DEPARTMENT OF B.C.A

FIRST SEMESTER

CORE PAPER II - PROBLEM SOLVING TECHNIQUES

UNIT I: Introduction: History, characteristics and limitations of Computer. **Hardware/Anatomy of Computer**: CPU, Memory, Secondary storage devices, Input Devices and Output devices. **Types of Computers**: PC, Workstation, Minicomputer, Main frame and Supercomputer. **Software**: System software and Application software. **Programming Languages**: Machine language, Assembly language, High-level language, 4GL and 5GL-Features of good programming language. **Translators**: Interpreters and Compilers.

UNIT II: Data: Data types, Input, Processing of data, Arithmetic Operators, Hierarchy of operations and Output. Different phases in Program Development Cycle (PDC). **Structured Programming: Algorithm**: Features of good algorithm, Benefits and drawbacks of algorithm. **Flowcharts**: Advantages and limitations of flowcharts, when to use flowcharts, flowchart symbols and types of flowcharts. **Pseudocode**: Writing a pseudocode. **Coding, documenting and testing a program**: Comment lines and types of errors. **Program design**: Modular Programming.

UNIT III: Selection Structures: Relational and Logical Operators -Selecting from Several Alternatives – Applications of Selection Structures. **Repetition Structures**: Counter Controlled Loops –Nested Loops– Applications of Repetition Structures.

UNIT IV: Data: Numeric Data and Character Based Data. **Arrays**: One Dimensional Array - Two Dimensional Arrays – Strings as Arrays of Characters.

UNIT V: Data Flow Diagrams: Definition, DFD symbols and types of DFDs. **Program Modules**: Subprograms-Value and Reference parameters- Scope of a variable - Functions – Recursion. **Files**: File Basics-Creating and reading a sequential file- Modifying Sequential Files.

TEXT BOOK:

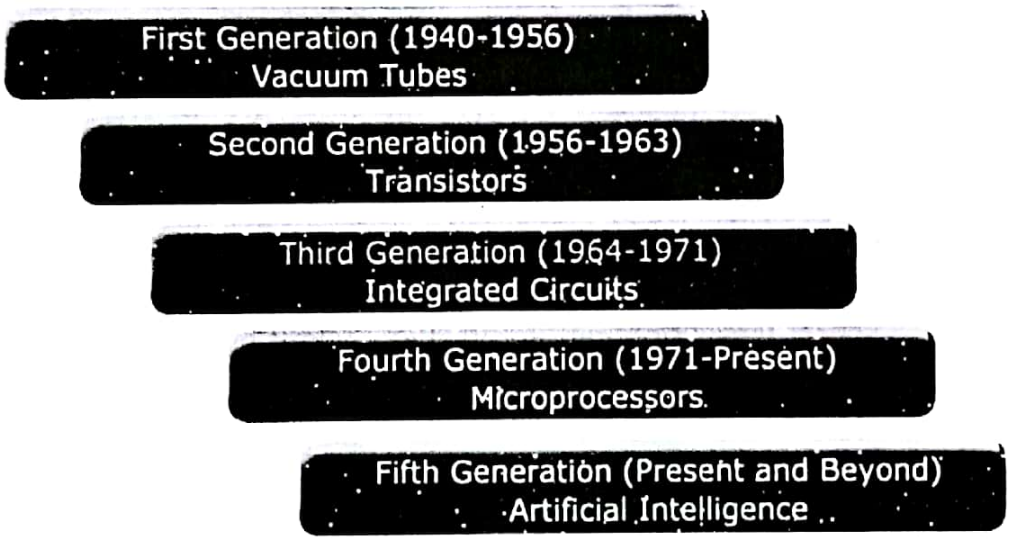
Stewart Venit, "*Introduction to Programming: Concepts and Design*", **Fourth** Edition, 2010, Dream Tech Publishers.

UNIT I

Introduction: History, characteristics and limitations of Computer. **Hardware/Anatomy of Computer**: CPU, Memory, Secondary storage devices, Input Devices and Output devices. **Types of Computers**: PC, Workstation, Minicomputer, Main frame and Supercomputer. **Software**: System software and Application software. **Programming Languages**: Machine language, Assembly language, High-level language, 4GL and 5GL-Features of good programming language. **Translators**: Interpreters and Compilers.

DEFINITION: A computer is a mechanical or electronic device that can efficiently store, retrieve and manipulate large amounts of information at high speed and with great accuracy. It can execute tasks without human intervention by carrying out a list of instructions called a **program**. The evolution of computers has been distinctly divided into five generations. The various application areas of computers are education, business, communication, science, engineering, entertainment, banking, health etc.

HISTORY OF COMPUTERS



- **Charles Babbage** designed a computer called "**Analytical Machine**" in mid-1800s. It was built with hundreds of axles and gears and could process up to 40-digit numbers. Unfortunately, Babbage never finished his work since his ideas were too advanced for the existing technology and also he could not get enough financial aid for his project.
- In 1945, a team at the University of Pennsylvania completed their work on the world's first fully-operable electronic computer called **ENIAC**-Electronic Numerical Integrator and Calculator. It can perform 5,000 additions per second with incredible accuracy. It was 80 feet long, 8 feet high and weighing 33 tons with 17,000 **vacuum tubes** and consumed 1,75,000 watts of electricity.

- In 1955, about 300 computers were built by IBM and were used largely by business, universities and government agencies.
- Computers later became significantly faster and reliable as large and heat producing vacuum tubes were replaced by small **transistors**. Due to their low energy needs and size, they can be packed closer in a compact design.
- In early 1960s, DEC (Digital Equipment Corporation) took feature of small, efficient packages of transistors called as **Integrated Circuits (ICs)** to create **minicomputer**, which are smaller and also cheap in price.
- Later **mainframes** come into existence.
- In 1970s, computers have become a house hold appliance. This was achieved by the invention of **microchip**, a piece of silicon about a size of a stamp but can hold numerous components within it.
- In 1974, the cousin of microchip came into existence called **microprocessor,** after which actual **PC (Personal Computer)** arrived. It was a relatively inexpensive machine, small enough to fit on a desktop. PC attracted many professionals and hobbyists. In 1977, a better and more useful computer **APPLE II** came. In 1981, IBM brought widely popular **IBM PC** which assured the future of PC.
- IBM threw out Macintosh and accounted 95% of PC market in 1984. IBM relayed on MS Windows OS for its operations which includes word processors, photo editing programs, web browsers and a few computer games.
- Now many PCs from IBM, Dell, Gateway, Compaq , Acer, Samsung etc are in the market.
- Supercomputers are even more powerful than mainframes and can process over one billion instructions per second.
- Among all improvements for the last 10 years in IT industry, the Internet stood first. **The Internet** is a world-wide collection of networks, interlinked computers that are able to share resources and data via cable or phone lines.
- The two main attractions of Internet are
    - **WWW (World Wide Web)** is a vast collection of linked documents (web pages) created by Internet users and stored on thousands of Internet-connected computers.
    - **E-Mail (Electronic Mail)** allows anyone with access to the Internet to use computer to exchange messages with little or no cost with another Internet user anywhere in the world.

CHARACTERISTICS OF COMPUTER

1. **Speed**: A computer is a fast electronic device that can solve problem in few seconds. The speed of a computer generally depends upon its hardware configuration.

2. **Storage Capacity:** A computer can store huge amount of data in its different storage area in different formats. Generally there are two storage areas: Main memory and Secondary memory.
3. **Accuracy:** A computer carries out operation with great accuracy.
4. **Reliability:** A computer usually produces results without errors-if error means, mostly there are human generated and so they are trustworthy machines.
5. **Versatility:** They can perform many different tasks for different purposes.
6. **Diligence:** Computers can perform repetitive calculations any number of times with the same accuracy.
7. They don't suffer from tiredness, fatigue, lack of concentration.

LIMITATIONS OF COMPUTER

Although a computer is far better in performance than a human being, it fails in certain ways as follows:

(i) Computers cannot think and they can't do any job unless they are first programmed with specific instructions for same. They work as per stored instructions. Algorithms are designed by humans to make a computer perform a special task.

(ii) Computers are incapable of decision making as they do not possess the essential elements necessary to take a decision i.e. knowledge, information, wisdom, intelligence and the ability to judge.

(iii) Though computers are helpful in storage of data and can contain the contents of encyclopedias even, but only humans can decide and implement the policies.

(iv) Although the trend today is to make computers intelligent by inducing artificial intelligence (AI) in them, they still do not have any decision-making abilities of their own. Thus, their IQ level is zero. They need guidance to perform various tasks.

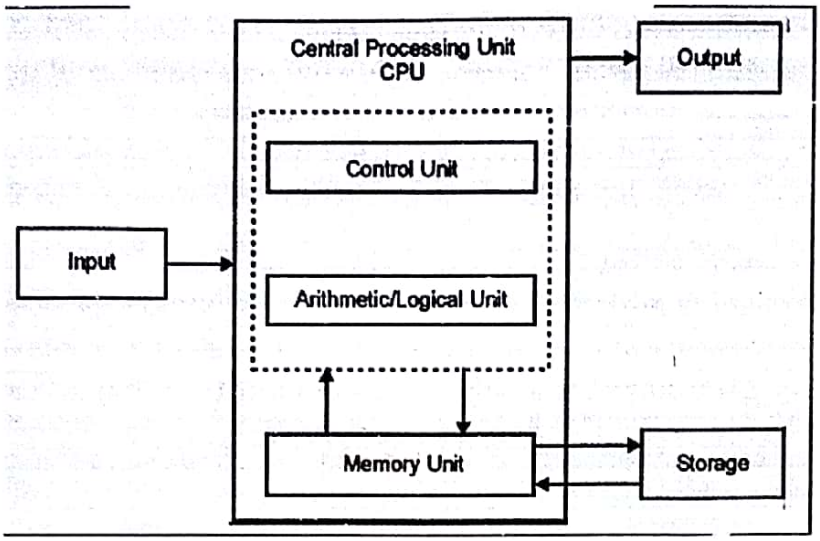
(v) Overall, a computer was built by humans and new technology comes out all the time. Hardware will eventually fade out and deteriorate.

HARDWARE/ANATOMY OF COMPUTER

A Computer must have the ability to input, store, manipulate and output data. These functions are carried out by the five main components of a computer system:

- CPU (Central Processing Unit)
- Internal Memory (RAM and ROM)
- Mass storage devices (Disk Drives)
- Input devices
- Output devices

Components used by a computer but located outside the system unit are called as **Peripherals.** All the physical equipment that makes up the computer system is known as **Hardware.**

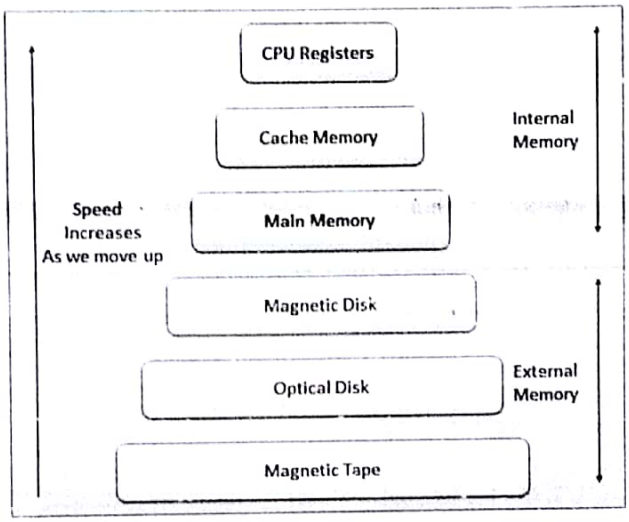


1. **CENTRAL PROCESSING UNIT**: The function of any computer revolves around CPU-"the brain" of the computer. It controls all other hardware components. The main operations of the CPU include four phases.
   - Fetching instructions from the memory.
   - Decoding the instructions to decide what operations are to be performed.
   - Executing the instructions.
   - Storing the results back in the memory.

   The three main components of CPU are:
   - **ALU (Arithmetic Logic Unit):** ALU performs arithmetic and logical operations on the data.
   - **CU (Control Unit):** Control unit controls the flow of data and information. It maintains the sequence of operations being performed by CPU. CU guides ALU about the operations that are to be performed and also suggests the I/O device to which the data is to be communicated. It uses the **Program Counter (PC)** register for retrieving the next instruction and **Status Register** for handling conditions like overflow of data.
   - **Registers:** Special purpose temporary storage units are called as registers-PC, IR, MAR, MBR, MDR, AC etc.

2. **MEMORY UNIT**: The memory unit of a computer is used to store data, instructions for processing data, and intermediate results of processing and final processed information. There are two types of memory like Primary and Secondary memory.
   - **Primary Memory**: Primary memory is available as built-in memory. It is represented as a set of locations with each location occupying 8-bits. The commonly used primary memories are:

- **ROM (Read Only Memory or Permanent Memory)** stores data and instructions even though the computer is turned off. Generally it is used to store the Basic Input/Output System (BIOS), which performs the Power On Self Test (POST).
- **RAM (Random Access Memory)** or Read/Write Memory or Volatile memory retains the data as long as there is power supply. There is also SRAM and DRAM in which SRAM is fast and expensive than DRAM.
- **Cache Memory** is placed between CPU and main memory. It is used to store the data and related instructions which are often used. Performance of cache is measured by Hit Ratio.



➢ **Secondary Memory:** External storage devices with non-volatile memory are connected externally to the computer.

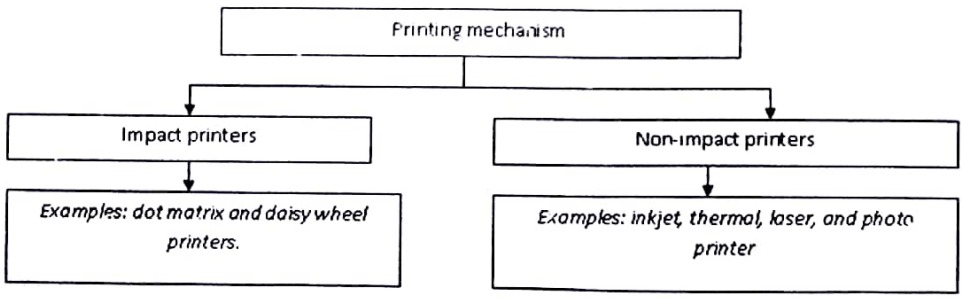
- **Magnetic storage** devices store information that can be read, erased and rewritten a number of times. Eg: Floppy disk, hard disk, magnetic tapes.
- **Optical storage devices** use the laser beam to read and write data. **Eg**: CD-ROM, CD-RW, DVD-ROM.
- **Magneto-optical storage devices** have high storage capacity as they use laser beams and magnets for reading and writing data to the device. The end-user can modify the data stored for multiple times. **Eg:** Sony minidisc.
- **USB Drive (Universal Serial Bus Drive)** is a removable storage. It is pretty fast, compact and larger in capacity. **Eg:** Pendrive.

3. **INPUT DEVICES:** An input unit is an electronic device which is used to feed data and control signals to a computer-they are connected to computer using cables. **Eg:** Keyboard, Mouse, Scanner, Trackball, Light pen, Joystick, OCR, Digital Camera.

