cause the integer value 10 to be assigned to both x and y.
Similarly, x=y=10.5 will cause the integer value 10 to be assigned to both x and y, truncation occurs when the floating point value 10.5 is assigned to the integer variable y.

‘C’ also contains the five additional assignment operators :

1. +=,
2. -=,
3. *=,
4. /=,
5. %=.

Assignment operators have a lower precedence than any of the other operators that have been discussed earlier. The decreasing order of precedence is given below, the associativity of assignment operators is right to left.

Conditional Operators

There is one conditional operator (?:) in 'C' language. An expression that makes use of the conditional operator is called a conditional expression.
A conditional expression is written in the form

Expression 1? Expression 2: Expression 3

When evaluating a conditional expression, expression 1 is evaluated first. If expression 1 is true, then expression 2 is evaluated and this becomes the value of the conditional expression. If expression 1 is false, then expression 3 is evaluated and this becomes the value of the conditional expression.
Example: (i< 1) ? 0:200
i is integer variable here.
The expression (i<1) is evaluated first, if it is true the entire conditional expression takes on the value 0. Otherwise, the entire conditional expression takes on the value 200.
Conditional expression frequently appears on the right hand side of a simple assignment statement. The resulting value of the conditional expression is assigned to the identifier on the left. The conditional operator has its own precedence, just above the assignment operator. The associativity is right-to-left.

Increment and decrement operators
C has very useful two operators generally not found in any other programming languages.

1. "++" Increment Operator: The increment operator adds 1 to the operand.
2. "—"Decrement Operator: The decrement operator subtracts 1 to the operand.

Both of them are Unary operators and are extensively used in "for" and "while" loops.
Example:

1. x++=>x=x+1.   3. y--=>y=y-1
2. ++x=>x=x+1.   4. --y=>y=y-1

In general ++x and x++ are the same when used in forming statements independently, they behave differently when they are used in expressions on the right hand side of an assignment statement.

Example: x=3; y=++x; } Output: y=4, x=4. Add 1 to x and assign that to y	Example: x=3; y=x++; } Output: y=4, x=3 y=x+1=>Add 1 to y after assigning x=> x remains

	the same and only y changes.

Bitwise operators

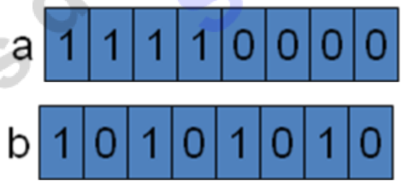
Manipulation of data at bit level is of major importance when it comes to high end applications using micro processors and controllers etc. These operators are used for testing the bits, or shifting them right or left. They cannot be applied to data types float or double.

Corresponding bits of both operands are combined by the usual *logic operations*. There are six bitwise operators:

1. **&** – Bitwise AND
   Result is **1** if both operand bits are **1**
2. **|** – Bitwise OR
   Result is **1** if either operand bit is **1**
3. **^** – Bitwise Exclusive OR
   Result is **1** if operand bits are different
4. **~** – One's Complement
   Each bit is reversed
5. **<<** – Shift left
   Multiply by 2
6. **>>** – Shift right
   Divide by 2

Example:

unsigned int c, a, b;



a	1	1	1	1	0	0	0	0
b	1	0	1	0	1	0	1	0

c=a&b;

1	**0**	**1**	**0**	**0**	**0**	**0**	**0**

c = a | b;

1	**1**	**1**	**1**	**1**	**0**	**1**	**0**

c = a ^ b;

0	**1**	**0**	**1**	**1**	**0**	**1**	**0**

c = ~a;

0	**0**	**0**	**0**	**1**	**1**	**1**	**1**

c = a >> 3;

1	**1**	**0**	**0**	**0**	**0**	**0**	**0**

c = a << 2;

0	**0**	**0**	**1**	**1**	**1**	**1**	**0**

Bit Masking:

In computer science, a **mask** is data that is used for bitwise operations. Using a mask, multiple bits in a byte, nibble, word (etc.) can be set either on, off or inverted from on to off (or vice versa) in a single bitwise operation.

Bitwise AND, OR, XOR and NOT can be used for masking of bits to get the intended value in that context.

