15BECS205 COMPUTER PROGRAMMING 3 2 0 4

OBJECTIVE:

- Understand, analyze and implement software development tools like algorithm, pseudo codes and programming structure
- Study, analyze and understand logical structure of a computer program, and different construct to develop a program in 'C' language

INTENDED OUTCOME:

- Understand the programming elements for solving computing related problems;
- Possess the ability to design and develop efficient computer programs for solving problems;
- Possess the ability to learn advanced programming techniques independently;
- Possess the ability to learn other high level programming languages independently;

UNIT-I POINTERS AND BUILT-IN-FUNCTIONS (10)

Introduction and features of pointers- Declaration of pointers- Void pointer- Array of pointers- Pointers to Pointers- Built-in-functions- String functions- Math functions- Character functions- Memory Management Functions- static and dynamic memory

UNIT-II STRUCTURES AND UNIONS (8)

Introduction and features of Structures- Declaration and initialization- Array of Structures- Pointers to structures- Passing structures as arguments to functions- Enumerated data type- typedef- Union

UNIT-III FILES (7)

Introduction- File operations- Open, read and close- Text modes- Binary modes- File functions-

fprintf,fscanf,getc,putc,fgetc,fputc,fseek,feof- Command line arguments

UNIT-IV PREPROCESSOR DIRECTIVES **(10)**

The #define Statement- Program Extendability- Program Portability- The # Operator- The ## Operator-The #include Statement- System Include Files- conditional compilation- The #if, #endif, #else, #elif, #ifndef Statements- #error, #line and #undef Statement

UNIT-V GRAPHICS IN C **(10)**

Graphics and Text mode- Video Adapter- Initialize Graphics Mode and resolution, graphics.h header file-Functions for drawing a Point on Screen, drawing lines, rectangle, circles, arcs, polygon- Functions to fill colors- Display Text in Graphics mode, outtext(), outtextxy(), justifying text.

Total: 45+15=60

REFERENCES:

1. Yashavant Kanetkar, "Let us C", BPB Publications, 13th Edition, 2013
2. H.M. Deitel and D.J. Deitel, 'C:How to Program', Prentice Hall, 7th Edition, 2012
3. A.P. Godse and D.A.Godse, "Advanced C Programming", Technical Publications, 1st Edition, 2008
4. How to solve it by Computer by R.J. Dromey, Prentice-Hall India EEE Series, 2012

KARPAGAM ACADEMY OF HIGHER EDUCATION

Faculty of Engineering

Department of Computer Science and Engineering

Lecture Plan

Subject Name: COMPUTER PROGRAMMING **Subject Code: 13BECS602**

S.No	Topic Name	No.of Periods	Supporting Materials	Teaching Aids
	UNIT- I POINTERS AND BUILT-IN-FUNCTIONS			
1	Introduction and features of pointers	1	T[2]-1	BB
2	Declaration of pointers	1	T[2]-1	BB
3	Void pointer	1	T[2]-5	PPT
4	Array of pointer	1	T[2]-6	PPT
5	Pointers to Pointers	1	T[2]-6	PPT
6	Built-in-functions	1	T[1]-95	PPT
7	String functions	1	T[1]-95	PPT
8	Math functions	1	T[1]-68	BB
9	Character functions	1	T[2]	PPT
10	Memory Management Functions	1	T[1]-12	BB
11	Static and dynamic memory	1	T[2]	BB
Total		**11**		
	UNIT- II STRUCTURES AND UNIONS			
12	Introduction and features of Structures	1	T[1]-200	PPT
13	Declaration	1	T[2]	PPT
14	Initialization	1	T[1] 201	BB
15	Array of Structures	1	T[1]214	PPT
16	Pointers to structures	1	T[1]214	PPT
17	Passing structures as arguments to functions	1	T[1]218	PPT
18	Passing structures as arguments to functions	1	T[2]218	PPT
19	Enumerated data type	1	T[2]218	PPT
20	typedef- Union	1	T[2]221	BB
21	Tutorial-Example Structure Program	1	T[2]	-
Total		**10**		
	UNIT- III FILES			
22	Introduction to Files	1	T[2]	PPT
23	File operations	1	T[2]	PPT

24	Open, read and close	1	T[2]	PPT
25	Text modes	1	T[1]-488	BB
26	Binary modes	1	T[1]-193	PPT
27	File functions	1	T[1]-266	BB
28	fprintf,fscanf,getc	1	T[1]-305	PPT
29	putc,fgetc,fputc,fseek,feof	1	T[1]-343	BB
30	Command line arguments	1	T[2]	PPT
31	Tutorial-Example File program	1	T[2]	PPT
Total		**10**		
	UNIT- IV PREPROCESSOR DIRECTIVES			
32	The #define Statement	1	T[2]-139	PPT
33	Program Extendability	1	T[2]-139	PPT
34	Program Portability	1	T[1]-140	PPT
35	The # Operator	1	T[2]-152	BB
36	The ## Operator	1	T[2]-159	PPT
37	The #include Statement	1	T[2]-162	BB
38	System Include Files	1	T[2]-163	PPT
39	conditional compilation- The #if, #endif, #else, #elif, #ifndef Statements	1	T[2]-133	PPT
40	#error, #line and #undef Statement	1	T[2]	PPT
41	Tutorial	1	T[2]-133	BB
Total		**10**		
	UNIT- V GRAPHICS IN C			
42	Graphics and Text mode	1	T[2]-248	PPT
43	Video Adapter	1	T[2]-465	BB
44	Initialize Graphics Mode and resolution	1	T[2]-465	BB
45	graphics.h header file	1	T[2]-255	PPT
46	Functions for drawing a Point on Screen	1	T[2]-248	PPT
47	drawing lines, rectangle, circles, arcs, polygon	1	T[1]-1087	PPT
48	Functions to fill colors	1	T[1]-1087	PPT
49	Display Text in Graphics mode	1	T[1]-690	BB
50	outtext(), outtextxy(), justifying text	1	T[1]-690	PPT
51	Revision	1	T[1]-752	BB
52	**Discussion on Previous University Question Papers**			
Total		**10**		
	Total Hours	**52**		

TEXT BOOKS

S.NO	Title of the book			Year of publication
1	Yashavant Kanetkar, "Let us C", BPB Publications, 13th Edition.			2013
2	A.P. Godse and D.A.Godse, "Advanced C Programming", Technical Publications, 1st Edition.			2008

GE2112 - FUNDAMENTALS OF COMPUTING AND PROGRAMMING

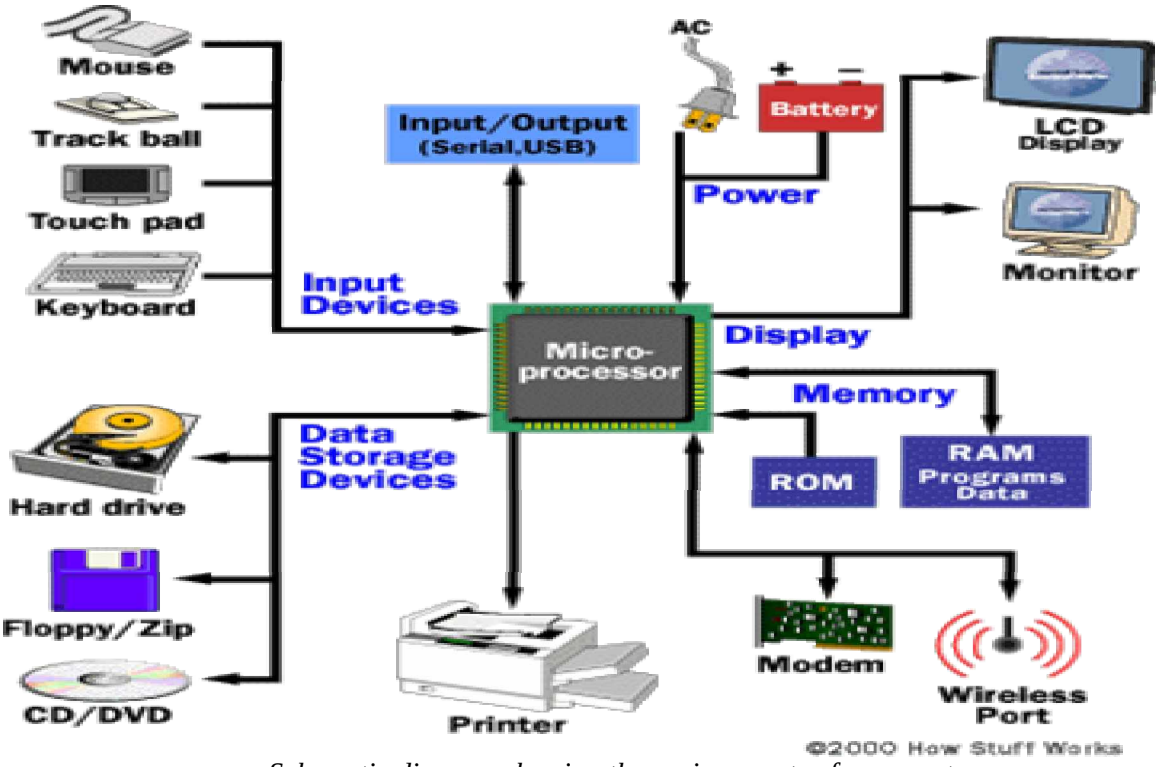
UNIT - I

INTRODUCTION TO COMPUTERS

Introduction – Characteristics of Computers – Evolution of Computers - Computer Generations – Classification of Computers – Basic Computer organization – Number Systems

INTRODUCTION

PARTS OF A COMPUTER



Schematic diagram showing the various parts of a computer.

Performance

Features that affect the performance of the computer include:

- **microprocessor**
- **Operating System**
- **RAM**
- **disk drives**
- **display**
- **input/output ports**

The microprocessor:

- Has a set of internal instructions stored in memory, and can access memory for its own use while working.
- Can receive instructions or data from you through a keyboard in combination with another device (mouse, touchpad, trackball, and joystick).

- Can receive and store data through several data storage devices (hard drive, floppy drive, Zip drive, CD/DVD drive).
- Can display data to you on computer monitors (cathode ray monitors, LCD displays).
- Can send data to printers, modems, networks and wireless networks through various input/output ports. Is powered by AC power and/or batteries.

CHARACTERISTICS OF COMPUTERS

The various characteristics of computers are as follows

1. **Speed**: Speed is the most important characteristics of computer .Computer having more speed to perform jobs instantaneously.

2. **Accuracy:** The computers are perfect, accurate and precise. Accuracy signifies the reliability of the hardware components of computers.

3. **Automatic:** A computer works automatically, once programs are stored and data are given to it, constant supervision is not required.

4. **Endurance:** A computer works continuously and will not get tired and will not suffer from lack of concentration.

5. **Versatility**: A computer can be put to work in various fields.

6. **Reduction of cost**: Though initial investment may be high, computer substantially reduces the cost of transaction.

EVOLUTION OF COMPUTERS

The Early development:

ABACUS:

ABACUS uses movable beads stung on wires above and below a cross bar and its operation are based on the idea of the place value notation. The beads of the counter represent digits. The value of the digit in each position is determined by adding the values of the beads pressed against the cross piece.

Pascal calculating machine:

It was the first real desktop calculating device that could add and subtract. It consists of a set of toothed wheels or gears with each wheel or gear having digits 0 to 9 engraved on it. Arithmetic operation could be performed by tunings these wheels.

Punched card Machine:

The presence and absence of the holes in the card represent the digits.

Charles Babbage's Engine:

The machine took input from the punched card. The Analytical engine had a memory which will perform arithmetic operations.

GENERATIONS OF COMPUTERS

The Zeroth Generation

The term Zeroth generation is used to refer to the period of development of computing, which predated the commercial production and sale of computer equipment. The period might be dated as extending from the mid-1800s. In particular, this period witnessed the emergence of the first electronics digital computers on the ABC, since it was the first to fully implement the idea of the stored program and serial execution of instructions. The development of EDVAC set the stage for the evolution of commercial computing and operating system software. The hardware component technology of this period was electronic vacuum tubes. The actual operation of these early computers took place without be benefit of an operating system. Early programs were written in machine language and each contained code for initiating operation of the computer itself. This system was clearly inefficient and depended on the varying competencies of the individual programmer as operators.

The First Generation, 1951-1956

The first generation marked the beginning of commercial computing. The first generation was characterized by high-speed vacuum tube as the active component technology. Operation continued without the benefit of an operating system for a time. The mode was called "closed shop" and was characterized by the appearance of hired operators who would select the job to be run, initial program load the system, run the user's program, and then select another job, and so forth. Programs began to be written in higher level, procedure-oriented languages, and thus the operator's routine expanded. The operator now selected a job, ran the translation program to assemble or compile the source program, and combined the translated object program along with any existing library programs that the program might need for input to the linking program, loaded and ran the composite linked program, and then handled the next job in a similar fashion. Application programs were run one at a time, and were translated with absolute computer addresses. There was no provision for moving a program to different location in storage for any reason. Similarly, a program bound to specific devices could not be run at all if any of these devices were busy or broken.

At the same time, the development of programming languages was moving away from the basic machine languages; first to assembly language, and later to procedure oriented languages, the most significant being the development of FORTRAN

The Second Generation, 1956-1964

The second generation of computer hardware was most notably characterized by transistors replacing vacuum tubes as the hardware component technology. In addition, some very important changes in hardware and software architectures occurred during this period. For the most part, computer systems remained card and tape-oriented systems. Significant use of random access devices, that is, disks, did not appear until towards the end of the second generation. Program processing was, for the most part, provided by large centralized computers operated under mono-programmed batch processing operating systems.

The most significant innovations addressed the problem of excessive central processor delay due to waiting for input/output operations. Recall that programs were executed by processing the machine instructions in a strictly sequential order. As a result, the CPU, with its high speed electronic component, was often forced to wait for completion of I/O operations which involved mechanical devices (card readers and tape drives) that were order of magnitude slower.

These hardware developments led to enhancements of the operating system. I/O and data channel communication and control became functions of the operating system, both to relieve the application

programmer from the difficult details of I/O programming and to protect the integrity of the system to provide improved service to users by segmenting jobs and running shorter jobs first (during "prime time") and relegating longer jobs to lower priority or night time runs. System libraries became more widely available and more comprehensive as new utilities and application software components were available to programmers.

The second generation was a period of intense operating system development. Also it was the period for sequential batch processing. Researchers began to experiment with multiprogramming and multiprocessing.

The Third Generation, 1964-1979

The third generation officially began in April 1964 with IBM's announcement of its System/360 family of computers. Hardware technology began to use integrated circuits (ICs) which yielded significant advantages in both speed and economy. Operating System development continued with the introduction and widespread adoption of multiprogramming. This marked first by the appearance of more sophisticated I/O buffering in the form of spooling operating systems. These systems worked by introducing two new systems programs, a system reader to move input jobs from cards to disk, and a system writer to move job output from disk to printer, tape, or cards. The spooling operating system in fact had multiprogramming since more than one program was resident in main storage at the same time. Later this basic idea of multiprogramming was extended to include more than one active user program in memory at time. To accommodate this extension, both the scheduler and the dispatcher were enhanced. In addition, memory management became more sophisticated in order to assure that the program code for each job or at least that part of the code being executed was resident in main storage. Users shared not only the system' hardware but also its software resources and file system disk space.

The third generation was an exciting time, indeed, for the development of both computer hardware and the accompanying operating system. During this period, the topic of operating systems became, in reality, a major element of the discipline of computing.

The Fourth Generation, 1979 - Present

The fourth generation is characterized by the appearance of the personal computer and the workstation. Miniaturization of electronic circuits and components continued and Large Scale Integration (LSI), the component technology of the third generation, was replaced by Very Large Scale Integration (VLSI), which characterizes the fourth generation. However, improvements in hardware miniaturization and technology have evolved so fast that we now have inexpensive workstation-class computer capable of supporting multiprogramming and time-sharing. Hence the operating systems that supports today's personal computers and workstations look much like those which were available for the minicomputers of the third generation. Examples are Microsoft's DOS for IBM-compatible personal computers and UNIX for workstation. However, many of these desktop computers are now connected as networked or distributed systems. Computers in a networked system each have their operating system augmented with communication capabilities that enable users to remotely log into any system on the network and transfer information among machines that are connected to the network. The machines that make up distributed system operate as a virtual single processor system from the user's point of view; a central operating system controls and makes transparent the location in the system of the particular processor or processors and file systems that are handling any given program.