- **Keyboard**: Keyboard is the most common input device used by more users. The classification of keys are:
    a. Function keys perform a specific task such as searching a file or refreshing a web page.
    b. Modifier keys are shift and control keys used to modify the casing style of a character or symbol.
    c. Cursor-movement keys are up, down, left and right keys which are used to modify the direction of the cursor on the screen.
    d. Spacebar key shifts the cursor to the right by one position.
    e. Numeric keypad holds numbers and operators.
- **Mouse:** Mouse allows the user to select elements (tools, icons, buttons) by pointing and clicking them. Mouse can be used to draw and paint on the screen. It is called as **Pointing Device** because it is used to change the position of cursor. Mouse has two buttons, a wheel on the top and a ball at its base.
- **Scanner:** It is used to convert documents and images as the digitized images understandable by computers-the digitized images can be black and white or colour. Proportion of red, green, blue will determine the colour proportion.

4. **OUTPUT DEVICES**: Output unit is an electronic device used to communicate the output obtained after processing a specific task to the user. Eg: Printers, Monitor, and Speaker.

- **Monitor**: Monitor or screen is the commonly used output device that produces visual presentation of textual and graphical information. The monitors can be Cathode Ray Tube (CRT) , Liquid Crystal Display (LCD) or Light Emitting Diode (LED). CRTs are large while LCDs and LEDs are thin, and occupy less space. A monitor can be characterized by its size and resolution. The monitor size is the length of the screen measured diagonally. Resolution or dot pitch is the number of picture elements (pixels) per unit square.
- **Printer**: Printer is used to produce a hard copy of the electronic text displayed on the screen. The various types of printers are dot matrix printers, inkjet printers and laser printers. The performance of a printer is measured in terms of Dot Per Inch (DPI) and Pages Per Minute (PPM).



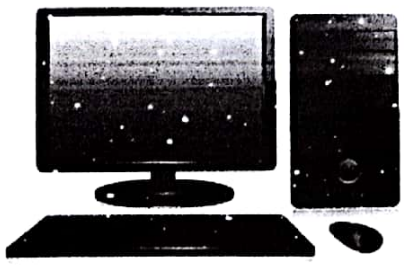
- **Speaker**: Speaker is an electromechanical unit that converts electrical signals into sound. They are used to give warning sounds and internet audio.

TYPES OF COMPUTERS

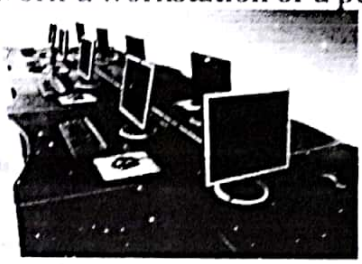
Computers can be broadly classified by their speed and computing power.

PC (Personal Computer)	It is a single user computer system having moderately powerful microprocessor.
Workstation	It is also a single user computer system, similar to personal computer with more powerful microprocessor.
Mini Computer	It is a multi-user computer system, capable of supporting many users simultaneously.
Main Frame	It is a multi-user computer system, capable of supporting hundreds of users simultaneously. Software technology is different from minicomputer.
Supercomputer	It is an extremely fast computer, which can execute hundreds of millions of instructions per second.

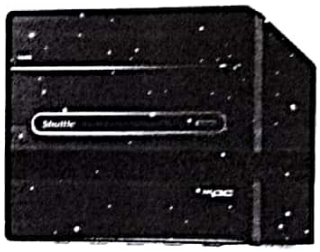
1. **PC (Personal Computer):** A PC can be defined as a small, relatively inexpensive computer designed for an individual user. PCs are based on the microprocessor technology that enables manufacturers to put an entire CPU on one chip. Businesses use personal computers for word processing, accounting, desktop publishing, and for running spreadsheet and database management applications. At home, the most popular use for personal computers is playing games and surfing the Internet. Although personal computers are designed as single-user systems, these systems are normally linked together to form a network. In terms of power, now-a-days high-end models of the Macintosh and PC offer the same computing power and graphics capability as low-end workstations by Sun Microsystems, Hewlett-Packard, and Dell. The principal characteristics of personal computers are that they are single-user systems and are based on microprocessors.



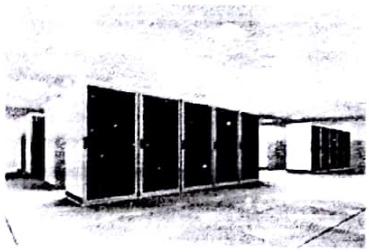
2. **Workstation:** Workstation is a computer used for engineering applications (CAD/CAM), desktop publishing, software development, and other such types of applications which require a moderate amount of computing power and relatively high quality graphics capabilities. Workstations generally come with a large, high-resolution graphics screen, large amount of RAM, inbuilt network support, and a graphical user interface. Most workstations also have mass storage device such as a disk drive, but a special type of workstation, called diskless workstation, comes without a disk drive. Common operating systems for workstations are UNIX and Windows NT. Like PC, workstations are also single-user computers like PC but are typically linked together to form a local-area network, although they can also be used as stand-alone systems. In networking, workstation refers to any computer connected to a local-area network-a workstation or a personal computer.



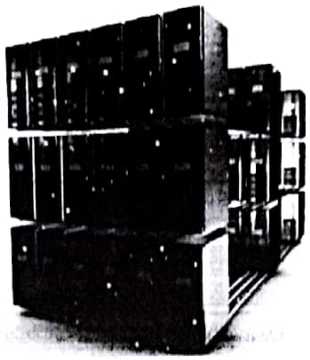
3. **Minicomputer:** It is a midsize multi-processing system capable of supporting up to 250 users simultaneously.



4. **Mainframe:** Mainframe is very large in size and is an expensive computer capable of supporting hundreds or even thousands of users simultaneously. Mainframe executes many programs concurrently and supports simultaneous execution of programs. The chief difference between a supercomputer and a mainframe is that a supercomputer channels all its power into executing a few programs as fast as possible, whereas a mainframe uses its power to execute many programs concurrently. In some ways, mainframes are more powerful than supercomputers because they support more simultaneous programs. But supercomputers can execute a single program faster than a mainframe.

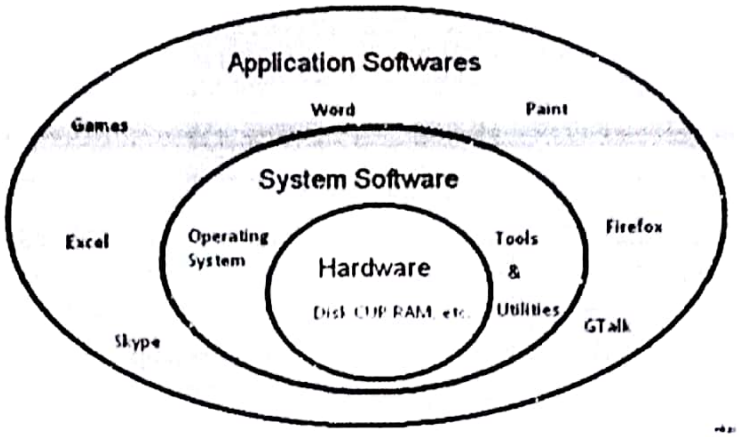


5. **Supercomputer:** Supercomputers are one of the fastest computers currently available. Supercomputers are very expensive and are employed for specialized applications that require immense amount of mathematical calculations (number crunching). For example, weather forecasting, scientific simulations, (animated) graphics, fluid dynamic calculations, nuclear energy research, electronic design, and analysis of geological data (e.g. in petrochemical prospecting). The best known supercomputer manufacturer is Cray Research.



SOFTWARE

Software refers to a set of computer programs that are required to enable the hardware to work and perform these operations effectively.



Two tasks of computer software are:

- Control and co-ordinate the hardware components and manage their performances (**System Software**)
- Enable the users to accomplish their required tasks (**Application Software**)

1. **SYSTEM SOFTWARE**: Depending upon the task performed, the system software can be classified into two:

   ➢ **System Management Programs** enable the users to perform certain utility functions like creating backup files, recovering damaged files and

merging files. They minimize the human intervention during processing and help in maximizing productivity of a computer system.

Eg: Operating system, Utility programs, Device drivers

- ✓ Operating System is the principal component of system software and is responsible for overall management of computer resources. It also provides an interface between the computer and the user. The major functions of OS are process management, memory management, file management, I/O management, ensuring security of access to computer resources, commands and instructions' interpretation and assignment of utility programs.
- ✓ Utility Programs are small programs which provide additional capabilities to OS-they are not an essential part of OS (ie) OS can execute most of the programs without having the utility programs. Some of the utility tasks are search, print, disk defragmenter, system profiler, encryption, virus scanner, and backup and data recovery.
- ✓ Device Driver acts as a translator between I/O devices and the computer. It interprets the input provided by the user into the computer understandable form and directs it to the OS. It is a special software that enables a hardware device to perform an operation according to the command of the OS.

➢ **System Development Programs** are also called as programming software which allows the users to develop programs in different programming languages.

Eg: Language translators, linkers, debuggers, editors.

- ✓ Language translators are compilers, interpreters, assemblers.
- ✓ Linkers are used to link the runtime libraries.
- ✓ Debuggers are of two types. Machine level debugger debugs the object code of the program and shows all the lines where bugs are detected. Symbolic debugger debugs the original source code (HLL).
- ✓ Editors are text editor, digital audio editor, graphics editor, binary file editor, HTML editor, source code editor.

2. **APPLICATION SOFTWARE:** Application software includes a variety of programs that are designed to meet the information processing needs of users. They can be broadly classified into two as:

➢ **Standard Application Programs** are also known as general purpose application programs perform certain common information processing tasks for the users. These programs significantly increase the productivity of users but also decreases the time and cost.

Eg:

- ✓ Word Processor helps to create, edit, print documents like letters, reports etc.

- ✓ Spreadsheet helps to manipulate and calculate large amounts of tabular data. It is used as a computational tool by applying formulae on the value of cells.
- ✓ DBMS is computer software that helps us to store and maintain records in a database. Database is a set of records stored in a structured manner whereas record is a set of similar or dissimilar values related to an entity. DBMS allows the user to perform various operations such as search, update, delete on the data stored in the database. The advantages of DBMS are:
  - enables the user to organize all the information in a strategic manner.
  - helps the user to maintain consistency in the information.
  - helps to maintain data integrity.
  - allows the user to maintain the security of the confidential information by protecting the database by password.
  - accessing information from any system connected in a network.

Data models available are relational, hierarchical and network. DBMS includes a query language system which enables a user to perform various operations.

- ✓ DTP (Desk Top Publishing) enables the user to perform various activities required for publishing a page or a set of pages. **Eg**: Organizing the content layout by setting margins and justification properties. There will be page layout software that enables the user to view changes done at the time of designing the document. They use 'WYSIWYG' (What You See Is What You Get) technique. The software can manipulate on different components like text, graphics and images. **Eg**: Quark Xpress, InDesign, Microsoft Publisher, Apple Pages.
- ✓ Web Brower is software that is used to access the Internet and WWW. **Eg**: Internet Explorer from Microsoft, Firefox from Mozilla, Netscape Navigator from Netscape, Chrome from Google.

  IE2.0=1995 by Microsoft using Macintosh OS.

  IE3.0=1996 using Windows 95 OS. It has features like Internet Mail, Windows Address Book, and Windows Media Player.

  IE4.0=1997 including Microsoft Outlook Express (E-mail Software).

  IE8.0=A most secured web browser.

➢ **Unique Application Programs or end-user application programs** refer to the development of own programs to accomplish certain tasks that are unique to end-users' areas of operations. These programs are usually developed in any high level language by skilled programmers who have deep knowledge in programming environment. **Eg**: Inventory management, pay bills, examination results, ticket reservation, income tax processing etc.

PROGRAMMING LANGUAGES

To write computer programs, we need programming languages as computers don't understand natural languages. A programming language is a set of symbols and the rules governing their use that is utilized in constructing programs.

Why natural languages are not used?

- Natural languages are ambiguous, vaguely structured and have very large and changing vocabularies.
- In case of natural languages, we can understand even while using poor grammar and vocabulary, while we cannot expect the system to do like that.

The fundamentals types of programming languages are as:

1. ***MACHINE LANGUAGE*** is the native language of computers. Computers are digital devices which have only 2 states-ON (1) and OFF (0). Since a machine language program consists of a sequence of bits-0's and 1's, it is the only language which the computer can understand directly. The numbers specified and the order in which they appear, tell the computer what to do. Instruction in machine language consists of 2 parts:

OPCODE	OPERAND

OPCODE (operation code) tells the computer what functions are to be done.
OPERAND tells where to find or store the data.

Advantages:

- Translation free
- High speed

Disadvantages:

-Machine dependent.

-**Complex language**: It is very difficult for humans to read and write machine language programs as all instructions have to be converted into binary code. So the need is hardware expert rather than programmer.

-**Error prone**: Programmer has to remember all opcodes and hard to keep track of program logic.

-**Tedious**: Hard to modify and correct a program and so it is time-consuming.

2. ***ASSEMBLY LANGUAGE*** is a symbolic representation of machine language. There is usually a one-to-one correspondence between the two (ie) each assembly language instruction translates into one machine language instruction. It was the stepping stone for all subsequent language development created by IBM. Comparatively, people can easily recognize assembly language than machine language. It uses meaningful abbreviations for machine specific instructions called as mnemonics, which is usually 3 letters long. Whenever an assembly language program is carried out by a computer, it must be translated into machine language. This is done by assembler. The principal register to be used is accumulator designated by 'A'. One operand should be at A, all operations are carried out at A and the result will be stored at A.

Syntax:

[label] <opcode> <operands> [;comment]

Eg: ADD A,B

Functions of an assembler:

1. Allows the programmer to use mnemonics and symbolic names in writing source codes.
2. Checks the syntax of assembly program and generates error messages or syntax errors.
3. Translates the register address to system address.
4. Assembles all the instruction in the main memory for execution.
5. Provides linking facility among the subroutines.
6. Generation of output on required output medium.

Advantages:

- Easy to understand and use. (no need to memorize opcodes)
- Less error prone
- Efficiency (fast and use less memory)
- More control over hardware

Disadvantages:

-Machine dependent.
-Slow development time.
-Harder to learn.
-No standardization.

3. ***HIGH LEVEL LANGUAGE (HLL)*** usually contains English words and phrases. A single instruction in HLL usually translates into many instructions in machine language. A HLL program must be translated into machine language as they should be understood by a computer. HLL have several advantages over machine and assembly languages as,

+HLL is easier to learn and use as they are closer to natural languages
+Resultant programs are easier to read and modify
+ A given HLL doesn't differ very much from computer to computer (portable)
+ HLL permits faster development of program in lesser cost
+HLL is useful in developing complex software as it supports many complex data structures
+The programmer doesn't need to learn the instruction set of each computer (machine independent)
+HLL supports the ideas of abstraction so that the programmers can concentrate on the solution rather than on lower level details.
+ Compilers are designed efficiently to point out the errors (easy debugging).
+ Documentation can be made easily.

But sometimes HLL are less powerful and produce less efficient programs than their assembly language counterparts.

Eg:

- First HLL was **FORTRAN** (FORmula TRANslator) used for engineering and scientific applications in 1950.
- **Ada** (named after Ada Auguta Byron) was used mostly in US department of Defence.
- **BASIC** (Beginner's All-purpose Symbolic Instruction Code) was a popular and easy to learn language developed in 1960s.
- **C and C++** were developed Dennis Ritchie and Bjarne Stroustrap respectively.
- **COBOL** (COmmon Business Oriented Language) was used to write program for business applications.
- **Java** is a very popular modern language especially for networking and web applications.
- **Pascal** was a popular language used as a basis of teaching programming techniques.
- **Visual Basic** is a version of BASIC that is well-suited for software that runs on GUIs.