***Why the name Bit* Mask?**

- When a bitwise AND is performed, the *zeros* in the MASK constant **hide** the corresponding bits in your data variable (because the result contains only the bits that are *ones* in both operands). You can think of the *zeros* in the MASK constant as being opaque and the *ones* as being transparent. Thus the expression MASK is like covering the bit pattern of data with the bit pattern of MASK so that only the bits under the *ones* of MASK are visible.

Example:

```
        0 0 0 0 0 0 1 0     (MASK constant)
AND    1 0 0 1 0 1 1 0     (your data variable)
        ---------------
        0 0 0 0 0 0 1 0     (result of operation)
```

Uses of Bit Masks

- XOR(^) can swap two variables without using an intermediate, temporary variable which is useful if you are short on available RAM or want that sliver of extra speed. Usually, when **not using** ^, you will do:

```
temp = a;

a = b;

b = temp;
```

Using ^, no "temp" is needed:

```
a ^= b;  ⎫
b ^= a;  ⎬ This will swap "a" and "b" integers. Both must be integers.
a ^= b;  ⎭
```

Special operators

There are many special operators such as comma, sizeof, pointer operators(& and *) and member selection operators(->) etc , out of which the first two will be discussed.

The comma operator

Comma (,) operator is used to link the related expressions together. Comma used expressions are linked from left to right and the value of the right most expression is the value of the combined expression. The comma operator has the lowest precedence of all operators.

Example:

1.sum=(X=5,Y=3,X+Y);

The result will be sum=8. The comma operator is also used to separate variables during declaration.

2.for(i=0;i<8;i++).

The size of operator:

The size of operator is not a library function but a keyword, which returns the size of the operand in bytes. The size of operator always, precedes its operand. The information obtained from this operator can be very useful when transferring a program to a different computer. This operator can be used for dynamic memory allocation. The various expressions and results of sizeof operator are

Expression	**Result**
sizeof(char)	1
sizeof(int)	2
sizeof(float)	4
sizeof(double)	8

Example:

```
int sum=100;
M=sizeof(sum);
```

Precedence of C Operators

Category	**Operator**	**Associativity**
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type) * & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Example:

x* = -3 *(y+z)/4

is equivalent to

x= x*(-3 *(y+z)/4)

In this example first (y+z) will be evaluated, and then multiplication will take place. Then division, and finally multiplication will take place.

Note: *Parentheses can always be used to control precedence and associativity within an expression.*

Type Conversion

Type conversion (often a result of *type casting*) refers to changing an entity of one *data type*, expression, function argument, or return value into another. This is done to take advantage of certain features of type hierarchies. For instance, values from a more limited set, such as integers, can be stored in a more compact format and later converted to a different format enabling operations not previously possible, such as division with several decimal places' worth of accuracy.

Automatic (or Implicit) type conversion

Automatic type conversion (or standard conversion) happens whenever the compiler expects data of a particular type, but the data is given as a different type, leading to an automatic conversion by the compiler without an explicit indication by the programmer.

