

UNIT-I

INTRODUCTION TO COMPUTERS & ALGORITHMS & FLOWCHARTS

Topics: Generation of computers, Computer system memory hierarchy, Input / Output, RAM/ROM, Software & Hardware, Understand bit, byte, KB, MB, GB and their relations to each other, Operating System overview, Computer Networks Overview.

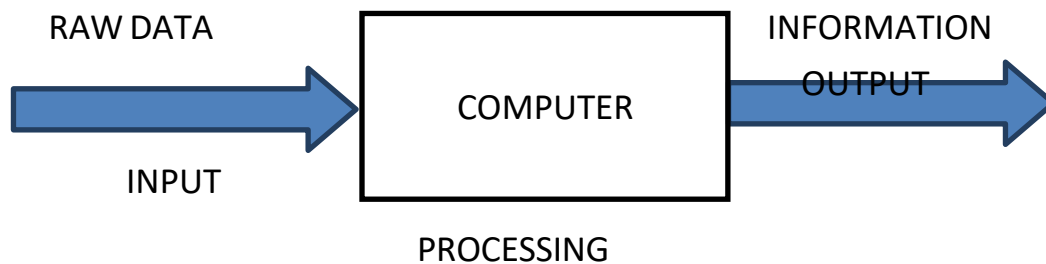
Algorithms and Flow Charts – Examples of Flow charts for loops and conditional statements.
--8 Hrs

Why Computer?

- Increases the productivity and efficiency.
- Can crunch data at lighting speed.
- Can store vast amounts of information and reduce wastage of resources otherwise.
- Helps sort, organize, and search through information.
- Connects to the vast world of information through the Internet.
- Gets a better understanding of data.
- Allows to keep connected with each other.
- Can help in making a decision using the vast data.

Introduction to computers

- Computer is an electronic device that accepts input as data, processes it & stores temporarily into a storage device and gives out useful output.



- A program is a sequence of instructions written using a computer programming language to perform a specified task.
- Computers have power to perform billions of calculations in fraction of seconds. They are very accurate and are reliable.

Data?

- Data represents facts and figures.
- Examples
 - Ram scored 99% marks.
 - Max. Temperature today at Dehradun was 25 degrees centigrade.
 - Mumbai had a rainfall of 220mm on 20th July 2020.
 - BSE : Sensex was trading at 40,000 points.
 - The above examples talk about facts and figures.
- The examples in the previous slide demonstrate that they are some facts but does not make much sense.
- Now lets analyze the following:
 - Ram scored 99% marks and stood first to the class.
 - He was the highest scorer in the University.
 - He secured a rank amongst the first 10 students in the state.

Characteristics of computers

- *Speed*
- *Accuracy*
- *Automatic*
- *Diligence*
- *Memory*
- *No I.Q.*
- *Economical*
- *Versatile*

Applications of computers

- Word Processing
- Internet
- Digital Audio or Video Composition
- Desktop Publishing
- e-Business
- Bioinformatics
- Health care
- GIS and Remote Sensing
- Meteorology
- Multimedia and Animation
- Legal System

- Retail Business

Generation of computers

The computers of today have originated in the second half of the twentieth century. As the technology improved in physics and electronics there was evolutionary developments in the hardware and software of computers. Sooner the computer started to evolve, and the technological advancement marked a generation of computers.

1. First Generation of Computers:

Computers were developed between 1940–1956 and were called the first-generation computers. They were large and limited to basic calculations. They consisted of large devices like the vacuum tubes. The input method of these computers was a ***machine language*** known as the 1GL or the first-generation language.

The data was entered using physical methods such as ***punch cards, paper tape, and magnetic tape*** into those computers.

Examples of the first generation computers include ENIAC, EDVAC, UNIVAC, IBM-701, and IBM-650. J.P.Eckert and J.W.Mauchy invented the first successful electronic computer called ENIAC, that stands for “***Electronic Numeric Integrated and Calculator***”.

These computers were large and very unreliable. They would heat up and frequently shut down and could only be used for very basic computations.

Advantages:

1. They were made up of vacuum tubes which were the only electronic component available during those days.
2. These computers could do calculations in milliseconds.

Disadvantages:

1. Heavy, Bulky, and very big in size, weighed about 30 tons.
2. These computers were very costly.
3. The vacuum tubes required a large cooling system.
4. Limited programming capabilities
5. Large amount of energy consumption.
6. Not reliable and frequent maintenance was required.

2 Second generation:

Second generation Computers were developed between 1957-1963. These computers were more reliable and in place of vacuum tubes, they used transistors (semiconductors). Smaller in size compared to first generation computers. The input for these computers were higher level languages like COBOL, FORTRAN etc.

- Second generation computers were first to store instructions in memory, which moved from a magnetic drum to magnetic core technology.
- Second generation computers were first developed for the atomic energy industry.
- Examples of the second-generation computers include IBM 1620, IBM 7094, Honeywell 400, CDC 1604, CDC 3600, UNIVAC 1108. They were faster than their predecessors.

Advantages:

1. Due to the presence of transistors instead of vacuum tubes, the size of electronic components decreased.
2. Less heating effect coz. energy consumed was less as compared to 1st generation of computers.
3. Assembly language and punch cards were used as input.
4. Low cost than first generation computers.
5. Better speed, could calculate in microseconds.
6. Better portability as compared to first generation

Disadvantages:

1. A cooling system was required.
2. Constant maintenance was required.
3. Only used for specific purpose.

3 Third Generation of Computers:

Computers developed in the third generation period of 1964 – 1971 differed from the first and the second generations by the fact that a new circuit element like IC's (Integrated Circuits) were used. An integrated circuit is a small device that can contain thousands and thousands of devices like transistors, resistors and other circuit elements that make up a computer. Robert Noyce and Jack Kilby in 1958-1959 invented Integrated Circuit or the IC chips. With the invention of IC's, it became possible to fit thousands of circuit elements

into a small region and hence the size of the computers eventually became smaller and smaller. Computers that used IC's during that period were, PDP-8 (Personal Data Processor), PDP-11, ICL 2900, IBM 360 series, IBM 370/168, Honeywell-6000 series.

- These computers had few megabytes of main memory and magnetic disks which could store few tens of megabytes of data per disk drive.
- High level programming languages like COBOL and FORTRAN were standardized by ANSI
- Some more high level programming languages like PL/I PASCAL and BASIC were introduced at this time.
- Third generation computers were the first to implement time sharing operating systems.
- Input to these computers could now be provided using keyboards and mouse.

Advantages:

1. Faster than second generation computers and could perform 1 million transactions per second.
2. Smaller, cheaper and more reliable than their predecessors.
3. These computers had faster and larger primary memory and secondary storage.
4. Widely used for scientific as well as business applications.
5. During this generation of computers, standardization of existing high-level languages and invention of new high-level languages was done.
6. Had time sharing operating system which allowed interactive use of computer by one or more users simultaneously thereby improving the productivity of the users.

Disadvantages:

1. IC chips were difficult to maintain.
2. Highly sophisticated technology was required for the manufacturing of IC chips.
3. A cooling system was required.

4 Fourth Generation of Computers:

Fourth Generation of computers were developed between 1972 – 1989. These computers used VLSI technology or the Very Large Scale Integrated (VLSI) circuits technology and were based on microprocessor. Intel was the first company to develop a microprocessor. A microprocessor is used in a computer for any logical and arithmetic operation and to perform execute any program. Graphics User Interface (GUI)

technology used provided more comfort to users.

The first “personal computer” or PC developed by IBM, belonged to this generation. VLSI circuits had almost about 5000 transistors on a very small chip and were capable of performing many high-level tasks and computations. These computers were thus very compact and thereby required a small amount of electricity to run.

Examples, STAR 1000, CRAY-X-MP (Supercomputer), DEC 10, PDP 11, CRAY- This generation of computers had the first “supercomputers” that could perform many calculations accurately.

They were also used in networking and used higher and more complicated languages as their inputs. The computer languages like languages like C, C+, C++, DBASE etc. were the input for these computers.

Advantages:

1. Fastest in computation and size was reduced as compared to the previous generation of computer.
2. Heat generated was negligible.
3. Small in size as compared to previous generation computers.
4. Required less maintenance.
5. All types of high-level language could be used in these types of computers.

Disadvantages:

1. The Microprocessor design and fabrication were very complex.
2. Cooling system was required in many cases due to the presence of ICs.
3. Advance technology was required to make the ICs.

5 Fifth Generation of Computers:

This is the present generation of computers and is the most advanced one. The generation began around 1990 and is the present generation of computers. The methods of input include the modern high-level languages like Python, R, C#, Java etc. The aim of the fifth generation is to make a device, which could respond to natural language input and was capable of learning and self-organizing.

This generation is based on ULSI (Ultra Large Scale Integration) technology resulting in the production of microprocessor chips having millions and billions of electronic components.

Examples include: Intel P4, i3 – i10, AMD Athlon, Desktop, Laptop, NoteBook etc.

Advantages:

1. They are most reliable and work faster.
2. They are available in different sizes and unique features.
3. They come with more user-friendly interfaces with multimedia features.

Disadvantages:

1. They tend to be sophisticated and complex tools.
2. They give more power to companies to watch what you are doing and even allow malicious users to infect your computer.
3. They may make the human brain dull.

Computer System Memory Hierarchy

Memory Hierarchy Design is divided into two main types:

- a) **External Memory or Secondary Memory** Comprising of Magnetic Disk, Optical Disk, Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via I/O Module.
- b) **Internal Memory or Primary Memory** Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.

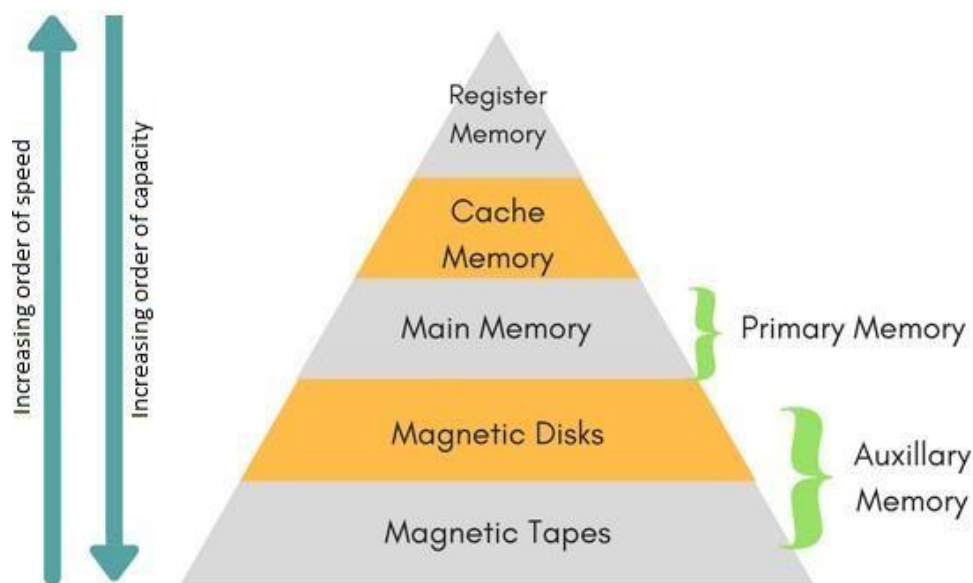


Fig. 1 Memory hierarchy

The following characteristics of Memory Hierarchy can be made from above Fig. 1:

- i. **Capacity:** It is the total volume of information the memory can store. As we move from top to bottom in the hierarchy, the capacity increases.
- ii. **Access Time:** It is the time interval between the read/write request and the availability of the data. As we move from bottom to top in the hierarchy, the access time decreases.
- iii. **Performance:** Earlier computer system were designed without Memory Hierarchy, due to large difference in access time the speed gap increased between the CPU registers and Main Memory, which resulted in lower performance of the system and thus, enhancement was required. The performance of the system increased by minimizing the levels of memory hierarchy to access & manipulate data.
- iv. **Cost per bit:** As we move from bottom to top in the Hierarchy, the cost per bit increases i.e. Internal Memory is costlier than External Memory.

Functional components of a Digital Computer

Von Neumann proposed design **architecture** for an **electronic digital computer** with the following basic functional components as shown in Fig. 2.

- **Input-Output** unit
- **Main Memory** that stores data and instructions
- **Processing unit** contains arithmetic logical unit and processor registers
- **Control unit** that contains an **instruction register** and **program counter**
- **Secondary** device.

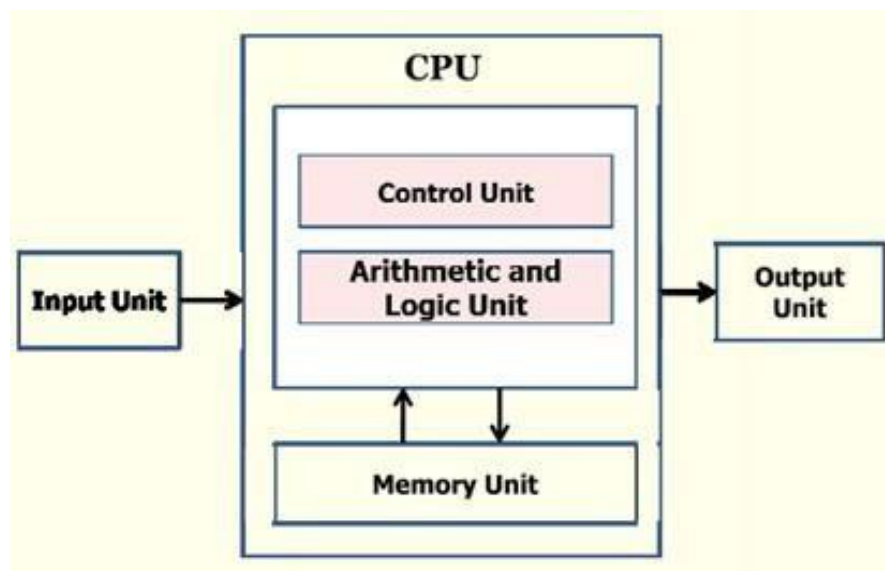


Fig. 2 Components of a Digital Computer

- **Input Unit:** The input unit consists of input devices that are attached to the computer. These devices take input and convert it into binary language that the computer understands. Some of the common input devices are keyboard, mouse, joystick, scanner etc.
- **Central Processing Unit (CPU):** Once the information is entered into the computer by the input device, the processor processes it. The CPU is called the brain of the computer because it is the control center of the computer. It first fetches instructions from memory and then interprets them so as to know what is to be done. If required, data is fetched from memory or input device. Thereafter CPU executes or performs the required computation and then either stores the output temporarily in the memory or displays on the output device. The CPU has three main components which are responsible for different functions – Arithmetic Logic Unit (ALU), Control Unit (CU) and Memory registers.
- **Arithmetic Logical Unit (ALU):** The ALU, as its name suggests performs mathematical calculations and takes logical decisions. Arithmetic calculations include addition, subtraction, multiplication and division. Logical decisions involve comparison of two data items to see which one is larger or smaller or equal.
- **Control Unit:** The Control unit coordinates and controls the data flow in and out of CPU and also controls all the operations of ALU, memory registers and also input/output units. It is also responsible for carrying out all the instructions stored in the program. It decodes the fetched instruction, interprets it and sends control signals to input/output devices until the required operation is done properly by ALU and memory.
- **Memory Registers:** A register is a temporary unit of memory in the CPU. These are used to store the data that is directly used by the processor. Registers can be of different sizes(16 bit, 32 bit, 64 bit and so on) and each register inside the CPU has a specific function like storing data, storing an instruction, storing address of a location in memory etc.
- **Memory :** Memory attached to the CPU is used for storage of data and instructions and is called internal memory. The internal memory is divided into many storage locations, each of which can store data or instructions. Each memory location is of the same size and has an address. With the help of the

address, the computer can read any memory location easily without having to search the entire memory. When a program is executed, its data is copied to the internal memory and is stored in the memory till the end of the execution. The internal memory is also called the Primary memory or Main memory. This memory is also called as RAM, i.e. Random Access Memory.

- **Output Unit** : The output unit consists of output devices that are attached with the computer. It converts the binary data coming from CPU to human understandable form. The common output devices are monitor, printer, plotter etc.

Computer Memories

- Computer memory is an internal storage area used to store data and programs.
- Computer memory can be categorized into two major types which can be used to **store data** and **programs**.
 1. Primary or Main memory
 2. Secondary or Auxiliary memory
- While the main memory holds instructions and data when a program is executing, the auxiliary or the secondary memory holds data and programs not currently in use and provides long-term storage.
- The primary memory is volatile, so the data can be retained in it, only when the power is on. Moreover, it is very expensive and therefore limited in capacity.
- The secondary memory stores data or instructions permanently, even when the power is turned off. It is cheap and can store large volumes of data. Moreover, data stored in auxiliary memory is highly portable, as the users can easily move it from one computer to the other.
- The only drawback of secondary memory is that data can be accessed from it at a very slow speed as and when compared with the data access speed of primary memory.
- A memory is just like a human brain. The memory is divided into large number of small parts called cells. Each location or cell has a unique address.