4. ***4GL*** are non-procedural languages which are commonly used to access the databases (ie) the computer is instructed what rather than how. They are easier to write, but the user does not have any control over it. With increase in power and speed of hardware and with diminishing costs, the use of 4GLs has spread. The components of 4GL are query languages, report generators and application generators. The advantage of 4GL is shorter development and debugging time while requirement of more disk space is its demerit.

5. ***5GL*** can process natural languages (ie) they can accept, interpret and execute instructions in natural languages. So the user will be free to learn any programming language. 5GLs are closely related to artificial intelligence and expert systems.

Features of good programming language:

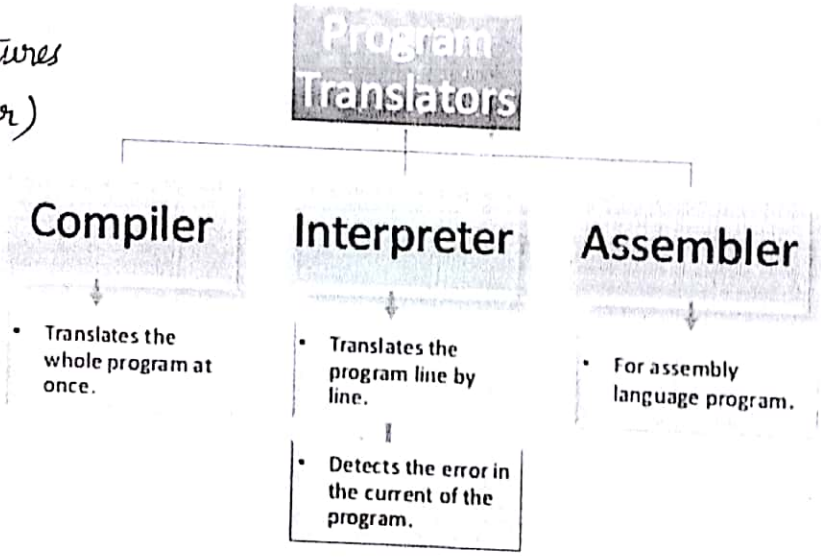
1. **Ease of use**-easy in writing codes and clarity in syntax, easy to grasp and readable.
2. **Portability**-code should be distributed across multiple platforms. PL should be independent of any particular hardware or OS.
3. **Naturalness for the application**-The language should have a syntax which allows the program structure to show the underlying logical structure of algorithm or flowchart (ie) promote structural programming.
4. **Reliability**-The language should have the support of error detection as well as prevention which makes a system failure free.

5. **Performance**-The application developed by a PL should tap the maximum performance of available hardware in terms of high speed and efficiency with low cost.
6. **Compact code**-Develop the code with minimum number of lines
7. **Maintainability**- Easy to accept changes
8. **Reusability**- Facilitate the adoption of code in similar applications.
9. Ability to provide interface to other language. (like native methods in Java)
10. **Concurrency support**-Extent to which inherent languages support the construction of ~~the~~ code with multiple threads of control (parallel processing).
11. **Standardization**- Extent to which PL has been supported by recognized bodies like ANSI, ISO. Non-standard PL will soon become obsolete and surely produce poor code.

TRANSLATORS

Computers only understand machine code (binary code). To get around the issue, the high-level and low-level program code (source code) needs to pass through a translator. A translator will convert the source code into machine code (object code). There are several types of translator programs, each able to perform different tasks.

(Explain features of assembler)



Interpreter: Interpreter programs are able to read, translate and execute one statement at a time from a high-level language program. The interpreter stops when a line of code is reached that contains an error. Interpreters are often used during the development of a program. They make debugging easier as each line of code is analysed and checked before execution. Interpreted programs will launch immediately, but program may run slower than a complied file. No executable file is produced. The program is interpreted again from scratch every time the user launches it.

Compiler: System software that stores the complete program, scans it, translates the complete program into object code and then creates an executable code is called a compiler. Compilers are used to translate a program written in a high-level language into machine code (object code). Once compiled, the translated program file can then be directly used by the computer and is independently executable. Compiling may take some time but the translated program can be used again and again without the need for recompilation. An error report is often produced after the full program has been translated. Errors in the program code may cause a computer to crash. These errors can only be fixed by changing the original source code and compiling the program again. On the face of it, compilers compare unfavourably with interpreters because they –

- are more complex than interpreters
- need more memory space
- take more time in compiling source code

The steps in compiling source code into executable code using compiler are:

- **Pre-processing** – In this stage pre-processor instructions, typically used by languages like C and C++ are interpreted, i.e. converted to assembly level language.
- **Lexical analysis** – Here all instructions are converted to **lexical units** like constants, variables, arithmetic symbols, etc.
- **Parsing** – Here all instructions are checked to see if they conform to **grammar rules** of the language. If there are errors, compiler will ask ~~you~~ us to fix them before ~~you~~ can proceed.
- **Compiling** – At this stage the source code is converted into **object code**.
- **Linking** – If there are any links to external files or libraries, addresses of their executable will be added to the program. Also, if the code needs to be rearranged for actual execution, they will be rearranged. The final output is the **executable code** that is ready to be executed.

COMPILER	INTERPRETER
Translates high-level languages into machine code	Temporarily executes high-level languages, one statement at a time
An executable file of machine code is produced (object code)	No executable file of machine code is produced (no object code)
Compiled programs no longer need the compiler	Interpreted programs cannot be used without the interpreter
Error report produced once entire program is compiled	Error message produced immediately and program stops at that point
Compiling may be slow, but the resulting program code will run quickly (directly on the processor)	Interpreted code is run through the interpreter (IDE), so it may be slow

UNIT II

Data: Data types, Input, Processing of data, Arithmetic Operators, Hierarchy of operations and Output. Different phases in Program Development Cycle (PDC).**Structured Programming**: **Algorithm**: Features of good algorithm, Benefits and drawbacks of algorithm. **Flowcharts**: Advantages and limitations of flowcharts, when to use flowcharts, flowchart symbols and types of flowcharts. **Pseudocode:** Writing a pseudocode. **Coding, documenting and testing a program**: Comment lines and types of errors. **Program design**: Modular Programming.

DATA

Data is the entity which holds the value that will be manipulated in the program. There are two fundamental types of data.

1. **Numeric data** consists of numbers-this type of data is usually subdivided into integer or floating point data. Integer data refers to whole numbers while float data refers to the number with decimal point.
2. **Character or Alphanumeric data** consists of characters-characters usually enclose within single quotes or double quotes (strings). Special characters like , : : ! etc is also a valid character.

INPUT: The input operation transmits data from an outside source to the program. Often this data will be typed at the keyboard by the user. There will be a set of input statements in all programming languages to scan the input.

Eg for input statements:

BASIC-INPUT
C-scanf(), getchar(),getc(),gets()
JAVA-readLine(), nextLine()
C++ – cin

Usually it is good practice to have a prompt before entering input as it will help the user to understand what type of data has to be given as input. There may be other forms of input like the user may input by clicking or moving the mouse. There will be another common form of input which does not involve the user at all (ie) data can be transmitted to a program from a data file stored on disk.

VARIABLE is the quantity that can change value during the execution of program. So to refer to the data in the subsequent program statements, we will use the variable name. Al most all the programming languages follow the same rule to form a variable name (ie) start with alphabet, now use of blank space etc. While naming a variable, give a meaningful name to the relevant of data, so that all the end-users can understand easily. A variable has to be declared along with its data type in order to ensure that kind of data it can hold.

Eg: Addition of two numbers

Start
Read two values in a,b
Compute c->a+b
Display c
Stop

The above sequence is a pseudocode and it is not relevant to any language. Variables are a,b and c. Input is caught by a and b.

PROCESSING OF DATA: The read values are now at a and b. They both are summed up and the value is stored at c. To know the result, the c's value is displayed.

VARIABLE	BEFORE	AFTER
a	10	10
b	20	20
C	Undefined	30

Please note that the values of a and b don't change even though it appears at the right side of the statement. But the value of c is undefined when the user gives 10 and 20 to a and b respectively. Therefore in an assignment statement, RHS of the expression will be computed before LHS.

Let x=100;
:
:
x=200;

What happens if a variable that already has a value is reassigned with another value? In such a case the latest value will overwrite the old value in its memory location (ie) x holds 200 and not 100.

Operators: They are the symbols which indicate what operation has be done with the data. The types of operators are:

ARITHMETIC OPERATORS	
Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponential	^

RELATIONAL OPERATORS	
Less than	<
Less than or equal to	<=
Greater than	>
Greater than or equal to	>=
Equal to	=
Not equal to	<>

LOGICAL OPERATORS
AND
OR
NOT

Hierarchy of operations:

1. First perform the operations in parenthesis.
2. Next perform exponentials
3. Do multiplications and divisions from left to right if there is more than one.
4. Finally do additions and subtractions from left to right if there is more than one.

Eg: 3*(6+2)/12-(7-5)^2*3
=3*8/12-2^2*3
=3*8/12-4*3
=24/12-4*3
=2-4*3
=2-12
=-10

OPERATORS	ORDER OF EXECUTION
Arithmetic operators	First
Relational operators	Second
Logical operators	Third

OUTPUT: A program's output is data sent from memory to the screen, printer or to a file. The output is normally the results of the program's processing part.

Eg for output statements:

BASIC-INPUT
C-printf(), putchar(),putc(),puts()
JAVA-System.out.println()
C++ – Cout

The output of a program can be well understood if they are prompted with some information. For example, instead of printing the result of sum of 2 numbers 'c', it will be more attractive if we write as display "The sum =",c. Of course each programming language has its own output statements to create various kinds of screen. Therefore the basic building blocks of a program are input, processing and output.

PROGRAM DEVELOPMENT CYCLE (PDC)

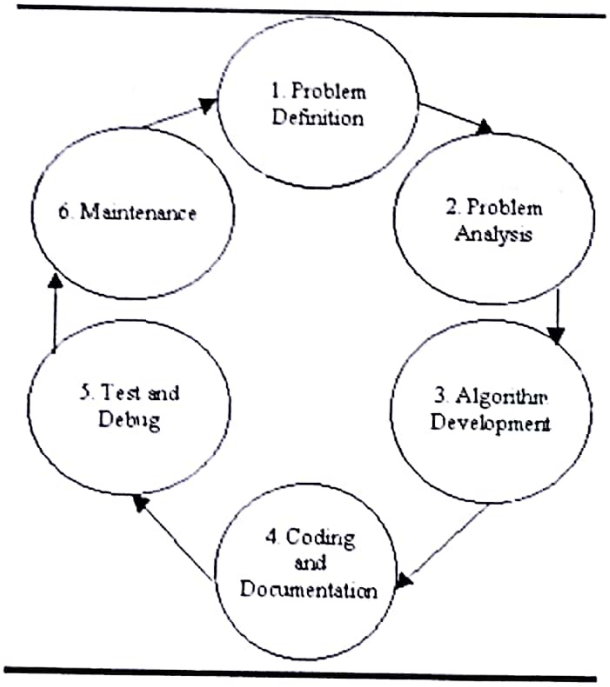
In developing a computer program to solve a given problem, we follow some time-honoured problem solving principles. We must

- Completely understand the problem
- Devise a plan to solve it
- Carry out the plan
- Review the results

When this approach is applied to program writing, it entails carrying out the following tasks.

- Analyze the problem
- Design the program
- Code the program
- Test the program

This process is called a **Program Development Cycle**. The word Cycle is used here because we have to often return to previous steps and make modifications before the process is complete. But in large applications, the cycle is rarely ever complete-these programs are continually evaluated and modified to meet changing demands. The team of software designers, developers are required to interact with each other at each stage to bring the end product as per the client's requirements.



a) Problem Definition: A program is a set of instructions that instructs a computer how to perform a specific task. It is written in a high level language that can be implemented on a number of different processors. A collection of programs can be compared to a recipe book, where each recipe can be assumed as a program. Every recipe has a list of ingredients (fixed data) and a list of instructions detailing

exactly what to do with those ingredients. When we follow a recipe, we are actually executing a program. Firstly, the problem is analyzed precisely and completely. Based on understanding, the developer knows about the scope within which the problem needs to be developed. This step collects all the facts concerning the problem specifications.

- What is the required output?
- What is the supplied input?
- How will we obtain the required output from the given input?

These questions are directly related to the building blocks of a problem but in reverse order. If the requirements of the user are not properly understood, then the output is bound to fall short of the end-user's expectation. Proper analysis of user requirements is quite essential for developing the software within time frame, controlled cost and also lead to faster and accurate development of program. We choose variables to represent the given input and the required output. We also finalize the formulae to be carried out in the program.

b) Problem Analysis: After analyzing the problem, the developer needs to develop various solutions to solve the given problem. From these solutions, the optimum solution (by experimenting with all the solutions) is chosen, which can solve the problem comfortably and economically.

c) Algorithm Development: After selecting the appropriate solution, algorithm is developed to depict the basic logic of the selected solution. An algorithm depicts the solution in logical steps (sequence of instructions). Further, algorithm is represented by flowcharts and pseudo codes. These tools make program logic clear and they eventually help in coding.

ALGORITHM FOR SIMPLE INTEREST

Start
Read values for P, N and R
Compute SI=(P*N*R)/100
Display SI
Stop

d) Testing the Algorithm for Accuracy: Before converting the algorithms into actual code, it should be checked for accuracy. The main purpose of checking algorithm is to identify major logical errors at an early stage, because logical errors are often difficult to detect and correct at later stages. The testing also ensures that the algorithm is a "true" one and it should work for both normal as well as unusual data.

e) Coding: After meeting all the design considerations, the actual coding of the program takes place in the chosen programming language. Depending upon application domain and available resources, a program can be written by using

computer languages of different levels such as machine, assembly or high-level languages.

f) Test and Debug: Testing actually take place throughout the program development cycle at each step. For example, in the design phase, we should imagine that we are the computer and run through the algorithm using simple input data to see if the output comes or not. This is called as desk-checking. It is common for the initial program code to contain errors. A program compiler and programmer-designed test data machine tests the code for syntax errors. The results obtained and compared with results calculated manually from this test data. Depending upon the complexity of the program, several rounds of testing may be required.

g) Documentation: Once the program is free from all the errors, it is the duty of the program developers to ensure that the program is supported by suitable documentation. These documents should be supplied to the program users. Documenting a program enables the user to operate the program correctly. It also enables other persons to understand the program clearly so that it may, if necessary, be modified, or corrected by someone other than the original programmer.

h) Implementation: After performing all the above-mentioned steps, the program is installed on the end user's machine. In this stage, users are also provided with all the essential documents so that they can understand how the program works. The implementation can be viewed as the final testing because only after using the program, the user can point out the drawbacks, if any to the developers. Based on the feedback, the programmers can modify or enhance the program.

i) Maintenance and Enhancement: After the program is implemented, it should be properly maintained taking care of the changing requirements of its users and system. The program should be regularly enhanced by adding additional capabilities. This phase is also concerned with detecting and fixing the errors, which were missed in testing phase. Since this step generates user feedback, the programming cycle continues as the program modified or reconstructed to meet the changing needs.

STRUCTURED PROGRAMMING

Structured Programming is a method for designing and coding programs in a systematic, organized manner. Following the steps of the program development cycle, designing a program in a top-down fashion, using comments to document a program are some of the examples of structured programming. A well-structured design leads in a natural way, to well-structured, easy-to-read code. One of the goals

of structured programming is to create a program that is easy for programmers to read and for users to run.

ALGORITHMS

Definition: An algorithm is defined as a sequence of clear and explicit instructions that, when provided with a set of input values produces an output and then terminates.