CLASSIFICATION OF COMPUTERS

There are four classifications of digital computer systems: super-computer, mainframe computer, minicomputer, and microcomputer.

Super-computers are very fast and powerful machines. Their internal architecture enables them to run at the speed of tens of **MIPS** (Million Instructions per Second). Super-computers are very expensive and for this reason are generally not used for CAD applications. Examples of super-computers are: Cray and CDC Cyber 205.

Mainframe computers are built for general computing, directly serving the needs of business and engineering. Although these computing systems are a step below super-computers, they are still very fast and will process information at about 10 MIPS. Mainframe computing systems are located in a centralized computing center with 20-100+ workstations. This type of computer is still very expensive and is not readily found in architectural/interior design offices.

Minicomputers were developed in the 1960's resulting from advances in microchip technology. Smaller and less expensive than mainframe computers, minicomputers run at several MIPS and can support 5-20 users. CAD usage throughout the 1960's used minicomputers due to their low cost and high performance. Examples of minicomputers are: DEC PDP, VAX 11.

Microcomputers were invented in the 1970's and were generally used for home computing and dedicated data processing workstations. Advances in technology have improved microcomputer capabilities, resulting in the explosive growth of personal computers in industry. In the 1980's many medium and small design firms were finally introduced to CAD as a direct result of the low cost and availability of microcomputers. Examples are: IBM, Compaq, Dell, Gateway, and Apple Macintosh.

The average computer user today uses a microcomputer. These types of computers include PC's, laptops, notebooks, and hand-held computers such as Palm Pilots.

Larger computers fall into a mini-or mainframe category. A mini-computer is 3-25 times faster than a micro. It is physically larger and has a greater storage capacity.

A mainframe is a larger type of computer and is typically 10-100 times faster than the micro. These computers require a controlled environment both for temperature and humidity. Both the mini and mainframe computers will support more workstations than will a micro. They also cost a great deal more than the micro running into several hundred thousand dollars for the mainframes.

Processors

The term processor is a sub-system of a data processing system which processes received information after it has been encoded into data by the input sub-system. These data are then processed by the processing sub-system before being sent to the output sub-system where they are decoded back into information. However, in common parlance processor is usually referred to the microprocessor, the brains of the modern day computers.

There are two main types of processors: CISC and RISC.

CISC: A **Complex Instruction Set Computer** (CISC) is a microprocessor Instruction Set Architecture (ISA) in which each instruction can indicate several low-level operations, such as a load from memory, an arithmetic operation, and a memory store, all in a single instruction. The term was coined in contrast to Reduced Instruction Set Computer (RISC).

Examples of CISC processors are the VAX, PDP-11, Motorola 68000 family and the Intel x86/Pentium CPUs.

RISC: Reduced Instruction Set Computing (RISC), is a microprocessor CPU design philosophy that favors a smaller and simpler set of instructions that all take about the same amount of time to execute. Most types of modern microprocessors are RISCs, for instance ARM, DEC Alpha, SPARC, MIPS, and PowerPC.

The microprocessor contains the CPU which is made up of three components--the control unit supervises all that is going on in the computer, the arithmetic/logic unit which performs the math and comparison operation, and temporary memory. Because of the progress in developing better microprocessors, computers are continually evolving into faster and better units.

Notebooks

A laptop computer (also known as notebook computer) is a small mobile personal computer, usually weighing around from 1 to 3 kilograms (2 to 7 pounds). Notebooks smaller than an A4 sheet of paper and weighing around 1 kg are sometimes called sub-notebooks and those weighing around 5 kg a desk note (desktop/notebook). Computers larger than PDAs but smaller than notebooks are also sometimes called "palmtops". Laptops usually run on batteries.

Notebook Processor:

A notebook processor is a CPU optimized for notebook computers. All computing devices require a CPU. One of the main characteristics differentiating notebook processors from other CPUs is low-power consumption.

The notebook processor is becoming an increasing important market segment in the semiconductor industry. Notebook computers are an increasingly popular format of the broader category of mobile computers. The objective of a notebook computer is to provide the performance and functionality of a desktop computer in a portable size and weight. Wireless networking and low power consumption are primary consideration in the choice of a notebook processor.

Integrated Components

Unlike a desktop computer, a notebook has most of the components built-in or integrated into the computer. For desktop systems, determining which computer to buy is generally not based on what type of keyboard or mouse that is available. If you don't like the keyboard or mouse, you can always purchase something else. However, in the case of a notebook computer, the size of the keyboard or type of pointing device may be something that you need to consider unless you intend to use a regular mouse or full-sized keyboard. There are some notebooks that have a keyboard that expands when the notebook is opened which is a nice feature if you find the normal keyboard to be too small. Pointing devices vary from a touch pad to a stick within the keyboard to a roller or track-ball. Most notebooks have the video, sound, and speakers integrated into the computer and some notebooks even have a digital camera built-in which is very handy for video conferencing.

BOOTING:

In computing, booting is a bootstrapping process that starts operating systems when the user turns on a computer system. A boot sequence is the set of operations the computer performs when it is switched on which load an operating system.

Everything that happens between the times the computer switched on and it is ready to accept commands/input from the user is known as *booting*.

The process of reading disk blocks from the starting of the system disk (which contains the Operating System) and executing the code within the bootstrap. This will read further information off the disk to bring the whole operating system online.

Device drivers are contained within the bootstrap code that support all the locally attached peripheral devices and if the computer is connected to a network, the operating system will transfer to the Network Operating system for the "client" to log onto a server

The Process of loading a computer memory with instructions needed for the computer to operate. The process and functions that a computer goes through when it first starts up, ending in the proper and complete loading of the Operating System. The sequence of computer operations from power-up until the system is ready for use

COLD BOOTING:

The cold booting is the situation, when all the computer peripherals are OFF and we start the computer by switching ON the power.

WARM BOOTING:

The warm booting is the situation, when we restart the computer by pressing the RESET button and pressing CTRL+ ALT + DEL keys together.

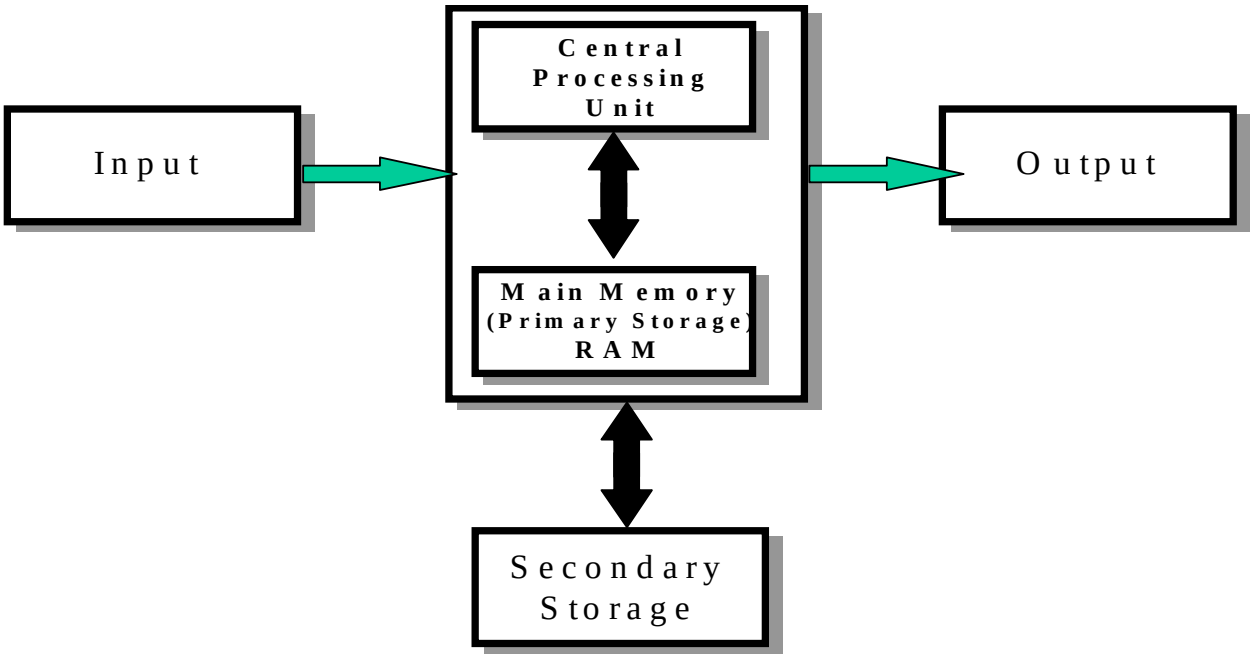
Graphic User Interface (GUI)

A program interface that takes advantage of the computer's graphics capabilities to make the program easier to use. Well-designed graphical user interfaces can free the user from learning complex command languages. On the other hand, many users find that they work more effectively with a command-driven interface, especially if they already know the command language.

BASIC COMPUTER ORGANIZATION:

A standard fully featured desktop configuration has basically four types of featured devices

1. Input Devices 2. Output Devices 3. Memory 4. Storage Devices



Introduction to CPU

- CPU (Central Processing Unit)
- The Arithmetic / Logic Unit (ALU)
- The Control Unit
- Memory
- Main
- External
- Input / Output Devices
- The System Bus

CPU Operation

The fundamental operation of most CPUs

- To execute a sequence of stored instructions called a program.
  - The program is represented by a series of numbers that are kept in some kind of computer memory.
  - There are four steps that nearly all CPUs use in their operation: fetch, decode, execute, and write back.
  - **Fetch:**
    - Retrieving an instruction from program memory.
    - The location in program memory is determined by a program counter (PC)
    - After an instruction is fetched, the PC is incremented by the length of the instruction word in terms of memory units.

- **Decode :**
  - The instruction is broken up into parts that have significance to other portions of the CPU.
  - The way in which the numerical instruction value is interpreted is defined by the CPU's instruction set architecture (ISA).
  - Opcode, indicates which operation to perform.
  - The remaining parts of the number usually provide information required for that instruction, such as operands for an addition operation.
  - Such operands may be given as a constant value or as a place to locate a value: a register or a memory address, as determined by some addressing mode.
- **Execute :**
  - During this step, various portions of the CPU are connected so they can perform the desired operation.
  - If, for instance, an addition operation was requested, an arithmetic logic unit (ALU) will be connected to a set of inputs and a set of outputs.
  - The inputs provide the numbers to be added, and the outputs will contain the final sum.
  - If the addition operation produces a result too large for the CPU to handle, an arithmetic overflow flag in a flags register may also be set.
- **Write back :**
  - Simply "writes back" the results of the execute step to some form of memory.
  - Very often the results are written to some internal CPU register for quick access by subsequent instructions.
  - In other cases results may be written to slower, but cheaper and larger, main memory.
  - Some types of instructions manipulate the program counter rather than directly produce result data.

Input Devices

Anything that feeds the data into the computer. This data can be in alpha-numeric form which needs to be keyed-in or in its very basic natural form i.e. hear, smell, touch, see; taste & the sixth sense …feel?

Typical input devices are:

1. Keyboard
2. Mouse
3. Joystick
4. Digitizing Tablet
5. Touch Sensitive Screen
6. Light Pen
7. Space Mouse
8. Digital Stills Camera
9. Magnetic Ink Character Recognition (MICR)
10. Optical Mark Reader (OMR)
11. Image Scanner
12. Bar Codes
13. Magnetic Reader
14. Smart Cards
15. Voice Data Entry
16. Sound Capture
17. Video Capture

The Keyboard is the standard data input and operator control device for a computer. It consists of the standard QWERTY layout with a numeric keypad and additional function keys for control purposes.

The Mouse is a popular input device. You move it across the desk and its movement is shown on the screen by a marker known as a 'cursor'. You will need to click the buttons at the top of the mouse to select an option.

Track ball looks like a mouse, as the roller is on the top with selection buttons on the side. It is also a pointing device used to move the cursor and works like a mouse. For moving the cursor in a particular

direction, the user spins the ball in that direction. It is sometimes considered better than a mouse, because it requires little arm movement and less desktop space. It is generally used with Portable computers.

Magnetic Ink Character Recognition (MICR) is used to recognize the magnetically charged characters, mainly found on bank cheques. The magnetically charged characters are written by special ink called magnetic ink. MICR device reads the patterns of these characters and compares them with special patterns stored in memory. Using MICR device, a large volume of cheques can be processed in a day. MICR is widely used by the banking industry for the processing of cheques.

The **joystick** is a rotary lever. Similar to an aircraft's control stick, it enables you to move within the screen's environment, and is widely used in the computer games industry.

A **Digitising Tablet** is a pointing device that facilitates the accurate input of drawings and designs. A drawing can be placed directly on the tablet, and the user traces outlines or inputs coordinate positions with a hand-held stylus.

A **Touch Sensitive Screen** is a pointing device that enables the user to interact with the computer by touching the screen. There are three types of Touch Screens: ***pressure-sensitive, capacitive surface and light beam.***

A **Light Pen** is a pointing device shaped like a pen and is connected to a VDU. The tip of the light pen contains a light-sensitive element which, when placed against the screen, detects the light from the screen enabling the computer to identify the location of the pen on the screen. Light pens have the advantage of 'drawing' directly onto the screen, but this can become uncomfortable, and they are not as accurate as digitising tablets.

The **Space mouse** is different from a normal mouse as it has an X axis, a Y axis and a Z axis. It can be used for developing and moving around 3-D environments.

Digital Stills Cameras capture an image which is stored in memory within the camera. When the memory is full it can be erased and further images captured. The digital images can then be downloaded from the camera to a computer where they can be displayed, manipulated or printed.

The **Optical Mark Reader (OMR)** can read information in the form of numbers or letters and put it into the computer. The marks have to be precisely located as in multiple choice test papers.

Scanners allow information such as a photo or text to be input into a computer. Scanners are usually either A4 size (flatbed), or hand-held to scan a much smaller area. If text is to be scanned, you would use an Optical Character Recognition (OCR) program to recognise the printed text and then convert it to a digital text file that can be accessed using a computer.

A **Bar Code** is a pattern printed in lines of differing thickness. The system gives fast and error-free entry of information into the computer. You might have seen bar codes on goods in supermarkets, in libraries and on magazines. Bar codes provide a quick method of recording the sale of items.

Card Reader This input device reads a **magnetic strip** on a card. Handy for security reasons, it provides quick identification of the card's owner. This method is used to run bank cash points or to provide quick identification of people entering buildings.

Smart Card This input device stores data in **a microprocessor embedded in the card**. This allows information, which can be updated, to be stored on the card. This method is used in store cards which accumulate points for the purchaser, and to store phone numbers for cellular phones.

Output Devices

Output devices display information in a way that you can you can understand. The most common output device is a monitor. It looks a lot a like a TV and houses the computer screen. The monitor allows you to 'see' what you and the computer are doing together.

Brief of Output Device

Output devices are pieces of equipment that are used to get information or any other response out from computer. These devices display information that has been held or generated within a computer. Output devices display information in a way that you can understand. The most common output device is a monitor.

Types of Output Device

Printing: Plotter, Printer
Sound : Speakers
Visual : Monitor

A **Printer** is another common part of a computer system. It takes what you see on the computer screen and prints it on paper. There are two types of printers; Impact Printers and Non-Impact Printers.

Speakers are output devices that allow you to hear sound from your computer. Computer speakers are just like stereo speakers. There are usually two of them and they come in various sizes.

Memory or Primary Storage

Purpose of Storage

The fundamental components of a general-purpose computer are arithmetic and logic unit, control circuitry, storage space, and input/output devices. If storage was removed, the device we had would be a simple calculator instead of a computer. The ability to store instructions that form a computer program, and the information that the instructions manipulate is what makes stored program architecture computers versatile.

- **Primary storage**, or **internal memory**, is computer memory that is accessible to the central processing unit of a computer without the use of computer's input/output channels
- Primary storage, also known as main storage or memory, is the main area in a computer in which data is stored for quick access by the computer's processor.

Primary Storage

Primary storage is directly connected to the central processing unit of the computer. It must be present for the CPU to function correctly, just as in a biological analogy the lungs must be present (for oxygen storage) for the heart to function (to pump and oxygenate the blood). As shown in the diagram, primary storage typically consists of three kinds of storage:

Processors Register

It is the internal to the central processing unit. Registers contain information that the arithmetic and logic unit needs to carry out the current instruction. They are technically the fastest of all forms of computer storage.