For example, *if the computer has 64k words, then this memory unit has $64 * 1024 = 65536$ memory locations.*

*The address of these locations varies from **0 to 65535**.*

1. Primary or Main memory

- Primary memory holds only those data and instructions on which the computer is currently working.
- It stores **instructions** and **data** of a program in execution. It is **volatile in nature**, it can retain content as long as power is ON. It is **expensive** and also limited in **capacity** as compared to secondary memory i.e. it stores only the relevant data or instructions currently required for a program or application.
- They are made up of **semiconductors**. **Very fast** compared to secondary memory but **slower** than Internal CPU memory (**Registers**).
- Examples include RAM, ROM, etc and can vary in a size from 4 to 16 GB.

Characteristics of Main Memory:

- These are semiconductor memories.
- It is known as the main memory.
- A computer cannot run without the primary memory.
- Usually volatile memory.
- Data is lost in case power is switched off.
- It is the working memory of the computer.
- Faster than secondary memories.

RAM is of two types :-

- Static RAM (SRAM)
- Dynamic RAM (DRAM)

Static RAM (SRAM): Flip Flop (latch) is used to represent 1 and 0's.

The word static indicates memory **refresh** is not required, it retains memory contents as long as power is being supplied.

These are popular because of the following reasons:

- **Simpler Design**
- **Faster** and used as **cache memory**
- **Highly Reliable**
- **Expensive** and Long life

The Fig.3 illustrates a static ROM chip. They find applications in **digital Cameras**, high end **mobile phones**.



Fig. 3 Static RAM

Dynamic RAM (DRAM):

It is made up of combinations of **Capacitor** and **Transistor**. The Capacitor gradually **discharges** and must be continually **refreshed** in order to maintain the data. This is achieved by a **refresh circuit** that rewrites the data several times per second. It is used as **RAM** as **shown in Fig.4**.

The following are the reasons it is preferred over static RAM:

- It is **cheaper, smaller, higher capacity**, forms major part of main memory.
- It has **Short** data lifetime
- **Slower** as compared to **SRAM**
- **Smaller/Denser** in size, **Less expensive**
- **More power** consumption



Fig. 4 DRAM

2. Secondary Memory or Auxiliary memory

It is used to store **Data** and **programs** that are not currently in use or not immediately required by the program. It stores the information permanently as long as user desires and is not dependent on **power** unlike primary memory. **It's Cheaper** and can store **very large** quantity of data. Memory is organized into a number of partitions called as **cells** just like housing colony. Each cell has a unique **address** that can be used to identify and is used to store data permanently. The examples include **Hardisk, Pen Drives, CD-ROM, DVD, Magnetic Tapes** etc.

Read Only Memory

ROM stands for **Read Only Memory** Data stored in ROM also known as **firmware**. Generally used in PC to store instructions needed to **Boot-up** a computer. **Non-volatile**. Information is **programmed/stored permanently** in such memories ROM extensively used in electronic items like **washing machine, Dishwasher, Cameras** and **microwave oven** among others. Stores BIOS program used in the **Computer Booting Process**.

Types of ROMs and their characteristics

a. MROM (Masked ROM)

These are **Hard-wired** devices that contain pre-programmed set of **data** or **instructions** supplied by the manufacturers. These programs once written cannot be changed.

MROM are used in electronic devices like Washing machine, Smart TV's, Microwave, refrigerator etc.

b. PROM (Programmable Read Only Memory)

PROM can be modified only once by the user/manufacturer. It can be **programmed only once** and is **not erasable**. They are used in devices such as cell phones, RFID tags, medical devices etc.

c. EPROM (Erasable and Programmable Read Only Memory)

EPROM can be erased by **ultra-violet light**. These can be modified by writing many a times by erasing and writing the contents. These are used in smart memory, memory sticks etc.

d. EEPROM (Electrically Erasable and Programmable Read Only Memory)

- EEPROM is **programmed** and **erased electrically**. It can be erased and reprogrammed about **ten thousand times**. **Selective location** can be modified and programmed. They can be **erased one byte** at a time, rather than the entire chip. These find applications in flash memory, remote keyless systems, microcontrollers etc.

Cache Memory

- Cache memory is a very high-speed semiconductor memory which can speed up the CPU.
- It acts as a buffer between the CPU and the main memory.
- It is used to hold those parts of data and program which are most frequently used by the CPU.
- The parts of data and programs are transferred from the disk to cache memory by the operating system, from where the CPU can access them.
- Cache memory is basically a portion of memory made of high-speed static RAM (SRAM) instead of the slower and cheaper dynamic RAM (DRAM) which is used for main memory.

It's the special memory having speed of access time much faster than RAM and hence is suitable for CPU to synchronize. It's made of **SRAM** (Static RAM) and is smaller and

much expensive than RAM. It is placed **between CPU and RAM**, as shown in Fig. 5, which acts like a buffer between CPU and RAM. CPU **looks** for Instructions and Data **first** in the **Cache memory** then into **RAM**. The idea is to fetch **most frequently accessed data and instructions** so that they are immediately made available to CPU.

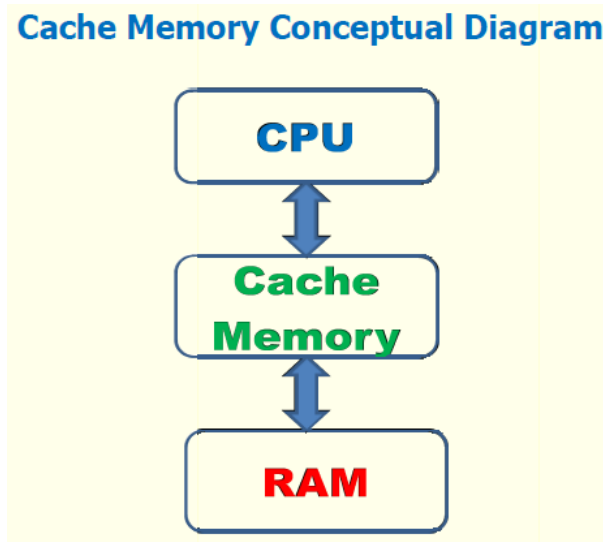


Fig. 5 Cache Memory

Advantages of cache memory –



1. Cache memory is faster than main memory.
2. It consumes less access time as compared to main memory.
3. It stores the program that can be executed within a short period of time.
4. It stores data for temporary use.

Disadvantages –

1. Cache memory has limited capacity.
2. It is very expensive.

SEQUENTIAL AND RANDOM ACCESS:

Memory devices can be accessed either randomly or sequentially.

Sequential access	Random access
Data is read sequentially in a specified order. 	Data is read in an arbitrary manner. 
If the 99th record is desired after the 1st one, then all the records have to be traversed to reach the desired one. Therefore, the time required to return data can vary depending on the position of the record.	Random access always returns data in constant time.
Sequential access devices can store a large number of records at a very low cost.	Random access devices are expensive than sequential access devices.
Magnetic tapes support sequential access.	RAM (Random Access Memory) supports random access.
Sequential-access is faster if records are to be accessed in the same order.	Random-access is faster if records are to be accessed in a random order.

Computer - Memory Units

Memory unit is the amount of data that can be stored in the storage unit. This storage capacity is expressed in terms of Bytes.

1. **Bit (Binary Digit):** A binary digit is logical 0 and 1 representing a passive or an active state of a component in an electric circuit. A bit is a binary digit, the smallest unit of data on a computer. A bit can hold only one of two values: 0 or 1, corresponding to the electrical values of off or on, respectively.
2. **Nibble:** A group of 4 bits is called nibble.
3. **Byte:** A group of 8 bits is called byte. A byte is the smallest unit, which can represent a data item or a character.
4. **Word:** A computer word, like a byte, is a group of fixed number of bits processed as a unit. The length of a computer word is called word-size or word length. It may be as small as 8 bits or may be as long as 96 bits.

Unit	Equivalent
1 kilobyte (KB)	1,024 bytes
1 megabyte (MB)	10^6 bytes or 10^3 KB
1 gigabyte (GB)	10^9 bytes or 10^3 MB
1 terabyte (TB)	10^{12} bytes or 10^3 GB
1 petabyte (PB)	10^{15} bytes or 10^3 TB
1 exabyte (EB)	10^{18} bytes or 10^3 PB
1 zettabyte (ZB)	10^{21} bytes or 10^3 EB
1 yottabyte (YB)	10^{24} bytes or 10^3 ZB

Practice Problems:

Q 1: Karan has 600 MB of data. Kiran has 2000 MB of data. Will it all fit on Alice's 4 GB pen drive?

Sol: Yes it fits: 600 MB + 2000 MB is 2600 MB. 2600 MB is 2.6 GB, so it will fit on the 4 GB drive no problem. Equivalently we could say that the 4 GB drive has space for 4000 MB.

Q2: Janahavi has 100 small images, each of which is 500 KB. How much space do they take up overall in MB?

Sol: 100 times 500 KB is 50000 KB, which is 50 MB.

Q3: Assume a group is recording the sound inside a haunted House for 20 hours as MP3 audio files. About how much data will that be, expressed in GB?

Sol: MP3 audio takes up about 1 MB per minute. 20 hours, 60 minutes/hour, $20 * 60$ yields 1200 minutes. So that's about 1200 MB, which is 1.2 GB.

Similarly, one 1 GB is 1,024 MB, or 1,073,741,824 ($1024 \times 1024 \times 1024$) bytes. A terabyte (TB) is 1,024 GB; 1 TB is about the same amount of information as all of the books in a large library. A petabyte (PB) is 1,024 TB. 1 PB of data, if written on a stack of CDs would create a stack of height a mile high. An exabyte (EB) is 1,024 PB. A zettabyte (ZB) is 1,024 EB. Finally, a yottabyte (YB) is 1,024 ZB.

Computer languages

- Programming language is a language specifically designed to express computations that can be performed using the computer.
- These languages are used to create programs that are used to give the instructions.
- A programming language is a computer language that is used by programmers to communicate with computers.
- Programs is a set of instructions that can be executed by a computer to perform a specific task.

Computer languages can be classified into three types:

- Machine Language
- Assembly Language
- High Level Language

1. Machine Level Language (MLL)

- Program is a set of **instructions** which are **executed** by the **CPU** using **data** to achieve an objective. CPU **fetches** an instruction, **decodes** or figures it out and then **executes** the instruction.
- Instructions are written in terms of 1s and 0s only i.e in Binary meaning ON/OFF, where ON represents 1 and 0 indicates OFF.
- These MLL differs from Computer to Computer based on **hardware architecture**. MLL is **Fastest** in execution & Efficient Code can be written. Extremely **difficult** to write and can make **mistakes**. MLL does not need a translator because it is already in the machine-readable form.

For example: To representation 80 in the computer system it is represented as

1010000. So, it is very difficult to learn. To overcome this problem the assembly language was invented.

2. Assembly Level Language (ALL)

- Assembly language is easier to learn than Machine language.
- Assembly language uses mnemonics to represent data such as **Mov, Add, Sub, End**, etc.
- Assembly language is easy to understand and readable by human being as compared to machine language.
- Modifications and error fixing can be done in assembly language.
- It is slower in execution as compared to machine language.

For example

Stop 0000

Start 0001

Error 1111

Add 1010

- It is easy to remember the assembly language because some alphabets and mnemonics are used. Start, Stop, Error, Add are all examples of Mnemonics.
- System software Assembler is used to convert mnemonics into machine-understandable form. It is highly dependent on the machine architecture.

3. High level language

High level language uses English like language and is simpler to write a program compared to Assembly or Machine language. It is programmer friendly language. Knowledge of Hardware is not required. It is easy to understand and debug. It needs software like compiler or interpreter for translation. It is used widely for programming

and is faster in development and completion of coding tasks. It is easy to maintain and is portable i.e. can run or execute on any platform as it is independent of hardware architecture as shown in fig. 6.

Examples: **FORTRAN, COBOL, C, C++, JAVA** etc

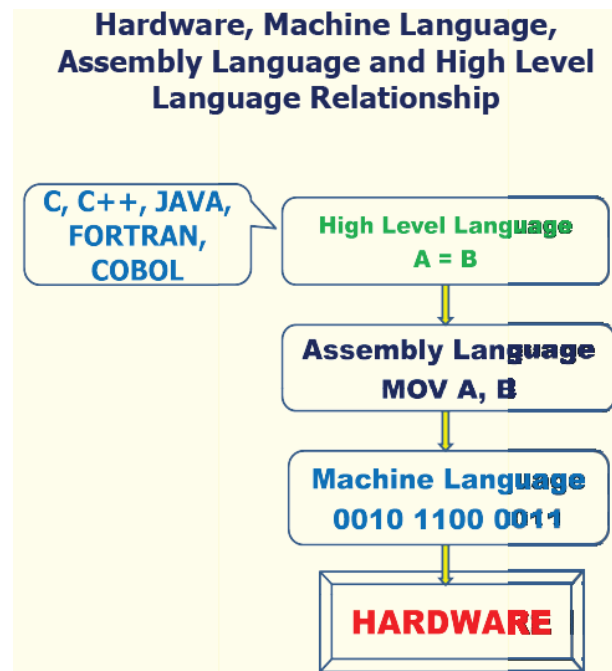


Fig. 6 Relationship between HLL, ALL and MLL

Translators

- Assembler
- Compiler
- Interpreter
- Linker Loader

Computers understand only machine language. Support system software's to accept a program in high level language and help it execute on a given machine.

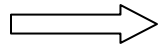
1. Assembler

System program that transforms code in Assembly language to Machine language

(object code)

Assembly Language

MOV A, B



Machine Language

0010 1100 0011

2. Compiler

Translates a computer program written in a **high level programming** language to an **executable code** for a target machine. i.e. **Source Code** to **Assembly or Machine code**.

High Level Language

A = B

Assembly Language

MOV A, B

Machine Language

0010 1100 0011

Examples of compilers: GNU or Turbo C Compiler

3. Interpreter

Reads a program in high level language and executes the instructions **without** having to **compile** the program first. Takes a **line of source code** and executes that line. Then proceeds to do the same with the subsequent line.

Examples: **Python, PERL, BASIC** languages

Compiler	Interpreter
Translates the entire program into machine code and then allows linking and loading for execution.	Converts one line of the source program and then executes that line. Proceeds to the next statement in this manner.
Compiled program runs faster	Comparatively slower.

Produces complete error/warning list	Halts at the first error.
Compile once run many times	Source code required each time.
Ex: C, C++	JAVA, Python etc.

Linker and Loader

Linker

Linker is alternatively referred as binder, link editor. Assume, we write some code that uses built in function **POW** to compute power (x^y ie say some base **X** raise to exponent **Y**, the code for which already exists in the header file <math.h>.

Now, during compilation, Linker joins or patches your code with the externally available code of **POW** in the libraries. Then it combines the output of a compiler or assembler to a single executable file.

Loader

Loader is program part of O.S that loads the executable program from the secondary memory into the RAM (Main Memory) for the CPU to execute the code. It verifies memory permissions availability etc and then hands over to the CPU the starting address for execution as shown in the fig. 7.

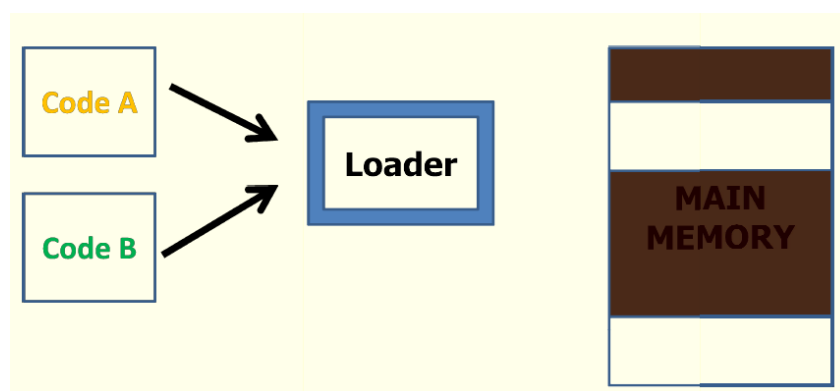


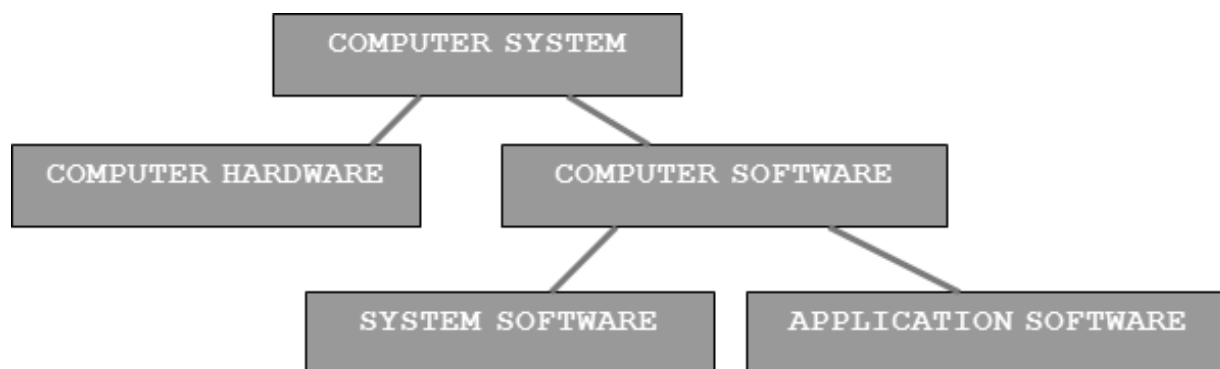
Fig. 7 Loader: Loads the programs combined by linker into main memory

Software & Hardware

A Computer is an electronic device that can perform various operations & calculations at a greater speed than an ordinary machine or human mind.