Algorithms are one of the basic tools that are used to develop the problem solving logic. Algorithms help a programmer in breaking down the solution of a problem into a number of steps. Corresponding to each step, a program statement can be written. Algorithms can have steps that repeat or require decision (comparison) until the task is completed. Different algorithms may be written for a same problem with a different set of instructions in more or less the same time, space and effort. Algorithms are not computer programs as they cannot be executed by a computer.

Properties of a good algorithm:

- There must be no ambiguity in any instruction (ie) simple and concise.
- There should not be any uncertainty about which instruction is to be executed next.
- The algorithm should terminate after a finite number of steps (ie) the algorithm cannot be open-ended.
- The algorithm must be general enough to deal with any contingency (ie) must be complete and definitely solve the given problem.
- It should be use most efficient logic to solve the given problem within minimum time (time complexity).
- It uses minimal system memory for its execution (space complexity).
- It should able to generate more accurate results for a wide range of results.
- It should be easily turned to program of any language.
- It is designed with standard conventions so that others are able to easily modify it while adding additional functionality.

Example: Largest of 2 numbers-Value1 and Value2

Step 1: Start
Step 2: Read two numbers-Value1,Value2
Step 3: Compare Value1 and Value2. If Value1>Value2, Max=Value1.
Step 4: If Value2>Value1, Max=Value2.
Step 5: Display the value at MAX as result.
Step 6: Stop.

Benefits of algorithm:

1. It is a step-wise representation of a solution to a given problem, which makes it easy to understand.

2. An algorithm uses a definite procedure.
3. Writing algorithm and program separately simplifies the overall task by dividing it into two simpler tasks. While writing the algorithm, we can focus on solving the problem instead of concentrating on a particular language (Reduced Complexity).
4. It is not dependent on any programming language, so it is easy to understand for anyone even without programming knowledge (Increased Flexibility).
5. Every step in an algorithm has its own logical sequence so it is easy to debug.
6. By using algorithm, the problem is broken down into smaller pieces or steps and hence it is easier for programmer to convert it into an actual program.

Drawbacks of algorithm:

1. Writing algorithm is time consuming.
2. Difficult to show branching and looping in algorithms.
3. Big tasks are difficult to put in algorithms.

FLOWCHARTS

A flowchart can be defined as the pictorial representation of a process which describes the sequence and flow of the control and information in a process. The flow of information is represented in a step-by-step form in a flowchart. Flowchart uses different symbols for depicting different activities, which are performed at different stages of a process. Flowchart details the operations/steps in a pictorial format which is easier to understand than reading it in a textual format. A flowchart can be likened to the blueprint of a building.

Flowcharts are generally drawn in the early stages of formulating computer solutions. Flowcharts play a vital role in the programming of a problem and are quite helpful in understanding the logic of complicated and lengthy problems. Once the flowchart is drawn, it becomes easy to write the program in any high level language. A flowchart is a must for the better documentation of a complex program.

Flowchart Symbols:

Symbol	When it is used:
Start / Stop	This shapes indicates the start or end of a flowchart
Process	A rectangular box represents a process, this is doing something. E.g. total = num1+num2
Input/ Output	A parallelogram represents input or output. E.g. Input num1 Output total
Decision	A diamond shape represents a decision, YES or NO e.g. is it a weekday?
Sub-process	This shape is used when you want to divert to another flowchart to complete a task before returning to the original flowchart.

NOTE:

1. **Connector**: A small, labelled, circular flow chart shape used to indicate a jump in the process flow. Connectors are generally used in complex or multi-sheet diagrams.

2. **Arrow:** used to show the flow of control in a process. An arrow coming from one symbol and ending at another symbol represents that control passes to the symbol the arrow points to.

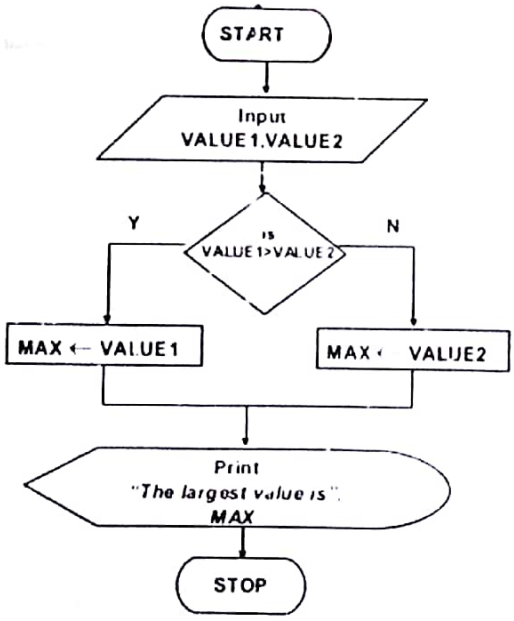
Basic guidelines for drawing a flowchart with the above symbols are that:

- In drawing a proper flowchart, all necessary requirements should be listed out in logical order.
- The flowchart should be neat, clear and easy to follow.
- There should not be any room for ambiguity in understanding the flowchart.
- The flowchart is to be read left to right or top to bottom.
- A process symbol can have only one flow line coming out of it.

When to use flowcharts?

1. To communicate to others how a process is done.
2. A flowchart is generally used when a new project begins in order to plan for the project.
3. A flowchart helps to clarify how things are currently working and how they could be improved. It also assists in finding the key elements of a process, while drawing clear lines between where one process ends and the next one starts.
4. Developing a flowchart stimulates communication among participants and establishes a common understanding about the process. Flowcharts also uncover steps that are redundant or misplaced.
5. Flowcharts are used to help team members, to identify who provides inputs or resources to whom, to establish important areas for monitoring or data collection, to identify areas for improvement or increased efficiency, and to generate hypotheses about causes.
6. It is recommended that flowcharts be created through group discussion, as individuals rarely know the entire process and the communication contributes to improvement.
7. Flowcharts are very useful for documenting a process (simple or complex) as it eases the understanding of the process.
8. Flowcharts are also very useful to communicate to others how a process is performed and enables understanding of the logic of a process.

Example: Largest of 2 numbers-Value1 and Value2



Advantages of flowchart:

1. The flowchart is an excellent way of communicating the logic of a program (ie) the diagrammatical representation of logic is easy to understand.
2. It is easy and efficient to analyze problem using flowchart.

3. During program development cycle, the flowchart plays the role of a guide or a blueprint and makes program development process easier.
4. After successful development of a program, it needs continuous timely maintenance during the course of its operation. The flowchart makes program maintenance easier.
5. It helps the programmer to write the program code easily.
6. It is easy to convert the flowchart into any programming language code as it does not use any specific programming language concept.
7. Appropriate documentation can be made by flowchart.

Disadvantages of flowchart:

1. The flowchart can be complex when the logic of a program is quite complicated.
2. Drawing flowchart is a time-consuming task.
3. Difficult to alter the flowchart. Sometimes, the designer needs to redraw the complete flowchart to change the logic of the flowchart or to alter the flowchart.
4. Since it uses special sets of symbols for every action, it is quite a tedious task to develop a flowchart as it requires special tools to draw the necessary symbols.
5. In the case of a complex flowchart, other programmers might have a difficult to understand the logic and process of the flowchart.
6. It is just a visualization of a program; it cannot function like an actual program.

Types of Flowchart:

1. **High-Level Flowchart:** A high-level (also called first-level or top-down) flowchart shows the major steps in a process. It illustrates a "birds-eye view" of a process. It can also include the intermediate outputs of each step (the product or service produced), and the sub-steps involved. Such a flowchart offers a basic picture of the process and identifies the changes taking place within the process. It is significantly useful for identifying appropriate team members (those who are involved in the process) and for developing indicators for monitoring the process because of its focus on intermediate outputs. Most processes can be adequately portrayed in four or five boxes that represent the major steps or activities of the process. In fact, it is a good idea to use only a few boxes, because doing so forces one to consider the most important steps. Other steps are usually sub-steps of the more important ones.
2. **Detailed Flowchart:** The detailed flowchart provides a detailed picture of a process by mapping all of the steps and activities that occur in the process. This type of flowchart indicates the steps or activities of a process and includes such things as decision points, waiting periods, tasks that frequently must be redone (rework), and feedback loops. This type of flowchart is useful

for examining areas of the process in detail and for looking for problems or areas of inefficiency.

3. **Deployment or Matrix Flowchart:** A deployment flowchart maps out the process in terms of who is doing the steps. It is in the form of a matrix, showing the various participants and the flow of steps among these participants. It is chiefly useful in identifying who is providing inputs or services to whom, as well as areas where different people may be needlessly doing the same task.

PSEUDOCODE

Pseudocode is a generic way of describing algorithm without referring to any specific programming language. Analyzing a detailed algorithm before developing a program is very time consuming. Hence there arises a need of a specification that only focuses on the program logic. Pseudo means false or imitation. It is used by the programmer for developing a computer program. Pseudocode is not written in syntax as per any programming language, rather written in normal English language. Hence it is an artificial and informal language that helps programmers to develop algorithms.

Guidelines:

- For looping and selection, the keywords that are to be used include Do While...EndDo; Do Until...Enddo; Case...EndCase; If...Endif; Call ... with (parameters); Call; Return ....; Return; When;
- As verbs, use the words Generate, Compute, Process, etc. Words such as set, reset, increment, compute, calculate, add, sum, multiply, ... print, display, input, output, edit, test , etc. with careful indentation tend to foster desirable pseudocode.
- Do not include data declarations in pseudocode.
  Eg: x=5;
- Use comments wherever necessary.
- Use proper indentation to increase readability.
- Always end multiline structures properly.
- Write only one statement per line.
- Capitalize initial keyword.
- Keep statements language independent.
- Output Statements can be represented by Write / display / print
- Input Statements can be represented by Read / get / input

Benefits of Pseudocode:

- Easy to understand as it uses simple English.
- Developing source code from pseudocode is easy.
- Can be done easily on a word processor.
- Implements structured concepts very well.

Disadvantages of Pseudocode:

- It's not visual.
- There is no accepted standard, so it varies widely from user to user.

Eg-Computing average marks for a class:

SET total to zero
SET grade counter to one
WHILE grade counter is less than or equal to ten
  Input the next grade
  Add the grade into the total
EndWhile
SET the class average as total divided by ten
PRINT the class average.

CODING, DOCUMENTING AND TESTING A PROGRAM

Once a suitable program has been created to solve a given problem, it is time to code the program (ie) translation of pseudocode of the design into corresponding statements in a particular programming language. The result of the program can be seen in the computer. Here the developers should be familiar with the programming language like C,C++ or Java to write correct program.

The notion of documenting a program means providing explanatory material about the program for other programmers and users. Annotating each part of program is necessary, (ie) documentation is a program with comments. A comment is text inserted into the program for explanatory purpose, but ignored by the computer while executing. How they are ignored means a certain character or a combination of characters indicates a particular statement in a program as comment line.

Eg:

BASIC	' (apostrophe) or REM
PASCAL	Text enclosed between * and *
JAVA/C/C++	/*.....*/ or //
VB	' (apostrophe)

Comments are of two types as:

1. **Header comments**: Appear at the beginning of a program that provides general information about the program.
2. **Step comments**: Appear throughout the program to explain the purpose of specific portions of code.

Certain commercial program includes another form of documentation to help customers learn to use the software available as user's guide or on-screen help called as external documentation. It is necessary to test a program at every phase of the program development cycle to ensure that it is error-free. The final and most

important, testing takes places when the code has been completed-we run the program with various sets of input data (test data) until we are convinced that our program is working properly.

Types of Errors: If a test run turns up problems with the program, we must debug it-eliminate the errors. The elimination of error may be easy or difficult depending upon the type of the error. There are 2 types of errors.

1. **Syntax Error**: Violation of the programming language's rules for creating valid statements is called as syntax error. **Eg**: Misspelling of keyword, omitting a punctuation mark. Syntax errors are usually listed after compilation. Usually the compiler will highlight the error in simple English along with line number-so syntax errors are easy to find and correct.
2. **Logic Error**: Failing to use the proper combination of statements to accomplish a certain task is called as logic error. It may occur due to poor analysis, faulty design, wrong formula, incorrect sequence of statements, and division by zero etc.

Unlike syntax errors, logic errors are not found by the translator. They can be found only by a careful reading of program specifications or by a careful reading of program specifications or by running the program with test data. Certain logic error may lead a program to crash or hang. Hence extensive testing is the best way to ensure that a program's logic is sound.

PROGRAM DESIGN

A computer program is basically a set of logical instructions. A program cannot get the solution of a problem by simply providing input to the computer without preparing the base for solving the problem. All the activities which have to be performed by a user in order to solve a problem using computer are grouped into three phases:

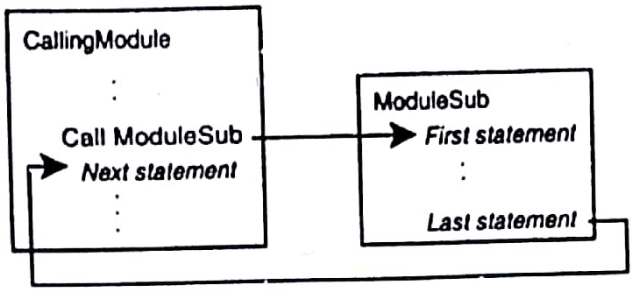
1. **Identifying the purpose:** It is the first stage of problem solving using a computer which will' identify parameters and constraints and collect information relevant to that.
2. **Developing a program:** For developing a program a user has to perform the following activities:
   - Identifying the logical structure- A user writes algorithm, draw flowchart to represent the flow of information and writing pseudocode to specify the programming specifications.
   - Writing the computer program
   - Debugging the program
3. **Executing the program**

MODULAR PROGRAMMING: A good way to begin the job of designing is to identify major tasks which can be divided into subtasks. The individual fundamental sub

tasks are called modules. The process of breaking down a problem into simpler sub problems is called as **Top-Down Design;** splitting a program into module is called Modular Programming.

Features of a module are:

- A module is relatively short.
- A module is assigned with a task.
- A module is self-contained and independent of other modules.



There will a number of modules in a complex program. To execute every module, there will be one parent module/calling module. Once a module is called, the control goes to that called module and returns to the calling module. Usually every program has a special module called as main module (ie) where every program execution begins. It is not a sub module of anyone.

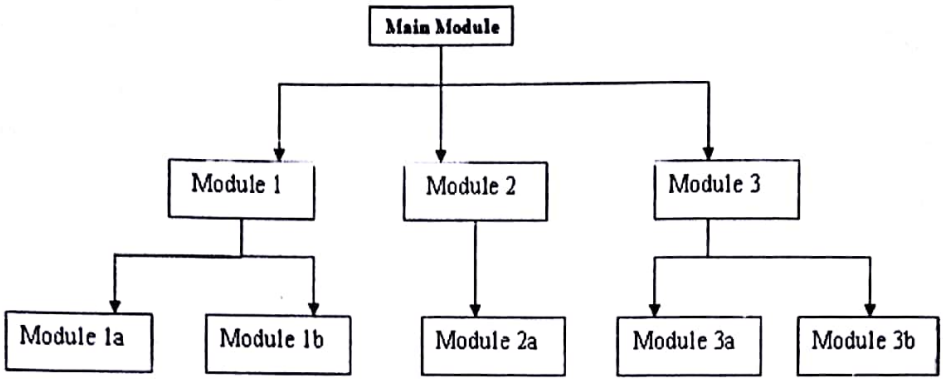
Benefits of modular programming are:

1. Program readability increases and reduces the time to locate the errors.
2. Programmer productivity is increased and hence easier to design code and test the program.
3. A singe module can be used in more than one place and reduces the amount of code.
4. Different modules can be programmed by different set of programmers at the same time.