Main memory

It contains the programs that are currently being run and the data the programs are operating on. The arithmetic and logic unit can very quickly transfer information between a processor register and locations in main storage, also known as a "memory addresses". In modern computers, electronic solid-state random access memory is used for main storage, and is directly connected to the CPU via a "memory bus" and a "data bus".

Cache memory

It is a special type of internal memory used by many central processing units to increase their performance or "throughput". Some of the information in the main memory is duplicated in the cache memory, which is slightly slower but of much greater capacity than the processor registers, and faster but much smaller than main memory.

Memory

Memory is often used as a shorter synonym for **Random Access Memory (RAM).** This kind of memory is located on one or more microchips that are physically close to the microprocessor in your computer. Most desktop and notebook computers sold today include at least 512 megabytes of RAM (which is really the minimum to be able to install an operating system). They are upgradeable, so you can add more when your computer runs really slowly.

The more RAM you have, the less frequently the computer has to access instructions and data from the more slowly accessed hard disk form of storage. Memory should be distinguished from storage, or the physical medium that holds the much larger amounts of data that won't fit into RAM and may not be immediately needed there.

Storage devices include hard disks, floppy disks, CDROMs, and tape backup systems. The terms auxiliary storage, auxiliary memory, and secondary memory have also been used for this kind of data repository.

RAM is temporary memory and is erased when you turn off your computer, so remember to save your work to a permanent form of storage space like those mentioned above before exiting programs or turning off your computer.

TYPES OF RAM:

There are two types of RAM used in PCs - Dynamic and Static RAM.

Dynamic RAM (DRAM): The information stored in Dynamic RAM has to be refreshed after every few milliseconds otherwise it will get erased. DRAM has higher storage capacity and is cheaper than Static RAM.

Static RAM (SRAM): The information stored in Static RAM need not be refreshed, but it remains stable as long as power supply is provided. SRAM is costlier but has higher speed than DRAM.

Additional kinds of integrated and quickly accessible memory are **Read Only Memory (ROM),** Programmable ROM (PROM), and Erasable Programmable ROM (EPROM). These are used to keep special programs and data, such as the BIOS, that need to be in your computer all the time. ROM is "built-in" computer memory containing data that normally can only be read, not written to (hence the name read only).

ROM contains the programming that allows your computer to be "booted up" or regenerated each time you turn it on. Unlike a computer's random access memory (RAM), the data in ROM is not lost when the computer power is turned off. The ROM is sustained by a small long life battery in your computer called the CMOS battery. If you ever do the hardware setup procedure with your computer, you effectively will be writing to ROM. It is non volatile, but not suited to storage of large quantities of data because it is expensive to produce. Typically, ROM must also be completely erased before it can be rewritten,

PROM (Programmable Read Only Memory)

A variation of the ROM chip is programmable read only memory. PROM can be programmed to record information using a facility known as PROM-programmer. However once the chip has been programmed the recorded information cannot be changed, i.e. the PROM becomes a ROM and the information can only be read.

EPROM (Erasable Programmable Read Only Memory)

As the name suggests the Erasable Programmable Read Only Memory, information can be erased and the chip programmed a new to record different information using a special PROM-Programmer. When EPROM is in use information can only be read and the information remains on the chip until it is erased.

Storage Devices

The purpose of storage in a computer is to hold data or information and get that data to the CPU as quickly as possible when it is needed. Computers use disks for storage: hard disks that are located inside the computer, and floppy or compact disks that are used externally.

- Computers Method of storing data & information for long term basis i.e. even after PC is switched off.
- It is non – volatile
- Can be easily removed and moved & attached to some other device
- Memory capacity can be extended to a greater extent
- Cheaper than primary memory

Storage Involves Two Processes

a) Writing data b) Reading data

Floppy Disks

The floppy disk drive (**FDD**) was invented at IBM by Alan Shugart in 1967. The first floppy drives used an 8-inch disk (later called a "**diskette**" as it got smaller), which evolved into the 5.25-inch disk that was used on the first IBM Personal Computer in August 1981. The 5.25-inch disk held 360 kilobytes compared to the 1.44 megabyte capacity of today's 3.5-inch diskette.

The 5.25-inch disks were dubbed "**floppy**" because the diskette packaging was a very **flexible plastic envelope**, unlike the rigid case used to hold today's 3.5-inch diskettes.

By the mid-1980s, the improved designs of the read/write heads, along with improvements in the magnetic recording media, led to the less-flexible, 3.5-inch, 1.44-megabyte (MB) capacity FDD in use today. For a few years, computers had both FDD sizes (3.5-inch and 5.25-inch). But by the mid-1990s, the 5.25-inch version had fallen out of popularity, partly because the diskette's recording surface could easily become contaminated by fingerprints through the open access area.

When you look at a floppy disk, you'll see a plastic case that measures 3 1/2 by 5 inches. Inside that case is a very thin piece of plastic that is coated with microscopic iron particles. This disk is much like the tape inside a video or audio cassette. Basically, a floppy disk drive reads and writes data to a small, circular piece of metal-coated plastic similar to audio cassette tape.

At one end of it is a small metal cover with a rectangular hole in it. That cover can be moved aside to show the flexible disk inside. But never touch the inner disk - you could damage the data that is stored on it. On one side of the floppy disk is a place for a label. On the other side is a silver circle with two holes in it. When the disk is inserted into the disk drive, the drive hooks into those holes to spin the circle. This causes the disk inside to spin at about 300 rpm! At the same time, the silver metal cover on the end is pushed aside so that the head in the disk drive can read and write to the disk.

Floppy disks are the smallest type of storage, holding only 1.44MB.

3.5-inch Diskettes (Floppy Disks) features:

- Spin rate: app. 300 revolutions per minute (rpm)
- High density (HD) disks more common today than older, double density (DD) disks
- Storage Capacity of HD disks is 1.44 MB

Floppy Disk Drive Terminology

- **Floppy disk** - Also called diskette. The common size is 3.5 inches.
- **Floppy disk drive** - The electromechanical device that reads and writes floppy disks.
- **Track** - Concentric ring of data on a side of a disk.
- **Sector** - A subset of a track, similar to wedge or a slice of pie.

It consists of a read/write head and a motor rotating the disk at a high speed of about 300 rotations per minute. It can be fitted inside the cabinet of the computer and from outside, the slit where the disk is to be inserted, is visible. When the disk drive is closed after inserting the floppy inside, the monitor catches the disk through the Central of Disk hub, and then it starts rotating.

There are two read/write heads depending upon the floppy being one sided or two sided. The head consists of a read/write coil wound on a ring of magnetic material. During write operation, when the current passes in one direction, through the coil, the disk surface touching the head is magnetized in one direction. For reading the data, the procedure is reverse. I.e. the magnetized spots on the disk touching the read/write head induce the electronic pulses, which are sent to CPU.

The major parts of a FDD include:

- **Read/Write Heads**: Located on both sides of a diskette, they move together on the same assembly. The heads are not directly opposite each other in an effort to prevent interaction between write operations on each of the two media surfaces. The same head is used for reading and writing, while a second, wider head is used for erasing a track just prior to it being written. This allows the data to be written on a wider "clean slate," without interfering with the analog data on an adjacent track.
- **Drive Motor**: A very small spindle motor engages the metal hub at the center of the diskette, spinning it at either 300 or 360 rotations per minute (RPM).
- **Stepper Motor**: This motor makes a precise number of stepped revolutions to move the read/write head assembly to the proper track position. The read/write head assembly is fastened to the stepper motor shaft.

- **Mechanical Frame**: A system of levers that opens the little protective window on the diskette to allow the read/write heads to touch the dual-sided diskette media. An external button allows the diskette to be ejected, at which point the spring-loaded protective window on the diskette closes.
- **Circuit Board**: Contains all of the electronics to handle the data read from or written to the diskette. It also controls the stepper-motor control circuits used to move the read/write heads to each track, as well as the movement of the read/write heads toward the diskette surface.

Electronic optics check for the presence of an opening in the lower corner of a 3.5-inch diskette (or a notch in the side of a 5.25-inch diskette) to see if the user wants to prevent data from being written on it.

Hard Disks

Your computer uses two types of memory: primary memory which is stored on chips located on the motherboard, and secondary memory that is stored in the hard drive. Primary memory holds all of the essential memory that tells your computer how to be a computer. Secondary memory holds the information that you store in the computer.

Inside the hard disk drive case you will find circular disks that are made from polished steel. On the disks, there are many tracks or cylinders. Within the hard drive, an electronic reading/writing device called the head passes back and forth over the cylinders, reading information from the disk or writing information to it. Hard drives spin at 3600 or more rpm (Revolutions Per Minute) - that means that in one minute, the hard drive spins around over 7200 times!

Optical Storage

- Compact Disk Read-Only Memory (CD-ROM)
- CD-Recordable (CD-R)/CD-Rewritable (CD-RW)
- Digital Video Disk Read-Only Memory (DVD-ROM)
- DVD Recordable (DVD-R/DVD Rewritable (DVD-RW)
- Photo CD

Optical Storage Devices Data is stored on a reflective surface so it can be read by a beam of laser light.
Two Kinds of Optical Storage Devices

- CD-ROM (compact disk read-only memory)
- DVD-ROM (digital video disk read-only memory)

Compact Disks

Instead of electromagnetism, CDs use pits (microscopic indentations) and lands (flat surfaces) to store information much the same way floppies and hard disks use magnetic and non-magnetic storage. Inside the CD-Rom is a laser that reflects light off of the surface of the disk to an electric eye. The pattern of reflected light (pit) and no reflected light (land) creates a code that represents data.

CDs usually store about 650MB. This is quite a bit more than the 1.44MB that a floppy disk stores. A DVD or Digital Video Disk holds even more information than a CD, because the DVD can store information on two levels, in smaller pits or sometimes on both sides.

Recordable Optical Technologies

- CD-Recordable (CD-R)
- CD-Rewritable (CD-RW)

- PhotoCD
- DVD-Recordable (DVD-R)
- DVD-RAM

CD ROM - Compact Disc Read Only Memory.

Unlike magnetic storage device which store data on multiple concentric tracks, all CD formats store data on one physical track, which spirals continuously from the center to the outer edge of the recording area. Data resides on the thin aluminum substrate immediately beneath the label.

The data on the CD is recorded as a series of microscopic pits and lands physically embossed on an aluminum substrate. Optical drives use a low power laser to read data from those discs without physical contact between the head and the disc which contributes to the high reliability and permanence of storage device.

To write the data on a CD a higher power laser are used to record the data on a CD. It creates the pits and land on aluminum substrate. The data is stored permanently on the disc. These types of discs are called as WORM (Write Once Read Many). Data written to CD cannot subsequently be deleted or overwritten which can be classified as advantage or disadvantage depending upon the requirement of the user. However if the CD is partially filled then the more data can be added to it later on till it is full. CDs are usually cheap and cost effective in terms of storage capacity and transferring the data.

The CD's were further developed where the data could be deleted and re written. These types of CDs are called as CD Rewritable. These types of discs can be used by deleting the data and making the space for new data. These CD's can be written and rewritten at least 1000 times.

CD ROM Drive

CD ROM drives are so well standardized and have become so ubiquitous that many treat them as commodity items. Although CD ROM drives differ in reliability, which standards they support and numerous other respects, there are two important performance measures.

- ü Data transfer rate
- ü Average access

Data transfer rate: Data transfer rate means how fast the drive delivers sequential data to the interface. This rate is determined by drive rotation speed, and is rated by a number followed by 'X'. All the other things equal, a 32X drive delivers data twice the speed of a 16X drive. Fast data transfer rate is most important when the drive is used to transfer the large file or many sequential smaller files. For example: Gaming video.

CD ROM drive transfers the data at some integer multiple of this basic 150 KB/s 1X rate. Rather than designating drives by actual KB/s output drive manufacturers use a multiple of the standard 1X rate. For example: a 12X drive transfer data at (12*150KB/s) 1800 KB/s and so on.

The data on a CD is saved on tracks, which spirals from the center of the CD to outer edge. The portions of the tracks towards center are shorter than those towards the edge. Moving the data under the head at a constant rate requires spinning the disc faster as the head moves from the center where there is less data per revolution to the edge where there is more data. Hence the rotation rate of the disc changes as it progresses from inner to outer portions of the disc.

CD Writers

CD recordable and CD rewritable drives are collectively called as CD writers or CD burners. They are essentially CD ROM drives with one difference. They have a more powerful laser that, in addition to reading discs, can record data to special CD media.

Pen Drives / Flash Drives

- Pen Drives / Flash Drives are flash memory storage devices.
- They are faster, portable and have a capability of storing large data.
- It consists of a small printed circuit board with a LED encased in a robust plastic
- The male type connector is used to connect to the host PC
- They are also used a MP3 players

Printers

Printers are hardware devices that allow you to create a hard copy of a file. Today a printer is a necessary requirement for any home user and business. Allowing individuals to save their work in the format of paper instead of electronically.

Types of Printers

- **Impact printers**
  - In case of Impact printer an inked ribbon exists between the print head and paper ,the head striking the ribbon prints the character.
- **Non Impact Printers**
  - Non Impact printers use techniques other than the mechanical method of head striking the ribbon

Impact printers

Impact printers are basically divided into 2 types

- Serial/Character printers
  - Dot matrix printers
- Daisy wheel printers
  - Line Printers

Non-Impact Printers

Non Impact Printers are divided into 3 categories

- Thermal printers
- Ink jet printers
- Laser printers

Classification

Printers are classified by the following characteristics:

Quality of type: The output produced by printers is said to be either *letter quality* (as good as a typewriter), *near letter quality*, or *draft quality*. Only daisy-wheel, ink-jet, and laser printers produce letter-quality type. Some dot-matrix printers claim letter-quality print, but if you look closely, you can see the difference.

Speed: Measured in characters per second (cps) or pages per minute (ppm), the speed of printers varies widely. Daisy-wheel printers tend to be the slowest, printing about 30 cps. Line printers are fastest (up to 3,000 lines per minute). Dot-matrix printers can print up to 500 cps, and laser printers range from about 4 to 20 text pages per minute.

Impact or non-impact: Impact printers include all printers that work by striking an ink ribbon. Daisy-wheel, dot-matrix, and line printers are impact printers. Non-impact printers include laser printers and ink-jet printers. The important difference between impact and non-impact printers is that impact printers are much noisier.

Graphics: Some printers (daisy-wheel and line printers) can print only text. Other printers can print both text and graphics.

Fonts: Some printers, notably dot-matrix printers, are limited to one or a few fonts. In contrast, laser and ink-jet printers are capable of printing an almost unlimited variety of fonts. Daisy-wheel printers can also print different fonts, but you need to change the daisy wheel, making it difficult to mix fonts in the same document.

Dot Matrix Printers

A dot matrix printer or impact matrix printer refers to a type of computer printer with a print head that runs back and forth on the page and prints by impact, striking an ink-soaked cloth ribbon against the paper, much like a typewriter. Unlike a typewriter or daisy wheel printer, letters are drawn out of a dot matrix, and thus, varied fonts and arbitrary graphics can be produced. Because the printing involves mechanical pressure, these printers can create carbon copies and carbonless copies. The standard of print obtained is poor. These printers are cheap to run and relatively fast.

The moving portion of the printer is called the print head, and prints one line of text at a time. Most dot matrix printers have a single vertical line of dot-making equipment on their print heads; others have a few interleaved rows in order to improve dot density. The print head consists of 9 or 24 pins each can move freely within the tube; more the number of pins better are the quality of output. Dot Matrix Printer Characters are formed from a matrix of dots.

The speed is usually 30 - 550 characters per second (cps). These types of printers can print graphs also. They can only print text and graphics, with limited color performance. Impact printers have one of the lowest printing costs per page. These machines can be highly durable, but eventually wear out. Ink invades the guide plate of the print head, causing grit to adhere to it; this grit slowly causes the channels in the guide plate to wear from circles into ovals or slots, providing less and less accurate guidance to the printing wires. After about a million characters, even with tungsten blocks and titanium pawls, the printing becomes too unclear to read.

Daisy Wheel Printer

A daisy wheel printer is a type of computer printer that produces high-quality type, and is often referred to as a letter-quality printer (this in contrast to high-quality dot-matrix printers, capable of near-letter-quality, or NLQ, output). There were also, and still are daisy wheel typewriters, based on the same principle. The DWP is slower the speed range is in 30 to 80 CPS.

The system used a small wheel with each letter printed on it in raised metal or plastic. The printer turns the wheel to line up the proper letter under a single pawl which then strikes the back of the letter and drives it into the paper. In many respects the daisy wheel is similar to a standard typewriter in the way it forms its letters on the page, differing only in the details of the mechanism (daisy wheel vs typebars or the type ball used on IBMs electric typewriters).