- The computer hardware cannot think and make decisions on its own.
- The hardware needs a software (a set of programs) to instruct what has to be done.
- A program is a set of instructions that is arranged in a sequence to guide a computer to find a solution for the given problem.
- The process of writing a program is called programming.

It comprises of many physical and tangible components that we can touch or feel, called the **Hardware** and programs and commands that drive the hardware, called the **Software**.



The Software is the set of instructions fed in form of programs to manage the computer system and control the hardware components. For example:

- The antivirus that protects our computer system is a type of Software.
- The media players that we use to play multimedia files such as movies, music etc. are Software.
- The Microsoft Office we use to edit the documents is Software.

Depending on its function and application, Software can be divided into three major types:

a. System Software and application software

System Software	application software
------------------------	-----------------------------

It is a collection of programs that enable the users to interact with hardware components efficiently	It is a collection of programs written for a specific application. Like, we have library system, inventory control system, etc
It controls and manages the hardware	It uses the services provided by the system software to interact with hardware components
System software is machine dependant	It is machine independent
The programmer must understand the architecture of the machine and hardware details to write a system software	The programmer ignores the architecture of the machine and hardware details to write an application software
Interacts with the hardware directly	Interacts with the hardware indirectly through system calls provided by system software
Writing a system software is a complicated task	Writing application programs is relatively very easy
Example: compiler, operating system	Example: MS-WORD, PAINT

a. System Software

These are the software that directly allows the user to interact with the hardware components of a computer system. As the humans and machines follow different languages, there has to be an interface that will allow the users to interact with the core system, this interface is provided by the software. The system software is the most important part of the a computer system that controls the hardware. This System Software can be further divided into four major types:

- i. **The Operating System** – It is the main program that manages and controls the inter-action of the components of a computer system. For ex., Microsoft Windows, Linux, Mac OS etc. are vendors who supply the Operating System (OS).
- ii. **The Language Processor** – The hardware components present in the computer system does not understand human language. There are three types of languages involved in the world of human-machine interaction:
 - **Machine-Level Language:** The machines only understand the digital signals or the binary codes or the binary language which consist of 0's and 1's. These are totally machine dependent language.
 - **Assembly-Level Language:** These are the Low-Level Languages (LLL), that forms a correspondence between machine level instruction and general assembly level instructions. Assembly language uses a mnemonics to represent each low-level machine instruction or operation-code also called the op-codes. For ex. ADD instruction is used to add two quantities, the HALT instruction is used to stop a program etc. It is a machine dependent language and varies from processor to processor.
 - **High-Level Language:** These are the simple English like statements, that humans use to program and code as it is easy to read and understand by the humans. For ex., Java, C, C++, Python etc.

The machine level language is very complex to understand and code, therefore the users prefer the *High-Level Language* or the *HLL* for coding. These codes need to be converted into the machine language so that the computer can easily understand and work accordingly. This operation is performed by the Language Processor which is made up of further three components:

- **Assembler:** Language processor used to convert the assembly language into machine level language.
 - **Compiler:** Language processor used to convert High-Level Language into machine level language at once, thus execution time is fast. The error detection is difficult in a compiler. Programming Languages like C, C++ and Simula use compilers.
 - **Interpreter:** Language processor used to convert High-Level Language into machine level language line-by-line, thus execution time is slow. Error-detection is easier in an interpreter as it reports as soon as a bug (error) is caught and restarts the process. Programming Languages like Python, Ruby, Visual Basic and Java uses an interpreter.
- iii. **The Device Drivers** – The device drivers are the system software programs that act as an interface between the various Input-Output devices and the users or the operating system. For ex., the Printers, CD-ROM, DVD, Web cameras come with a driver disk that is needed to be installed into the system to make the device run in the system.
- iv. **The BIOS** – *It stands for Basic Input Output System* and is a small firmware that controls input-output devices attached to the system. This software is also responsible for starting the OS or initiating the booting process.

Operating System overview

An operating system (OS) is a system software program that controls the system's hardware and that interacts with the user and application software. OS is the computer's master control program. It provides the tools (commands) that enable users to interact with the PC. When a command is entered, the OS translates it into code that the machine can understand. OS also ensures that the results are displayed on screen, printed, and so on.

- The primary goal of an operating system is to make the computer system *convenient and efficient to use*.
- An operating system ensures that the system resources (such as CPU, memory, I/O devices, etc) are utilized efficiently.

For example, there may be many service requests on a web server and each user request need to be serviced. Similarly, there may be many programs residing in the

main memory.

Therefore, the system needs to determine which programs are active and which need to wait for some I/O operation.

The Operating System performs the following functions:

- Displays the on-screen elements with which user can interact.
- Loads programs (such as word processing and spreadsheet programs) into the computer's memory to be used by the user.
- Coordinates programs working with the computer's hardware and other software.
- Manages the way information is stored on and retrieved from disks.

Operating systems can be organized into four major types: real-time, single user/single-tasking, single-user/multitasking, and multi- user/multitasking.

I. Real-time operating system (RTOS):

A RTOS is a very fast, relatively small OS. Real-time OS are often the embedded OS, they are built into the circuitry of a device and are not loaded from a disk drive. A real-time operating system is needed to run real-time applications and they responds to certain inputs extremely quickly—thousandths or millionths of a second (milliseconds or microseconds, respectively). Real-time applications are needed to run medical diagnostics equipment, life-support systems, machinery, scientific instruments, and industrial systems.

RTOS is used in the following make of mobiles:

- Symbian - Used in cell phones, mainly ones made by Nokia.

- Embedded Linux - Android is a subset, used in many devices like Samsung etc.
- BlackBerry OS - For BlackBerry phones.
- iOS - Subset of Mac OS X, used in Apple's mobile devices.

II. Single-User/Single-Tasking Operating Systems

An operating system that allows a single user to perform just one task at a time is a single-user/single tasking operating system. To a user, a "task" is a function such as printing a document, writing a file to disk, editing a file, or downloading a file from a network server. To the operating system, a task is a process, and small and simple OS can only manage a single task at a time. MS-DOS is one example of a single-tasking OS, and the Palm OS, used on the Palm handheld computers. Such operating systems are limited in characteristic and they take up very little space on disk or in memory while running and do not require a powerful and expensive computer.

III. Single-User/Multitasking Operating Systems

A single-user/multitasking operating system allows a single user to perform two or more functions at once. It takes a special operating system to keep two or more tasks running at once. The most commonly used personal computers usually run such OS, including Microsoft Windows and the Macintosh Operating System. The multitasking features of these OS have greatly increased the productivity of people in a large variety of jobs because they can accomplish more in a shorter period of time.

A disadvantage of a single-user/multitasking operating system is the increased size and complexity it needs to support multitasking.

IV. Multi-User/Multitasking Operating Systems

A multi-user/multitasking operating system is an operating system that allows multiple users to use programs that are simultaneously running on a single network server; called a ***terminal server*** as shown in Fig. 8. This is not at all the same as connecting to a network server for the sake of accessing files and printers. In multi-user OS, it gives each user a complete environment, called a user session, on the server. Each user's applications

run within their user session on the server separate from all other user sessions. The software that makes this possible is called a terminal client. In a multi-user/multitasking operating system environment, all or most of the processing occurs at the server. Examples of multi-user OS include UNIX, VMS (Virtual Memory System) used in older mid-range computers by Digital Equipment Corporation (DEC), and mainframe operating systems such as MVS (Multiple Virtual Storage) used in IBM Mainframe System/370 and System/390.

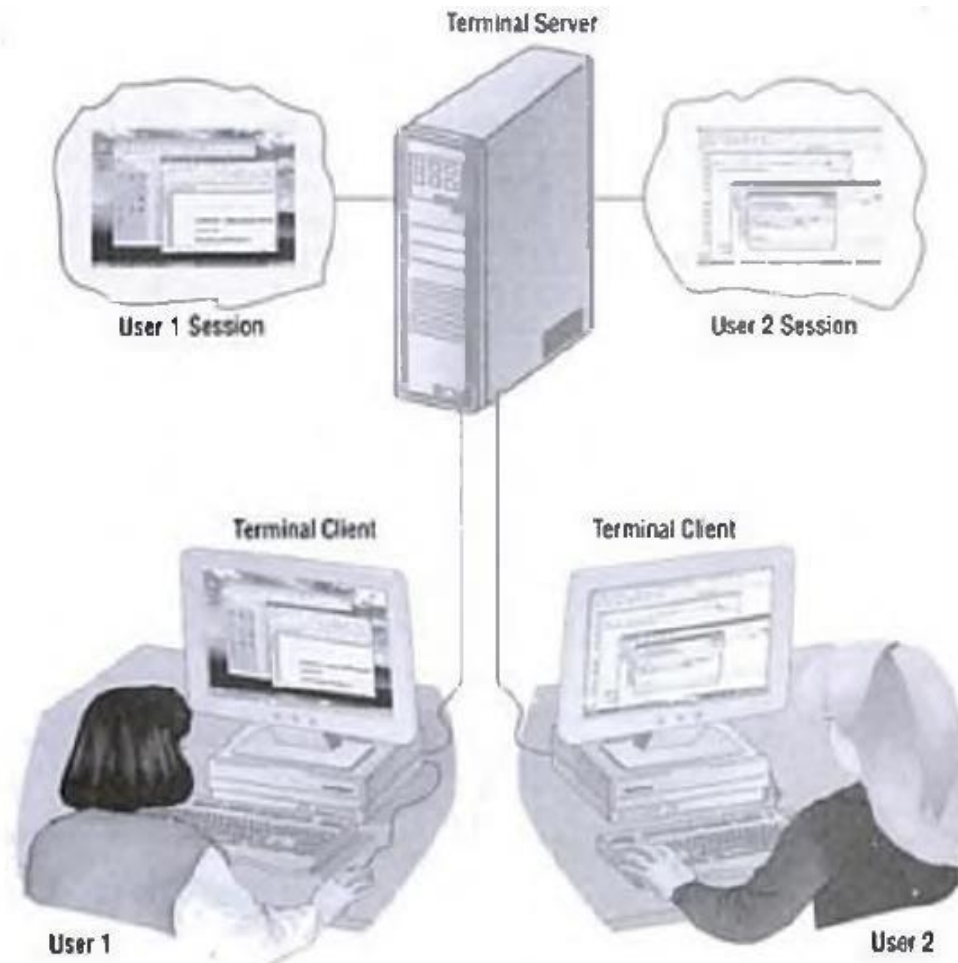


Fig. 8 Multi user/multi tasking operating each user runs his/her own session on the server

The advantage of these operating systems is that they can be managed by making changes to one server, rather than to many terminal clients. They also allow the user to work with applications that require a more powerful

computer. A disadvantage is that if the server goes down, then the user work is disrupted.

1. The small pictures on the desktop are called **icons** that represent links to resources on the PC or network.
2. **Multi-tasking** is the ability to perform two or more tasks at the same time.
3. You interact with a command-line interface by typing strings of characters at **Command Prompt**.
4. A running program may take up the whole screen or it may appear in a rectangular frame, called **Window**
5. A program that lets you back up data files is an example of **Utility Program**.
6. In a graphical user interface, you use a mouse (or other pointing device) to move a screen by **dragging** around the screen.
7. A **Operating System** supports an application that responds to certain input very, very quickly.
8. The process of moving from one open window to another is called **switching**
9. In Windows, when you right-click some objects on the screen, a special **menu** appears.
10. In a GUI **scrolling buttons** let you view parts of a program or file that do not fit in the window.

Computer Networks

A **computer network** is a system in which multiple **computers** are connected to each other to share information and resources. Connectivity may be established by the means of **Physical cables**, Optical wires or **Wireless** technologies. An example of computer network is shown in fig. 9. Communication between devices/computers also called nodes takes place by the means of established **rules** called as **Protocols**. Nodes may be **Computers, Printers, Scanners, Servers** etc. Resource on a computer network may be any device such as printers, scanners, modem or sharing software. Example of Resource sharing may be a **high-speed printer**.

Devices in a computer network are identified by a name and a unique address called a host address (IP Address) that helps in identifying and locating the device on a network.

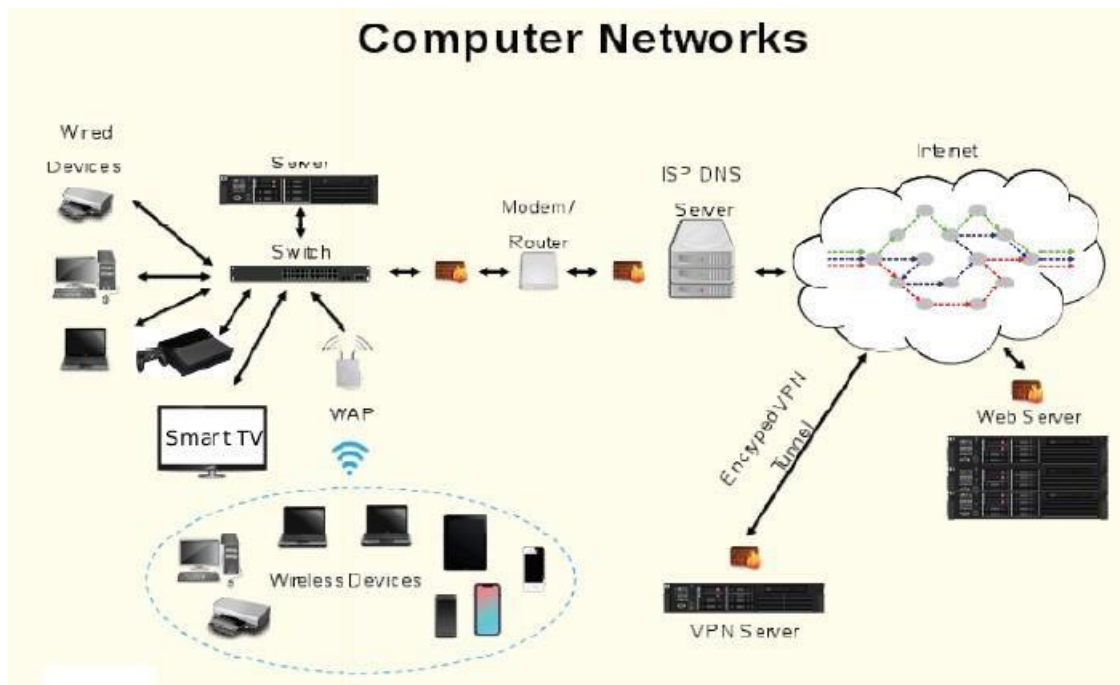


Fig. 9 Typical Computer Network Connectivity

Advantages of Comp. Networks

- File Sharing
- Resource Sharing
- Increased Storage Capacity
- Load Sharing
- Facilitate communications

I. Network Types

Broadly computer network can be classified into different types such as LAN and WAN.

a. Local Area Networks (LANs)

A local area network (LAN) is a data communication system consisting of several devices such as computers and printers. This type of network contains computers that are relatively near each other and are physically connected using cables, infrared links, or wireless media. A LAN can consist of just two or three PCs connected together to share

resources, or it can include hundreds of computers of different kinds. Any network that exists within a single building, or even a group of adjacent buildings, is considered a LAN.

A LAN is not a system that connects to the public environment (such as the Internet) using phone or data lines. It is often helpful to connect separate LANs together so they can communicate and exchange data. In a large company, for example, two departments located on the same floor of a building may have their own separate LANs, but if the departments need to share data, then they can create a link between the two LANs.

b. Wide Area Networks (WANs)

Typically, a wide area network (WAN) is two or more LANs connected together, generally across a wide geographical area separated by thousands of kilometers. For example, a company may have its corporate headquarters and manufacturing plant in one city and its marketing office in another. Each site needs resources, data, and programs locally, but it also needs to share data with the other sites. To accomplish this feat of data communication, the company can attach devices that connect over public utilities to create a WAN. (Note, however, that a WAN does not have to include any LAN systems. For example, two distant mainframe computers can communicate through a WAN, even though neither is part of a local area network.)

These remote LANs are connected through a telecommunication network (a phone company) or via the Internet through an Internet service provider (ISP) that contracts with the telecommunication networks to gain access to the Internet's backbone.