Hierarchy Chart is a solution approach that suggests a top-down solution of a problem. Since it is difficult to comprehend the solution for a large problem, we can decompose into smaller tasks-find the solution for the smaller tasks and combine all the solution to form the large problem. It is also called **Structure Chart** and it does not provide any details about program logic.

Summary:

1. Write modular programs.
2. Use descriptive variable names.
3. Provide a welcome message for the users.
4. Use prompt before input and output.
5. Document programs.



UNIT III

Selection Structures: Relational and Logical Operators -Selecting from Several Alternatives – Applications of Selection Structures. **Repetition Structures:** Counter Controlled Loops –Nested Loops– Applications of Repetition Structures.

RELATIONAL OPERATORS: A **relational operator** is a programming language construct or **operator** that tests or defines some kind of relation between two entities. There are six basic relational operators which will produce boolean value as answer (true/false). All the operators can be applied to both numeric and character-based data.

Equal to	=
Not equal to	≠
Greater than	>
Greater than or equal to	≥
Lesser than	<
Lesser than or equal to	≤

Note: ≠, ≥ and ≤ are not available in keyboard. So most programming languages use a special pair of characters to represent these relational operators.

Not equal to	!= (or) <>
Greater than or equal to	>=
Lesser than or equal to	<=

Eg:
3 < 4 true
5<= 9 true
4 == 7 false
8.3 != 2.1 true
"h"!="B" true

LOGICAL OPERATORS: The concept of logical operators is simple. They allow a program to make a decision based on multiple conditions. Each operand is considered a condition that can be evaluated to a true or false value. Then the value of the conditions is used to determine the overall value of the op1 operator op2 or !op1 grouping. They also produce boolean value (true/false) as answer.

AND	Produce true if all conditions are true else false
OR	Produce true if any one of the conditions is true else false
NOT or !	Negate the condition to opposite value

Eg:
(5>=9) AND (3=3) False

(5>=9) AND (3=3) True
!(3=3)False

X	Y	X AND Y	X OR Y	NOT X	NOT Y
true	true	true	true	false	false
true	false	false	true	false	true
false	true	false	true	True	false
false	false	false	false	true	true

Depending upon the sequence of statements, the control structures can be classified as,

- Selection or Decision structures
- Sequence structures
- Repetition or Loop or Iterative structures

SELECTION STRUCTURES

A selection structure consists of a test condition together with one or more groups or blocks of statements. The result of the "test" determines which of these blocks is executed. A selection structure is called

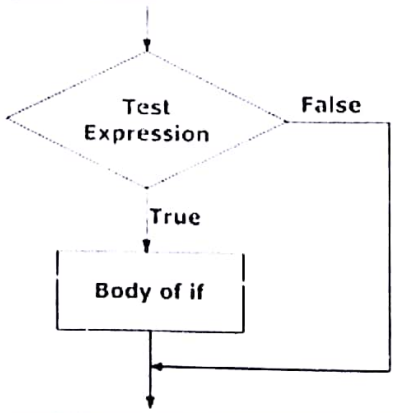
1. **A single-alternative or if-then structure** if it contains single block of statements to be either executed or skipped.
2. **A dual-alternative or if-then-else structure** if it contains two blocks of statements, one of which is to be executed while the other is to be skipped.
3. **A multiple-alternative structure** if it contains more than two blocks of statements, one of which is to be executed while the others are to be skipped.

1. **IF THEN:** The simplest type of selection structure is the If..then or single alternative structure.

Syntax:

```
If (test expression) then
    Body of if
End if
```

Flowchart:



The test condition or expression will be evaluated first. If its result is true, the statements enclosed between then and end if will be evaluated. If the condition is false, the block of statements will be skipped. In both cases of true or false, execution then proceeds to the program statement following end if.

Eg:

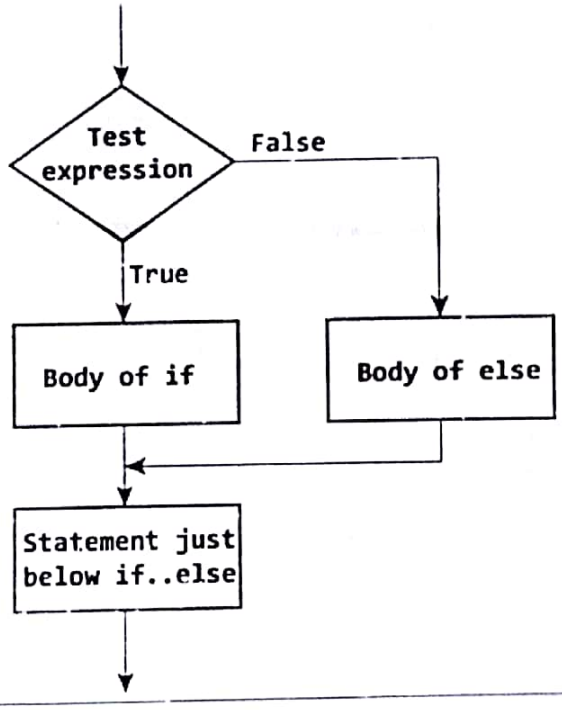
```
Write "Enter a number:";
Input n
If (n>0) then
        Write "Number is positive"
If (n<0) then
        Write "Number is negative"
If (n=0) then
        Write "Number is zero"
```

2. **IF..THEN..ELSE**: If-then-else or dual alternative structure has the following general form:

Syntax:

```
If (test expression) then
     Body of if
Else
     Body of else
End if
```

Flowchart:



If test condition is true, then the block of statements between then and else will be executed. If the test condition is false, then the block of statements between else and end if will be executed. In either case, execution then proceeds to the program statements following end if.

Eg:

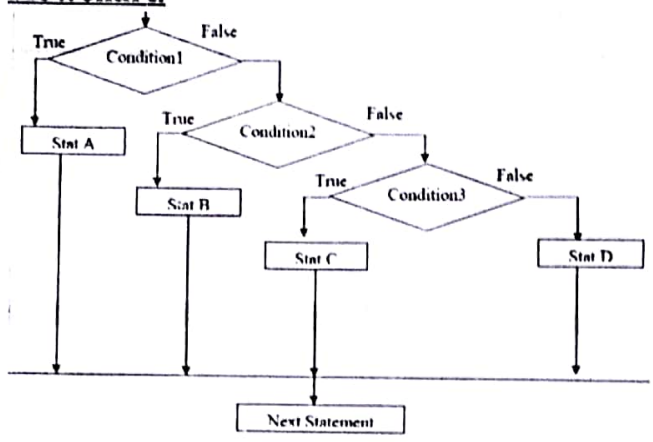
```
Write "Enter a number:";
Input n
If (n>0) then
        Write "Number is positive"
else
If (n<0) then
        Write "Number is negative"
else
        Write "Number is zero"
```

3. **ELSE IF LADDER (several alternative using if)**: We can construct a multiple alternative structure using several applications of If-then or If-then-else structure. Here more conditions upon certain parameters will be checked and all those conditions are grouped together.

Syntax:

```
If (condition1) then
      Stat A
Else If (condition2) then
      Stat B
Else If (condition3) then
      Stat C
.....
Else
      Body of else
End if
```

Flowchart:



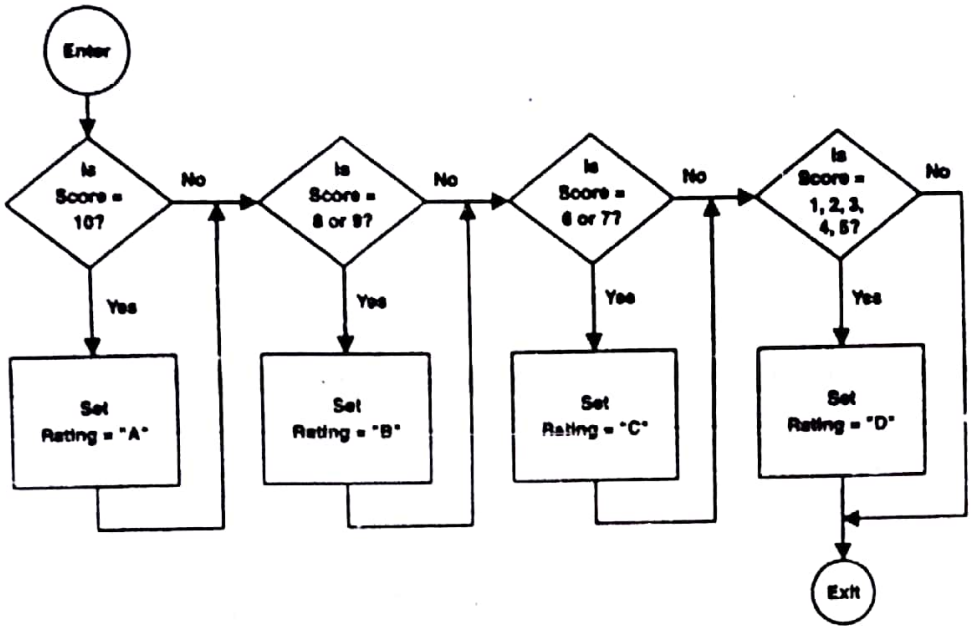
Eg: A program segment that translates a numerical score (represented by an integer from 1 to 10) into a character grade using following rules.

- If the score is 10, rating is A
- If the score is 8 or 9, rating is B
- If the score is 6 or 7, rating is C
- If the score is below 7, rating is D

Solution-1: Sequence of if statements

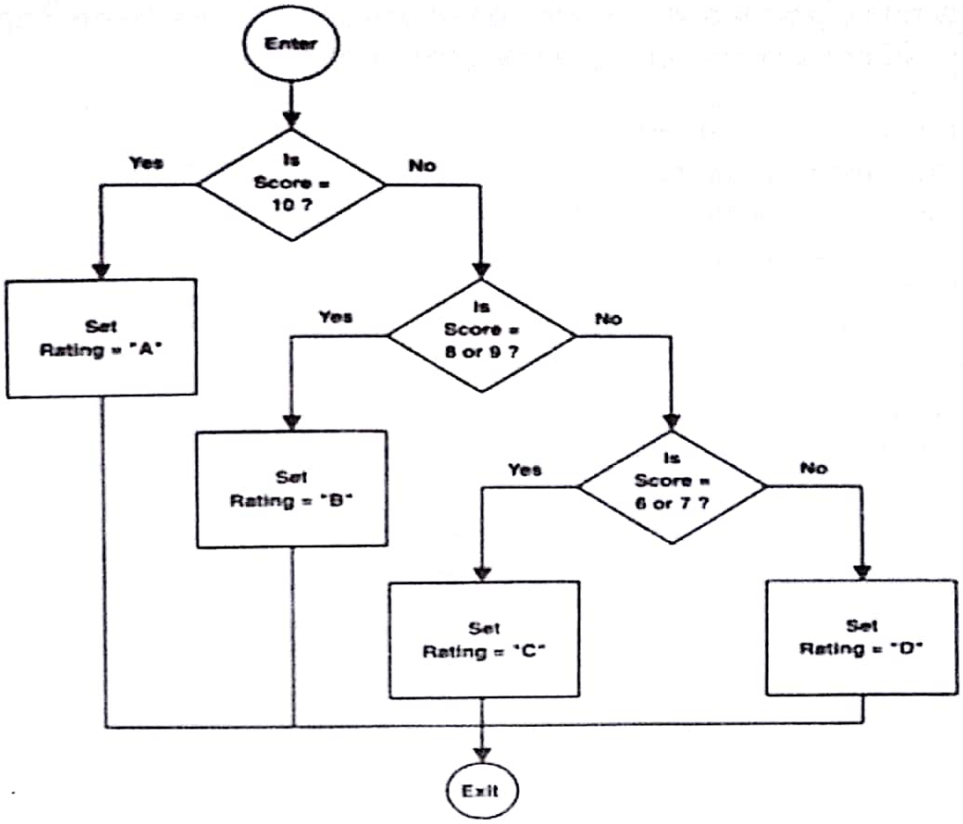
```
If score=10 then
      Set rating="A"
End if
If (score=8 or score=9) then
      Set rating="B"
End if
```

If (score=6 or score=7) then
    Set rating="C"
End if
If (score>=1 or score<=5) then
    Set rating="D"
End if



Solution-2: Sequence of if-then-else statements

If (score=10) then
    Set rating="A"
else If (score=8) or (score=9) then
    Set rating="B"
else If (score=6) or (score=7) then
    Set rating="C"
else
    Set rating="D"



4. **SEVERAL ALTERNATIVES USING CASE STRUCTURE**: To make the multiple alternatives more effective, many programming languages contain a statement called case or switch. This statement contains a single test expression, which should gives the result as integer or character determines the block of the code to be executed.

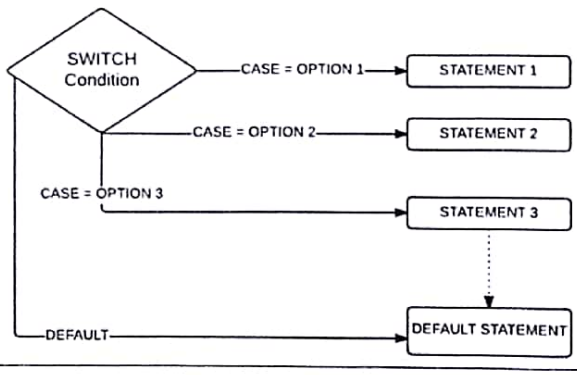
The test expression will be evaluated first. The result will be compared against the list of values. Which value is perfectly matched, the corresponding block of statements will be executed. If no match is found, the block of statements enclosed under keyword "default" will be executed.

Syntax:

```
switch (test expression)
  case option1:.....
  case option2:.....
  case option3:.....

  default:.....
end case
```

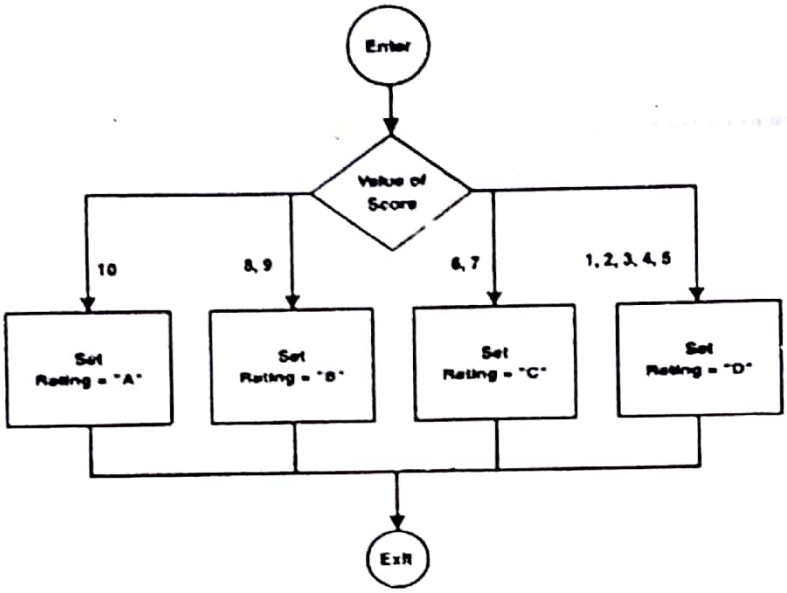
Flowchart:



Eg: A program segment that translates a numerical score (represented by an integer from 1 to 10) into a character grade using following rules.

- If the score is 10, rating is A
- If the score is 8 or 9, rating is B
- If the score is 6 or 7, rating is C
- If the score is below 7, rating is D

```
switch (score)
        case 10: set rating="a"
        case 8,9: set rating="b"
        case 6,7: set rating="c"
        default: set rating="d"
end case
```



Note: To make it easier to read the code for a selection structure, indent the statements that make up the true and false block.