Daisy wheel printers were fairly common in the 1980s, but were always less popular than dot matrix printers (ballistic wire printers) due to the latter's ability to print graphics and different fonts. With the introduction of high quality laser printers and inkjet printers in the later 1980s daisy wheel systems quickly disappeared but for the small remaining typewriter market.

Line Printer

The **line printer** is a form of high speed impact printer in which a line of type is printed at a time.
The wheels spin at high speed and paper and an inked ribbon are stepped (moved) past the print position. As the desired character for each column passes the print position, a hammer strikes the paper and ribbon causing the desired character to be recorded on the continuous paper. The speed is 300 to 2500 lines per minute (LPM). This technology is still in use in a number of applications. It is usually both faster and less expensive (in total ownership) than laser printers. In printing box labels, medium volume accounting and other large business applications, line printers remain in use

Line printers, as the name implies, print an entire line of text at a time. Two principle designs existed. In *drum printers*, a drum carries the entire character set of the printer repeated in each column that is to be printed. In *chain printers* (also known as *train printers*), the character set is arranged multiple times around a chain that travels horizontally pas the print line. In either case, to print a line, precisely timed hammers strike against the back of the paper at the exact moment that the correct character to be printed is passing in front of the paper. The paper presses forward against a ribbon which then presses against the character form and the impression of the character form is printed onto the paper.

These printers were the fastest of all impact printers and were used for bulk printing in large computer centers. They were virtually never used with personal computers and have now been partly replaced by high-speed laser printers.

Thermal Printers

Direct thermal printers create an image by selectively heating coated paper when the paper passes over the thermal print head. The coating turns black in the areas where it is heated, creating the image. More recently, two-color direct thermal printers have been produced, which allow printing of both red (or another color) and black by heating to different temperatures.

Thermal Printer Characters are formed by heated elements being placed in contact with special heat sensitive paper forming darkened dots when the elements reach a critical temperature. A fax machine uses a thermal printer. Thermal printer paper tends to darken over time due to exposure to sunlight and heat. The standard of print produced is poor. Thermal printers are widely used in battery powered equipment such as portable calculators.

Direct thermal printers are increasingly replacing the dot matrix printer for printing cash register receipts, both because of the higher print speed and substantially quieter operation. In addition, direct thermal printing offers the advantage of having only one consumable - the paper itself. Thus, the technology is well-suited to unattended applications like gas pumps, information kiosks, and the like.

Until about 2000, most fax machines used direct thermal printing, though, now, only the cheapest models use it, the rest having switched to either thermal wax transfer, laser, or ink jet printing to allow plain-paper printouts. Historically, direct thermal paper has suffered from such limitations as sensitivity to heat, abrasion (the coating can be fragile), friction (which can cause heat, thus darkening the paper), light (causing it to fade), and water. However, more modern thermal coating formulations have resulted in exceptional image stability, with text remaining legible for an estimated 50+ years.

Ink-Jet Printers

Inkjet printers spray very small, precise amounts (usually a few picolitres) of ink onto the media. They are the most common type of computer printer for the general consumer due to their low cost, high quality of output, capability of printing in vivid color, and ease of use. It is the most common printer used with home computers and it can print in either black and white or color.

Compared to earlier consumer-oriented printers, ink jets have a number of advantages. They are quieter in operation than impact dot matrix or daisywheel printers. They can print finer, smoother details through higher print head resolution, and many ink jets with photorealistic-quality color printing are widely available. For color applications including photo printing, ink jet methods are dominant.

Laser Printers
A laser printer is a common type of computer printer that produces high quality printing, and is able to produce both text and graphics. The process is very similar to the type of dry process photocopier first produced by Xerox.

Laser Printers use a laser beam and dry powdered ink to produce a fine dot matrix pattern. This method of printing can generate about 4 pages of A4 paper per minute. The standard of print is very good and laser printers can also produce very good quality printed graphic images too.

SCANNERS:

Technology today is rising to it's heights. For time saving and to have paperless offices we have a need of electronic version of invoice, Material ordering forms, Contract ordering data etc…for filing and database management. Even to automate the process of logging sales data into Excel, a scanner can help one with all of these tasks and more.

A scanner is an optical device that captures images, objects, and documents into a digital format. The image is read as thousands of individual dots, or pixels. It can convert a picture into digital bits of information which are then reassembled by the computer with the help of scanning software. The file of the image can then be enlarged or reduced, stored in a database, or transferred into a word processing or spreadsheet program.

Some of the key considerations for choosing the right scanner for your needs are given below.

a) How you intend to use the scanner?
b) Which type of scanner fits the exact usage?
c) Does one require a Black & White or a Colour quality output?
d) What is the Price and the Software bundles?

Depending upon the usage and the importance of the business if one would like to have quality photographs or other images, than colour quality will be an important characteristic. With both a black and white and a color quality output the bit depth, resolution and dynamic range are essential to selecting the right scanner for ones need.

Scanner Types:
Scanners create a digital reproduction of an image or document and come in a variety of shapes and sizes designed to perform different types of tasks. There are three types of office scanners usually seen in the market and the functions they serve are as follows:

a) Flatbed

The flatbed scanner consists of its own base with a flat piece of glass and cover just as is found on most copiers. The scanning component of flatbeds runs over the length of the image in order to gather data. Flatbeds are useful when a user needs to scan more than single page documents. Pages from a book, for example, can easily be scanned without having to copy each page individually first.

Scanning objects is also done by flatbeds. By placing a white sheet of paper over a bouquet of flowers a scanner can reproduce what appears to be a stock photo onscreen.

Flatbeds have large footprint and hence take up a lot of desk thus if space is a concern one may go for an alternative.

b) Sheetfed

Sheetfed scanners are only used if one wants to scan for anything other than sheets of paper. The scanning component of a sheetfed is stationary while the document being scanned passes over it's 'eyes' similar to a fax machine. It is so thin just a couple of inches deep, such that it can easily fit between keyboards and monitor.

Sheetfeds usually work best in conjunction with an automatic document feeder for large projects. Pictures and other documents which are smaller than a full page can also be scanned using a sheetfed scanner. They have been known to bend pictures and reproduce less than quality images.

c) Slide

There is a need for accurate reproduce of very small images. For such application the resolution required is very sharp and slide types of scanner create a totally different scanner market. Slides are usually inserted into a tray, much like a CD tray on ones computer, and scanned internally. Most slide scanners can only scan slides, though some newer models can also handle negative strips.

Scanner Uses:

A scanner can do far more than simply scan a photograph, and many of its uses could go a long way to helping a small business. Below are indicated some of the applications for the scanner in a business environment.

1) Graphics

Graphic images are an important part of many businesses specially in marketing and sales functions. Scanners, like digital cameras, enable users to convert photographs, slides, and three-dimensional objects into files that can be pasted into a brochure, inserted into a presentation or posted on the Internet. Using accompanying software, these images can be edited, cropped, or manipulated to fit space and size requirements.

2) Data-Entry

Scanners automatically convert the data into digital files using OCR (Optical Character Recognition) software; this would save time and money which one would pay to someone to manually enter the reams of data into the computer. In conjunction with the software, a scanner reads each page and transfers the text to any number of programs. A form letter can be saved to a word processing program, sales figures to a spreadsheet, even a brochure to web-editing software.

3) Digital-Files

One observes that there are numerous papers filed in three-ring binders or different kinds of manual filing in the offices for records. The process of the manual paper flow can be avoided by using scanners of Digital type. Such scanners can help to create electronic filing cabinets for everything from invoices to expense reports. Forms can be reproduced online, and searchable databases can provide relevant information in seconds.

Pointer:

A symbol that appears on the display screen and that you move to select objects and commands. Usually, the pointer appears as a small angled arrow. Text -processing applications, however, use an I-beam pointer that is shaped like a capital I.

Pointing device:

A device, such as a mouse or trackball that enables you to select objects on the display screen.

Icons:

Small pictures that represent commands, files, or windows. By moving the pointer to the icon and pressing a mouse button, you can execute a command or convert the icon into a window. You can also move the icons around the display screen as if they were real objects on your desk.

Desktop:

The area on the display screen where icons are grouped is often referred to as the desktop because the icons are intended to represent real objects on a real desktop.

Windows:

You can divide the screen into different areas. In each window, you can run a different program or display a different file. You can move windows around the display screen, and change their shape and size at will.

Menus:

Most graphical user interfaces let you execute commands by selecting a choice from a menu.

In addition to their visual components, graphical user interfaces also make it easier to move data from one application to another. A true GUI includes standard formats for representing text and graphics. Because the formats are well-defined, different programs that run under a common GUI can share data. This makes it possible, for example, to copy a graph created by a spreadsheet program into a document created by a word processor.

Character User Interface/Text User Interface (CUI/TUI)

Short for Character User Interface or Command-line User Interface, CUI is another name for a command line. Early user interfaces were CUI. That is they could only display the characters defined in the ASCII set. Examples of this type of interface are the command line interfaces provided with DOS 3.3 and early implementations of UNIX and VMS.

This was limiting, but it was the only choice primarily because of 2 hardware constraints. Early CPUs did not have the processing power to manage a GUI. Also, the video controllers and monitors were unable to display the high resolution necessary to implement a GUI.

FUNCTIONS OF CPU:

Process Management

The CPU executes a large number of programs. While its main concern is the execution of user programs, the CPU is also needed for other system activities. These activities are called processes. A process is a program in execution. Typically, a batch job is a process. A time-shared user program is a process. A system task, such as spooling, is also a process. For now, a process may be considered as a job or a time-shared program, but the concept is actually more general.

In general, a process will need certain resources such as CPU time, memory, files, I/O devices, etc., to accomplish its task. These resources are given to the process when it is created. In addition to the various physical and logical resources that a process obtains when it is created, some initialization data (input) may be passed along.

We emphasize that a program by itself is not a process; a program is a passive entity. It is known that two processes may be associated with the same program; they are nevertheless considered two separate execution sequences.

The operating system is responsible for the following activities in connection with processes managed.

- The creation and deletion of both user and system processes
- The suspension and resumption of processes.
- The provision of mechanisms for process synchronization
- The provision of mechanisms for deadlock handling.

Memory Management

Memory is central to the operation of a modern computer system. Memory is a large array of words or bytes, each with its own address. Interaction is achieved through a sequence of reads or writes of specific memory address. The CPU fetches from and stores in memory. In order for a program to be executed it must be mapped to absolute addresses and loaded in to memory.

In order to improve both the utilization of CPU and the speed of the computer's response to its users, several processes must be kept in memory.

The operating system is responsible for the following activities in connection with memory management.

- Keep track of which parts of memory are currently being used and by whom.
- Decide which processes are to be loaded into memory when memory space becomes available.
- Allocate and de-allocate memory space as needed.

Secondary Storage Management

The main purpose of a computer system is to execute programs. These programs, together with the data they access, must be in main memory during execution. Since the main memory is too small to permanently accommodate all data and program, the computer system must provide secondary storage to backup main memory. Most modem computer systems use disks as the primary on-line storage of information, of both programs and data.

Most programs, like compilers, assemblers, sort routines, editors, formatters, and so on, are stored on the disk until loaded into memory, and then use the disk as both the source and destination of their processing. Hence the proper management of disk storage is of central importance to a computer system.

There are few alternatives. Magnetic tape systems are generally too slow. In addition, they are limited to sequential access. Thus tapes are more suited for storing infrequently used files, where speed is not a primary concern.

The operating system is responsible for the following activities in connection with disk management

- Free space management
- Storage allocation
- Disk scheduling.

Input Output System

One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user. For example, in UNIX, the peculiarities of Input/Output devices are hidden from the bulk of the operating system itself by the INPUT/OUTPUT system. The Input/Output system consists of:

- A buffer caching system
- A general device driver code
- Drivers for specific hardware devices.

Only the device driver knows the peculiarities of a specific device.

File Management

File management is one of the most visible services of an operating system. Computers can store information in several different physical forms; magnetic tape, disk, and drum are the most common forms. Each of these devices has it own characteristics and physical organization. For convenient use of the computer system, the operating system provides a uniform logical view of information storage. The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, the file. Files are mapped, by the operating system, onto physical devices.

A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic or alphanumeric. Files may be free-form, such as text files, or may be rigidly formatted. In general files are a sequence of bits, bytes, lines or records whose meaning is defined by its creator and user. It is a very general concept.
The operating system implements the abstract concept of the file by managing mass storage device, such as tapes and disks. Also files are normally organized into directories to ease their use. Finally, when multiple users have access to files, it may be desirable to control by whom and in what ways files may be accessed.

The operating system is responsible for the following activities in connection with file management:

- The creation and deletion of files
- The creation and deletion of directory
- The support of primitives for manipulating files and directories
- The mapping of files onto disk storage.
- Backup of files on stable (non volatile) storage.

Protection System

The various processes in an operating system must be protected from each other's activities. For that purpose, various mechanisms which can be used to ensure that the files, memory segment, CPU and other resources can be operated on only by those processes that have gained proper authorization from the operating system.

Protection refers to a mechanism for controlling the access of programs, processes, or users to the resources defined by a computer controls to be imposed, together with some means of enforcement. An unprotected resource cannot defend against use (or misuse) by an unauthorized or incompetent user.

Networking

A distributed system is a collection of processors that do not share memory or a clock. Instead, each processor has its own local memory, and the processors communicate with each other through various communication lines, such as high speed buses or telephone lines. Distributed systems vary in size and function. They may involve microprocessors, workstations, minicomputers, and large general purpose computer systems.

The processors in the system are connected through a communication network, which can be configured in the number of different ways. The network may be fully or partially connected. The communication network design must consider routing and connection strategies, and the problems of connection and security. A distributed system provides the user with access to the various resources the system maintains. Access to a shared resource allows computation speed-up, data availability, and reliability.

Command Interpreter System

One of the most important components of an operating system is its command interpreter. The command interpreter is the primary interface between the user and the rest of the system. Many commands are given

to the operating system by control statements. When a new job is started in a batch system or when a user logs-in to a time-shared system, a program which reads and interprets control statements is automatically executed.

NUMBER SYSTEMS

Binary	Decimal	Octal	Hexadecimal
0000	00	0	0
0001	01	1	1
0010	02	2	2
0011	03	3	3
0100	04	4	4
0101	05	5	5
0110	06	6	6
0111	07	7	7
1000	08	10	8
1001	09	11	9
1010	10	12	A
1011	11	13	B
1100	12	14	C
1101	13	15	D
1110	14	16	E
1111	15	17	F

DECIMAL NUMBERS

In the decimal number systems each of the ten digits, 0 through 9, represents a certain quantity. The position of each digit in a decimal number indicates the magnitude of the quantity represented and can be assigned a weight. The weights for whole numbers are positive powers of ten that increases from right to left, beginning with $10^o = 1$ that is $\mathbf{10^3\ 10^2\ 10^1\ 10^o}$

For fractional numbers, the weights are negative powers of ten that decrease from left to right beginning with 10^{-1} that is $\mathbf{10^2\ 10^1\ 10^o.\ 10^{-1}\ 10^{-2}\ 10^{-3}}$
The value of a decimal number is the sum of digits after each digit has been multiplied by its weights as in following examples

Express the decimal number 87 as a sum of the values of each digit.

The digit 8 has a weight of 10 which is 10 as indicated by its position. The digit 7 has a weight of 1 which is 10^o as indicated by its position.

$$\mathbf{87 = (8 \times 10^{1)} + (7 \times 10^0)}$$

Express the decimal number 725.45 as a sum of the values of each digit.

$$725.45 = (7 \times 10^2) + (2 \times 10^1) + (5 \times 10^0) + (4 \times 10^{-1}) + (5 \times 10^{-2}) = 700 + 20 + 5 + 0.4 + 0.05$$

BINARY NUMBERS

The binary system is less complicated than the decimal system because it has only two digits, it is a base-two system. The two binary digits (bits) are 1 and 0. The position of a 1 or 0 in a binary number indicates its weight, or value within the number, just as the position of a decimal digit determines the value of that digit. The weights in a binary number are based on power of two as:

$$\ldots\ldots 2^4\ 2^3\ 2^2\ 2^1\ 2^0 .\ 2^{-1}\ 2^{-2}\ \ldots.$$

With 4 digits position we can count from zero to 15.In general, with n bits we can count up to a number equal to $2^n - 1$. Largest decimal number = $2^n - 1$.A binary number is a weighted number. The right-most bit is the least significant bit (LSB) in a binary whole number and has a weight of $2^0 = 1$. The weights increase from right to left by a power of two for each bit. The left-most bit is the most significant bit (MSB); its weight depends on the size of the binary number.