II. Hybrid Networks

Between the LAN and WAN structures, you will find hybrid networks such as campus area networks (CANs) and metropolitan area networks (MANs). In addition, a new form of network type is emerging called home area networks (HANs). The following section explains these networks.

a. Campus Area Networks (CANs)

A campus area network (CAN) follows the same principles as a local area network, only on a larger and more diversified scale. With a CAN, different campus offices and organizations can be linked together. For example, in a typical university setting, a bursar's office might be linked to a registrar's office. In this manner once a student has paid his or her tuition fees to the bursar, this information is transmitted to the registrar's system so the student can enroll for classes. Some university departments or organizations might be linked to the CAN even though they already have their own separate LANs.

b. Metropolitan Area Networks (MANs)

The metropolitan area network (MAN) are large-scale networks that connects multiple corporate LANs together. MANs usually are not owned by a single organization; their communication devices and equipment are usually maintained by a group or single network provider that sells its networking services to corporate customers. MANs often take the role of a high-speed network that allows for the sharing of regional resources.

c. Home Area Networks (HANs)

A home area network (HAN) is a network contained within a user's home that connects a person's digital devices, from multiple computers and their peripheral devices, such as a printer to telephones, VCRs, DVDs, televisions, video games, home security systems, "smart" appliances, fax machines, and other digital devices that are wired into the network.

III. Intranets and Extranets

The need to access corporate Web sites has created two classifications known as intranets and extranets.

Much of the technology available on the Internet is also available for private network use. The company's internal version of the Internet is called an intranet. An intranet uses the same Web server software that

gives the public access to Web sites over the Internet. The major difference is that an intranet usually limits access to employees and selected contractors having ongoing business with the company. Web pages have become very popular and, using integrated software, many operating systems offer complete Web server software or at least a personal Web server. This gives users the ability to create Web pages on their local computers that can be viewed by other members of the same network. Just, like on the Internet, users can allow others (again, usually employees) to browse their Web site and to upload or download files, video clips, audio clips, and other such media. Users also can set controls and limit who may access the Web site. Extranets are becoming a popular method for employees to exchange information using the company's Web site or e-mail while traveling or working from home.

An extranet is a partially accessible internal company Web site for authorized users physically located outside the organization. Whereas an intranet resides completely within the company's internal network and is accessible only to people that are members of the same company or organization, an extranet provides various levels of accessibility to outsiders. You can access an extranet only if you have a valid username and password, and your identity determines which parts of the extranet you can view.

Algorithms:

- An algorithm is a sequence of steps to solve a particular problem or algorithm is an ordered set of unambiguous steps that produces a result and terminates in a finite time.

OR

- Algorithm may be defined as a finite and ordered sequence of steps which when performed leads to the solution of a problem in a definite amount of time.
- A well-defined algorithm always provides an answer and is guaranteed to terminate.
- Algorithms are mainly used to achieve *software re-use*.

A good algorithm must have the following characteristics:

- Be precise
- Be unambiguous
- Not even a single instruction must be repeated infinitely
- After the algorithm gets terminated, the desired result must be obtained.

How/Why Algorithms?

Algorithms are usually written in combination of our spoken language and one or more programming languages, in advance of writing a program.

- Independent of Computer Languages
- Easily converted into Programs
- Step by Step Process Assists in error correction
- English and Math like

Algorithms vs. Programs

Both are sets of instructions on how solve a problem Algorithms

- In plain language understandable to People

Programs:

- Converting Algorithms into Instructions usually in a HLL understandable a computer(compiler).

Algorithms Terminology

1. Arithmetic Operators

+ - * / Mod or %

2. Relationship Operators

> < >= <= !=

3. Input/output

- Input, Read, Get, Scan (for taking Input)
- Output, Write, Show, Display, Print (for displaying result)

4. Constant vs. Variable

PI <- 3.142 (Constant)

Area <- PI * Radius * Radius (Variable)

5. Assignment vs. Equality

← Assignment Operator

= Equality (Comparison)

Say A is 10 and B is 20

$A \leftarrow B$ (Assignment operator: used to assign value to a variable)

Is $(A = B)$ (Equality operator used to compare two variables)

NOTE: WAA abbreviation for Write an Algorithm

The algorithm and flowchart include following three types of control structures:

1. Sequence: In the sequence structure, statements are placed one after the other and the execution takes place starting from up to down.

2. Branching (Selection): In branch control, there is a condition and according to a condition, a decision of either TRUE or FALSE is achieved. In the case of TRUE, one of the two branches is explored; but in the case of FALSE condition, the other alternative is taken.

3. Loop (Repetition): The Loop or Repetition allows a statement(s) to be executed repeatedly based on certain loop condition e.g. WHILE, FOR loops.

Advantages of algorithm

- It is a stepwise representation of a solution to a given problem, which makes it easy to understand.
- An algorithm uses a definite procedure.
- It is not dependent on any programming language, so it is easy to understand for anyone even without programming knowledge.
- Every step in an algorithm has its own logical sequence so it is easy to debug.

How To Write Algorithms:

Step 1: Define your algorithms input

Step 2: Define the variables

Step 3: Outline the algorithm's operations

Step 4: Output the results of your algorithm's operations

Algorithms based on Sequence:

Sequence means that each step of the algorithm is executed in the specified order.

Q: WAA to find the Sum of Two Numbers

START

1. $a \leftarrow 0, b \leftarrow 0, \text{sum} \leftarrow 0$
2. Read a, b
3. $\text{sum} \leftarrow a + b$
4. Write "Sum is: ", sum

STOP

Q: WAA to Exchange Two Numbers using third variable

START

1. a, b, temp
 2. Read a, b
 3. $\text{temp} \leftarrow a$
 $a \leftarrow b$
 $b \leftarrow \text{temp}$
 4. Write "After Exchange a is", a
 5. Write "After Exchange b is", b
- STOP

Algorithms based on Decision Making- Control Flow

Decision statements are used when the outcome of the process depends on some condition. For example, if $x=y$, then print "EQUAL". Hence, the general form of the if construct can be given as if condition then process.

1. If statement :

Syntax or General Format of the IF Construct

```
IF (CONDITION IS TRUE) THEN  
{
```

```
Statement 1
Statement 2
```

```
.....
Statement N
}
```

Q: WAA to ONLY print a message if GUESS number is 100

START

1. num ← 0

2. Read num

3. IF (num = 100) THEN

{

Print "You have Guessed the lucky Number"

Print "Congratulations !!!"

}

STOP

2. IF-ELSE statement:

IF – ELSE Construct Syntax:

```
IF (CONDITION_1 IS TRUE) THEN
{
Statement 1
Statement 2
.....
Statement N
}
ELSE
{
Statement A
Statement B
.....
Statement Z
}
```

Q: WAA to find if number is ODD or EVEN

```

START
1. num ← 0
2. Read num
3. IF (num mod 2 = 0) THEN
    {
    Print "Number is Even"
    }
ELSE
    {
    Print "Number is Odd"
    }
STOP

```

Q: WAA to calculate area ONLY of a valid Triangle using Herons Formula.

```

START
1. a, b, c, s, area
2. Read a, b, c
3. IF ( (a + b) > c AND (b + c) > a AND (a + c) > b) THEN
    {
    s (a + b + c)/2
    area ← SQRT(s*(s-a)*(s-b)*(s-c))
    Write "Area is:", area
    }
ELSE
    {
    Write "Invalid Triangle"
    }
STOP

```

3. IF-ELSEIF-ELSE statement:

Syntax:

```

IF (Condition_A is TRUE) THEN
{
Sa

```

```

Sb
.....
Sn
}
ELSE IF (Condition_B is TRUE) THEN
{
Sa
Sb
.....
Sn
}
ELSE
{
Na
Nb
.....
Nn
}

```

Q: WAA to find a number is Positive, Negative or Zero

START

1. num 0

2. Read num

3. IF (num = 0) THEN

{

Print "Number is ZERO"

}

ELSE IF (num < 0) THEN

{

Print "Number is Negative"

}

ELSE

{

Print "Number is Positive"

}

STOP

Algorithms based on Repetition

Repetition, which involves executing one or more steps for a number of times, can be implemented using constructs such as while, do-while, and for loops. These loops execute one or more steps until some condition is true.

1. while loop:

Syntax:

```
    WHILE (CONDITION_TRUE) DO
        Ia
        Ib

        ....
        In
    END-WHILE
```

Q: WAA to display Numbers from 1 to N

```
START
1. num 1, N
2. Get N // Last Number
3. WHILE (num <= N) DO
    Print num " "
    num num + 1
END-WHILE
STOP
```

2. Repeat...for Count

Syntax of the Repeat...for Count

// Count Up

```
REPEAT for count 1, 2, 3,...N
{
    Sa
    Sb
    ...
    Sn
```

}

OR

// Count Down

REPEAT for count N, N-1, N-2,...0

{

Sa

Sb

...

Sn

}

Q: WAA to find the sum of the series $1 + 3 + 5 + 7 + 9 \dots N$ for the first 30 terms

START

1. num 1, sum 0

2. REPEAT for count 1, 2, 3...30

{

sum sum + num

num num + 2

}

3. Write "Sum of 30 terms is:", sum

STOP

Q: WAA to determine if an input number is a perfect number. Ex: 6, 28

START

1. num, i 0, fact_sum 0

2. READ num

3. WHILE (i <= num /2) DO

IF (num mod i = 0) THEN

fact_sum fact_sum + i

END-IF

i i + 1

END-WHILE

4. IF (fact_sum = num) THEN

WRITE num, "is perfect"

ELSE

WRITE num, "is NOT perfect"

END-IF

STOP

Q: WAA to generate the Fibonacci Series 0,1,1,2,3,5,8..N. For N >= 1

START

1. first 0, second 1,next 1, N

2. READ N

3. WRITE first, second

4. WHILE (next <= N) DO

WRITE next

first second

second next

next first + second

END-WHILE

STOP




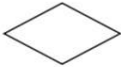
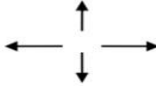

Flowcharts

- **“Flowchart** is diagrammatic /graphical representation of sequence of steps to solve a problem.”
Flowchart is a graphical or symbolic representation of a process.
- It is basically used to design and document virtually complex processes to help the viewers to visualize the logic of the process, so that they can gain a better understanding of the process.
- When designing a flowchart, each step in the process is depicted by a different symbol and is associated with a short description. The symbols in the flowchart are linked together with arrows to show the flow of logic in the process

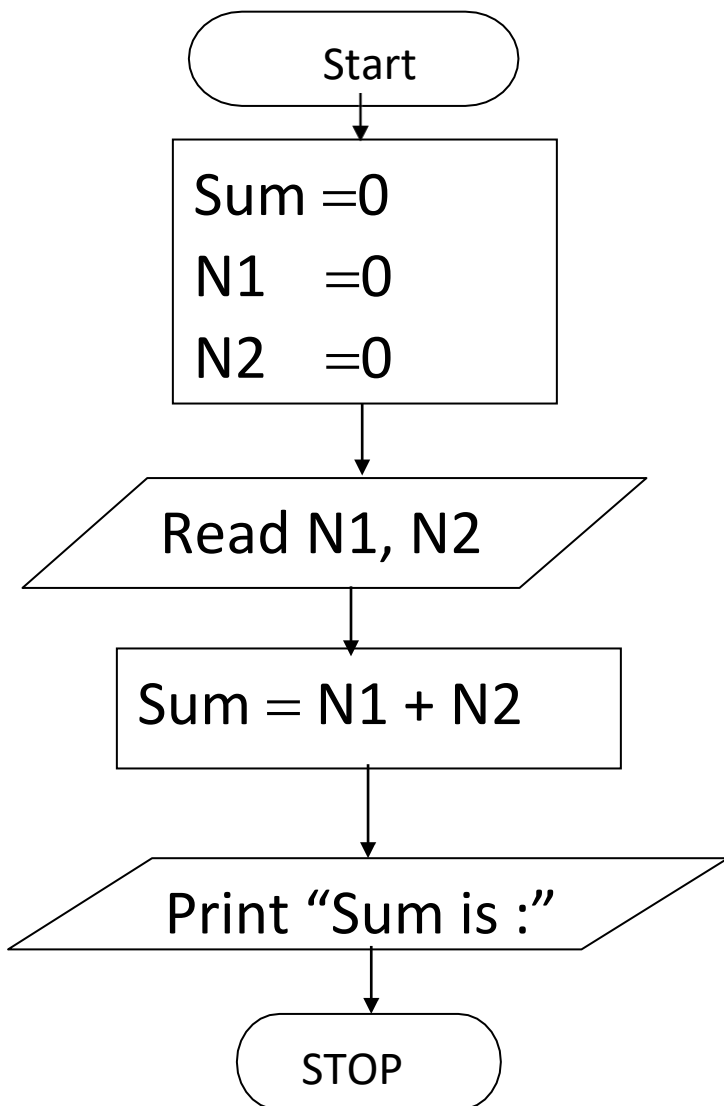
Advantages of flowchart:

- Flowchart is an excellent way of communicating the logic of a program.
- Easy and efficient to analyze problem using flowchart.
- During program development cycle, the flowchart plays the role of a blueprint, which makes program development process easier.
- After successful development of a program, it needs continuous timely maintenance during the course of its operation. The flowchart makes program or system maintenance easier.
- It is easy to convert the flowchart into any programming language code.

To draw a flowchart following standard symbols are use:-

Symbol Name	Symbol	function
Oval		Used to represent start and end of flowchart
Parallelogram		Used for input and output operation
Rectangle		Processing: Used for arithmetic operations and data-manipulations
Diamond		Decision making. Used to represent the operation in which there are two/three alternatives, true and false etc
Arrows		Flow line Used to indicate the flow of logic by connecting symbols
Circle		Page Connector

Q: Draw a flowchart to add two numbers?