APPLICATIONS OF SELECTION STRUCTURES: Two important applications of selection structures are:

1. ***Defensive Programming***: It involves the inclusion of statements within a program to check, during execution for improper data. The program segment that catches and reports an error of this sort is called as error trap.

   Eg-1: Avoiding a potential division by zero

```
Write "Enter a number to find its reciprocal"
Input num
If num<>0 then
        Set ans=1/num
        Write "Reciprocal=",ans
```

```
        Else
            Write "Don't enter zero please"
        End if
```

Eg-2: Dealing with square-root: Most programming languages nave built-in function that compute square root of a number as sqrt(number) where the argument many be a number, variable or expression whose value should be non-negative.

```
Write "Enter a number to find square root"
Input num
If num>=0 then
    Write "Square root=",sqrt(num)
Else
    Write "Enter a positive number"
End if
```

2. ***Menu-Driven Programs*:** To make the programs more user-friendly, we can go for menu programs. Menu screen will displays a list of choices (program's major functions) and the user has to select one choice at a time. The concept of menu enhances user-friendliness. Programs that present their options in this fashion, instead of by requiring their users to memorize commands, are said to be menu-driven. Selection of a choice will lead to another detailed submenu or directly into the specified task.

Eg-1:

```
If (opt=1) then
    call function1
Else If (opt=2) then
    call function2
Else If (opt=3) then
    call function3
.....
```

Eg-2:

```
select (opt)
    case 1: call function1
    case 2: call function2
    case 3: call function3
    .....
End case
```

Examples:

1.
```
If amt>0 then                    [amt=5,-1]
    Write amt
End if
Write amt
```

2. If amt>0 then [amt=5,-1]
    Write amt
Else
    Set amt=-amt
    Write amt
End if

3. Write "Enter CP and SP"
Input CP,SP
Set amt=SP-CP
If amt>0 then
    Write "Profit=",amt
Else
    Set amt=-amt
    Write "Loss=",amt
End if
Write "Calculation completed"

4. Construct a multiple-alternative structure that displays "low" if x is equal to 0, "medium", if x is equal to 1 or 2, or "high", if x is greater than 2 but less than or equal to 10. Use a set of if-then, if-then-else and case statement.

REPETITION STRUCTURES

All programming languages provide statements to create a loop or repetition structure-a block of code that, under certain conditions, will be executed repeatedly. The condition that determines whether a loop is re-entered or exited is usually constructed with the help of relational and logical operators. All repetition structures can be divided into two fundamental types: Pre-test loops and Post-test loops.

Pre-test Loop /Entry-controlled Loop	Post-test Loop/Exit-controlled Loop
Exit condition occurs before the body of the loop.	Exit condition occurs after the body of the loop.
Condition will be evaluated first.	Body of the loop will be executed first.
If the condition is false at the first time itself, the body of the loop will not be executed at least once.	Even though the condition is false at the first time itself, the body of the loop will be executed at least once.
The variables appearing in the exit condition must be initialized.	The variables appearing in the exit condition may be initialized in the body of the loop.

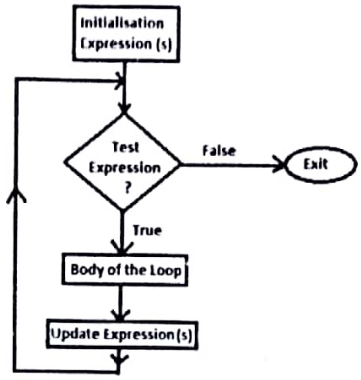
1. **While loop:** Here the condition is evaluated first. If the condition is true, the body of the loop will be executed. Again the condition is checked. The body of the

loop will be executed as long as the condition is true. If the condition is false, the body of the loop will be skipped. This comes under the category of pre-test loop, since the condition is checked before the body of the loop. So if the condition is false at the first time itself, the body of the loop will not be executed at least once.

Syntax:

```
while (condition)
    body of the while loop
end while
```

Flowchart:



Eg-1:

```
Input num
While num>0
        Write num^2
        Input num
End while
```

Eg-2:

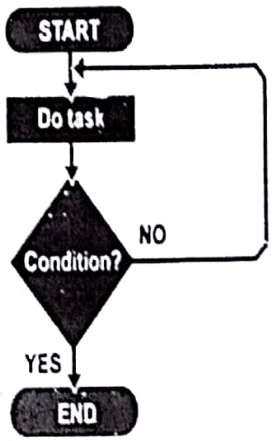
```
Set num=1
While num<3
        Write 2*num
        Set num=num+1
End while
```

2. **Repeat..until loop:** Here the body of the loop is executed first. Then the condition is checked. If the condition is false, the body of the loop will be executed. Again the condition is checked. The body of the loop will be executed as long as the condition is false. If the condition is true, the body of the loop will be skipped. This comes under the category of post-test loop, since the condition is checked after the body of the loop. So if the condition is false at the first time itself, the body of the loop will be executed at least once.

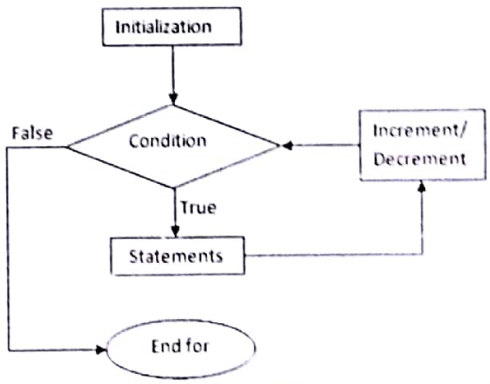
Eg-1:

```
Set num=1
repeat
```

write 2*num
set num=num+1
until num=3

Syntax:	Flowchart:
repeat body of the loop until condition	

3. Counter-Controlled Loops: A counter controlled loop is one that is executed a fixed number, N, where N is known prior to entering the loop for the first time. Counter (N) will keep the number of passes through the loop. When the counter reaches the preset number, the loop is exited. For loop comes under the category of built-in counter controlled loop. We can also construct while or repeat-until loop as counter controlled loop.

Syntax:	Flowchart:
for counter=initial value step increment to limit value body of the loop end for	

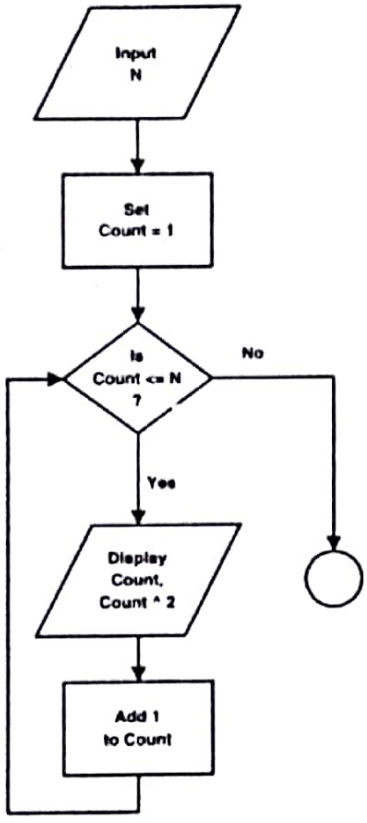
Here counter must be a variable, increment must be a constant, initial and final values may be constants, variables or expressions. On entering the loop, the counter will be initialized to initial values. If the initial value is not greater than limit value, then the body of the loop is executed. If counter is greater than limit value, the loop is skipped. On subsequent pass through the loop, counter is incremented

by the value of increment and if the new value of counter is not greater than the limit value, the body of the loop will be again executed else control goes to end for.

Note: We can also "step backwards" through a loop (ie) increment value can be negative and here if the initial value is greater than the limit value, the body of the loop will be executed else not.

Eg-1: Print square of numbers from 1 to n

Prompt for and input the value for n
Initialize the variable count to 1
While count<=n
    Write count,count^2
    Add 1 to count
End while



Eg-2:

for k=10 step -1 to 1
    write k
end for

Eg-3:

write "Before loop"
for k=5 step 1 to 1

```
        write "Inside loop"
end for
write "After loop"
```

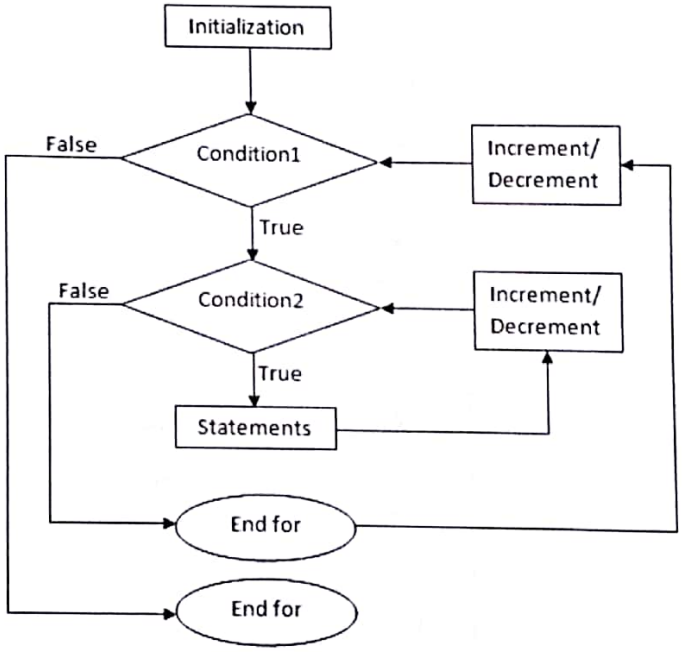
Eg-4:

```
set n=1
for k=n step 1 to n*3
        write n,k
end for
```

Eg-5: Print numbers 10,20,30...100

4. **Nested Loops**: Programs sometimes employ one loop which is contained entirely within another-in such a case, the loops are nested. The larger loop is called as the outer loop and the one lying within it is called as the inner loop.

Flowchart:



Eg-1: Nested for loop

```
for i=1 step 1 to 5                    //condition1
   write "Outer loop iteration", i
   for j=1 step 1 to 2                 //condition2
         write"i=",i;"j=",j
   end for
end for
```

Output:

Outer loop iteration 1
i = 1; j = 1
i = 1; j = 2
Outer loop iteration 2
i = 2; j = 1
i = 2; j = 2
Outer loop iteration 3
i = 3; j = 1
i = 3; j = 2
Outer loop iteration 4
i = 4; j = 1
i = 4; j = 2
Outer loop iteration 5
i = 5; j = 1
i = 5; j = 2

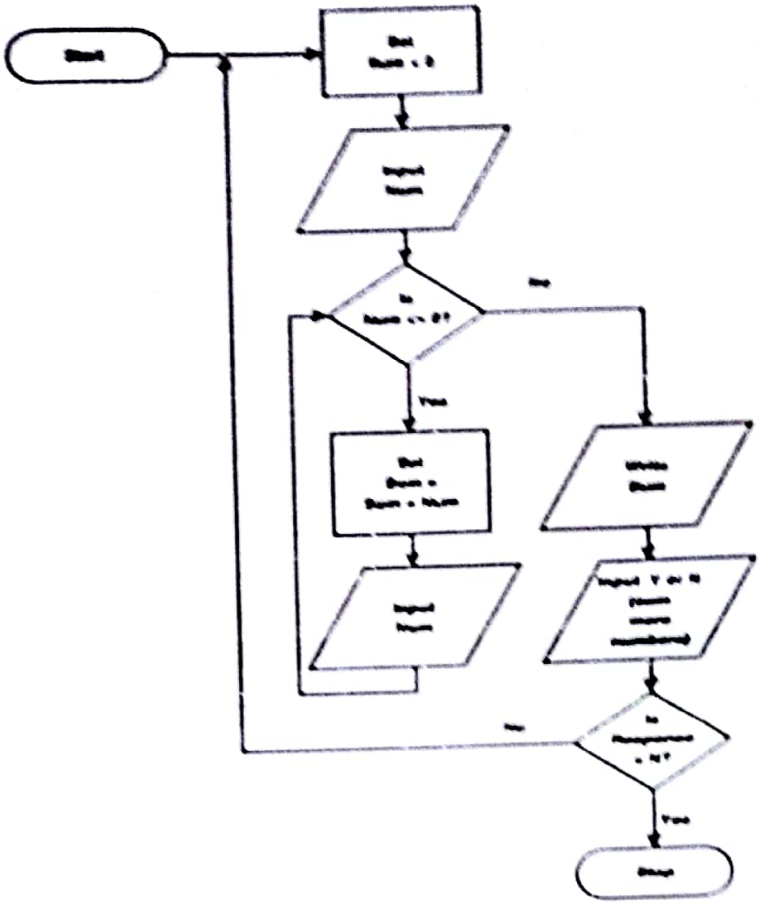
Other kinds of nesting of for loop are:

1. for i=..
   for j=....
   end for(j)
   for k=.....
   end for(k)
   end for(i)

for i=....
   for j=....
      for k=....
      end for(k)
   end for(j)
end for(i)

Eg-2: Nesting other kinds loop

Initialize sum to 0 (ie) set sum=0
Repeat
   Prompt for and Input a number as num
   While num<>0 (use 0 as a sentinel value)
      Set sum=sum+num
      Prompt for and input a number as num
   End while
   Write sum
   Prompt user for another set of numbers and input response (y/n)
Until response<>"n"



Note: If a loop's exit condition is never satisfied, then the loop will never be exited- it will become an infinite loop. It is therefore important to make sure, through a suitable prompt, how the user is to terminate the action of the loop.

APPLICATIONS OF REPETITION STRUCTURES:

1. ***Using sentinel-controlled loops to input data:*** Loops are often used to input large amounts of data: one data is entered at a time. The exit condition must cause the loop to be exited after all data have been input. Usually it is a best way to make the user to enter special item called sentinel value (signal to end the input). It acts as end-of-data marker else the program will be caught in infinite loop. For example,
   - For a list of positive numbers as input, -1 can act as sentinel.
   - For a list of non-zero numbers as input, 0 can act as sentinel.

   Eg:

```
Write "Enter the number of hours worked. Enter -1 to exit"
Input hours
While hours<>-1
        Write "Enter the rate of pay"
        Input rate
        Set salary=hours*rate
```

```
        Write hours,rate,salary
        Write "Enter the number of hours worked. Enter -1 to exit"
        Input hours
End while
```

2. ***Data validation***: This refers to the action of getting input continuously until user enters a valid input. To have the user enter a positive number at some point during program, we have the pseudocode like this:-

Eg:

```
Repeat
  Write "Input a positive number"
  Input num
Until num>0
```

Here the prompt "!nput a positive number" will be printed repeatedly till the user enters a positive number. If we want to improve the code, we can rewrite the above pseudocode as,

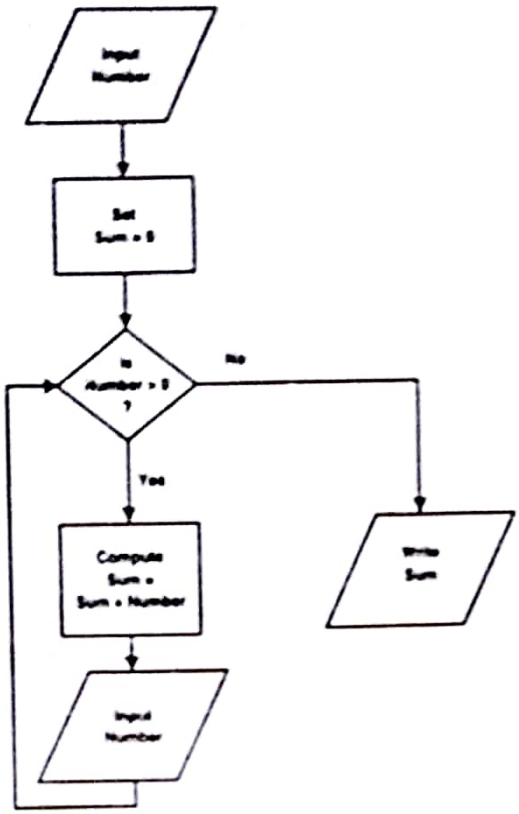
```
Write "Input a positive number"
Input num
While num<=0
  Write "Number should be positive. Try again"
  Input num
End while
```

So the data validation is an example of defensive programming, writing code that checks during execution of improper data.