BINARY-TO-DECIMAL CONVERSION

The decimal value of any binary number can be found by adding the weights of all bits that are 1 and discarding the weights of all bits that are 0

Example

Let's convert the binary whole number **101101** to decimal

Weight:	2^5	2^4	2^3	2^2	2^1	2^0
x						
Binary no:	**1**	**0**	**1**	**1**	**0**	**1**
Value	**32**	**0**	**8**	**4**	**0**	**1**

Sum = 45

HEXADECIMAL NUMBERS

The hexadecimal number system has sixteen digits and is used primarily as a compact way of displaying or writing binary numbers because it is very easy to convert between binary and hexadecimal. Long binary numbers are difficult to read and write because it is easy to drop or transpose a bit. Hexadecimal is widely used in computer and microprocessor applications. The hexadecimal system has a base of sixteen; it is composed of 16 digits and alphabetic characters. The maximum 3-digits hexadecimal number is FFF or decimal 4095 and maximum 4-digit hexadecimal number is FFFF or decimal 65.535.

BINARY-TO-HEXADECIMAL CONVERSION

Simply break the binary number into 4-bit groups, starting at the right-most bit and replace each 4-bit group with the equivalent hexadecimal symbol as in the following example

Convert the binary number to hexadecimal: 1100101001010111

Solution:

1100 1010 0101 0111

C A 5 7 = CA57

HEXADECIMAL-TO-DECIMAL CONVERSION

One way to find the decimal equivalent of a hexadecimal number is to first convert the hexadecimal number to binary and then convert from binary to decimal.

Convert the hexadecimal number 1C to decimal:

1 C
0001 1100 = $2^4 + 2^3 + 2^2$ = 16 +8+4 = 28

DECIMAL-TO-HEXADECIMAL CONVERSION

Repeated division of a decimal number by 16 will produce the equivalent hexadecimal number, formed by the remainders of the divisions. The first remainder produced is the least significant digit (LSD).
Each successive division by 16 yields a remainder that becomes a digit in the equivalent hexadecimal number. When a quotient has a fractional part, the fractional part is multiplied by the divisor to get the remainder.

Convert the decimal number 650 to hexadecimal by repeated division by 16

650 /16 = 40.625 0.625 x 16 = 10 = A (LSD)
40 /16 = 2.5 0.5 x 16 = 8 = 8
2/16 = 0.125 0.125 x 16 = 2 = 2 (MSD)

The hexadecimal number is 28A

OCTAL NUMBERS

Like the hexadecimal system, the octal system provides a convenient way to express binary numbers and codes. However, it is used less frequently than hexadecimal in conjunction with computers and microprocessors to express binary quantities for input and output purposes.

The octal system is composed of eight digits, which are:

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

To count above 7, begin another column and start over: 10, 11, 12, 13, 14, 15, 16, 17, 20, 21 and so on. Counting in octal is similar to counting in decimal, except that the digits 8 and 9 are not used.

OCTAL-TO-DECIMAL CONVERSION

Since the octal number system has a base of eight, each successive digit position is an increasing power of eight, beginning in the right-most column with 8^o. The evaluation of an octal number in terms of its decimal equivalent is accomplished by multiplying each digit by its weight and summing the products.

Let's convert octal number 2374 in decimal number.

Weight	8^3	8^2	8^1	8^0
Octal number	**2**	**3**	**7**	**4**

$$2374 = (2 \times 8^3) + (3 \times 8^2) + (7 \times 8^1) + (4 \times 8^o) = 1276$$

DECIMAL-TO-OCTAL CONVERSION

A method of converting a decimal number to an octal number is the repeated division-by-8 method, which is similar to the method used in the conversion of decimal numbers to binary or to hexadecimal.

Let's convert the decimal number 359 to octal. Each successive division by 8 yields a remainder that becomes a digit in the equivalent octal number. The first remainder generated is the least significant digit (LSD).

359/8 = 44.875 **0.875 x 8 = 7 (LSD)**
44 /8 = 5.5 **0.5 x 8 = 4**
5/8 = 0.625 **0.625 x 8 = 5 (MSD)**
The number is 547.

OCTAL-TO-BINARY CONVERSION

Because each octal digit can be represented by a 3-bit binary number, it is very easy to convert from octal to binary.

Octal/Binary Conversion

Octal Digit	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Binary	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**

Let's convert the octal numbers 25 and 140.

Octal Digit	**2**	**5**		**1**	**4**	**0**
Binary	**010**	**101**		**001**	**100**	**000**

BINARY-TO-OCTAL CONVERSION

Conversion of a binary number to an octal number is the reverse of the octal-to-binary conversion**.**

Let's convert the following binary numbers to octal:

1 1 0	**1 0 1**		**1 0 1**	**1 1 1**	**0 0 1**	
6	**5**	**= 65**	**5**	**7**	**1**	**= 571**

GE2112 - FUNDAMENTALS OF COMPUTING AND PROGRAMMING

UNIT II

COMPUTER SOFTWARE

Computer Software –Types of Software – Software Development Steps – Internet Evolution - Basic Internet Terminology – Getting connected to Internet Applications.

COMPUTER SOFTWARE

Hardware

Hardware is the machine itself and its various individual equipment.
It includes all mechanical, electronic and magnetic devices such as monitor, printer, electronic circuit, floppy and hard disk.

Software

Software refers to the set of computer programs, which are used in applications and operating systems. It is the collection of programs, which increase the capabilities of the hardware. Software guides the computer at every step where to start and stop during a particular job. The process of software development is called *programming*.

TYPES OF SOFTWARE

Application Software:

Application Software is a set of programs for a specific application. Application software is useful for word processing, accounting, and producing statistical report, Graphics, Excel and Data Base. Programming languages COBOL, FORTRAN, C++, VB, VC, Java

Types of Application Software

Application software enables users to perform the activities and work that computers were designed for. The specific type of application used depends on the intended purpose, and there are application programs for almost every need. Three broad types of application software available for business users are individual, collaboration, and vertical programs (see Table 5-1)**.**

Individual application software refers to programs individuals use at work or at home. Examples include word processing, spreadsheet, database management, and desktop publishing programs.

Collaboration software (also called **groupware)** enables people at separate PC workstations to work together on a single document or project, such as designing a new automobile engine.

Vertical application software is a complete package of programs that work together to perform core business functions for a large organization. For example, a bank might have a mainframe computer at its corporate headquarters connected to conventional terminals in branch offices, where they are used by managers, tellers, loan officers, and other employees. All financial transactions are fed to the central computer for processing. The system then generates managers' reports, account statements, and other essential documents.

Other Application Software Models

Shareware

Shareware is software developed by an individual or software publisher who retains ownership of the product and makes it available for a small "contribution" fee. The voluntary fee normally entitles users to receive online or written product documentation and technical help.

Freeware

Freeware is software that is provided free of charge to anyone wanting to use it. Hundreds of freeware programs are available, many written by college students and professors who create programs as class projects or as part of their research.

Open Source Software

An open source software program is software whose programming code is owned by the original developer but made available free to the general public, who is encouraged to experiment with the software, make improvements, and share the improvements with the user community

Application Software for Individual Use

The thousands of application programs that individuals use to perform computing tasks at work and at home can be grouped into four types:

Productivity software
Software for household use
Graphics and multimedia software
Communication software

Productivity Software

Productivity software is designed to improve efficiency and performance on the job and at home, and is the largest category of application software for individual use. In-depth knowledge and skill in using productivity software applications can make a potential employee more valuable to a business, organization, or agency.

Word Processing

A word processing program can be used to create almost any kind of printed document. Word processors are the most widely used of all software applications because they are central to communication. Whatever the type of document created with a word processing program, the essential parts of the procedure remain the same:

Create (enter) text
Edit the text
Format the document
Save and print the file

Desktop Publishing

Desktop publishing (DTP) software allows users to create impressive documents that include text, drawings, photographs, and various graphics elements in full color. Professional-quality publications can be produced with DTP software. Textbooks such as this one may be designed and laid out with a desktop

publishing application such as *PageMaker*, *QuarkXpress*, or *Adobe InDesign*. Major word processors offer limited desktop publishing features sufficient for creating simple newsletters and brochures.

Spreadsheets

Spreadsheet software is an electronic version of the ruled worksheets accountants used in the past. Spreadsheet software provides a means of organizing, calculating, and presenting financial, statistical, and other numerical information. Businesses find spreadsheets particularly useful for evaluating alternative scenarios. The spreadsheet uses "what if" calculations to evaluate possibilities. By entering various data values and formulas into a spreadsheet, questions can be answered quickly and accurately.

For the individual user, spreadsheets fulfill many purposes, including:

Preparing and analyzing personal or business budgets
Reconciling checkbooks
Analyzing financial situations
Tracking and analyzing investments
Preparing personal financial statements
Estimating taxes

Database Management

In a computerized database system, data are stored in electronic form on a storage medium, such as hard or floppy disks or CDs. A **database** is a collection of data organized in one or more tables consisting of individual pieces of information, each located in a **field**, and a collection of related fields, each collection making up one **record** (see Figure 5-1). A commercial database program typically allows users to create a form for entering data. A user can design an electronic form to make entering information into the database easier. The information entered using such a form will become a record in a table. Users can add, remove, or change the stored data.

Presentation Graphics

Presentation graphics software allows users to create computerized slide shows that combine text, numbers, animation, graphics, sounds, and videos. A **slide** is an individual document that is created in presentation graphics software. A **slide show** may consist of any number of individual slides. For example, an instructor may use a slide show to accompany a lecture to make it more engaging and informative. Microsoft PowerPoint and Corel Presentations are two popular presentation software programs.

Software for Household Use

Numerous software applications designed for use in the household are available for purchase. Among the many products available are applications for managing personal finances, preparing tax returns, preparing legal documents, playing games, and education and reference.

Graphics and Multimedia Software

Graphics and multimedia software allows both professional and home users to work with graphics, video, and audio. A variety of applications software is focused in this area, including painting and drawing software, image-editing software, video and audio editing software, Web authoring software, and computer-aided design (CAD) software.

Communications Software

One of the major reasons people use computers is to communicate with others and to retrieve and share information. Communications software allows users to send and receive e-mail, browse and search the Web, engage in group communications and discussions, and participate in videoconferencing activities.

Automatic Multimedia Tagging Software
Advances in Speech Recognition Software
Pattern Recognition Software
Distributed Computing

System Software:

When you switch on the computer the programs written in ROM is executed which activates different units of your computer and makes it ready for you to work. This set of programs can be called system software. System software are general programs designed for performing tasks such as controlling all operations required to move data into and out of the computer System Software allows application packages to be run on the computer.Computer manufactures build and supply this system software with the computer system.

An **operating system** is the most important piece of software on a personal computer. The location of the operating system identifies the **boot drive** for the personal computer, which is typically the hard drive. Once started, the operating system manages the computer system and performs functions related to the input, processing, output, and storage of information, including:

Managing main memory, or RAM
Configuring and controlling peripheral devices
Managing essential file operations, including formatting or copying disks, and renaming or deleting files
Monitoring system performance
Providing a user interface

Windows

Windows 2000 Professional Windows 2000 Professional, introduced in late 1999, was designed for use with business computers and was the successor to Windows 98 for office environments. Incorporating the power of Windows NT, Windows 2000 Professional was used to link

Windows XP Professional Microsoft's **Windows XP** Professional was designed for the latest computers that are fast, powerful, and have lots of memory and hard disk space. It combined the more powerful features of Windows 2000 and Windows NT and included many cosmetic changes. Windows XP contained many new and improved features and was extremely user-friendly. Icons had a three-dimensional look, and provided impressive features for managing photo and music files. It was the first Windows version to be copy-protected.

Windows Vista Windows Vista, released in 2007, improves and expands on Windows XP's capabilities. It provides much more robust security features than any earlier version. The centerpiece of this security system is **User Access Control (UAC)**, a protection system that prompts the user for administrator-level credentials whenever an operation is attempted that might affect system stability or security in some way. Home networking is easier than ever with Windows Vista.

Macintosh Operating System

The Macintosh, the first commercial GUI, was originally released in 1984 and has been updated many times since. It included a virtual desktop, pull-down menus, dialog boxes, and icons representing common commands and programs. With its impressive graphics and ease of use, it quickly became the model for other GUIs.

OS/2

IBM's **OS/2** GUI operating system was the company's response to the popularity of Microsoft Windows and the Apple Mac OS. The latest version is called OS/2 Warp. In addition to running native application programs, OS/2 can also run programs written for DOS and Windows systems.

Linux

Linux is a UNIX-based operating system that runs on a number of computer platforms including PCs, servers, and handheld devices. The Linux kernel (the central module or basic part) was developed mainly by Linus Torvalds. Torvalds designed Linux as an **open-source software program**, which means that the developer retains ownership of the original programming code but makes it free to the general public, who is encouraged to experiment with the software, make improvements, and share the improvements with the entire user community. During recent years, many improvements and refinements have been made rendering Linux an extremely popular and functional operating system for both large and small computers.

Server Operating Systems

Some operating systems are designed specifically for use with local area networks, allowing multiple users to connect to the server and to share network resources such as files and peripheral devices such as printers. The kind of operating system selected for use with a network server depends on network architecture and processing requirements.

Novell Netware

NetWare, developed by Novell, Inc. during the 1980s, is a popular and widely used operating system for microcomputer-based local area networks. Network users have the option of working with or without network resources.

Windows

Microsoft's **Windows NT Server** was one of Microsoft's earlier entries into the client/server market. It supported the connection of various peripheral devices and multitasking operations in which networked computers could process applications at the same time. Windows NT Server was replaced by Windows 2000 Server.

UNIX

Developed in the early 1970s by programmers at Bell Laboratories, the **UNIX** operating system was originally designed for servers and large computer systems

Linux Linux (pronounced LIN-UKS) is one of the fastest-growing server operating systems. Linux is an open-source software program based on the UNIX operating system
Solaris Solaris is a Unix-based operating environment developed by Sun Microsystems. It was originally developed to run on Sun's SPARC workstations but now runs on many workstations from other manufacturers

Compiler: It is a program translator that translates the instruction of a higher level language to machine language

.

An interpreter is another type of program translator used for translating higher level language into machine language.It takes one statement of higher level languages, translate it into machine language and immediately execute it.

SOFTWARE DEVELOPMENT STEPS

Software Development Life Cycle

The product developed which achieves customer satisfaction is not done in a single step. It involves series of steps in a software development process. This is needed to develop quality products with error free products to achieve customer satisfaction. There are many models available in the software development process.

But majority of software development process follow the model named as software development life cycle. This software develop life cycle has number of steps in it. The below article describes about the software development life cycle and the steps involved into it.
Software development life cycle model is also called as waterfall model which is followed by majority of systems. This software development life cycle process has the following seven stages in it namely
1. System Requirements Analysis
2. Feasibility study
3. Systems Analysis and Design
4. Code Generation
5. Testing
6. Maintenance
7. Implementation

Let us discuss each of these to have an overview about teach of the following steps in software development life cycle.

1. System Requirements Analysis:
The first essential or vital thing required for any software development is system. Also the system requirement may vary based on the software product that is going to get developed. So a careful analysis has to be made about the system requirement needed for the development of the product. After the analysis and design of the system requirement phase the system required for the development would be complete and the concentration can be on the software development process.

2. Feasibility study:
After making an analysis in the system requirement the next step is to make analysis of the software requirement. In other words feasibility study is also called as software requirement analysis. In this phase development team has to make communication with customers and make analysis of their requirement and analyze the system. By making analysis this way it would be possible to make a report of identified area of problem. By making a detailed analysis on this area a detailed document or report is prepared in this phase which has details like project plan or schedule of the project, the cost estimated for developing and executing the system, target dates for each phase of delivery of system developed and so on. This phase is the base of software development process since further steps taken in software development life cycle would be based on the analysis made on this phase and so careful analysis has to be made in this phase.

3. Systems Analysis and Design:
This is an important phase in system development .Here analysis is made on the design of the system that is going to be developed. In other words database design, the design of the architecture chosen, functional specification design, low level design documents, high level design documents and so on takes place. Care

must be taken to prepare these design documents because the next phases namely the development phase is based on these design documents. If a well structured and analyzed design document is prepared it would reduce the time taken in the coming steps namely development and testing phases of the software development life cycle.