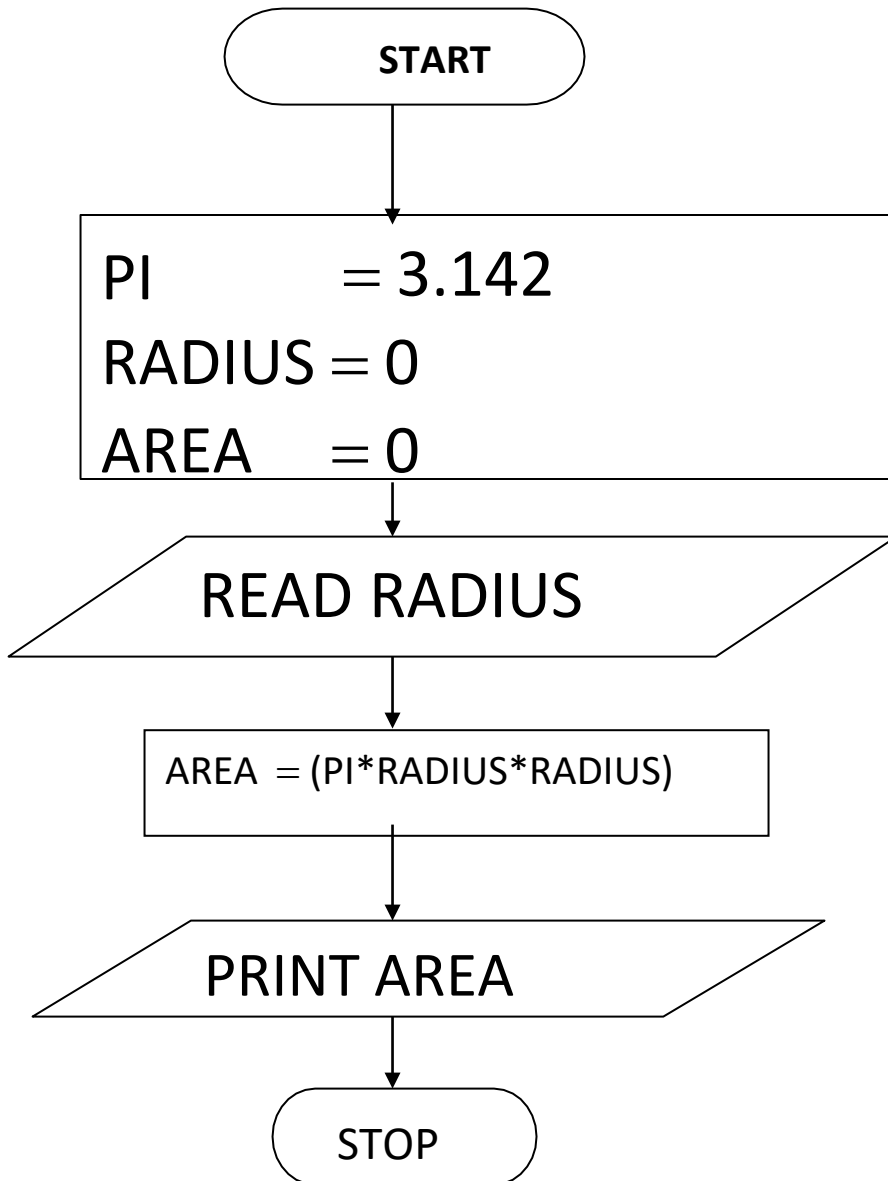


Sample Output

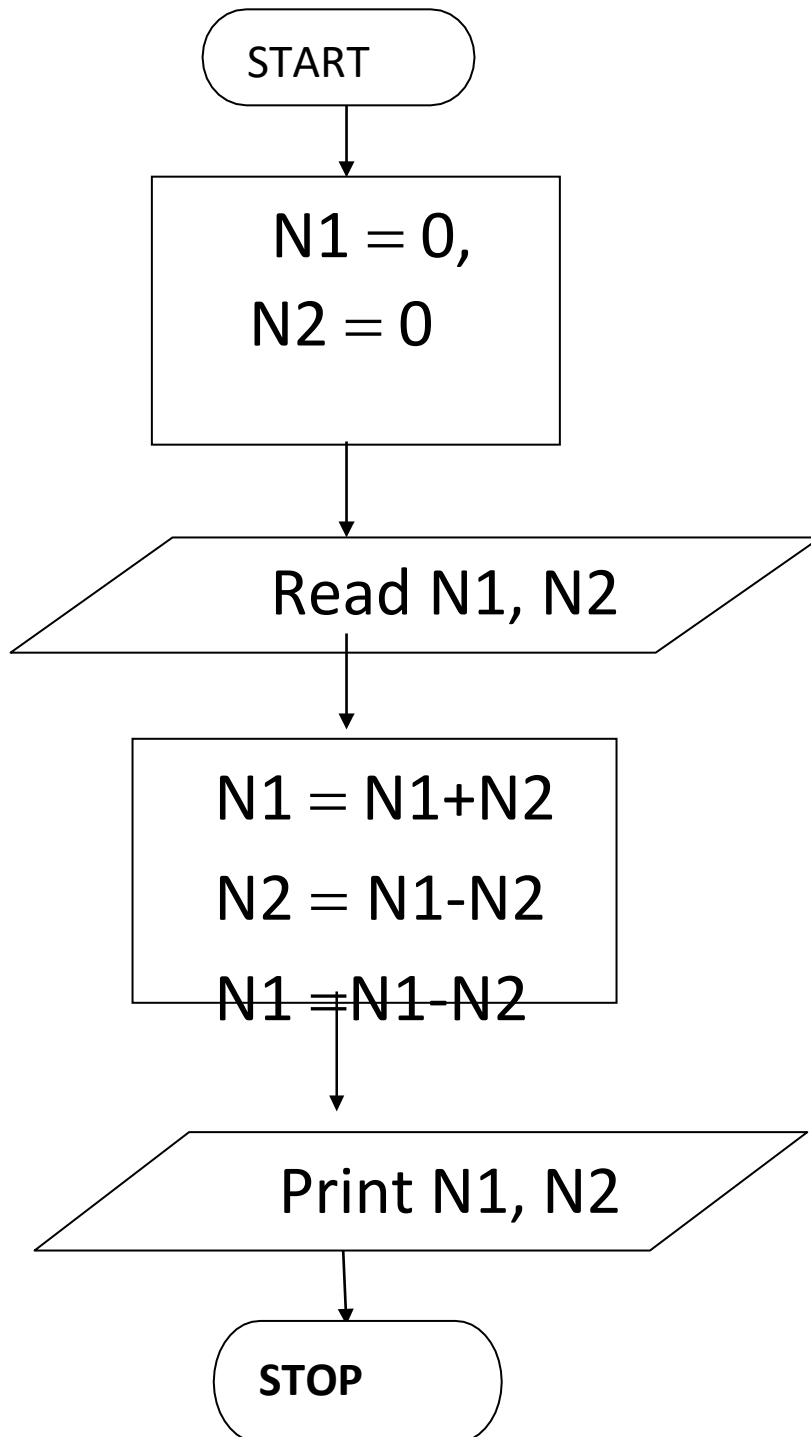
Assume N1 = 50, N2 = 50

Sum is: 100

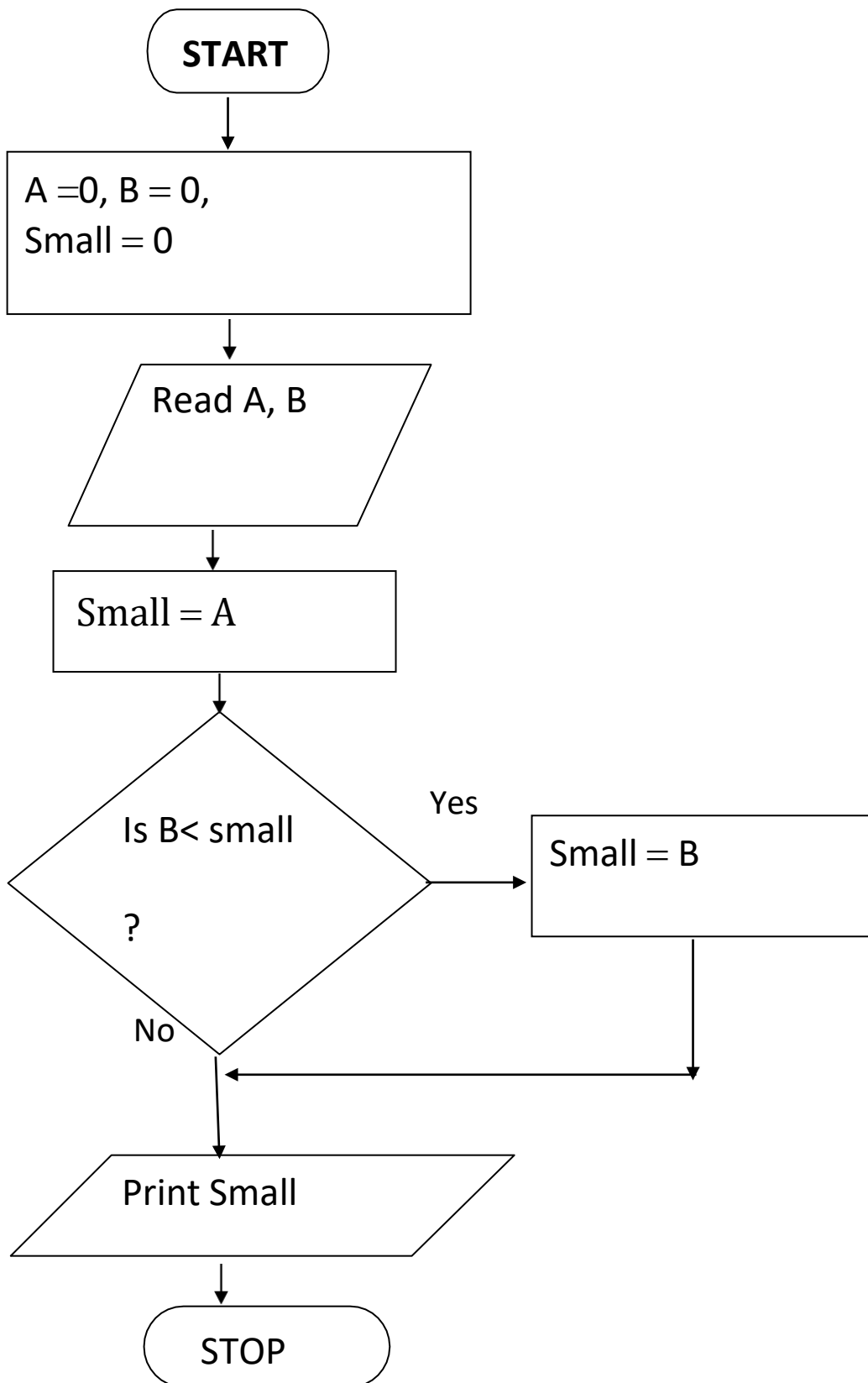
Q: Draw a flowchart to find area of a circle.



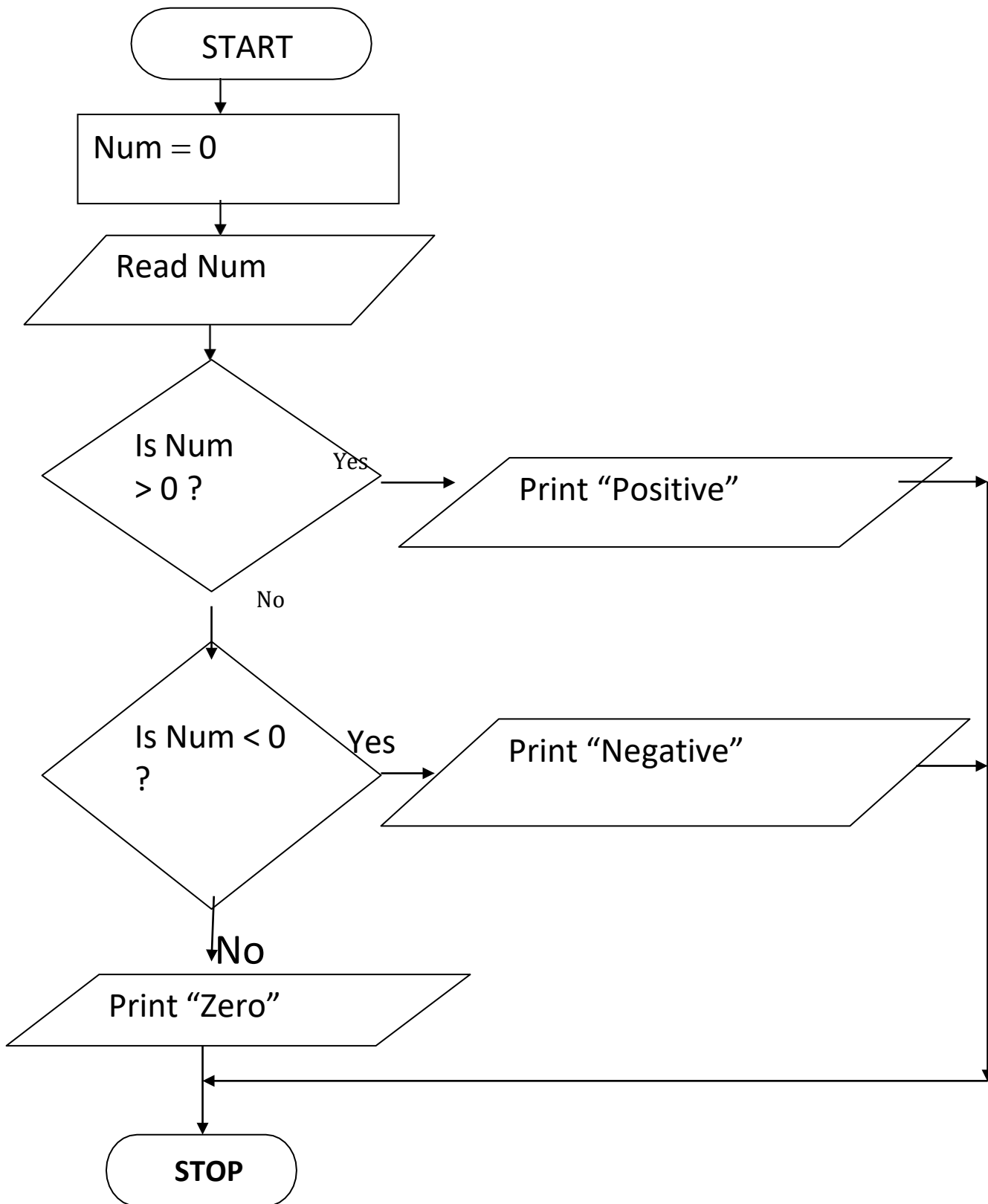
Q: Draw a flowchart to swap two numbers.



Q: Draw a flowchart to find the smallest of two numbers.



Q: Draw a flowchart to find whether a number is positive, negative and zero.



UNIT-II

INTRODUCTION TO C PROGRAM & OPERATORS

Topics: First C program - Hello world, How to open a command prompt on Windows or Linux How to read and print on screen - printf(), scanf(), getchar(), putchar() Variables and Data types - Variables, Identifiers, data types and sizes, type conversions, difference between declaration and definition of a variable, Constants.

Life cycle of a C program (Preprocessing, Compilation, Assembly, Linking, Loading, Execution), Compiling from the command line, Macros.

Operators – equality and assignment, Compound assignment operators, Increment and decrement operators, Performance comparison between pre and post increment/decrement operators, bitwise operators (AND, OR, NOT and XOR), Logical Operators, comma operator, precedence and associativity, Logical operators (AND, OR,NOT).

Why Program Computers?

- Computer is a machine. Depends on Instructions to accomplish something.
- Correct Instructions results expected output.
- Series of meaningful instructions is a program.
- People who write these instructions known as programmers.
- Cooking dish certain steps are followed. Particular order, quantity etc. Same in computer programming Wrong instruction, did not give the desired result.
- On RUNNING/EXECUTING a computer program we get desired output
- You need a COMPILER to run a program on a computer. It makes a program understandable to a computer. Since Computer can understand only 0s and 1s
- COMPILATION or BUILD is the process of making a program executable on a computer
- Program written in high level language called a SOURCE CODE.
- Compiler transforms to MACHINE CODE
- We shall use Code::Blocks/Online GDB/Dev C++ Compiler for typing, editing, debugging and executing a program.

Terminology

- Integrated development environment (IDE) see the program output in one place with an IDE . Example: Code::blocks, Eclipse, Visual Studio
- Editor
- Compilation
- Debugging: Removing them called debugging.
- Errors in code called as Bugs.

Introduction

We all program our lives in some way. Computer Program is a list of Instructions

Example Program: ISBT (Inter State Bus Terminus) to Graphic Era University, Dehradun

- Head west on NH 7 toward ISBT Circle
- At ISBT Circle, take the 1st exit onto NH 307 2 Kms
- Turn left onto Post Office Rd 1 Kms
- Turn right onto Bell Road 200 Metres
- Make a left towards Graphic Era University

Let us say your parents give you one thousand rupees per month towards your expenses. Assume your hobby is listening to music and you purchase them from a website selling songs. Let us say the cost of a song is 12 rupees. Now you want to write a simple program to tell you the total cost for say N number of songs you download from the site. If you had to write the steps they would be...

Enter "Number of songs you would like to download today?"

NEnter "the cost per song:" 12

Multiply the Number of songs with cost per song: $N \times$

12Show the final cost: Rs. _____

- This sequence of instructions is a program.
- Now computer programs are similar but have a **SYNTAX** and written in some programming language.
- Computer languages may specify that end of each statement be followed by a special character like say ; in C
- Failure to use the **terminating** ; leads to syntax error message
- Every language has its unique syntax
- Logic is known as **SEMANTICS**

```
// C Program
#include <stdio.h>
int main ()
{
    int N_Songs ;
    float Song_Cost = 12, Final_Price;
    printf("Enter Number of Songs \n");
    scanf("%d", &N_Songs);
    Final_Price = N_Songs * Song_Cost;
    printf("Cost of Songs is %0.2f \n",Final_Price);
    return 0;
}
```

Key parts of a program

- Variables and Constant
- Name and types of values
- Variables vary and contain value(s)
- Names for Memory location
- Operators = *
- Data type Character and Non-Character
- Must Declare memory locations to be reserved

Why Learn C Programming Language?

- Close to 50 years still widely used
- Language that is the basis for most modern languages
- Understand the working of a computer system
- Extensive applications. For example, compilers, operating systems embedded applications to name a few.
- Major portions of OS like Windows, Linux, UNIX still in C. Integration with new devices through device drivers in C.
- Embedded world mobile phones, washing machines, digital cars, cameras, IoT applications use C.
- Games also due to speed and quick response
- Few languages have the ability to interact with hardware like C
- Lean, Mean and efficient
- Lean, mean efficient language

Features of C language:

- Structured
- High Level/Middle Level
- Robust
- Portable
- Extensible
- Supports pointers

Structured:

C is structured procedure-oriented language, it divides the problem into smaller modules called functions or procedures.

High Level/Middle Level:

Programs written in C must be translated by the Compiler (system software) into machine level language.

While in middle level languages programmers can write a code that can be translated by the assemblers into machine code, which helps the programmers to interact directly with the hardware.

Portable:

C is portable that is code written on one machine can be easily ported or executed on other machines as long as that machine supports the same C compiler.

Robust:

C is robust language that is programs written in C do not crash so easily. It recovers quickly whenever programs results into an erroneous condition.

Extensible:

C is extensible language that is it allows new features and modifications to be made to the existing programs written in C.

Supports pointers:

C supports the use of pointers that allows the programmers to manipulate the memory directly.

Journey of C Language

- In 1972 Dennis Ritchie worked on improvement of a language called B by Ken Thompson which in turn was derived from BCPL by Martin Richards.
- The result was the C programming language.
- Version 4 of Unix Kernel major portions were coded in C
- First languages used for writing an OS rather than an assembly language
- Language became popular
- 1978 K&R released the first specification via book The C programming language
- To address standardization 1983 ANSI committee formed
- ANSI released standard document in 1989
- Adopted by ISO in 1990
- C89 or C90 refer to same standard
- ISO C99 inline functions, one line comments, variable length arrays
- C11 and C18

The only way to learn a programming language is by writing programs in it.