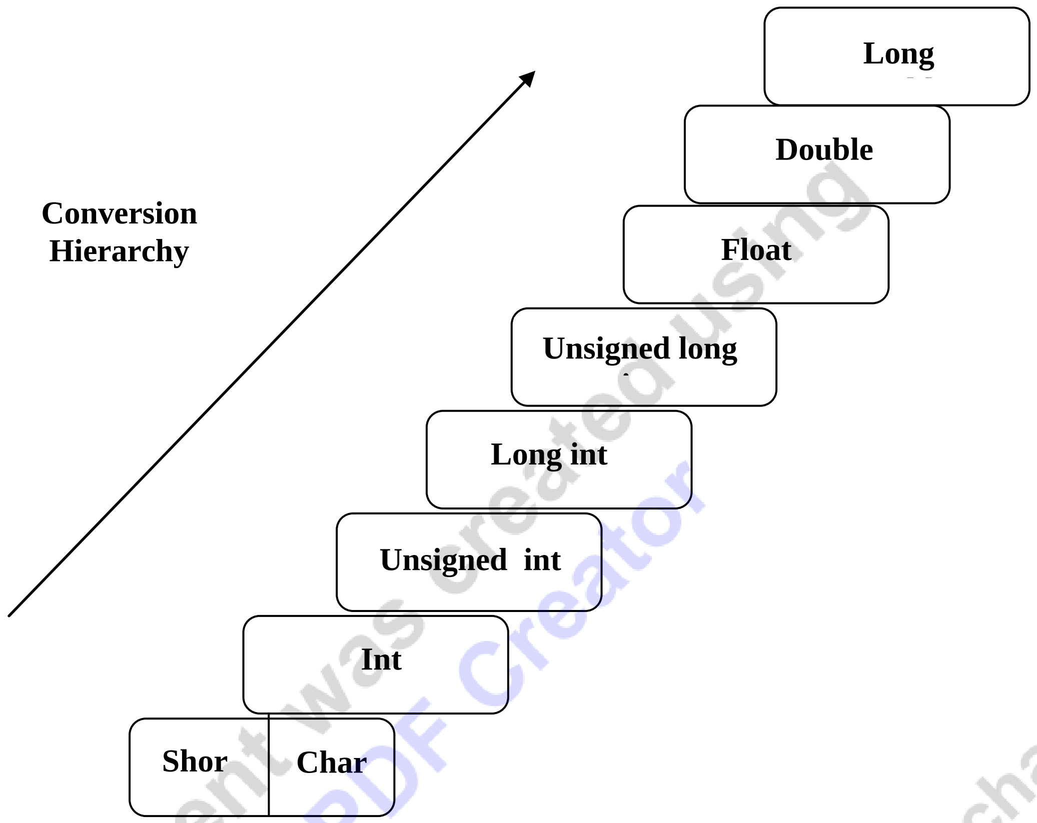
When an expression requires a given type that cannot be obtained through an implicit conversion or if more than one standard conversion creates an ambiguous situation, the programmer must explicitly specify the target type of the conversion. If the conversion is impossible it will result in an error or warning at compile time. Warnings may vary depending on the compiler used or compiler options.

This type of conversion is useful and relied upon to perform integral promotions, integral conversions, floating point conversions, floating-integral conversions, arithmetic conversions, pointer conversions.

int a = 5.6;

float b = 7;

In the example above, in the first case an expression of type float is given and automatically interpreted as an integer. In the second case (more subtle), an integer is given and automatically interpreted as a float.



Note: "C" uses a rule that all expressions except in assignments, any implicit type conversion are made from lower size type to higher size type.
Any automatic type conversion is an implicit conversion if not done explicitly in the source code.

Explicit type conversion (Type Casting)

Type casting is the use of direct and specific notation in the source code to request a conversion or to specify a member from an overloaded class. There are cases where no automatic type conversion can occur or where the compiler is unsure about what type to convert to, those cases require explicit instructions from the programmer or will result in error.

Common usage of type casting

Performing arithmetical operations with varying types of data type without an explicit cast means that the compiler has to perform an implicit cast to ensure that the values it uses in the calculation are of the same type. Usually, this means that the compiler will convert all of the values to the type of the value with the highest precision.

Example: The following is an integer division and so a value of 2 is returned.

float a = 5 / 2;

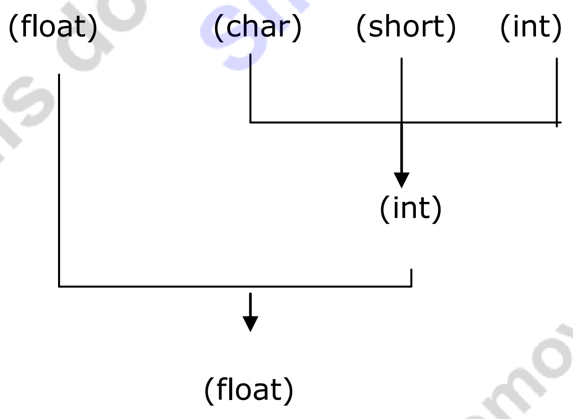
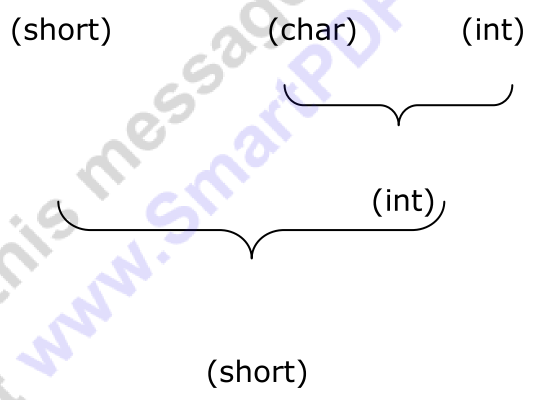
To get the intended behavior, you would either need to cast one or both of the constants to a **float**.