4. Code Generation:

This is the phase where actual development of the system takes place. That is based on the design documents prepared in the earlier phase code is written in the programming technology chosen. After the code is developed generation of code also takes place in this phase. In other words the code is converted into executables in this phase after code generation.

5. Testing:

A software or system which is not tested would be of poor quality. This is because this is the phase where system developed would be tested and reports are prepared about bugs or errors in system. To do this testing phase there are different levels and methods of testing like unit testing, system test and so on. Based on the need the testing methods are chosen and reports are prepared about bugs. After this process the system again goes to development phase for correction of errors and again tested. This process continues until the system is found to be error free. To ease the testing process debuggers or testing tools are also available.

To develop reliable and good quality Program/Software we need to follow the following 5 steps :

Requirement Specification.
Analysis.
Design.
Implementation.
Verification and testing.

INTERNET EVOLUTION

The Internet is a network of networks. Computer users on the Internet can contact one another anywhere in the world .In Internet a huge resource of information is accessible to people across the world .Information in every field starting from education, science, health, medicine, history, and geography to business, news, etc. can be retrieved through Internet .You can also download programs and software packages from anywhere in the **world .**In 1969 Department of Defense (DOD) of USA started a network called ARPANET (Advanced Research Projects Administration Network) . Around 1970, NSFNET (National Science Foundation Network) was created. With the advancement of modern communication facilities. By 1990 many computers were looking up to NSFNET giving birth to Internet .Internet is not a governmental organization. The ultimate authority of the Internet is the Internet Society. This is a voluntary membership organization whose purpose is to promote global information exchange. Internet has more than one million computers attached to it. Ten years of research brought Local Area Ethernet Networks (LANs) and workstations were developed to get connected to LAN. Computers connected to ARPANET used a standard or rule to communicate with each other with NCP (National Control Protocol). Protocol is a network term used to indicate the standard used by a network for communication. Rapid change in information technology suppressed NCP and brought TCP/IP (Transmission Control Protocol/Internet Protocol) in to the world of networking. The Internet is a rare example of a large democracy with no state of head, no official censors, no bosses, no board of directors. Nobody controls the Internet and in principle, any computer can speak to any other computer, as long as it obeys the technical rules of the TCP/IP protocol. This freedom of Internet helped it to move out of its original base in military and research institutions, into elementary and high schools, colleges, public libraries, commercial sectors.

BASIC INTERNET TERMINOLOGY

Blog - A **blog** is information that is instantly published to a Web site. Blog scripting allows someone to automatically post information to a Web site. The information first goes to a blogger Web site. Then the information is automatically inserted into a template tailored for your Web site.

Bookmark - a way of storing your favorite sites on the Internet. Browsers like Netscape or Internet Explorer let you to categorize your bookmarks into folders.

Browser - A software program that allows users to access the Internet. Examples:

Non-graphical a user interface for computers which allows you to read plain text, not pictures, sound, or video, on the Internet. It is strictly text based, non-Windows, and does not place high memory demands on your computer. An example is **lynx** .(http://lynx.browser.org/)

Graphical a user interface for computers which enables people to see color, graphics, and hear sound and see video, available on Internet sites. These features are usually designated by underlined text, a change of color, or other distinguishing feature; sometimes the link is not obvious, for example, a picture with no designated characteristic. Examples are **Netscape** and **Internet Explorer**.

CGI (**C**ommon **G**ateway **I**nterface script) - a specificiation for transferring information between a Web server and a CGI program, designed to receive and and return data. The script can use a variety of languages such as C, Perl, Java, or Visual Basic. Many html pages that contain forms use a cgi program to process the data submitted by users/clients.

Chat - real-time, synchronous, text-based communication via computer.

Cookie - Information (in this case URLs, Web addresses) created by a Web server and stored on a user's computer. This information lets Web sites the user visits to keep of a user's browsing pattterns and preferences. People can set up their browsers to accept or not accept cookies.

Domain Name - A method of identifying computer addresses. Your e-mail address has a domain address. If you have an "edu" at the end of your e-mail address that means your account is affiliated with an educational institution. A "com" extension means you have a business account. A government account has a .gov suffix.

FTP - Using file transfer protocol software to receive from upload) or send to (download) files (text, pictures, spreadsheets, etc.) from one computer/server to another.

Home page - Generally the first page retrieved when accessing a Web site. Usually a "home" page acts as the starting point for a user to access information on the site. The "home" page usually has some type of table of contents for the rest of the site information or other materials. When creating Web pages, the "home" page has the filename "index.html," which is the default name. The "index" page automatically opens up as the "home" page.

HTML - A type of text code in Hypertext Markup Language which, when embedded in a document, allows that document to be read and distributed across the Internet.

HTTP - The hypertext transfer protocol (http) that enables html documents to be read on the Internet.

Hypertext - text that is non-sequential, produced by writing in HTML (**H**yper**t**ext **M**arkup **L**anguage) language. This HTML coding allows the information (text, graphics, sound, video) to be accessed using HTTP (Hypertext Transfer Protocol).

Hyperlink - Text, images, graphics that, when clicked with a mouse (or activated by keystrokes) will connect the user to a new Web site. The link is usually obvious, such as underlined text or a "button" of some type, but not always.

Instant Messaging (IM) - a text-based computer conference over the Internet between two or more people who must be online at the same time. When you send an IM the receiver is instantly notified that she/he has a message.

IP Address - (Internet Protocol) The number or name of the computer from which you send and receive information on the Internet.

Modem - A device that connects your computer to the Internet, when you are not connected via a LAN (local area network, such as at work or on a campus.) Most people connect to a modem when using a home computer. The modem translates computer signals to analog signals which are sent via phone lines. The telephone "speaks" to the computer/server which provides your Internet access.

URL - A universal resource locator (a computer address) that identifies the location and type of resource on the Web. A URL generally starts with "http."
Intranet: It is a relatively smaller private network that uses the Internet protocols and connectivity. It is an extension of the Internet and is privately used by organizations.

Web Server: A web server is a computer program that accepts HTTP requests from web clients and provides them with HTTP responses.

IP Address: It is a way of numerically identifying an entity on a computer network. The original addressing system known as IPv4, used 32 bit addresses. With the growth of the Internet, IPv6 came to be used wherein the addresses are composed of 128 bits. You might want to know how to find your IP address.

Internet Service Provider: A company, which provides users with an access to the Internet, is known as an Internet service provider or Internet access provider. ISP, as it is called, offers email accounts and other services like remote storage of files for its customers. Here is a word about choosing a cheap ISP.

Internet Address

Every page on the Internet has a unique address. This address is used get the web page for user from Internet. The address on the Internet is known as URL (Uniform Resource Locator). A typical Internet address or URL would look like; http://www.mans.edu.eg/facscim/arabic/. The URL contains the components that specify the protocol, server, and pathname of an item

URL parts

The protocol is followed by a colon (**http:**),
The server is preceded by two slashes (**//www.mnmjec.edu.**)
Each segment of the pathname is preceded by a single slash /facscim/ /english/Tables/Default.htm).
A protocol is set of rules that tells the computer know how to interpret the information at that address
The first component, the protocol, defines the manner for interpreting computer information.
Many Internet pages use **HTTP** (**H**yper**T**ext **T**ransfer **P**rotocol).Other common Internet protocols that one might come across are **FTP** (File Transfer Protocol),

NEWS (Usenet news groups protocol), and **GOPHER** (an alternative transfer protocol). Gopher protocol is mostly out of date now. The second component, of the address is the server (**www.mnmjec.edu**), identifies the computer system that stores the information you seek and is always preceded by two slashes.

A server is a computer that has information stored on it and sends it to the client, when a request is made. Each server on the Internet has a unique address name whose text refers to the organization maintaining the server. Most of the Web pages will have **.htm** or **.html** as their secondary or extension name.

GETTING CONNECTED TO INTERNET APPLICATIONS

Types of Internet Connections

There are two main ways for users to connect to the Internet: through dial-up access or by using a LAN connection.

Dial-up Access Dial-up access allows access to the Internet over a standard telephone line by using a computer and a modem to dial into an ISP or VAN connection. Dial-up access is a feature typically included with the software provided by an ISP. Using a regular telephone line is usually the slowest telecommunications medium for setting up an individual Internet account through a dial-up ISP.

Local Area Network (LAN) Connection LAN connections provide faster and more direct Internet access by connecting users to an ISP on a direct wire, at speeds 30 or more times faster than can be achieved through a dial-up modem. Because they are more expensive than dial-up access, LAN connections are more commonly found in the workplace. Despite the increased cost there are approximately forty million LAN users in the United States using cable and DSL connections to connect from their homes.

Cable Modem Television cable companies provide a special modem and software for broadband (high-speed) Internet access. This service offers the advantage of simultaneous Web access and telephone calls, but is not available everywhere. In addition, the service slows down as more subscribers sign up in a neighborhood or location. The cost is about $50 monthly, plus a possible installation fee.

Digital Subscriber Line (DSL) DSL Internet service is as fast as cable modem and provides simultaneous Web access and telephone use, but the service is usually available only to users within three miles of the telephone carrier's central switching office. The line is dedicated to one household, and is not shared with neighbors. DSL service costs around $50 monthly, plus an installation fee.

Wireless The fastest growing segment of Internet service involves wireless connections to the Internet. Thousands of Wireless "hot-spot" portals are springing up, allowing access in public places and even aboard airplanes.

(i) Gateway Access

Gateway Access is also known as Level-One connection. It is the access to the Internet from a network, which is not on the Internet. The gateway allows the two different types of networks to "talk" to each other. But the users of the Gateway Internet have limited access to the Internet. They might not be able to use all the tools available on Internet. The local Internet Service Provider (ISP) normally defines this limitation.

Dial-up Connection

'Dial-up' connection is also known as Level Two connection. This provides connection to Internet through a dial-up terminal connection. The computer, which provides Internet access is known as 'Host' and the computer that receives the access, is 'Client' or 'Terminal'. The client computer uses modem to access a "host" and acts as if it is a terminal directly connected to that host. This type of connection is also known as 'Remote Modem Access' connection. And the host to which the client gets connected is actually connected to the Internet by a full time connection (See Leased Connection). In dial-up connection to Internet, Host carries all the command that are typed on a client machine and forward them to Internet. It also receives the

data or information from the Internet on behalf of the 'Client' and passes it to them. The client computer acts as a 'dumb' terminal connected to remote host. This type of connection can further be divided into two categories.

Shell Connection

In this type of Internet Connection, the user will get only textual matter of a Web Page. This connection does not support Graphics display. However the user will be able to surf the Internet, do FTP, receive mail. Shell Accounts were the only type of Internet access available for many years before the Internet entered in to the world of graphics and became more users friendly.

TCP/IP Connection

Today's graphical World Wide Web browsers provide easier access with multimedia sound and pictures. The major difference between Shell and TCP/IP account is that, Shell account can only display text and does not support graphics display, whereas TCP/IP can display both. Hence it is more popular Internet connection. Shell accounts are slowly phasing out from the Internet scenario.

To access any of these dial-up accounts you need the followings;

Computer, WebTV, personal digital assistant (PDA), or Web phone
Dial-up modem, digital subscriber line (DSL) modem, or cable modem
Telephone line or cable connection
Telecommunications software
Web browser
An account with an Internet Service Provider (ISP)

Leased Connection

Leased connection is also known as direct Internet access or Level Three connection. It is the secure, dedicated and most expensive, level of Internet connection. With leased connection, your computer is dedicatedly and directly connected to the Internet using high-speed transmission lines. It is on-line twenty-four hours a day, seven days a week. Leased Internet connections are limited to large corporations and universities who could afford the cost.

Newer Internet Applications

Not only does the content of the Internet change daily, but they very way in which the Internet is used and understood as a communication medium is constantly evolving.

Peer-to-Peer File Sharing

Peer-to-Peer (P2P) file sharing allows people to download material directly from other users' hard drives, rather than from files located on Web servers. Napster is famous for pioneering P2P file sharing.

Internet Telephony

Internet telephony is another increasingly popular way to use the Internet. By using this technology, also called **Voice over IP (VoIP)**, two or more users with sufficiently good connections can use the Internet to make telephone calls around the world. Once their voices are digitized and broken down into packets, they can be transmitted anywhere, just like any other form of data. There are no long distance telephone charges, and users only pay their normal ISP connection fees.

Streaming Audio and Video

An alternative to downloading a piece of music or video is to access it using **streaming techniques** (also known as webcasting). Streaming sends a continuous stream of data to the receiving computer, where it is immediately displayed. Old data is erased as new data arrives.

Webcams

Tiny video cameras called **webcams** allow conversations over the Web through live video transmission. Often mounted on top of a computer monitor, the cameras automatically create and transmit video to the PC. Despite the fact that the images are a bit grainy and jerky, millions are in use. As the technology improves, their popularity is sure to increase even further.

Audio Mail

Audio mail is a fledgling type of electronic mail that allows people to transmit messages by voice. As with e-mail, attachments can be included. The technology can be compared to voice mail, without telephone charges.

GE2112 - FUNDAMENTALS OF COMPUTING AND PROGRAMMING

UNIT III

PROBLEM SOLVING AND OFFICE APPLICATION SOFTWARE

Planning the Computer Program – Purpose – Algorithm – Flow Charts – Pseudocode -Application Software Packages- Introduction to Office Packages (not detailed commands for examination).

PLANNING THE COMPUTER PROGRAM

The Programming Process - Purpose

- **Understand the problem**
  - Read the problem statement
  - Question users
  - Inputs required
  - Outputs required
  - Special formulas
  - Talk to users
- **Plan the logic**
  - Visual Design Tools
    - Input record chart
    - Printer spacing chart
    - Hierarchy chart
    - Flowchart
  - Verbal Design Tools
    - Narrative Description
    - Pseudocode
- **Code the program**
  - Select an appropriate programming language
  - Convert flowchart and/or Pseudocode instructions into programming language statements
- **Test the program**
  - Syntax errors
  - Runtime errors
  - Logic errors
  - Test Data Set

- **Implement the program**
  - Buy hardware
  - Publish software
  - Train users
  - Implementation Styles
    - Crash
    - Pilot
    - Phased
    - Dual
- **Maintain the program**
  - Maintenance programmers
  - Legacy systems
  - Up to 85% of IT department budget

ALGORITHM

- **Algorithm**
  - Set of step-by-step instructions that perform a specific task or operation
  - "Natural" language NOT programming language
- **Pseudocode**
  - Set of instructions that mimic programming language instructions
- **Flowchart**
  - Visual program design tool
  - "Semantic" symbols describe operations to be performed

FLOWCHARTS

Definitions:

A flowchart is a schematic representation of an algorithm or a stepwise process, showing the steps as boxes of various kinds, and their order by connecting these with arrows. Flowcharts are used in designing or documenting a process or program.[

A flow chart, or flow diagram, is a graphical representation of a process or system that details the sequencing of steps required to create output.

A flowchart is a picture of the separate steps of a process in sequential order.

Types:

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, such as the example in the figure entitled High-Level Flowchart of Prenatal Care. 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.

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. For example, the Detailed Flowchart of Patient Registration reveals the delays that result when the record clerk and clinical officer are not available to assist clients.

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. See the Deployment of Matrix Flowchart.

ADVANTAGES OF USING FLOWCHARTS

The benefits of flowcharts are as follows:

1. Communication: Flowcharts are better way of communicating the logic of a system to all concerned.
2. Effective analysis: With the help of flowchart, problem can be analysed in more effective way.
3. Proper documentation: Program flowcharts serve as a good program documentation, which is needed for various purposes.
4. Efficient Coding: The flowcharts act as a guide or blueprint during the systems analysis and program development phase.
5. Proper Debugging: The flowchart helps in debugging process.
6. Efficient Program Maintenance: The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part

Advantages

Logic Flowcharts are easy to understand. They provide a graphical representation of actions to be taken.

Logic Flowcharts are well suited for representing logic where there is intermingling among many actions.

Disadvantages

Logic Flowcharts may encourage the use of GoTo statements leading to software design that is unstructured with logic that is difficult to decipher.

Without an automated tool, it is time-consuming to maintain Logic Flowcharts.