Comments in C Code

```
// C hello world example
/*
Graphic Era University
B.Tech 1stsem
*/
```

```
#include <stdio.h>
int main()
{
printf("Welcome to C Programming\n");
return 0;
}
```

- Can be placed anywhere in a program
- Clarity and explanation to others/self later on
- Only Part of the Source File Not the executable
- Cannot Nest Multiline Comments
- Avoid Trivial Comments Ex: int i
- May use comments to describe the logic in Pseudo code
- Program code change history can be maintained at the top of each program
- Code testing frequently comment out code

```
// C hello world example
/*
Author: Prof. Mahant @ GEHU
Purpose: Comments in C Code
*/
#include <stdio.h>
int main()
{
    printf("Welcome to C Programming\n");
    return 0;
}
```

Library

- Originally C library was considered as part of the UNIX operating system
- Users provided facilities for I/O, memory management etc
- User Community implementations were shared
- Incorporated as a part of standard C
- Libraries in ANSI C
- Declaration is in header files but actual code is in library files
- C lacks built-in capability to do tasks like memory management, I/O, manipulation of data etc
- Arrangement in the form of a standard library exists
- Compile your code and link them for usage
- Functions are specified as a part of the standard ISO C
- Small Library set of functions compared to other languages
- Developed and thoroughly tested,
- Optimised and error free
- Time saving
- Consistency of behaviour across operating platforms
- Header file contains function declarations, data type definitions, and macros
- As of now 29 Header files per C11

Preprocessor Directives (#include)

`#include <C_HeaderFile.H>`

Example

`#include <stdio.h>`

OR

`#include "U_HeaderFile.H"`

Example

`#include "projects.h"`

- `<C_HeaderFile.H>` Search for header file in the standard system directories (Can show a snapshot of standard system directories on Windows with Code::Blocks compiler)
- `"U_HeaderFile.H"` Searches in the location where your source file exists followed by the standard directories

C:\D:\Code Blocks\MinGW\include

Eg: `stdio.h`, `math.h`

- Contents of the header file are included at that point in the source code. Think of it as a copy paste into the source file
- Step is prior to compilation. Not apart of the C Compiler. That is why no ;
- Files shared across programs
- One header file Per Line
- Function prototype declarations (No code is present), Global variables, Constants
- No Code after # line
- Software Program as the name indicates does pre-processing prior to the actual compilation of the program in a high level language.
- Includes header file <stdio.h>
- Removes any comments in the source program

Macros Substitution

They are abbreviations of C code. They are substituted for all occurrences in your sourcefile. Example

#define PI 3.142

Ex:

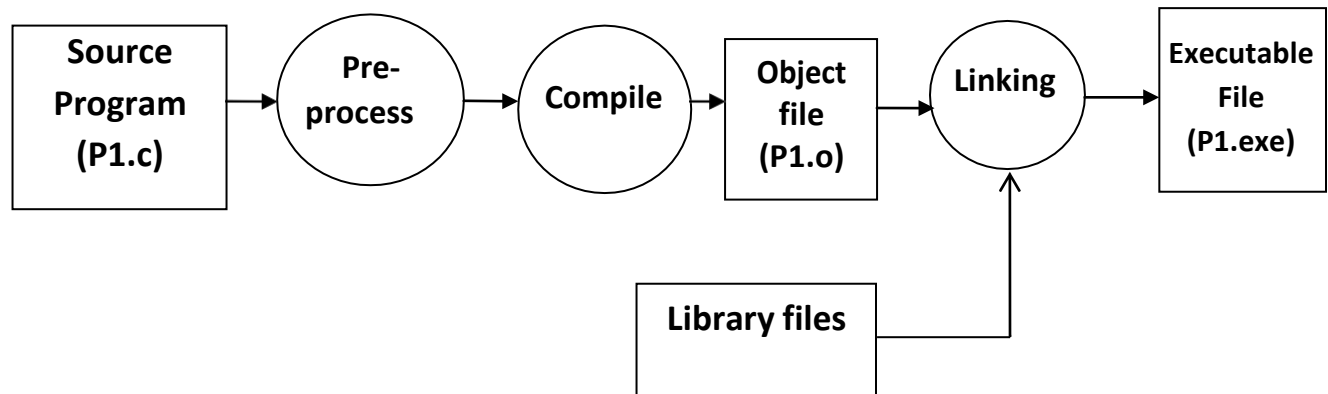
```
#include <stdio.h>
#define STRING "Hello World"
int main(void)
{
    /* Using a macro to print "HelloWorld" */
    printf(STRING);
    return 0;
}
```

Journey of a C Program from(Source Code to Executable)

- Create a file in code:blocks IDE and save the file with an extension .c For example: **hello.c**
- File is passed onto a software program called pre-processor. File after preprocessing becomes **hello.i**
- The file is taken in by the compiler and converted to a assembly file called **hello.s**
- The assembler converts the file into **hello.o** (object code)
- Some calls like say printf code not yet part of the program. These references need to be resolved. Library object code is stored in .a or .lib files.
- A software program called the linker resolves these references (linking the code)
- Output of the linker is .exe on windows and a.out on Linux platforms. The executable image is stored on the secondary storage device.
- Loader takes the image and loads into RAM. Intimates to the CPU starting address of

the code.

Life cycle of a C program



Source Program: Any C file say **P1.C** program is edited and given to a C compiler.

Figure above shows the different phases of execution of C program.

Pre-Processing: This is the first phase through which source code is passed. In this phase any statements defined in this section (before the main() function) are processed, if used in the program.

This phase includes:

- Removal of Comments
- Expansion of Macros
- Expansion of the included files.
- Conditional compilation

For ex. printf and scanf statements, if used in the program, will have been checked with their definitions stored in the header file <stdio.h>.

Compile: The next step is to compile and produce an; intermediate file that contains assembly level instructions **P1.s**.

During this phase the compiler checks for the syntax errors such as declaration of variables, initialization if any, the correct syntax of the C program etc. If any errors are encountered then they get displayed with corresponding line numbers.

The assembler converts the **P1.s** file after correction and successful compilation of the program to an **object file P1.o**.

Linking:

This is the final phase in which all the linking of function calls with their definitions are done. Linker knows where all these functions are implemented. Linker does some extra work also, it adds some extra code to our program which is required when the program starts and ends.

Linking produces the executable file **P1.exe (a.out by default on Linux/Unix machines)**, which is the machine code (binary code) which will be actually executed by the processor.

Loader:

Loader takes the image (**P1.exe**) generated and loads it into RAM and informs the CPU the starting address of the code for execution (running of the program).

BASIC CONCEPTS OF A C PROGRAM

- The structure of a C program is shown below:

```
int main()
{
    declaration section;

    statement-1 // Executable section
    starts
    statement-2
    statement-3
    statement-4 // Executable section ends
    return
    1;
}
```

User defined function-definition(s) → *optional*

Documentation Section

- This section allows to document by adding Comments to the program. Comments are portions of the code ignored by the compiler. The comments allow the user to make simple notes in the source-code.
- For ex. // this is an example for single line comment
/* this is an example for multiple line comment */

Preprocessor Directives

- The *preprocessor* accepts the source program and prepare the source program for compilation.
- The preprocessor-statements start with symbol #.
- The normal preprocessor used in all programs is **include**.
- The **#include** directive instructs the preprocessor to include the specified file-contents in the beginning of the program.

- Forex:

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

```
main()
```

Global declaration section

- If user wishes to declare any variable outside the main program which needs to be accessed by any part of the program then he may declare it in this section just before main function.
- Every C program should have a function called as **main()** and is an entry point to the program (a gateway to the program) that is always executed.
- The statements enclosed within left and right curly brace is called body of the function. The main() function is divided into 2 parts:

Declaration Section

- The variables that are used within the function main() should be declared in the declaration-section only.
- The variables declared inside a function are called local-variables. The compiler allocates the memory for these variables based on their types for ex. if the variable is an integer then it allocates 4 Bytes, if it's of **char** type then 1-Byte and so on. Ex: int p, t,r;

Executable Section

- This contains the instructions given to the computer to perform a specific task.
- The task may be to:
 - display a message
 - read data or
 - do some task ex. add two numbers etc.

User Defined Function-definition(s): If user has any user defined functions then those definitions are written outside the main function.

Example: **Program to display a message on the screen.**

```
#include<stdio.h>  int
main()
{
    printf("Welcome to C");
    return 1;
}
```

Output:

Welcome to C

DATA INPUT/OUTPUT FUNCTIONS

- There are many library functions for input and output operations in C language.

- Forex:

getch(), putchar(), scanf(), printf()

- For using these functions in a C-program there should be preprocessor statement #include<stdio.h> in the beginning of the program before the declaration of main function.

Input Function

- The input functions are used to read the data from the keyboard and store in memory-location.

- Forex:

scanf(), getchar(), getch(), getche(), gets()

scanf()

Reads values into built-in data types

Syntax: scanf("Format specifier",&var_name);

- Reads data obeying the format specified into a memory address
- scanf(" format string ", &v1,&v2..&vn);
- Format string shall be conversion specifiers without spaces or other characters
- Followed by the location (address) of memory locations.
- & known as reference operator
- number and type of format specifiers is programmers responsibility
- forgetting & can lead to program crash or unexpected behaviour
- ignore spaces, newlines, tabs while reading input

```
#include <stdio.h>
int main()
{
    int a;
    printf("Enter integer number:");
    scanf("%d", &a );
    printf("a is = %d\n", a);
    return 0 ;
}
```

Enter Integer Number: 55
a is = 55

Output Functions

- The output functions are used to receive the data from memory-locations through the variables and display on the monitor.

- Forex:

printf(), putchar(), putch(), puts() Types
of I/O Functions:

printf() (Most common Output Function)

- scanf and printf very widely used C functions

```
#include <stdio.h>
int main ()
{
    int i = 10 ;
    float j = 100;
    printf("Value of i = %d and j =
    %f \n", i, j);
    return 0;
}
```

Value of i = 10 and j = 100.000000

- Used for printing values of built-in datatypes
- C has no idea about what you are printing. So....
- Need to inform C, how to make sense of the data

printf(" format string ", expr1, expr2,expr3...exprn);

- format string informs C how the values are to be interpreted. It contains conversion specifiers and text
- Conversion specifiers begin with %

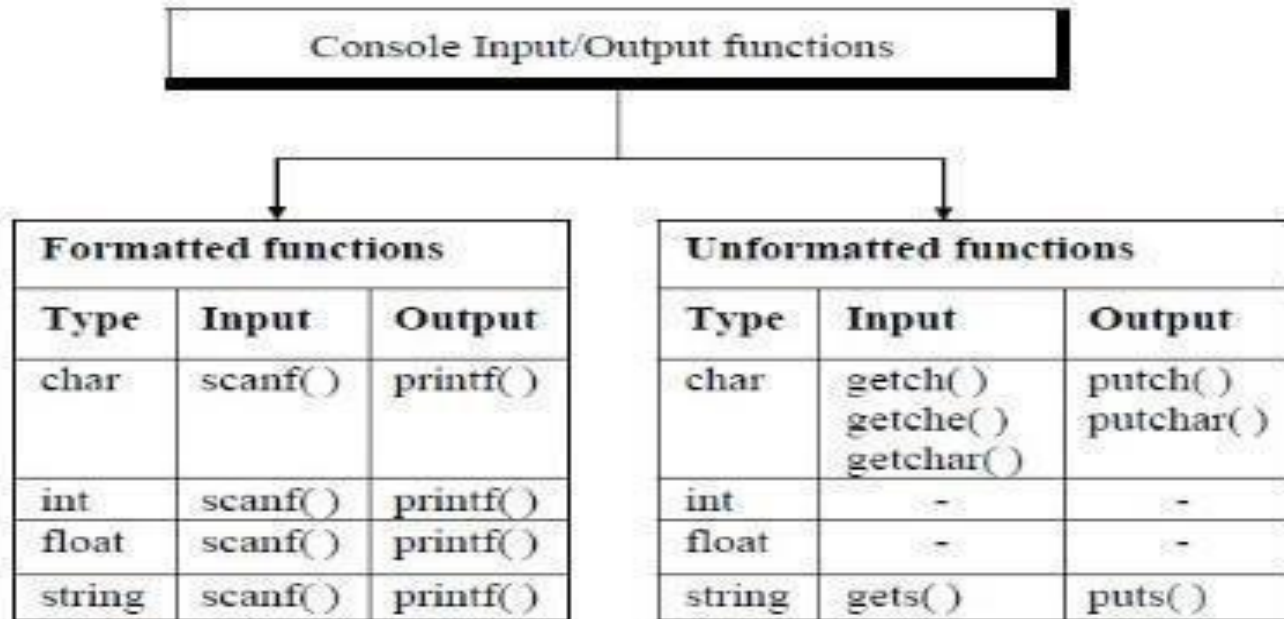
- Expressions may contain constants, variables or a combination evaluating to a built-in data type
- At the point of conversion specifiers values are substituted during printing
- Conversion specifiers specify the format of display
- For example %d indicates decimal integer
- Ordinary characters enclosed within "" are printed as is.
- No checking of number of format specifiers and expressions
- C does not perform any type checks in the printf() function call
- Programmer job to make sure that the data type of the variables MATCH and CORRESPOND to formatting character
- Programmers responsibility to specify correct specifier and data type
- Otherwise garbage results are printed

ILLUSTRATION

printf("Value of i = %f and j = %d \n", i,j); // Garbage Output

printf("Value of i = %d and j = %f \n", i); // i = 100, meaningless output

- There are 2 types of I/O Functions as shown below:



Character Set

Character-set refers to the set of alphabets, letters and some special characters that are valid in C language.

- Alphabets
- Digits
- White Spaces Tab Or New line Or Space
- Special Characters

1. ALPHABETS

Uppercase letters A-Z

Lowercase letters a-z

2. DIGITS

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

3. Special Characters

~ Tilde

! Exclamation mark

Number sign

\$ Dollar sign

% Percent sign

^ Caret

& Ampersand

* Asterisk

(Left parenthesis)

) Right parenthesis

_ Underscore

Special Characters

Symbol Meaning

+ Plus sign

| Vertical bar

\ Backslash

` Apostrophe

– Minus sign

= Equal to sign

{ Left brace

} Right brace

[Left bracket

] Right bracket

: Colon

” Quotation mark

; Semicolon

< Opening angle bracket

> Closing angle bracket
? Question mark
, Comma
. Period
/ Slash

Execution Character Set (Escape Sequence)

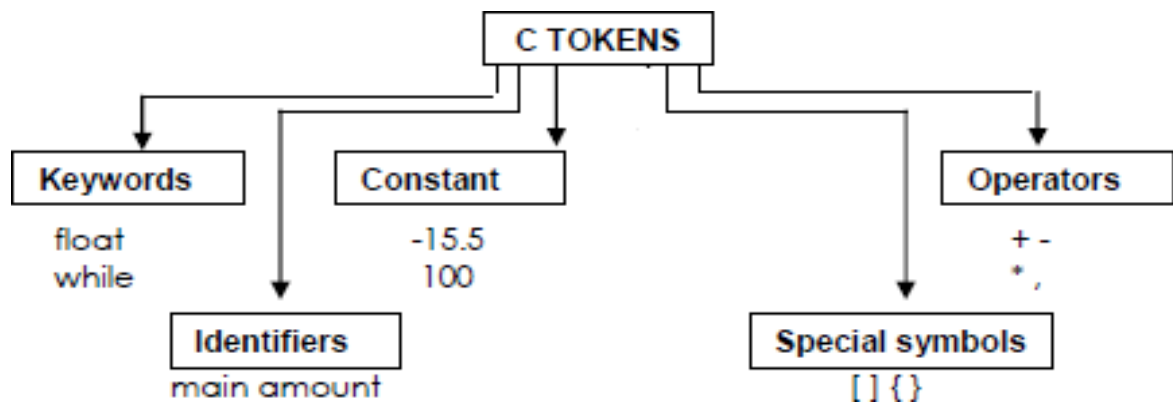
These are unprintable, which means they are not displayed on the screen or printer. Those characters perform other functions. Examples are backspacing, moving to a newline, or ringing a bell. Each one of character constants represents one character, although they consist of two characters. These character combinations are called as escape sequence.

- An escape sequence character begins with a backslash and is followed by one character.
- A backslash (\) along with some characters give rise to special print effects by changing (escaping) the meaning of some characters.
- The complete set of escape sequences are:

Character	ASCII value	Escape Sequence	
Null	000	\0	Null character
Alarm (bell)	007	\a	Beep Sound
Back space	008	\b	Moves previous position
Horizontal tab	009	\t	Moves next horizontal tab
New line	010	\n	Moves next Line
Vertical tab	011	\v	Moves next vertical tab
Form feed	012	\f	Moves initial position of next page
Carriage return	013	\r	Moves beginning of the line
Double quote	034	\"	Present Double quotes
Single quote	039	\'	Present Apostrophe
Question mark	063	\?	Present Question Mark
Back slash	092	\\	Present back slash
Octal number	\00		
Hexadecimal number	\x		

Tokens

- A token is a smallest element of a C program.
- One or more characters are grouped in sequence to form meaningful words. These meaningful words are called tokens.
- The tokens are broadly classified as follows
 - Keywords ex: **if, for, while, int, float, char**
 - Identifiers ex: **sum, length**
 - Constants ex: **10, 10.5, 'a', "sri"**
 - Operators ex: **+ - * /**



→ Special symbols ex: [], (), {}

Sample C Code Tokens

```
int main ( )  
{  
    const float PI = 3.142 ;  
    int a = 0 ;  
    a = a + 10 ;  
    printf( "hello world \n" ) ;  
    return 0 ;  
}
```

Keywords: int, const, float

Operators: =, +

Identifiers: a, printf, main, PI

Constants: 3.142

Strings: "hello world \n"

Special Characters: { } ()

Keywords

- Keywords are tokens which are reserved by C for a definite purpose.
- Each keyword has fixed meaning and that cannot be changed by the user.
- C supports about 32 keywords.