3. ***Computing sum and averages:*** These structures are useful in manipulating a large set of data. We add each successive number to the running total, the sum obtained so far. We are looping repeatedly applying the addition operation until all the numbers have been added.

Eg: Finding sum

```
Initialize the sum to 0
Prompt for and input number as num
While num>0
  Add num to sum
  Prompt for and input next number as num
End while
Write sum
```



Eg: Finding average

Initialize count to 0
Initialize sum to 0
Prompt for and input number as num
While num>0
  Add count by 1
  Add num to sum
  Prompt for and input next number as num
End while
Prompt for and write sum
Set avg=sum/count
Prompt for and write avg

UNIT IV

Data: Numeric Data and Character Based Data. **Arrays**: One Dimensional Array - Two Dimensional Arrays – Strings as Arrays of Characters.

DATA

Programming languages allow us to input, manipulate and output various kinds of data or data types.

NUMERIC DATA

Numeric data is the concept of numbers. Most programming languages distinguish numeric data into two types as: Integers and Real or Floating point numbers.

1. ***The Integer Data type*:** An integer is a positive, negative or zero whole number. Since they are relatively simple numbers, integers take only less storage space in the memory.
   Eg: 323,-567,0

   A variable can be declared or defined to be of integer type by placing proper statement at the beginning of the program.
   Eg:
   Pascal=n:integer;
   Basic=Let n=50; (here n will be considered as integer since 50 does not include any decimal point
   C/C++/Java=int n;
   SQL=n number;

   All the five arithmetic operators +,-,*,/,^ can be applied to integers. Addition, subtraction and multiplication of two integers will produce one integer as a result. But dividing one integer by another integer will not guarantee to give another integer (it is programming language dependent) (ie) 5/2 will produce the result as 2.5 in Basic and Pascal, while in C++/Java, the result is 2 and the decimal place is truncated. So if both the operands are integer, division will be taken as integer division.

2. ***The Real Data type:*** In programming language, a real or floating point number is a number with decimal point (ie) not an integer. Programming language has declaration statement to specify the variables as float.
   Eg:
   Pascal=n:real;
   Basic=Let n=50.78;
   C/C++/Java=float n;
   SQL=n number(6,2);

   In the above declarations, the computer allocates the memory needed to store real number will be twice as much memory as needed for an integer variable. We will determine whether or not a variable or arithmetic expression has an integer value with the aid of a function Int(x) where x is a

number, variable or expression. Int(x) will yield only the integer value. If x is a real variable, it is turned to integer by just discarding the fractional or decimal part. Some other built-in functions like trunc(), round(),ceil() and floor() will also turn a real number to an integer.

Eg-1:

num1=15.25 num2=0 num3=4.5

Int(num1+num2+num3)=10 instead of 10.75

If x is an integer, Int(x) is still valid.

Eg-2: The following pseudocode is used to display a table of squares of all integers from 1 to n. To improve the readability, the program causes a line to be skipped after every 3 rows.

```
Repeat
   Write "Enter a positive integer"
   Input n
Until (n>0) and (Int(n)=n);
For count=1 step 1 to n
   Write count,count^2
   If (Int(count/3))=count/3 then
         Write
   End if
End for
```

Scientific and Exponential Notation: 0.000000789 is called as scientific notation while the computer equivalent of scientific notation is known as exponential notation.

Eg:

$0.000000789=7.89\times10^{-7}$

$1,502,000,000=1.502\times10^{9}$

If "10 to a power" is replaced by letter E/e, it gives exponential notation.

(ie) $1.502\times10^{9}=1.502E9$

$7.89\times10^{-7}=7.89E-7$

To convert the exponential to scientific notation:

- If the exponential integer is positive, move the decimal point to the right.
- If the exponential integer is negative, move the decimal point to the left.

Eg:

1.67E-4=0.000167

4.2E6=4200000

Random Numbers: Random numbers are those whose values form an unpredictable sequence. The application areas are simulation, computer games, engineering, and mathematics. Most programming languages include a mathematical function to generate random number. Random(N) where N=positive integer. Usually it produces an integer from 1 to N (seed).

Eg:

- Random(5)=1,2,3,4 or 5
- Random(3)+5=6,7 or 8
- Random(2)-1=0 or 1

Write a pseudocode to count the number of 5 and 8 when rolling a pair of dice.

```
Set count5=0
Set count8=0
For i=1 step 1 to 1000
        Set d1=Random(6)
        Set d2=Random(6)
        Set sum=d1+d2
        If sum=5 then
                Set count5=count5+1
        Else
        If sum=8 then
                Set count8=count8+1
        End if
End for
Write "Number of 5s=",count5
Write "Number of 8s=",count8
```

CHARACTER-BASED DATA

There are two types of non-numeric data: Character and Strings.

1. ***Characters:*** A character is any symbol that is recognized as valid by the programming language we are using. Character set may vary from one programming language to other. Most programming languages recognize a common core of about 100 basic characters. All data, including characters are stored in the computer's memory in binary form. So we have to standardize the scheme of associating a number with each character. Thus a standard correspondence for a basic set of 128 characters is given by **ASCII code**-American Standard Code for Information Interchange.
    - In ASCII, each character is associated with a number from 0 to 127.
    - Uppercase letters have ASCII code from 65(A) to 90(Z).
    - Lowercase letters have ASCII code from 97(a) to 122(z).
    - Numbers have ASCII code from 48(0) to 57(9).
    - Blank space has ASCII code of 32.

Ordering of characters:

- Letters are in alphabetical order and all uppercase letters precede all lowercase letters.
- Digits (viewed as characters) retain their natural order ("1"<"2").
- The blank space precedes all digits and letters.

The ASCII Codes from 32 to 127

Code	Character	Code	Character	Code	Character
32	[blank]	64	@	96	
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_	127	[delete]

2. ***Strings***: A character string or string is a sequence of characters. In most programming languages, strings are enclosed within quotation marks. Under certain cases, a single character is also considered to be a string. If a string doesn't contain any characters, it is called as **null string**. It is represented by two consecutive quotation marks as"" or ''. The number of characters in it is called as **length of string**. Null string's length is 0. Some programming language has special data type as string while others allow declaring as group of characters. We can also apply relational operators to character strings (ie) during searching certain names, sorting of strings etc. String1 and String2 are equal if and only if both of them are of equal length and contains same set of characters else both of them are unequal.

   Eg:
   "B$? 12" Length=6
   "n" Length=1

Ordering of Strings: Relational operators can be used on strings for sorting based upon its ASCII code. To determine which of two unequal strings comes first, use the following procedure:

- Scan the strings from left to right, stopping at the first position for which the corresponding characters differ or when one of the strings ends.
- If two corresponding characters differ before either string ends, the ordering of these two characters determines the ordering of given strings.
- If one string ends before any prior of corresponding characters differ, then the shorter string precedes the longer one.

Eg:

a. "word"<>"Word" True
b. "Ann"="Ann " False
c. "*?/!"<"*?,3" False (/ is 47 while , is 44)
d. "Ann"<="Anne" True

Input 2 names and display them in alphabetical order

```
Write "Enter 2 names"
Input n1,n2
Write "Two names in alphabetical order is"
If n1<n2 then
      Write n1,n2
Else
      Write n2,n1
End if
```

Concatenation of Strings: The addition operator "+" is used to join two given strings (ie) set s1+s2. If s1="good" and s2="morning, then s3="goodmorning".

Pseudocode:

```
Write "Give person's first name"
Input name1
Write "Give person's last name"
Input name2
Set name=name1+name2
Write name
```

Note: A character string with digits may look like a number, but it is only string constant and not treated as numeric constant. For example, constant 75 will be stored in memory in its binary equivalent. But "75" will be stored in ASCII codes of 7 and 5 successively. String can also be ordered by using < and > symbols where string length and ASCII codes are compared ("123"<"25" and "24.0">"24" are true).

ARRAYS

Array is a collection of variables of the same type, all of which are referenced by the same name.

- Since an array stores many data under a same name, there must be way to refer the individual elements. Programming languages follow syntax to have a number enclosed within parenthesis or brackets to indicate a particular element. That number is called as **subscript**. Eg: num[4] means 4th element of array num.
- Arrays are stored in memory in a sequence of consecutive storage locations- one location for each array element.
- The computer must know how many storage locations to set or allocate for that array.

Advantages:

1. Array can reduce the number of variable names needed in a program (ie) we can use a single array name instead of a collection of variables to store related data.
2. Arrays help to create more efficient programs with greater generality and flexibility.
3. Once data are entered, they can be processed many times without having to be inputted again.
4. Arrays are purely random access data structure.

There are two types of arrays as one-dimensional and two-dimensional arrays.

1. ***ONE-DIMENSIONAL ARRAYS (Lists):*** A one-dimensional array is a list of related data of the same type (numbers, characters) referred to by a single variable name.

   Eg:

   Pascal: a=array[1..10] of integer;

   C++:int a[10];

   Visual Basic: DIM a(1 to 10) as integer;

   Pseudocode:

   ```
   Declare rain[12]
   Set sum=0
   For k=1 step 1 to 12
       Write "Enter the rainfall for month", k
       Input rain[k]
       Set sum=sum+rain[k]
   End for
   Set average=sum/12
   ```

```
For k=1 step 1 to 12
    Write "The rainfall for month", k,"=",rain[k]
End for
Write "Average rainfall for the year =",average
```

2. ***TWO-DIMENSIONAL ARRAYS (Matrix):*** It is sometimes necessary to use arrays whose elements are determined by two factors-there comes the concept of two-dimensional arrays. A two-dimensional array is a collection of elements of the same type stored inconsecutive memory locations, all of which are referenced by the same variable name using two subscripts.
Eg: N[4,2] means the second value of 4th entity. A two dimensional array can be viewed as a matrix with rows and columns. Declaration of two dimensional array will have two subscripts as S[4,5] means an array S with 4 rows and 5 columns and thus a capacity of 20 elements. Higher dimensional arrays though not often used, are allowed in programming language while we want to store more data.
Eg: Write a pseudocode to read and display 5 marks for 4 students.

	m1	m2	m3	m4	m5
student1	92	94	87	83	90
student2	78	86	64	73	84
student3	72	68	77	91	79
student4	88	76	93	69	52

```
Declare S[4,5]
For i=1 step 1 to 4
        Write "Give 5 marks for student", i
        For j=1 step 1 to 5
                Input s[i,j]
        End for
End for
For i=1 step 1 to 4
        Write "5 marks for student", i, "are"
        For j=1 step 1 to 5
                Write s[i,j]
        End for
End for
```

Strings as array of characters: Some programming languages have string as a data type while others will strings as array of characters. Array size is the total size allocated for a particular string during declaration itself while the length of the string is the number of characters currently stored in the given arrays. To calculate the length, every language contains a built-in function as length(string).
Eg:

Declare s[25] Number of characters allocated for s=25
Set s="India" Length of the string=5
Write length(s1)

Regarding the storage of strings at memory, each character will take one byte. In order to denote the end of string, a special symbol will be inserted at the end of string. If the allotted locations are greater than length of the string, remaining locations will be assigned with the blank spaces.

I	N	D	I	A	.		

Here '.' is shown as the terminating characters and count the characters until we encounter '.' symbol.

UNIT V

Data Flow Diagrams: Definition, DFD symbols and types of DFDs. **Program Modules:** Subprograms-Value and Reference parameters- Scope of a variable - Functions - Recursion. **Files:** File Basics-Creating and reading a sequential file-Modifying Sequential Files.

DATA FLOW DIAGRAM (DFD)

In the 1970s, Larry Constantine, the original developer of structured design, proposed data flow diagrams as a practical technique and is an important tool used by system analysts. A **Data Flow Diagram (DFD)** is a graphical representation of the "flow" of data through an information system, modelling its *process* aspects. A DFD is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated. DFD models a system by using external entities from which data flows to a process, which transforms the data and create output data flows which goto other processes, external entities or data stores.

It can provide an overview of what data a system would process, what transformation of data are done, what data are stored and which stored data are used, and where the result flows. Hence DFD is used to keep track of the data transmitted to and from each subprogram.

Symbols in DFD:

1. **Circle/Rounded Rectangle**-Represents a process. They do computations on data. Both inputs and outputs to a process are dataflow. Processes are numbered and given a name.
2. **Arrow**-Data flows are represented by arrows. The name of the data appears next to the line.
3. **Rectangle**-External entities are represented by rectangles. They are the sources and destinations of information entering or leaving the system. They can be an outside organization, a person, a system etc.
4. **Two Parallel Lines**- Processes may store or retrieve data from a data store. It is depicted by two parailel lines. If an arrow points to store, it denotes writing else reading operation.

Good conventions in developing DFD: DFDs serve the dual purpose of specifying what data are needed for processing and as documentation of what procedures transform data.

A good DFD should have:

- Process names, data stores names and data flow names must be meaningful according to context.
- Each process should have at least one input and one output.
- Each data sore should have at least one data flow in and one data flow out.
- Data stored in a system must go through a process.

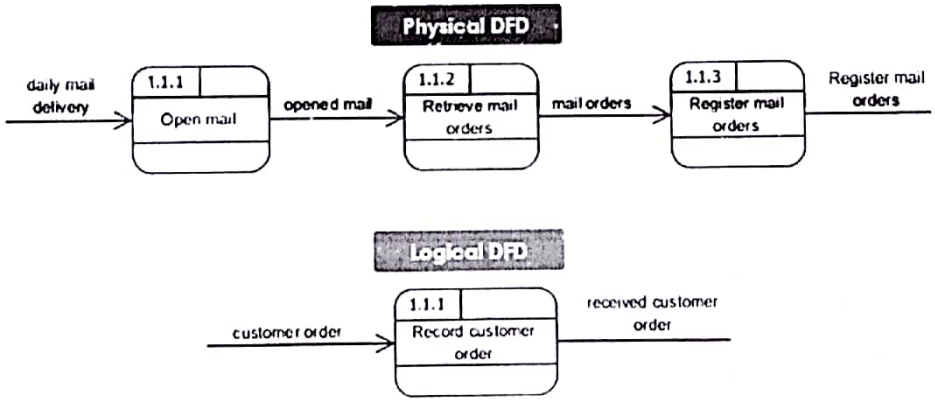
- DFDs must be developed top down with lower levels giving more details.
- Data should be conserved (ie) it can only transform input data to create output data.
- Dataflow should not act as signals to activate or initiate processes.
- DFD should not have more than 7-9 processes.

A good DFD should not have:

- Loops
- A process which is a pure decision.
- A data flow splits into flow with different names and meanings.
- Crossing lines.