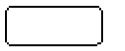
Logic Flowcharts may be used during detailed logic design to specify a module. However, the presence of decision boxes may encourage the use of GoTo statements, resulting in software that is not structured. For this reason, Logic Flowcharts may be better used during Structural Design

LIMITATIONS OF USING FLOWCHARTS

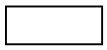
1. Complex logic: Sometimes, the program logic is quite complicated. In that case, flowchart becomes complex and clumsy.
2. Alterations and Modifications: If alterations are required the flowchart may require re-drawing completely.
3. Reproduction: As the flowchart symbols cannot be typed, reproduction of flowchart becomes a problem.
4. The essentials of what is done can easily be lost in the technical details of how it is done.

GUIDELINES FOR DRAWING A FLOWCHART

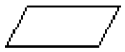
Flowcharts are usually drawn using some standard symbols; however, some special symbols can also be developed when required. Some standard symbols, which are frequently required for flowcharting many computer programs.



Start or end of the program



Computational steps or processing function of a program

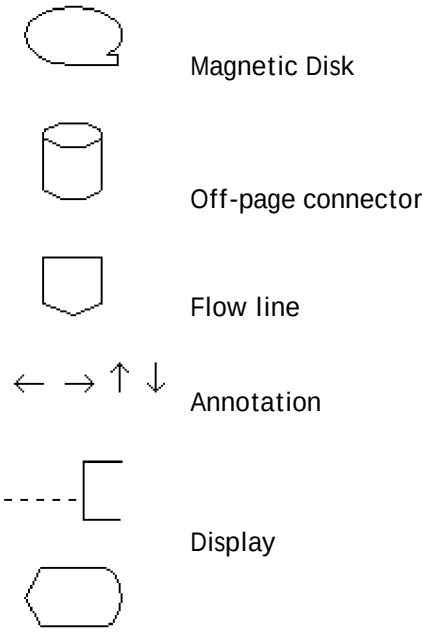


Input or output operation

Decision making and branching

Connector or joining of two parts of program

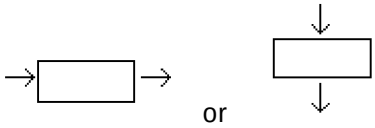
Magnetic Tape



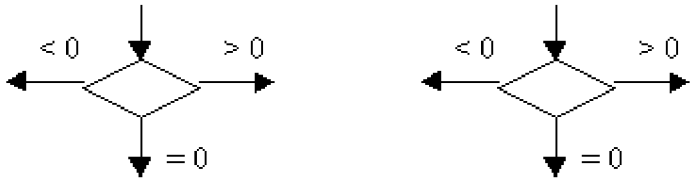
Flowchart Symbols

The following are some guidelines in flowcharting:

a. In drawing a proper flowchart, all necessary requirements should be listed out in logical order.
b. The flowchart should be clear, neat and easy to follow. There should not be any room for ambiguity in understanding the flowchart.
c. The usual direction of the flow of a procedure or system is from left to right or top to bottom.
d. Only one flow line should come out from a process symbol.

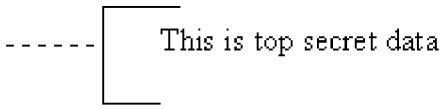


e. Only one flow line should enter a decision symbol, but two or three flow lines, one for each possible answer, should leave the decision symbol.



f. Only one flow line is used in conjunction with terminal symbol.

g. Write within standard symbols briefly. As necessary, you can use the annotation symbol to describe data or computational steps more clearly.

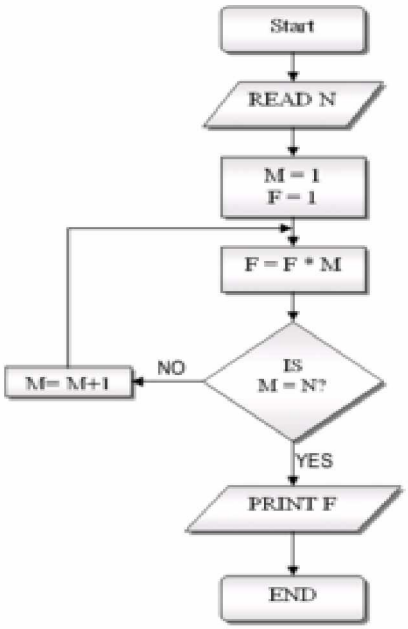


h. If the flowchart becomes complex, it is better to use connector symbols to reduce the number of flow lines. Avoid the intersection of flow lines if you want to make it more effective and better way of communication.
i. Ensure that the flowchart has a logical *start* and *finish.*
j. It is useful to test the validity of the flowchart by passing through it with a simple test data.

Examples

Sample flowchart

A flowchart for computing factorial N (N!) Where N! = 1 * 2 * 3 *...* N. This flowchart represents a "loop and a half" — a situation discussed in introductory programming textbooks that requires either a duplication of a component (to be both inside and outside the loop) or the component to be put inside a branch in the loop



Sample Pseudocode

```
ALGORITHM Sample
        GET Data
        WHILE There Is Data
                DO Math Operation
                GET Data
        END WHILE
END ALGORITHM
```

APPLICATION SOFTWARE PACKAGES

Shareware

Shareware is software developed by an individual or software publisher who retains ownership of the product and makes it available for a small "contribution" fee. The voluntary fee normally entitles users to receive online or written product documentation and technical help.

Freeware

Freeware is software that is provided free of charge to anyone wanting to use it. Hundreds of freeware programs are available, many written by college students and professors who create programs as class projects or as part of their research.

Open Source Software

An open source software program is software whose programming code is owned by the original developer but made available free to the general public, who is encouraged to experiment with the software, make improvements, and share the improvements with the user community

Application Software for Individual Use

The thousands of application programs that individuals use to perform computing tasks at work and at home can be grouped into four types:

Productivity software
Software for household use
Graphics and multimedia software
Communication software

INTRODUCTION TO OFFICE PACKAGES

Word Processing

A word processing program can be used to create almost any kind of printed document. Word processors are the most widely used of all software applications because they are central to communication. Whatever the type of document created with a word processing program, the essential parts of the procedure remain the same:

Create (enter) text
Edit the text
Format the document
Save and print the file

Spreadsheets

Spreadsheet software is an electronic version of the ruled worksheets accountants used in the past. Spreadsheet software provides a means of organizing, calculating, and presenting financial, statistical, and other numerical information. Businesses find spreadsheets particularly useful for evaluating alternative scenarios. The spreadsheet uses "what if" calculations to evaluate possibilities. By entering various data values and formulas into a spreadsheet, questions can be answered quickly and accurately.

For the individual user, spreadsheets fulfill many purposes, including:

Preparing and analyzing personal or business budgets
Reconciling checkbooks
Analyzing financial situations
Tracking and analyzing investments
Preparing personal financial statements
Estimating taxes

GE2112 - FUNDAMENTALS OF COMPUTING AND PROGRAMMING

UNIT IV

INTRODUCTION TO C

Overview of C – Constants, Variables and Data Types – Operators and Expressions – Managing Input and Output operators – Decision Making - Branching and Looping.

OVERVIEW OF C

As a programming language, C is rather like Pascal or Fortran.. Values are stored in variables. Programs are structured by defining and calling functions. Program flow is controlled using loops, if statements and function calls. Input and output can be directed to the terminal or to files. Related data can be stored together in arrays or structures.

Of the three languages, C allows the most precise control of input and output. C is also rather more terse than Fortran or Pascal. This can result in short efficient programs, where the programmer has made wise use of C's range of powerful operators. It also allows the programmer to produce programs which are impossible to understand. Programmers who are familiar with the use of pointers (or indirect addressing, to use the correct term) will welcome the ease of use compared with some other languages. Undisciplined use of pointers can lead to errors which are very hard to trace. This course only deals with the simplest applications of pointers.

A Simple Program

The following program is written in the C programming language.

```
#include <stdio.h>

main()
{
        printf("Programming in C is easy.\n");
}
```

A NOTE ABOUT C PROGRAMS

In C, lowercase and uppercase characters are very important! All commands in C must be lowercase. The C programs starting point is identified by the word

main()

This informs the computer as to where the program actually starts. The brackets that follow the keyword *main* indicate that there are no arguments supplied to this program (this will be examined later on).

The two braces, { and }, signify the begin and end segments of the program. The purpose of the statment

include <stdio.h>

is to allow the use of the *printf* statement to provide program output. Text to be displayed by *printf()* must be enclosed in double quotes. The program has only one statement

printf("Programming in C is easy.\n");

printf() is actually a function (procedure) in C that is used for printing variables and text. Where text appears in double quotes "", it is printed without modification. There are some exceptions however. This has to do with the \

and % characters. These characters are modifier's, and for the present the \ followed by the n character represents a newline character. Thus the program prints

```
Programming in C is easy.
```

and the cursor is set to the beginning of the next line. As we shall see later on, what follows the \ character will determine what is printed, ie, a tab, clear screen, clear line etc. Another important thing to remember is that all C statements are terminated by a semi-colon ;

Summary of major points:

- program execution begins at *main()*
- keywords are written in lower-case
- statements are terminated with a semi-colon
- text strings are enclosed in double quotes
- C is case sensitive, use lower-case and try not to capitalise variable names
- \n means position the cursor on the beginning of the next line
- *printf()* can be used to display text to the screen
- The curly braces {} define the beginning and end of a program block.

BASIC STRUCTURE OF C PROGRAMS

C programs are essentially constructed in the following manner, as a number of well defined sections.

```
/* HEADER SECTION                    */
/* Contains name, author, revision number*/

/* INCLUDE SECTION                   */
/* contains #include statements      */

/* CONSTANTS AND TYPES SECTION       */
/* contains types and #defines       */

/* GLOBAL VARIABLES SECTION          */
/* any global variables declared here */

/* FUNCTIONS SECTION                 */
/* user defined functions            */

/* main() SECTION                    */

int main()
{

}
```

VARIABLE

User defined variables must be declared before they can be used in a program. Variables must begin with a character or underscore, and may be followed by any combination of characters, underscores, or the digits 0 - 9.

LOCAL AND GLOBAL VARIABLES

Local

These variables only exist inside the specific function that creates them. They are unknown to other functions and to the main program. As such, they are normally implemented using a stack. Local variables cease to exist once the function that created them is completed. They are recreated each time a function is executed or called.

Global

These variables can be accessed (ie known) by any function comprising the program. They are implemented by associating memory locations with variable names. They do not get recreated if the function is recalled.

DEFINING GLOBAL VARIABLES

/ Demonstrating Global variables */*

Example:

```
#include <stdio.h>
int add_numbers( void );              /* ANSI function prototype */

/* These are global variables and can be accessed by functions from this point on */
int  value1, value2, value3;

int add_numbers( void )
{
        auto int result;
        result = value1 + value2 + value3;
        return result;
}

main()
{
        auto int result;

        result = add_numbers();
        printf("The sum of %d + %d + %d is %d\n",
                value1, value2, value3, final_result);
}
```

The scope of global variables can be restricted by carefully placing the declaration. They are visible from the declaration until the end of the current source file.

Example:

```
#include <stdio.h>
void no_access( void ); /* ANSI function prototype */
void all_access( void );

static int n2;                  /* n2 is known from this point onwards */

void no_access( void )
{
        n1 = 10;        /* illegal, n1 not yet known */
        n2 = 5;         /* valid */
}

static int n1;                  /* n1 is known from this point onwards */

void all_access( void )
{
        n1 = 10;        /* valid */
        n2 = 3;         /* valid */
}
```

AUTOMATIC AND STATIC VARIABLES

C programs have a number of segments (or areas) where data is located. These segments are typically,

_DATA Static data
_BSS Uninitialized static data, zeroed out before call to main()
_STACK Automatic data, resides on stack frame, thus local to functions
_CONST Constant data, using the ANSI C keyword const

The use of the appropriate keyword allows correct placement of the variable onto the desired data segment.

Example:

```
/* example program illustrates difference between static and automatic variables */
#include <stdio.h>
void demo( void );              /* ANSI function prototypes */

void demo( void )
{
        auto int avar = 0;
        static int svar = 0;

        printf("auto = %d, static = %d\n", avar, svar);
        ++avar;
        ++svar;
}


main()
{
        int i;
```

```
        while( i < 3 ) {
                demo();
                i++;
        }
}
```

AUTOMATIC AND STATIC VARIABLES

Example:

```
/* example program illustrates difference between static and automatic variables */
#include <stdio.h>
void demo( void );          /* ANSI function prototypes */

void demo( void )
{
        auto int avar = 0;
        static int svar = 0;

        printf("auto = %d, static = %d\n", avar, svar);
        ++avar;
        ++svar;
}


main()
{
        int i;

        while( i < 3 ) {
                demo();
                i++;
        }
}
```

Program output

```
auto = 0, static = 0
auto = 0, static = 1
auto = 0, static = 2
```

The basic format for declaring variables is

data_type var, var, ... ;

where *data_type* is one of the four basic types, an *integer*, *character*, *float*, or *double* type.

Static variables are created and initialized once, on the first call to the function. Subsequent calls to the function do not recreate or re-initialize the static variable. When the function terminates, the variable still exists on the _DATA segment, but cannot be accessed by outside functions.
Automatic variables are the opposite. They are created and re-initialized on each entry to the function. They disappear (are de-allocated) when the function terminates. They are created on the _STACK segment.

DATA TYPES AND CONSTANTS

The four basic data types are

- **INTEGER**
  These are whole numbers, both positive and negative. Unsigned integers (positive values only) are supported. In addition, there are short and long integers.

  The keyword used to define integers is,

  int

  An example of an integer value is 32. An example of declaring an integer variable called **sum** is,

  int sum;

  sum = 20;

- **FLOATING POINT**
  These are numbers which contain fractional parts, both positive and negative. The keyword used to define float variables is,

- *float*

  An example of a float value is 34.12. An example of declaring a float variable called **money** is,

```
float money;
money = 0.12;
```

- **DOUBLE**
  These are exponetional numbers, both positive and negative. The keyword used to define double variables is,

- *double*

  An example of a double value is 3.0E2. An example of declaring a double variable called **big** is,

```
double big;
big = 312E+7;
```

- **CHARACTER**
  These are single characters. The keyword used to define character variables is,
-
- *char*

An example of a character value is the letter **A**. An example of declaring a character variable called **letter** is,

```
    char letter;
letter = 'A';
```

Note the assignment of the character *A* to the variable *letter* is done by enclosing the value in **single quotes**. Remember the golden rule: Single character - Use single quotes.

Sample program illustrating each data type

Example:

```
#include < stdio.h >

main()
{
        int   sum;
        float money;
        char  letter;
        double pi;

        sum = 10;           /* assign integer value */
        money = 2.21;               /* assign float value */
        letter = 'A';               /* assign character value */
        pi = 2.01E6;                /* assign a double value */

        printf("value of sum = %d\n", sum );
        printf("value of money = %f\n", money );
        printf("value of letter = %c\n", letter );
        printf("value of pi = %e\n", pi );
}
```

Sample program output

```
value of sum = 10
value of money = 2.210000
value of letter = A
value of pi = 2.010000e+06
```

INITIALISING DATA VARIABLES AT DECLARATION TIME

In C variables may be initialised with a value when they are declared. Consider the following declaration, which declares an integer variable *count* which is initialised to 10.

```
int  count = 10;
```

SIMPLE ASSIGNMENT OF VALUES TO VARIABLES

The = operator is used to assign values to data variables. Consider the following statement, which assigns the value 32 an integer variable *count*, and the letter **A** to the character variable *letter*

```
count = 32;
letter = 'A'
```

Variable Formatters

%d	decimal integer
%c	character
%s	string or character array
%f	float
%e	double

HEADER FILES

Header files contain definitions of functions and variables which can be incorporated into any C program by using the pre-processor *#include* statement. Standard header files are provided with each compiler, and cover a range of areas, string handling, mathematical, data conversion, printing and reading of variables.

To use any of the standard functions, the appropriate header file should be included. This is done at the beginning of the C source file. For example, to use the function *printf()* in a program, the line

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

should be at the beginning of the source file, because the definition for *printf()* is found in the file *stdio.h* All header files have the extension .h and generally reside in the /include subdirectory.

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

The use of angle brackets <> informs the compiler to search the compilers include directory for the specified file. The use of the double quotes "" around the filename inform the compiler to search in the current directory for the specified file.

OPERATORS AND EXPRESSIONS

An expression is a sequence of operators and operands that specifies computation of a value, or that designates an object or a function, or that generates side effects, or that performs a combination thereof.