Rules for using keywords

- Keywords cannot be used as a variable or function.
- All keywords should be written in lower letters.
- Following keywords are as listed below:

break	case	Char	const	continue	default
double	else	Float	for	if	int
register	return	short	signed	sizeof	struct
switch	typedef	unsigned	void	while	do
long	auto	static	register	enum	return
extern	union				

Identifier

- Identifier is used to represent various part of a program such as variables, constants, functions etc.
- An identifier is a word consisting of sequence of
 - Letters
 - Digits or "_" (underscore)
- Forex:
Average, _Percent, total100

Few Valid Identifiers

Ex. Length, _breadth, area51, Num5_sol, _percent_ etc.

Few Invalid Identifiers

Ex. 10Average, -Sum, Sol?ution, Div/quot etc.

Rules for an Identifier

- Can start with a letter or underscore
- Can't start with a digit or special symbol or operator
- Can't contain special symbol or operator
- Can't be a keyword

Constants

- A constant is an identifier whose value remains fixed throughout the execution of the program.
- The constants cannot be modified in the program.
- For example:
1, 3.14512, 'z', "pcdnotes"

Different types of constants are:

1. Integer Constant:

- An integer is a whole number without any fraction part.
- There are 3 types of integer constants:
 - i) Integer & Decimal constants (0 1 2 3 4 5 6 7 8 9)
Integer Constants
For ex: 0, -9, 22
 - ii) Octal constants (0 1 2 3 4 5 6 7)
For ex: 021, 077, 033
 - iii) Hexadecimal constants (0 1 2 3 4 5 6 7 8 9 A B C D E F) For ex:
0x7f, 0x2a, 0x521

2. Floating Point Constant

- The floating point constant is a real number.
- The floating point constants can be represented using 2 forms:
 - i) Fractional Form
 - A floating point number represented using fractional form has an integer part followed by a dot and a fractional part.
 - For ex:
0.5, -0.99
 - ii) Scientific Notation (Exponent Form)
 - The floating point number represented using scientific notation has three parts namely:
mantissa E exponent
 - For ex:
9.86E3 imply 9.86×10^3

3. Character Constant

- A symbol enclosed within a pair of single quotes(') is called a character constant.
- Each character is associated with a unique value called an ASCII (American Standard Code for Information Interchange)code.

Forex: '9', 'a', '\n'

4. String Constant

- A sequence of characters enclosed within a pair of double quotes("")is called a string constant.
- The string always ends with NULL character (denoted by \0)character.
- Forex:

"9" "a" "sri" "\n"

Data in Programming

- Data is collection of raw facts
- Data can be numbers, characters, images, sound
- Programs transform data into meaningful information
- Primarily focus on Integers, Characters and Real Numbers (floating point)
- Ex: Roll Number Integer
- Numbers without decimal place are called as integers
- Example: 9, -8, 1008, -4995

Data Types in C

- The data type defines the type of data stored in a variable and hence in the memory-location.

Three basic data types in C:

int , float and char

- C supports three classes of datatypes:

1) Primary datatype

for ex: int, float, char and void

2) Derived datatypes

For ex: array

3) User defined datatypes For

ex: structure

Primary data types in C:

1) int

- An int is a keyword, used to define integers.
- Using *int* keyword, the programmer can inform the compiler that the data associated with this keyword should be treated as integer.
- C supports 3 different sizes of integer:
 - Short int
 - int
 - long int

2) float

- A float is a keyword which is used to define floating point numbers.
- Compiler allocates 4 bytes of memory.
- C supports 3 different sizes of float:
 - float
 - double
 - long double
- compiler allocates 8 bytes of memory to the data type **double**, while 16 bytes to **long double**.

3) char

- A **char** is a keyword which allows defining and store a single character.
- Compiler allocates 1 byte of memory.

4) void

- **void** is an empty data type indicates that no value is associated with this data type, and it does not occupy any space in the memory.
- Generally used with functions to indicate that the function does not return any value.

Range of data types for 16-bit processor

Data type	Bytes	Range of data type
char	1 byte	-128 to 127
unsigned char	1 byte	0 to 255
int	2 bytes	-32,768 to 32,767
unsigned int	2 bytes	0 to 65,535
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
float	4 bytes	3.4E-38 to 3.4E38

double	8 bytes	1.7E-308 to 1.7E308
long	8 bytes	-223372036854775808 to +9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

Qualifiers

- Qualifiers alter the meaning of primary data types to yield a new data type.

Size Qualifiers

- Size qualifiers alter the size of primary datatype.
- The keywords `long` and `short` are 2 size qualifiers. For example:

```
long int i; //The size of int is 2 bytes but, when long
            //keyword is used, variable will be of 4 bytes
short int i; //The size of int is 2 bytes but, when short
            //keyword is used, that variable will be of 1
            //byte
```

Sign Qualifiers

- Whether a variable can hold positive value, negative value or both values is specified by sign qualifiers.
- Keywords `signed` and `unsigned` are used for sign qualifiers.

```
unsigned int a; //unsigned variable can hold zero &
                // positive values only
Signed int b;  //signed variable can hold zero ,positive
                //and negative values
```

Constant Qualifiers

- Constant qualifiers can be declared with keyword **const**.
- An identifier declared by using a **const** keyword cannot be modified.

```
const int p=20; //the value of p cannot be changed in the
                //program.
```

Variables

- A variable is an identifier whose value can be changed during execution of the program. Variable is a name given to a memory-location where the data(value) can be stored.
- Using the variable-name, the data can be
→ stored in a memory-location and
→ accessed or manipulated

Variable Declarations:

//Showing some variable declarations

```
#include <stdio.h> #define LOWER 0 int main ()
{
int lower = 234;
char c = 'a', d, f;
short age ;
float fahr = 123.45;
int start = LOWER;
const double e = 2.7049658030;
}
```

/*Ritchie's advice: Declare each variable type on a separate line */

- Choose variable names that are related to the purpose of the variable, and that are unlikely to get mixed up typographically.
- Use short names for local variables, especially loop indices, and longer names for external variables.

Rules for defining a variable

- 1) The first character in the variable should be a letter or an underscore.
- 2) The first character can be followed by letters or digits or underscore.
- 3) No special symbols are allowed (other than letters, digits and underscore).
- 4) Length of a variable can be up to a maximum of 31 characters.
- 5) Keywords should not be used as variable-names.

- Valid variables:

area, rate_of_interest, _temperature_, celcius25 Invalid variables:

3fact, ?sum, sum-of-digits, length62\$, for, int, if

Declaration of Variable

- The declaration tells the compiler
- what is the name of the variable used
- what type of data is held by the variable
- The syntax is shown below:

data_type variable1, variable2, variable3;

where variable1, variable2, variable3 are variable-names of data_type that could be int, float or char type.

- Forex:

int a, b, c;

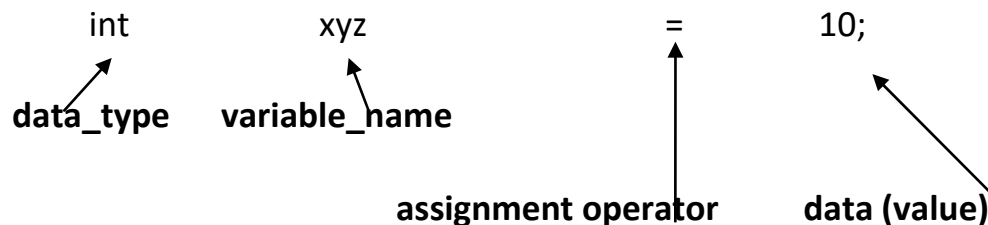
float x, y, z;

Initialization of Variable

- The variables are not initialized when they are declared. Hence, variables normally contain garbage values and hence they have to be initialized with valid data.

- Syntax is shown below:

- Forex:



float pi=3.1416; char
c='z';

Enum Data Type

- Enumeration: Listing things one after the other
- Variables that possess small range of values
- Represent Integer constant values
- Programmers create their own keyword/type
- Used to make the code readable and easy to maintain
- Two enum names can have same value.
- the compiler by default assigns values starting from 0
- Assign values to some name in any order. All unassigned names get value

- as value of previous name plus one
- value assigned to enum names must be some integral constant, i.e., the value must be in range from minimum
- possible integer value to maximum possible integer value

Suppose we need to assign numbers to days of the week

// DECLARATION

```
enum WEEK_DAY { SUN, MON, TUE, WED, THU, FRI, SAT};
```

// INSTANTIATE OBJECT

```
enum WEEK_DAY DAY ;
```

// DAY is variable of type ENUM

```
Int main()
```

```
{
```

```
    enum WEEK_DAY DAY;
```

```
    DAY= THU;
```

```
    printf("Value of Thursday is %d", DAY);
```

```
    return 0;
```

```
}    //    OUTPUT    4
```

Boolean Type Data

In programming TRUE or FALSE is frequently used

```
#define TRUE 1
```

```
#define FALSE 0
```

```
int test;
```

```
test=TRUE;
```

```
test=FALSE;
```

```
if (test==TRUE)
```

```
printf("PASS\n");
```

```
if (test==FALSE)
```

```
printf("FAIL\n");
```

```
_Bool test    //_Bool keyword
```

- test takes the values TRUE (1) and FALSE (0)
- _Bool is unsigned integer data type
- Can be assigned values 0 or 1 only.
- Assignment of Non Zero value reverts to 1

- For example test = 3

```

if(test)
printf("TRUE\n");
else
printf("FALSE\n");

```
- bool defined as a macro in <stdbool.h>
- also true and false macros are defined

```

#include    <stdio.h>
#include    <stdbool.h>
//boolasanalias to  _Bool intmain()
{
    Booltest;
    test=true    ;
    if (test)
    printf("TRUE  \n");
    else
    printf("FALSE  \n");
}

```

Common Conversion Specifiers

Decimal integer: %d
 Octal integer: %o
 Hexadecimal: %x
 float: %f
 double: %lf
 char: %c
 string: %s
 unsigned int : %u

% W.P[Conversion_Specifier]

- W The width specifies the minimum number of characters to be printed
- Width specification is ignored if number of letters is more than the width specified
- P specifies the Precision
- %3d width printing 10 would print b10 (right justified)

- %-3d width printing 10 would print 10b (left justified)

```
//Printing of int and float using
//various format specifiers and widths #include<stdio.h>
```

```
#include<stdlib.h>
int main()
{
    inti=101 ;
    floatj=1234.123f; char
    name[20]="Arjun";
    printf("[%d][%6d][%-6d][%6.4d]\n",i,i, i, i);
    printf("[%f][%12f][%-12f][%12.2f][%12.2e]\n", j,j,j,j,j);
    printf("[%s][%15s][%-15s][%15.2s]\n", name,name,name,name);

    return 0;
}
```

Output of the Code Snippet:

```
[101][ 101][101 ][ 0101]
[1234.123047][ 1234.123047][1234.123047][ 1234.12][ 1.23e+003]
[Arjun][Arjun][Arjun][ Ar]
```

OPERATOR

- An operator can be any symbol like + - * / that specifies what operation need to be performed on the data.
- Forex:
 - + indicates add operation
 - * indicates multiplication operation Operand
- An operand can be a constant or a variable.

Expression

An expression is combination of operands and operator that reduces to a single value.

- Forex:

Consider the following expression $a+b$ here a and b are operands while $+$ is an operator

Operator Table with Precedence and Associativity:

- The order in which different operators are used to evaluate in an expression is called precedence of operators.

Precedence	Operator	Description	Associativity
1	++	Postfix increment	Left-to-right
	--	Postfix decrement	
	()	Function call	
	[]	Array subscripting	
	.	Element selection by reference	
	->	Element selection through pointer	
2	++	Prefix increment	Right-to-left
	--	Prefix decrement	
	+	Unary plus	
	-	Unary minus	
	!	Logical NOT	
	~	Bitwise NOT (One's Complement)	
	(type)	Type cast	

	*	Indirection (dereference)	
	&	Address-of	
	sizeof	sizeof	
3	*	Multiplication	Left-to-right
	/	Division	
	%	Modulo (remainder)	
4	+	Addition	Left-to-right
	-	Subtraction	
5	<<	Bitwise left shift	Left-to-right
	>>	Bitwise right shift	
6	<	Less than	Left-to-right
	<=	Less than or equal to	
	>	Greater than	
	>=	Greater than or equal to	
7	==	Equal to	Left-to-right
	!=	Not equal to	
8	&	Bitwise AND	Left-to-right
9	^	Bitwise XOR	Left-to-right
		(exclusive or)	
10		Bitwise OR (inclusive or)	Left-to-right

11	&&	Logical AND	Left-to-right
12		Logical OR	Left-to-right
13	?:	Ternary conditional	Right-to-left
14	=	Direct assignment	Right-to-left
	+=	Assignment by sum	
	-=	Assignment by difference	
	*=	Assignment by product	
	/=	Assignment by quotient	
	%=	Assignment by remainder	
	<<=	Assignment by bitwise left shift	
	>>=	Assignment by bitwise right shift	
	&=	Assignment by bitwise AND	
	^=	Assignment by bitwise XOR	
	=	Assignment by bitwise OR	
15 lowest	,	Comma	Left-to-right

Memory Key: **PUMA SRE BIT3 LOG2 TAC**

Illustrations

x = -2 + 11 - 7 * 9 % 6 / 12

Operator	Precedence	Associativity
- (Unary)	1	R-L
*, %, /	2	L-R
+, -	3	L-R
=	4	R-L

CLASSIFICATION OF OPERATORS

Operator Name

For Example

Arithmetic operators

+ - * / %

Increment/decrement operators

++ --

Assignment operators

=

Relational operators
operators

<>== Logical
&& || ~

Conditional operator

?:

Bitwise operators

& | ^

Comma operator

,

Special operators

[], sizeof, →

ARITHMETIC OPERATORS

- These operators are used to perform arithmetic operations such as addition, subtraction, division, multiplication, modulus.

There are 5 arithmetic operators:

Operator	Meaning of Operator
+	addition
-	subtraction
*	multiplication
/	division
%	modulos

- Division symbol(/)

→ divides the first operand by second operand and

→ returns the quotient.

Quotient is the result obtained after division operation.

- Modulus symbol (%)

→ divides the first operand by second operand and

→ returns the remainder.

Remainder is the result obtained after modulus operation.

- To perform modulus operation, both operands must be integers.
- Program to demonstrate the working of arithmetic operators.

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

```
int main()
```

```
{
```

```
    int a=11,b=4,c;
```

```
    c=a+b;
```

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

```
    c=a-b;
```

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

```
    c=a*b;
```

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

```
    c=a/b;
```

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

```
    c=a%b;
```

```
    printf("Remainder when a is divided by b=%d ", c);
```

```
    return 1;
```

```
}
```

Output:

a+b=15

a-b=7

a*b=44

a/b=2

a%b=3 Remainder when a is divided by b= 3

Integer vs Float Arithmetic

```
int a=20,b=6,result;
```

```
result = a / b ; //3
```

```
result = a % b ; //2
```


Float result;

result=10 /3; // 3.00

result=10/3; // 3.333333

NOTE:

- a) **Unary+,-have highest precedence, next are *,/,%, operators and last are binary+,-**
-operators and the associativity of all arithmetic operators is left to right.
- b) **Mod works only on Integer type data**

INCREMENT OPERATOR

- ++ is an increment operator.
 - As the name indicates, increment means increase, i.e. this operator is used to increase the value of a variable by 1.
 - For example:
 If b=5
 then b++ or ++b; // b becomes 6
 - The increment operator is classified into 2 categories:
 - 1) Post increment Ex: b++
 - 2) Pre increment Ex: ++b
 - As the name indicates, post-increment means first uses the value of variable and then increases the value of variable by 1.
 - As the name indicates, pre-increment means first increases the value of variable by 1 and then uses the updated value of the variable.
 - For ex:
 If x= 10,
 then z= x++; assigns the value 10 to z & then increments value of x but z = ++x;
 assigns the value 11 to z
- Example: Program to illustrate the use of increment operators. #include<stdio.h>
- ```
int main()
{
 int x=10,y = 10, z ; z= x++;
 printf(" z=%d x= %d\n", z, x);
 z = ++y+x;
 printf(" z=%d y= %d", z, y++);
 return 1;
}
```

Output:

z=10 x=11

z=22 y=11

## DECREMENT OPERATOR

- -- is a decrement operator.
- As the name indicates, decrement means decrease, i.e. this operator is used to decrease the value of a variable by 1.