float a = type_cast<float>(5) / type_cast<float>(2);

Or, you would have to define one or both of the constants as a float.

float a = 5f / 2f;

Implicit Type Conversion	**Explicit Type Conversion**

char c='A';
short s=1; int i=2; float l=3;
l = (c + s) * i;

	char c='A';
	short s=1; int i=2;
	s = (short) (c + i);
	

Numeric conversion

After any numeric promotion has been applied, the value can then be converted to another numeric type if required, subject to various constraints.

A value of any integer type can be converted to any other integer type. This only gets complicated when overflow is possible, as in the case where you convert from a larger type to a smaller type. When converting to a signed integer type where overflow is possible, the result of the conversion depends on the compiler. Most modern compilers will generate a warning if a conversion occurs where overflow could happen. Should the loss of information be intended, the programmer may do explicit type casting to suppress the warning; bit masking may be a superior alternative.

Floating-point types can be converted between each other, but are even more prone to platform-dependence. If the value being converted can be represented exactly in the new type then the exact conversion will happen. Otherwise, if there are two values possible in the destination type and the source value lies between them, then one of the two values will be chosen. In all other cases the result is implementation-defined.

Floating-point types can be converted to integer types, with the fractional part being discarded.

```
double a = 12.5;

int b = a;

printf (“%d”, b); // Prints "12".
```

A value of an integer type can be converted to a floating point type. The result is exact if possible, otherwise it is the next lowest or next highest representable value (depending on the compiler).

Managing Input and Output Functions

Input Output operations are useful for program that interact with user, take input from the user and print messages. The standard library for input output operations used in C is stdlib. When working with input and output in C there are two important streams: standard input and standard output. Standard input or stdin is a data stream for taking input from devices such as the keyboard. Standard output or stdout is a data stream for sending output to a device such as a monitor console.

Note: #include <stdio.h> is the header file which contains the printf and scanf functions. So before using them include this header file in your program.

Reading and Writing A Character

The simplest of all input/output operations is reading a character from the standard input unit (usually the keyboard) and writing it to the standard output unit (usually the screen). Reading a single character can be done by using the function **getchar.** The getchar takes the following form :

```
variable_name = getchar();
```

variable_name is a valid C name that has been declared as char type. When this statement is encountered , the computer waits until a key is pressed and then assigns this character as a value to **getchar** function . Since **getchar** is used on the right-hand side of an assignment statement, the character value of **getchar** is in turn assigned to the variable name on the left.

The simplest of output operator is putchar to output a single character on the output device.

```
putchar(varname);
```

For example

```
char name;

name = getchar();
```

will assign the character 'H' to the variable name when we press the key H on the keyboard. Since getchar is a function ,it requires a set of parentheses as shown. **C** supports testing of the character keyed in by the user by including the file ctype.h & the functions which can be used after including this file are

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 a lower case letter?**
isprint(c)	**Is c a printable character?**
ispunct(c)	**Is c a punctuation mark ?**
isspace(c)	**Is c a white space character?**
isupper(c)	**Is c an upper case letter?**

The getchar() and putchar() are used only for one input and output respectively and are not formatted. Formatted input refers to an input data that has been arranged in a particular format, for that we have scanf.

Formatted Input and Output

Formatted input refers to an input data that has been arranged in a particular format. For example, consider the following data:

15.73 123 John

This line contains three pieces of data, arranged in a particular form. Such data has to be read conforming to the format of its appearance. For example, the first part of the data should be read into a variable float, the second into int, and the third part into char. This is possible in C using the **scanf** function.

scanf() is the input function which is used to read the input from an input device such as a keyboard.

The general form of scanf is

scanf("control string",& arg1); control string=%x; x=type of data used as variable.