Types of DFD:

1. **Logical DFD**: It visualizes the data flow that is essential for a business to operate. It focuses on the business and the information needed, not on how the system works or is proposed to work. It specifies who does the operations, whether it is done manually or with a computer and also where it is done. It is easily drawn during the fact gathering stage.
2. **Physical DFD**: It shows how the system is actually implemented. The processes represent the program, program modules and manual procedures. The data stores represent the physical files and databases. It shows controls for validating input data, for obtaining a record, for ensuring successful completion of a process and for system security. A logical DFD is the starting point in developing a physical DFD.



Features	Logical DFD	Physical DFD
Model	How the business operates	How the system will be implemented
Process	Essential sequence	Actual sequence
Data store	Collections of data	Physical files and databases, manual files
Type of data store	Permanent data collections	Master files, transaction files
System controls	Business controls	Controls for data validation, record status, system security

PROGRAM MODULES

Modular programming divides a large task into various modules-to pass data between a calling module and a submodule, programming language make use of **parameters**. The items listed in parenthesis in the call statement are known as **actual arguments/parameters**. Those items listed in the subprogram header are knows as **formal arguments/parameters**. Actual arguments can be constants, variables or expressions. Formal arguments must be variables.

Advantages:

- It enhances the usefulness of subprograms-they can be designed and coded independently of the main program and even used in several different programs.
- It makes it easier to test and debug a subprogram independently of the main program.
- It makes it easier for different programmers to design and code different subprograms.

Syntax:

1. call subprogram name(arg1,arg2,.....argn)
2. subprogram name (par1,par2,.....parn)

Note:

1. Number of actual arguments=number of formal arguments.
2. Data type of corresponding actual arguments=Data type of the corresponding formal arguments.

Main Program

```
Declare message as String
Write "enter a message"
Input message
call print(message)
End program
```

Sub Program

```
subprogram print(text String)
Write "*****",text,"*****"
End subprogram
```

Value and Reference Parameters: There are two types of subprogram parameters as:

1. **Value Parameters**: They have the property that changes to their values in the subprograms don't affect the value of the corresponding variables in the calling modules-they are useful in importing data value.
2. **Reference/Variable Parameters**: They have the property that changes in their values do affect the corresponding arguments in the calling module-they are both used to import and export data value.

NOTE:

1. Value parameters enhance the independence of subprograms from main program and each other.
2. In pseudocode, placing "As Ref" after the parameter name will make it as a reference parameter; if these words are omitted, then the parameter is value parameter.

Eg-1 for value parameters:

```
Main program
    Set n=1
    Call change(n)
    Write n
End program

Subprogram change(num)
    Set num=2
End subprogram
```

n	num
1	2

Output: 1

Eg-2 for variable parameters:

```
Main program
    Set n1=1
    Set n2=2
    Call change(n1,n2)
    Write n1," ",n2
End program

Subprogram change(num1,num2 As Ref)
    Set num1=2
    Set num2=1
End subprogram
```

n1	n2
1	2

n1	num2/n2
1	1

Output: 1 1

Background Details:

If the parameter is of value type, then:

- A temporary location will be allocated when subprogram starts.
- Value will be copied to that location.
- Changes are reflected only in the temporary location and not in the temporary location and not in the corresponding argument.

If the parameter is of variable type, then:

- It is assigned to the same storage location of the corresponding argument.
- Changes made are done at the variable itself.

Scope of a variable: The part of the program in which a given variable can be referenced is called the scope of the variable. According to the scope, the variables are of two types:

1. **Global variables** are the variables declared in main program whose scope is entire program.
2. **Local variables** are the variables declared in subprograms.

When a variable x is declared as both global and local variable, the local variable takes precedence.

Eg:

```
Declare g as integer
G=5
Main program
        Declare x as integer
        Set x=1
        Call sub
        Write x
End program

Subprogram sub
        Declare x as integer
        x=2
        Write g
End subprogram
```

FUNCTIONS

A function is a special type of subprogram-one whose name can be assigned a value. There can be two types of functions as built-in functions and user-defined functions.

1. ***BUILT-IN FUNCTIONS***: They are often referred as library. The code for these functions is supplied in separate modules and user doesn't need to write the code. **Eg**: sqrt(x),int(x), random(x),length(x),round(x), eof(filename).

   Built-in functions can take more than one parameters and return at least one value. The arguments can be variables, constants or expressions. Built-in functions differ from subprograms as:

   - Header and definition of a built-in-function don't appear in the program that calls that function.
   - When a built-in-function is called, the function name is assigned a value.
   - A built-in function is called by using the function name anywhere in the program that a constant of its type is followed.

Eg:

```
Declare Inputstring as String
Declare n,number as String
Repeat
    Write "Enter an integer"
    Input Inputstring
    Set number=val(Inputstring,n)
Until (n<>0) and (number=int(number))
```

If the string entered by the user is an integer, both conditions become true and the loop will be exited.

2. ***USER-DEFINED FUNCTIONS***: Programming languages allow the programmers to create their own function called as user-defined functions. In languages like C,C++, subprograms and functions are same. In language like Pascal, subprograms (procedures) and functions are different. Procedures may or may not take arguments and may or may not return values, that too one or many. Functions may or may not take arguments and will return only one value. We will begin a function's header with the word Function.

   Syntax:

```
Function name as return type
.
.
.
    name=expression
End function
```

   Eg:

```
Main program
    Declare num as Real
    Set num=cube(10)
    Write num
End program

Function cube(x) as Real
    Set cube=x^3
End function
```

   Output: 1000

Recursion: When a subprogram calls itself, the process is called as **recursion**. The subprogram is said to be **recursive**. C,C++ etc allow recursion while Visual Basic does not. Recursive algorithms can sometimes provide quick and simple solutions to complex problems.

Eg-Factorial for a given number

```
Main program
        Declare n as integer
        Input n
        Write fact(n)
End program

Function fact(x) as integer
        If (x=0) or (x=1) then
                Set fact=1
        Else
                Set fact=x*fact(x-1)
        End if
End function
```

FILES

A file is a collection of information that has been assigned a name and stored on a disk, separately from the program that created it. Files may contain programs called as **program files**; some may contain data called as **data files**. The information in a data file is broken into group of related data called as **records**. Input provided by the user while the program is running is called **interactive input** while input to a program from a data file is called as **batch file**. The advantages of batch processing are:

- Data files are suitable for large volume of data.
- Data files can avoid the situation to re-enter data.
- Data files can be used by more than one program.
- Data files can store the output of the program for future review or input to other program.

CATEGORIES OF FILES: All files are divided into 2 general types. They are:

1. Text files that contain only ASCII characters. **Eg**: Simple word processing files, program files.
2. Binary files that contain symbols that are not ASCII. **Eg**: OS files.

Advantages of text files over binary files:

1. Text files are easier to create, whether from within the programming language software or directly type in a text editor.
2. Text files can be displayed on screen or printed in a printer without using special software (ie) text files are universal in nature and computers can correctly interpret the contents of a file. **Eg:** E-mail system can create, transmit, receive and display text-files; but cannot necessarily receive or display binary files.

Categories of Data Files: data files can be divided into 2 according to the arrangement of data. Either type of file can be used to solve specific problem. They are:

1. **Sequential file** contains records that must be processed in the order in which they are created. It is better choice if we have to frequently read/display/modify the entire file. **Eg**: To view the 10th record, we must read the first 9 records.
2. **Direct access files or Random files** have the property that each record can be accessed independently of the rest. It is more efficient if we have to store a lot of data but expect to change or display small amounts of data.

CREATING A SEQUENTIAL FILE

Developing a program segment that creates a sequential file involves 3 basic steps:

1. The file must be **opened**.
   - **An external name**-the name under which the file will be saved.
   - **An internal name**-the name by which the file will be known by the program.
   - **The file mode**-states the purpose for opening a file (ie) output mode for writing and input mode for reading.
2. Data must be **written** to the file by writing its contents.
3. The file must be **closed**, terminating the process, saving the file's contents to disk and break the connections between internal and external file name.

To denote the end of file, a special symbol is placed at the end called as **End-Of-File (EOF) marker**. EOF may appear in the test condition of any loop or selection structure returns Boolean value-true or false. If file pointer has reached end-of-file, it will return the value as true else false.

Pseudocode for file operations:

1. Open "external name" For mode As internal name
2. Write internal name, data
3. Read internal name, variable1, variable2
4. EOF(internal name)
5. Close internal name

Eg:

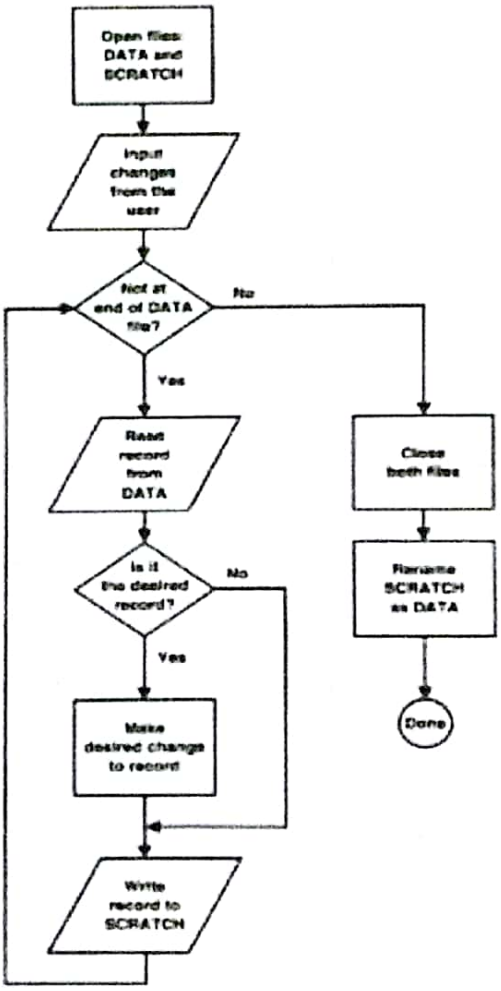
```
Open "GRADES" For Output As Newfile
Write "Enter the student's name and test score"
Write "Enter 0 for both to end"
Input student,score
While student <>"0"
        Write Newfile,Student.Score
        Write "Enter the stdent's name and test score"
        Write "Enter 0 for both when done"
```

```
        Input student,score
End while
Close Newfile
Open "Grades" For Input As Readfile
While not EOF(Readfile)
        Read Newfile,Student,Score
        Write Student,"  ", score
End while
Close Readfile
```

MODIFYING A SEQUENTIAL FILE

Three basic file operations are deleting, changing and inserting records within an existing sequential file. TO carry out any of these operations, the entire file has to be rewritten-every record must be read, temporarily stored in a second file called scratch file and process the desired records. After the completion of modification, written the records back to the given file.

Flowchart:



The general file modification process:

1. Open the given file as Input and scratch file as Output.
2. Input data concerning the change from user.
3. Read records from the given file and write them to the scratch file until we reach the desired record.
4. Make the change: Write the new or modified record to the scratch file; in the case of deletion, don't write the record into the scratch file.
5. Read rest of the records from the given file and write them to the scratch file.
6. Close both the files.
7. Replace the contents of scratch file on the given file.

1. ***Deleting records:***

- Successively read records from GRADES file.
- If the current record is not the one to be deleted, write it to SCRATCH; if it is one to the deleted, don't write it to SCRATCH.

Pseudocode:

```
Open "GRADES" For Input As Givenfile
Open "SCRATCH" For Output As Tempfile
Write "Enter the name of the student to be deleted"
Input Deletename
While not EOF(Givenfile)
        Read Givenfile, Student, Score
        If Student<>Deletename then
                Write Tempfile, Student, Score
        End if
End while
Close Givenfile,Tempfile
```

Eg:

GRADES	
Jones	86
Smith	94
Martin	73

Let "Martin" has to be deleted. Then after executing above pseudocode, the "SCRATCH" file will be:

SCRATCH	
Jones	86
Smith	94

Copy all the records of SCRATCH file back to GRADES file using the following pseudocode.

Pseudocode:

```
Open "GRADES" For Output As Targetfile
Open "SCRATCH" For Input As Sourcefile
While not EOF(Sourcefile)
        Read Sourcefile, Student, Score
        Write Targetfile, Student, Score
End while
Close Sourcefile,Targetfile
```

2. ***Changing records:***
   - Successively read records from GRADES file.
   - If the current record is the desired on, write the new record to a SCRATCH file. Otherwise, write the current record to the SCRATCH file.
   - Copy the SCRATCH file to the GRADES file.

Pseudocode:

```
Open "GRADES" For Input As Givenfile
Open "SCRATCH" For Output As Tempfile
Write "Enter the name of the student to be modified"
Input name
Write "Enter the new test score"
Input Newscore
While not EOF(Givenfile)
        Read Givenfile, Student, Score
        If Student=name then
                Write Tempfile, Student, NewScore
        Else
                Write Tempfile, Student, Score
        End if
End while
Close Givenfile,Tempfile
```

Eg:

GRADES	
Jones	86
Smith	94
Martin	73

Let Smith's has to be modified from 94 to 97. Then after executing above pseudocode, the "SCRATCH" file will be:

SCRATCH	
Jones	86
Smith	97
Martin	73

Copy all the records of SCRATCH file back to GRADES file using the following pseudocode.

Pseudocode:

```
Open "GRADES" For Output As Targetfile
Open "SCRATCH" For Input As Sourcefile
While not EOF(Sourcefile)
        Read Sourcefile, Student, Score
        Write Targetfile, Student, Score
End while
Close Sourcefile,Targetfile
```

After executing above pseudocode, the GRADES file will be,

GRADES	
Jones	86
Smith	97
Martin	73

3. ***Inserting records:***
   - Inserting records into a sequential file is the most complex of the three file modification operations.
   - Let the GRADES file be arranged with the records like name, score in alphabetical order. So if we want to insert a new record, it should be placed at the correct position.
     - Open GRADES file and a SCRATCH file.
     - Input the Newname and Newscore.
     - Read records from GRADES file and write them to the SCRATCH file until the desired location is reached.
     - Write new record into the SCRATCH file.
     - Read the rest of the records in GRADES file and write them to the SCRATCH file.

Two issues:

1. **How do we know when we have reached the proper location in GRADES file?** Since the names are in alphabetical order, the value of the string variable Student is increasing. So rewrite step-3 as, **Read records from GRADES and write them to SCRATCH file until Newname<Student** (new record is inserted just before the current one).
2. **What is the condition Newname<Student never occurs?** Here the new record will be inserted at the end of the file. There also Newname<Student is true for all names present in the file. Here the new record will be inserted at the beginning of the file.

Pseudocode:

```
Open "GRADES" For Input As Givenfile
Open "SCRATCH" For Output As Tempfile
Write "Enter the name of the student to be added"
Input Newname
Write "Enter the new test score"
Input Newscore
Repeat
    Read Givenfile, Student, Score
    If Newname<Student then
        Write Tempfile, Newname, NewScore
    Else
```

```
            Write Tempfile, Student, Score
        End if
Until (Newname<Student) or EOF(Givenfile);
If Student<Newname then
            Write Tempfile, Newname, NewScore
End if
Close Givenfile,Tempfile
```

Eg:

GRADES	
Jones	86
Smith	94

Let Martin has to be added. Then after executing above pseudocode, the "SCRATCH" file will be:

SCRATCH	
Jones	86
Martin	73
Smith	97

Pseudocode:

```
Open "GRADES" For Output As Targetfile
Open "SCRATCH" For Input As Sourcefile
While not EOF(Sourcefile)
        Read Sourcefile, Student, Score
        Write Targetfile, Student, Score
End while
Close Sourcefile,Targetfile
```

After executing above pseudocode, the GRADES file will be,

GRADES	
Jones	86
Martin	73
Smith	97