- For example:

If b=5

then b-- or --b; // b becomes 4

- Similar to increment operator, the decrement operator is classified into two categories:

i) Post decrement Ex: b--

ii) Pre decrement Ex: --b

- Forex:

If x=10,

Then z=x--; // z becomes 10,

but z = --x; // z becomes 9

Example: Program to illustrate the use of decrement operators.

```
int main()
{
 int x=10,y = 10, z ; z= x--;
 printf(" z=%d x= %d\n", z, x);
 z = --y;
 printf(" z=%d y= %d", z, y);
}
```

Output:

z=10 x=9

z=9 y=9

## TYPE CONVERSION

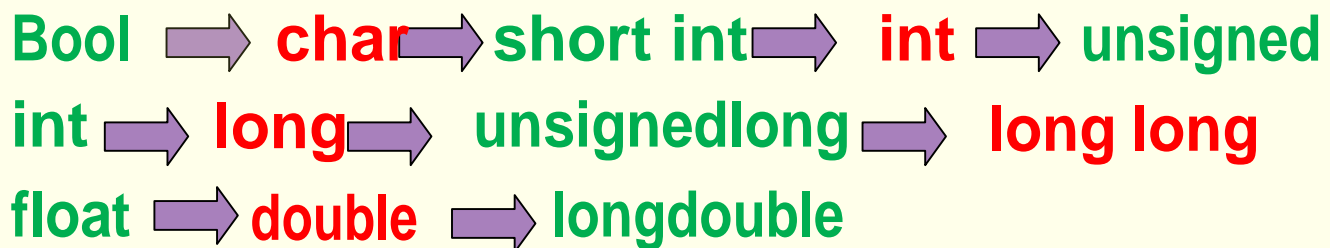
- Type conversion is used to convert data of one type to data of another type.
- Type conversion is of 2 types as shown in below figure:

## Implicit Type Conversion

- If a compiler converts one type of data into another type of data automatically, it is known as implicit conversion.
- There may be data loss whenever in implicit conversion takes place i.e while converting from **float** to **int** the fractional part will be truncated, **double** to **float** causes rounding of digit and **long** to **int** causes dropping of excess higher order bits.
- The conversion always takes place from lower rank to higher rank.

```
int main()// Implicit Conversion
{
 intage=2;
 char gender ='A'; //ASCII65
 age= age* gender;
 printf("%d \n", age); //130
} //Lower to a WIDER Type
```

Any implicit type conversions are made by converting the lower type to higher type as shown below:



For ex, int to float as shown in the above datatype hierarchy.

- Forex:

```
int a = 22;
```

```
float b=11;
```

```
float c = a/b; =0.500000
```

Rules (Promote lower to wider type):

- \_Bool to character
- character to short int
- short to int
- signed to unsigned int
- unsigned int to long int
- long to unsigned long int
- unsigned long int to long long
- long long to float
- float to double

### **Type Casting(Explicit Data Type Conversion)**

- When the data of one type is converted explicitly to another type with the help of some pre-defined types, it is called as explicit conversion.
- The syntax is shown below:

`data_type1 v1;`

`data_type2 v2= (data_type2) v1;`

where v1 can be expression or operand or value

```
#include<stdio.h>
int main()
{
 int i= 10,j= 20;
 float result;
 result =i/j;
 printf("%0.2f\n", result);// 0.00
 result =(float)i/j;
 printf("%0.2f \n", result);// 0.5
}
```

### **RELATIONAL OPERATORS**

- Relational operators are used to find the relationship between two operands.
- The output of relational expression is either true(1) or false(0).
- For example  
`a>b` //If a is greater than b, then `a>b` returns 1 else `a>b` returns 0.
- The 2 operands may be constants, variables or expressions.

There are 6 relational operators:

| Operator        | Meaning of Operator              | Example          |
|-----------------|----------------------------------|------------------|
| >               | Greaterthan7>4                   | returns true (1) |
| <               | Lessthan7<4                      | returns false(0) |
| >=              | Greater than or equal<br>to 7>=4 | returns true (1) |
| <=              | Less than or equal<br>to 7<=4    | return false(0)  |
| == Equal to     | 7==4                             | returns false(0) |
| != Not equal to | 7!=4                             | returns true(1)  |

- Forex:

#### Condition Return values

|        |              |
|--------|--------------|
| 8> 7   | 1 (or true)  |
| 8<7    | 0 (or false) |
| 8+7<15 | 0 (or false) |

- Example: Program to illustrate the use of all relational operators.

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

```
int main()
```

```
{ printf("7>4 : %d \n", 7>4);
 printf("7>=4 : %d \n", 7>=4);
 printf("7<4 : %d \n", 7<4);
 printf("7<=4 : %d \n", 7<=4);
 printf("7==4 : %d \n", 7==4);
 printf("7!=4 : %d ", 7!=4);
 return1;
```

```
}
```

*Output:*

```
7>4 : 1
7>=4 : 1
7<4 : 0
7<=4 :0
7==4 :0
7!=4 : 1
```

## LOGICAL OPERATORS

- These operators are used to perform logical operations like negation, conjunction and disjunction.

- The output of logical expression is either true(1) or false(0).
- There are 3 logical operators:

### Operator Meaning with Examples

&& Logical AND If c=3 and d=1 then ((c==3) && (d>2)) returns false.

|| Logical OR If c=3 and d=1 then ((c==3) || (d>3)) returns true.

! Logical NOT If c=3 then !(c==3) returns false.

- All non zero values(i.e. any value >0 or < 0 ) will be treated as true, while zero value(i.e. 0 ) will be treated as false.

### Truth table

| A | B | A&&B | A  B | !A |
|---|---|------|------|----|
| 0 | 0 | 0    | 0    | 1  |
| 0 | 1 | 0    | 1    | 1  |
| 1 | 0 | 0    | 1    | 0  |
| 1 | 1 | 1    | 1    | 0  |

- Example: Program to illustrate the use of all logical operators.

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

```
int main()
```

```
{
 printf("5 && 0 : %d \n", 5 && 0);
 printf("5 || 0 : %d \n", 5 || 0);
 printf("!0 : %d", !0);
 return 1;
}
```

*Output:*

```
5 && 0 : 0
```

```
5 || 0 : 1
```

```
!0 : 1
```

- Example: Program to illustrate the use of both relational & Logical operators.

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

```
main()
```

```
{
 printf("7>5 && 5< 8 : %d \n", 7>5 && 5<8);
 printf(" 7<5 || 5!=5 : % d \n", 7<5 || 5!=5);
}
```

```

 printf("!(3 ==3) : %d ", !(3==3));
 return 1;
 }

```

*Output:*

7>5 && 5<8 :1

7<5 || 5!=5 :0

!(3 ==3) : 0

## Equality Operators ( == != )

```

int main ()
{
 int a=20,b=10;
 int result ;
 if(a== 20)
 printf("Yes a is 20\n");
 if(b!=20)
 printf("Yes b is not 20\n");
}

```

## TERNARY OPERATOR (CONDITIONAL OPERATOR)

- The conditional operator is also called a ternary operator it has three parts.
- Conditional operators are used for decision making in C.
- The syntax is shown below:

(exp1)? exp2: exp3;

where exp1 is an expression evaluated to true or false;

If exp1 is evaluated to true, exp2 is executed; If exp1

is evaluated to false, exp3 is executed.

Example: Program to find biggest of 2 numbers using conditional operator.

```

#include<stdio.h>
int main()
{
 int a,b, big ;
 printf("Enter two different numbers:\n");
 scanf("%d%d", &a, &b);
 big=(a>b)? a :b;
 printf(" Biggest number is ", big);
 return 1;
}

```

*Output:*

Enter two different numbers: 88 79  
Biggest number is 88

Example: Program to find biggest of 3 numbers using conditional operator.

```
#include<stdio.h>
int main()
{
 int a,b,c big ;
 printf("Enter two different numbers:\n");
 scanf("%d%d%d", &a, &b, &c); big=(a>b)?
 ((a>c)?a:c) : (b>c)?b:c; printf(" Biggest of three
 number is ", big); return1;
}
```

*Output:*

Enter two different numbers: 88 99  
Biggest number is 99

## **ASSIGNMENT OPERATOR**

- The most common assignment operator is=.
- This operator assigns the value in right side to the left side.
- The syntax is shown below:

variable=expression;

- Forex:

c=5; //5 is assigned to c

b=c; //value of c is assigned to b 5=c; //

Error! 5 is a constant.

- The operators such as +=, \*= are called shorthand assignment operators.

- Forex,

a=a+10: can be written as a+=10;

- In the same way, we have:

### **Operator Example**

a-= a-=b a=a-b a\*=  
b c\*=b a=a\*b a/=

a/=b a=a/b

a%=b a%=b a=a%b



## NOTE:

```
int i= 2, j = 3; i *= j + 6 ;
i = i * (j + 6);
```

### Compact Assignments

= Direct assignment

+= Assignment by sum

-= Assignment by difference

\*= Assignment by product

/= Assignment by quotient

%= Assignment by remainder

<<= Assignment by bitwise left shift

>>= Assignment by bitwise right shift &= Assignment by bitwise AND

^= Assignment by bitwise XOR

|= Assignment by bitwise OR

| Operator | Example | Same as |
|----------|---------|---------|
| =        | a = b   | a = b   |
| +=       | a += b  | a = a+b |
| -=       | a -= b  | a = a-b |
| *=       | a *= b  | a = a*b |
| /=       | a /= b  | a = a/b |
| %=       | a %= b  | a = a%b |

## BITWISE OPERATORS

- These operators are used to perform logical operation (and, or, not) on individual bits of a binary number.
- There are 6 bitwise operators:

### Operators

&

|

^

~

<<

>>

### Meaning of operators

Bitwise AND

Bitwise OR

Bitwise exclusive OR

Bitwise complement

Shift left

Shift right

## Truth Table

| A | B | A&B | A B | A^B | ~A |
|---|---|-----|-----|-----|----|
| 0 | 0 | 0   | 0   | 0   | 1  |
| 0 | 1 | 0   | 1   | 1   | 1  |
| 1 | 0 | 0   | 1   | 1   | 0  |
| 1 | 1 | 1   | 1   | 0   | 0  |

## Bitwise AND operator &

The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

Let us suppose the bitwise AND operation of two integers 12 and 25.

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bit Operation of 12 and 25

00001100

& 00011001

\_\_\_\_\_

00001000 = 8 (In decimal)

## Example #1: Bitwise AND

```
#include <stdio.h>
int main()
{
 int a = 12, b = 25;
 printf("Output = %d", a&b);
 return 0;
}
```

## Output

12 = 00001100 (In Binary)

& 25 = 00011001 (In Binary)

```
=====
00001000 (8 in Binary)
Output = 8
```

## Bitwise OR operator |

The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. In C Programming, bitwise OR operator is denoted by |.

### Example #2: Bitwise OR

```
#include <stdio.h>
int main()
{
 int a = 12, b = 25;
 printf("Output = %d", a|b);
 return 0;
}
```

### Output

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bitwise OR Operation of 12 and 25

00001100

| 00011001

-----  
00011101 = 29 (In decimal)

Output = 29

## Bitwise XOR (exclusive OR) operator ^

The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by ^.

### Example #3: Bitwise XOR

```
#include <stdio.h>
int main()
{
 int a = 12, b = 25;
 printf("Output = %d", a^b);
 return 0;
}
```

## Output

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25

00001100

^ 00011001

\_\_\_\_\_

00010101 = 21 (In decimal)

Output = 21

## Bitwise complement operator ~

Bitwise complement operator is an unary operator (works on only one operand). It changes 1 to 0 and 0 to 1. It is denoted by ~.

35 = 00100011 (In Binary)

Bitwise complement Operation of 35

~ 00100011

\_\_\_\_\_

11011100 = 220 (In decimal)

## Twist in bitwise complement operator in C Programming

The bitwise complement of 35 (~35) is -36 instead of 220, but why?

For any integer  $n$ , bitwise complement of  $n$  will be  $\sim n$ . To understand this, you should have the knowledge of 2's complement.

## 2's Complement

Two's complement is an operation on binary numbers. The 2's complement of a number is equal to the complement of that number plus 1. For example:

| Decimal | Binary   | 2's complement                                     |
|---------|----------|----------------------------------------------------|
| 0       | 00000000 | $-(11111111+1) = -00000000 = -0(\text{decimal})$   |
| 1       | 00000001 | $-(11111110+1) = -11111111 = -256(\text{decimal})$ |
| 12      | 00001100 | $-(11110011+1) = -11110100 = -244(\text{decimal})$ |
| 220     | 11011100 | $-(00100011+1) = -00100100 = -36(\text{decimal})$  |

Note: Overflow is ignored while computing 2's complement.

The bitwise complement of 35 is 220 (in decimal). The 2's complement of 220 is -36. Hence, the output is -36 instead of 220.

**Bitwise complement of any number N is  $\sim(N+1)$ . Here's how:**

bitwise complement of  $N = \sim N$  (represented in 2's complement form)

2's complement of  $\sim N = -(\sim(\sim N)+1) = -(N+1)$

## Example #4: Bitwise complement

```
#include <stdio.h>
int main()
{
 printf("Output = %d\n",~35);
 printf("Output = %d\n",~~12);
 return 0;
```

```
}
```

## Output

Output = -36

Output = 11

## Shift Operators in C programming

There are two shift operators in C programming:

- Right shift operator
- Left shift operator.

### Right Shift Operator

Right shift operator shifts all bits towards right by certain number of specified bits. It is denoted by `>>`.

$212 = 11010100$  (In binary)

$212 \gg 2 = 00110101$  (In binary) [Right shift by two bits]

$212 \gg 7 = 00000001$  (In binary)

$212 \gg 8 = 00000000$

$212 \gg 0 = 11010100$  (No Shift)

### Left Shift Operator

Left shift operator shifts all bits towards left by a certain number of specified bits. The bit positions that have been vacated by the left shift operator are filled with 0. The symbol of the left shift operator is `<<`.

$212 = 11010100$  (In binary)

$212 \ll 1 = 110101000$  (In binary) [Left shift by one bit]

$212 \ll 0 = 11010100$  (Shift by 0)

$212 \ll 4 = 110101000000$  (In binary) = 3392 (In decimal)

## Example #5: Shift Operators

```
#include <stdio.h>
int main()
{
 int num=212, i=1;

 printf("Right shift by %d: %d\n", i, num>>i);

 printf("\n");

 printf("Left shift by %d: %d\n", i, num<<i);

 return 0;
}
```

Right Shift by 1: 106

Left Shift by 1: 424

```
Int main()
{
 int x= 2, y=3, z=-8;
 x=x<< y;
 printf("%d\n",x); //16
 x=x>> 2;
 printf("%d\n",x); // 4
 z=z>> 2;
 printf("%d\n",z); //-2 (?)
}
```

```
int main() // ~Complement Operator
{
 unsigned short x= 1;
 unsigned short i;
 i=~x;
 printf("%d\n",i); // 65534
}
```

```

int result; // signed
result = ~0 ;
printf("%d\n",result); //-1
}

```

//&, ^ ,| bitwise operators

```

int main()
{

 short i= 2 ,j=3;
 printf("%d\n", i & j); // 2
 printf("%d\n", i ^ j); // 1
 printf("%d\n", i | j); // 3
}

```

## **The sizeof operator**

The **sizeof** is a unary operator that returns the size of data (constants, variables, array, structure, etc).

```

#include <stdio.h>
int main()
{
 int a;
 float b;
 double c;
 char d;
 printf("Size of int=%lu bytes\n",sizeof(a));
 printf("Size of float=%lu bytes\n",sizeof(b));
 printf("Size of double=%lu bytes\n",sizeof(c));
 printf("Size of char=%lu byte \n",sizeof(d));
 printf("Size of char in bits=%d\n",sizeof(d)*8);

 return 0;
}

```

## **Output**



Size of int = 4 bytes  
Size of float = 4 bytes  
Size of double = 8 bytes  
Size of char = 1 byte  
Size of char in bits= 8

### **Comma Operator:**

The **comma operator** (represented by the token ,) is a binary **operator** that evaluates its first operand and discards the result, it then evaluates the second operand and returns the value (and type). The **comma operator** has the lowest precedence of any **C operator**.

```
int x, y;
x=(10,20,30);
printf("x=%d",x);
x=33, y=22;
printf("\nValue: %d\n", (x,(y+10)));
```

**Output: x=30**

**Value: 22**

### **ASCII (American Standard Code for Information Interchange)**

- Unique to each Programming Language
- Characters are represented as 0s and 1s i.e. in binary numbers
  - Each character is mapped to a number
- Different mapping schemes exist like Extended Binary Coded Decimal Interchange Code (EBCDIC, ASCII)
- Uppercase Characters are from 'A' (65) to 'Z' (90)
  - Lowercase from 'a' (97) to 'z' (122)
  - Digits '0' (48) to '9' (57)
- Example AB is stored as 6566 in consecutive memory locations

```
// C program to print ASCII Value of
Character
int main()
{
 char c = 'A';
 printf("The ASCII value of %c is %d", c, c);
}
```

Output

The ASCII value of A is 65

For example, the ASCII value of 'A' is 65.

What this means is that, if you assign 'A' to a character variable, 65 is stored in the variable rather than 'A' itself.