&(ampersand) is the address operator used to store the entered value into the address location of the corresponding variable.
The control string specifies the field format in which the data is to be entered and the arguments arg1 specifies the address of location where the data is stored. Control string and arguments are separated by commas.

*** Field (or format) specifications, consisting of the conversion character %, a data type character (or type specifier), and an optional number, specifying the field width.

*** Blanks, tabs, or new lines are ignored.

For formatted output you use printf

printf("control string", arg1, arg2,...argn);

/* No Ampersand is used while outputting as the address is already known while scanning and the result is placed in the known address */

Scanf()	**Printf()**
Input function	Output function
To read values at run time	To display values
Call by address fn.	Call by value fn.
Deals with keyboard	Deals with monitor
We can’t write \n and \t	Not allowed
It can contain only variables.	It can contain variables, messages and expressions.

Inputting and Outputting Integer Numbers

The field specification for reading an integer number is: **% w d**

The percent (%) indicates 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.

The field specification for outputting an integer value is: **%wd**

Example:

```
#include <stdio.h>
main()
{
    int i ;
   printf("Please enter an integer \n");
   scanf("%d",&i);
  printf("\n The entered integer=%d",i);
}
```

Output: Please enter an integer

```
          12
         The entered integer=12.
```

Inputting and Outputting Real Numbers

Unlike integers numbers, the field width of real numbers is not to be specified and therefore scanf reads real numbers using the simple specification %f for both the notations, namely, decimal point notation and exponential notation. For example, the statement

scanf("%f %f %f" , &x, &y , &z);

472.34 43.21E-1 678

will assign the value 472.34 to x, 4.321 to y, and 678.0 to z. The input field specifications may be seperated by any arbitrary blank spaces.

If the same data has typo be written on screen, the statement to be used is

printf("The values given as input are %f%f%f",x,y,z);

Output: 472.34 43.21E-1 678

Inputting and outputting Character Strings

We have already seen how a single character can be read from the terminal using the **getchar** function. The same can be achieved using the **scanf** function also. In addition, a **scanf** function can input strings containing more than one character strings.

- **gets()** is a function which recieves a string from keyboard until the user presses enter key. Spaces and tabs are acceptable as part of input string, which ios not acceptable when entering input using scanf function. It is an unformatted function like getchar(). It can read multiword strings like arrays.

  Eg: gets (arrayname);

  Input: "G R I E T"

  If scanf is used only G is read and all the other letters are ignored, whereas using gets(), the entire GRIET is accepted as it is considered to be a string.

  The similar theory can be used for printf and puts as the output depends on the input read by the compiler.

The corresponding argument should be a pointer to a character array. However %c may be used to read a single character when the argument is a pointer to a **char** variable.

scanf and printf Format Codes:

Data type	Format
Int	%d
Float	%f
Char	%c
Long	%ld
Double	%lf

Unsigned	%u

Note: % is called format specifier.

Some more formatting codes are as follows:

- %e read a floating point value
- %h read a short integer
- %i read a decimal, hexadecimal, or octal integer
- % o read an octal integer
- %x read a hexadecimal integer
- %[..] read a string of word(s)

Miscellaneous

Comments

- A comment is a non-executable statement.
- A "comment" is a sequence of characters beginning with a forward slash/asterisk combination (/*) that is treated as a single white-space character by the compiler and is otherwise ignored.
- A comment can include any combination of characters from the representable character set, including newline characters, but excluding the "end comment" delimiter (*/).
- Comments can occupy more than one line but cannot be nested.
- The compiler ignores the characters in the comment.

Use comments to document your code. This example is a comment accepted by the compiler:

```
/*      Comments      can      contain      keywords      such      as
  for and while without generating errors. */
```

Comments can appear on the same line as a code statement:

```
printf( "Hello\n" );  /* Comments can go here */
```

Since comments cannot contain nested comments, this example causes an error:

/* This is example of /* Nested commnet */ This causes error */

% is the conversion character used for displaying the value of the variable. %d (format string) indicates that the variable is of integer type.
\n is an escape sequence used to transfer the control to the next line.

Note: Please mail to deepthi@griet.in or himabindu@griet.in if any queries in regards to this document.