ARITHMETIC OPERATORS:

The symbols of the arithmetic operators are:-

Operation	Operator	Comment	Value of Sum before	Value of sum after
Multiply	*	sum = sum * 2;	4	8
Divide	/	sum = sum / 2;	4	2
Addition	+	sum = sum + 2;	4	6
Subtraction	-	sum = sum -2;	4	2
Increment	++	++sum;	4	5
Decrement	--	--sum;	4	3
Modulus	%	sum = sum % 3;	4	1

Example:

```
#include <stdio.h>

        main()
        {
                int sum = 50;
                float modulus;

                modulus = sum % 10;
                printf("The %% of %d by 10 is %f\n", sum, modulus);
        }
```

PRE/POST INCREMENT/DECREMENT OPERATORS

PRE means do the operation first followed by any assignment operation. POST means do the operation after any assignment operation. Consider the following statements

```
    ++count;        /* PRE Increment, means add one to count */
  count++;        /* POST Increment, means add one to count */
```

Example:

```
#include <stdio.h>

        main()
        {
                int count = 0, loop;

                loop = ++count;  /* same as count = count + 1; loop = count;  */
                printf("loop = %d, count = %d\n", loop, count);

                loop = count++;  /* same as loop = count;  count = count + 1;  */
                printf("loop = %d, count = %d\n", loop, count);
        }
```

If the operator precedes (is on the left hand side) of the variable, the operation is performed first, so the statement

```
loop = ++count;
```

really means increment *count* first, then assign the new value of *count* to *loop*.

THE RELATIONAL OPERATORS

These allow the comparison of two or more variables.

==	*equal to*
!=	*not equal*
<	*less than*
<=	*less than or equal to*
>	*greater than*
>=	*greater than or equal to*

Example:

```
#include <stdio.h>

        main()   /* Program introduces the for statement, counts to ten */
        {
                int  count;

                for( count = 1; count <= 10; count = count + 1 )
                        printf("%d ", count );

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

RELATIONALS (AND, NOT, OR, EOR)

Combining more than one condition

These allow the testing of more than one condition as part of selection statements. The symbols are

LOGICAL AND &&

Logical and requires all conditions to evaluate as TRUE (non-zero).

LOGICAL OR ||

Logical or will be executed if any ONE of the conditions is TRUE (non-zero).

LOGICAL NOT !

logical not negates (changes from TRUE to FALSE, vsvs) a condition.

LOGICAL EOR ^

Logical eor will be excuted if either condition is TRUE, but NOT if they are all true.

Example:

The following program uses an *if* statement with logical AND to validate the users input to be in the range 1-10.

```
#include <stdio.h>

        main()
        {
                int number;
                int valid = 0;

                while( valid == 0 ) {
                        printf("Enter a number between 1 and 10 -->");
                        scanf("%d", &number);
                        if( (number < 1 ) || (number > 10) ){
                                printf("Number is outside range 1-10. Please re-enter\n");
                                valid = 0;
                        }
                        else
                                valid = 1;
                }
                printf("The number is %d\n", number );
        }
```

Example:

NEGATION

```
        #include <stdio.h>

        main()
        {
                int flag = 0;
                if( ! flag ) {
                        printf("The flag is not set.\n");
```

```
                    flag = ! flag;
            }
            printf("The value of flag is %d\n", flag);
    }
```

Example:

Consider where a value is to be inputted from the user, and checked for validity to be within a certain range, lets say between the integer values 1 and 100.

```
#include <stdio.h>

main()
{
        int number;
        int valid = 0;

        while( valid == 0 ) {
                printf("Enter a number between 1 and 100");
                scanf("%d", &number );
                if( (number < 1) || (number > 100) )
                        printf("Number is outside legal range\n");
                else
                        valid = 1;
        }
        printf("Number is %d\n", number );
}
```

THE CONDITIONAL EXPRESSION OPERATOR or TERNARY OPERATOR

This conditional expression operator takes THREE operators. The two symbols used to denote this operator are the ? and the :. The first operand is placed before the ?, the second operand between the ? and the :, and the third after the :. The general format is,

condition ? expression1 : expression2

If the result of condition is TRUE (non-zero), expression1 is evaluated and the result of the evaluation becomes the result of the operation. If the condition is FALSE (zero), then expression2 is evaluated and its result becomes the result of the operation. An example will help,

*s = (x < 0) ? -1 : x * x;*

If x is less than zero then s = -1
*If x is greater than zero then s = x * x*

Example:

```
#include <stdio.h>

        main()
        {
                int input;

                printf("I will tell you if the number is positive, negative or zero!"\n");
                printf("please enter your number now--->");
                scanf("%d", &input );
                (input < 0) ? printf("negative\n") : ((input > 0) ? printf("positive\n") : printf("zero\n"));


}
```

BIT OPERATIONS

C has the advantage of direct bit manipulation and the operations available are,

Operation	Operator	Comment	Value of Sum before	Value of sum after
AND	&	sum = sum & 2;	4	0
OR	|	sum = sum | 2;	4	6
Exclusive OR	^	sum = sum ^ 2;	4	6
1's Complement	~	sum = ~sum;	4	-5
Left Shift	<<	sum = sum << 2;	4	16
Right Shift	>>	sum = sum >> 2;	4	0

Example:

```
/* Example program illustrating << and >> */
        #include <stdio.h>

        main()
        {
                int  n1 = 10, n2 = 20, i = 0;

                i = n2 << 4;  /* n2 shifted left four times */
                printf("%d\n", i);
                i = n1 >> 5;  /* n1 shifted right five times */
                printf("%d\n", i);
        }
```

Example:

```
        /* Example program using EOR operator  */
        #include <stdio.h>
```

```
main()
{
        int  value1 = 2, value2 = 4;

        value1 ^= value2;
        value2 ^= value1;
        value1 ^= value2;
        printf("Value1 = %d, Value2 = %d\n", value1, value2);
}
```

Example:

```
/* Example program using AND operator  */
#include <stdio.h>

main()
{
        int  loop;

        for( loop = 'A'; loop <= 'Z'; loop++ )
                printf("Loop = %c, AND 0xdf = %c\n", loop, loop & 0xdf);
}
```

MANAGING INPUT AND OUTPUT OPERATORS

Printf ():

printf() is actually a function (procedure) in C that is used for printing variables and text. Where text appears in double quotes "", it is printed without modification. There are some exceptions however. This has to do with the \ and % characters. These characters are modifier's, and for the present the \ followed by the n character represents a newline character.

Example:

```
#include <stdio.h>

        main()
        {
                printf("Programming in C is easy.\n");
                printf("And so is Pascal.\n");
        }
@ Programming in C is easy.
        And so is Pascal.
```

FORMATTERS for *printf* are,

Cursor Control Formatters

\n	newline
\t	tab

\r	carriage return
\f	form feed
\v	vertical tab

Scanf ():

Scanf () is a function in C which allows the programmer to accept input from a keyboard.

Example:

```
#include <stdio.h>

        main()    /* program which introduces keyboard input */
        {
                  int  number;

                  printf("Type in a number \n");
                  scanf("%d", &number);
                  printf("The number you typed was %d\n", number);
        }
```

FORMATTERS FOR scanf()

The following characters, after the % character, in a scanf argument, have the following effect.

d	read a decimal integer
o	read an octal value
x	read a hexadecimal value
h	read a short integer
l	read a long integer
f	read a float value
e	read a double value
c	read a single character
s	read a sequence of characters
[...]	Read a character string. The characters inside the brackets

ACCEPTING SINGLE CHARACTERS FROM THE KEYBOARD

Getchar, Putchar

getchar() gets a single character from the keyboard, and *putchar()* writes a single character from the keyboard.

Example:

The following program illustrates this,

```
#include <stdio.h>

main()
{
        int  i;
        int ch;

        for( i = 1; i<= 5; ++i ) {
                ch = getchar();
                putchar(ch);
        }
}
```

The program reads five characters (one for each iteration of the for loop) from the keyboard. Note that *getchar()* gets a single character from the keyboard, and *putchar()* writes a single character (in this case, *ch*) to the console screen.

DECISION MAKING

IF STATEMENTS

The *if* statements allows branching (decision making) depending upon the value or state of variables. This allows statements to be executed or skipped, depending upon decisions. The basic format is,

```
if( expression )
    program statement;
```

Example:

```
if( students < 65 )
    ++student_count;
```

In the above example, the variable *student_count* is incremented by one only if the value of the integer variable *students* is less than 65.

The following program uses an *if* statement to validate the users input to be in the range 1-10.

Example:

```
#include <stdio.h>

main()
{
        int number;
        int valid = 0;

        while( valid == 0 ) {
```

```
                printf("Enter a number between 1 and 10 -->");
                scanf("%d", &number);
                /* assume number is valid */
                valid = 1;
                if( number < 1 ) {
                        printf("Number is below 1. Please re-enter\n");
                        valid = 0;
                }
                if( number > 10 ) {
                        printf("Number is above 10. Please re-enter\n");
                        valid = 0;
                }
        }
        printf("The number is %d\n", number );
}
```

IF ELSE

The general format for these are,

```
if( condition 1 )
   statement1;
else if( condition 2 )
   statement2;
else if( condition 3 )
   statement3;
else
   statement4;
```

The *else* clause allows action to be taken where the condition evaluates as false (zero).

The following program uses an *if else* statement to validate the users input to be in the range 1-10.

Example:

```
#include <stdio.h>

main()
{
        int number;
        int valid = 0;

        while( valid == 0 ) {
                printf("Enter a number between 1 and 10 -->");
                scanf("%d", &number);
                if( number < 1 ) {
                        printf("Number is below 1. Please re-enter\n");
                        valid = 0;
                }
```

```
                else if( number > 10 ) {
                        printf("Number is above 10. Please re-enter\n");
                        valid = 0;
                }
                else
                        valid = 1;
        }
        printf("The number is %d\n", number );
}
```

This program is slightly different from the previous example in that an *else* clause is used to set the variable *valid* to 1. In this program, the logic should be easier to follow.

NESTED IF ELSE

/ Illustates nested if else and multiple arguments to the scanf function. */*

Example:

```
#include <stdio.h>

main()
{
        int  invalid_operator = 0;
        char  operator;
        float  number1, number2, result;

        printf("Enter two numbers and an operator in the format\n");
        printf(" number1 operator number2\n");
        scanf("%f %c %f", &number1, &operator, &number2);

        if(operator == '*')
                result = number1 * number2;
        else if(operator == '/')
                result = number1 / number2;
        else if(operator == '+')
                result = number1 + number2;
        else if(operator == '-')
                result = number1 - number2;
        else
                invalid_operator = 1;

        if( invalid_operator != 1 )
                printf("%f %c %f is %f\n", number1, operator, number2, result );
        else
                printf("Invalid operator.\n");
```

BRANCHING AND LOOPING

ITERATION, FOR LOOPS

The basic format of the for statement is,

```
for( start condition; continue condition; re-evaulation )
        program statement;
```

Example:

```
/* sample program using a for statement */
    #include <stdio.h>

    main()   /* Program introduces the for statement, counts to ten */
    {
        int  count;

        for( count = 1; count <= 10; count = count + 1 )
            printf("%d ", count );

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

The program declares an integer variable *count*. The first part of the *for* statement

```
for( count = 1;
```

initialises the value of *count* to 1. The *for* loop continues whilst the condition

```
count <= 10;
```

evaluates as TRUE. As the variable *count* has just been initialised to 1, this condition is TRUE and so the program statement

```
printf("%d ", count );
```

is executed, which prints the value of *count* to the screen, followed by a space character.

Next, the remaining statement of the *for* is executed

```
count = count + 1 );
```

which adds one to the current value of *count*. Control now passes back to the conditional test,

```
count <= 10;
```

which evaluates as true, so the program statement

```
printf("%d ", count );
```

is executed. *Count* is incremented again, the condition re-evaluated etc, until count reaches a value of 11.

When this occurs, the conditional test

```
count <= 10;
```

evaluates as FALSE, and the *for* loop terminates, and program control passes to the statement

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

which prints a newline, and then the program terminates, as there are no more statements left to execute.

THE WHILE STATEMENT

The *while* provides a mechanism for repeating C statements whilst a condition is true. Its format is,

```
while( condition )
    program statement;
```

Somewhere within the body of the *while* loop a statement must alter the value of the condition to allow the loop to finish.

Example:

```
/* Sample program including while  */
    #include <stdio.h>

main()
{
    int  loop = 0;

    while( loop <= 10 ) {
        printf("%d\n", loop);
        ++loop;
    }
}
```

The above program uses a *while* loop to repeat the statements

```
printf("%d\n", loop);
++loop;
```

whilst the value of the variable *loop* is less than or equal to 10.

Note how the variable upon which the *while* is dependant is initialised prior to the *while* statement (in this case the previous line), and also that the value of the variable is altered within the loop, so that eventually the conditional test will succeed and the *while* loop will terminate.

This program is functionally equivalent to the earlier *for* program which counted to ten.

THE DO WHILE STATEMENT

The *do { } while* statement allows a loop to continue whilst a condition evaluates as TRUE (non-zero). The loop is executed as least once.

Example:

```
/* Demonstration of DO...WHILE    */
#include <stdio.h>

main()
{
      int  value, r_digit;

      printf("Enter the number to be reversed.\n");
      scanf("%d", &value);
      do {
            r_digit = value % 10;
            printf("%d", r_digit);
            value = value / 10;
      } while( value != 0 );
      printf("\n");
}
```

The above program reverses a number that is entered by the user. It does this by using the modulus % operator to extract the right most digit into the variable *r_digit*. The original number is then divided by 10, and the operation repeated whilst the number is not equal to 0.

SWITCH CASE:

The *switch case* statement is a better way of writing a program when a series of *if elses* occurs. The general format for this is,

```
switch ( expression ) {
            case  value1:
                        program statement;
                        program statement;
                        ......
                        break;
            case  valuen:
                        program statement;
                        .......
                        break;
            default:
                        .......
                        .......
                        break;
}
```

The keyword *break* must be included at the end of each case statement. The default clause is optional, and is executed if the cases are not met. The right brace at the end signifies the end of the case selections.

Example:

```
#include <stdio.h>

        main()
        {
                int menu, numb1, numb2, total;

                printf("enter in two numbers -->");
                scanf("%d %d", &numb1, &numb2 );
                printf("enter in choice\n");
                printf("1=addition\n");
                printf("2=subtraction\n");
                scanf("%d", &menu );
                switch( menu ) {
                        case 1: total = numb1 + numb2; break;
                        case 2: total = numb1 - numb2; break;
                        default: printf("Invalid option selected\n");
                }
                if( menu == 1 )
                        printf("%d plus %d is %d\n", numb1, numb2, total );
                else if( menu == 2 )
                        printf("%d minus %d is %d\n", numb1, numb2, total );
        }
```

The above program uses a *switch* statement to validate and select upon the users input choice, simulating a simple menu of choices.

GE2112 - FUNDAMENTALS OF COMPUTING AND PROGRAMMING

UNIT V

FUNCTIONS AND POINTERS

Handling of Character Strings – User-defined Functions – Definitions – Declarations - Call by reference – Call by value – Structures and Unions – Pointers – Arrays – The Preprocessor – Developing a C Program : Some Guidelines

HANDLING OF CHARACTER STRINGS

STRINGS:

String is the collection of characters.Header file used is **string.h**

Datatype variable_name [] = "love.";

Example:

```
#include <stdio.h>
main()
{
        static char  string1[] = "Bye Bye ";
        static char  string2[] = "love.";
        char  string3[25];
        int  n = 0, n2;

        for( ; string1[n] != '\0'; ++n )
                string3[n] = string1[n];

        n2 = n;  n = 0;

        for( ; string2[n] != '\0'; ++n )
                string3[n2 + n] = string2[n];

        n2 += n;

        for(n = 0; n < n2 ; ++n)
                printf("%c", string3[n]);
}
```

BUILT IN FUNCTIONS FOR STRING HANDLING

You may want to look at the section on arrays first!. The following macros are built into the file string.h

strcat	*Appends a string*
strchr	*Finds first occurrence of a given character*
strcmp	*Compares two strings*
strcmpi	*Compares two strings, non-case sensitive*
strcpy	*Copies one string to another*
strlen	*Finds length of a string*
strlwr	*Converts a string to lowercase*
strncat	*Appends n characters of string*
strncmp	*Compares n characters of two strings*
strncpy	*Copies n characters of one string to another*

strnset *Sets n characters of string to a given character*
strrchr *Finds last occurrence of given character in string*
strrev *Reverses string*
strset *Sets all characters of string to a given character*
strspn *Finds first substring from given character set in string*
strupr *Converts string to uppercase*

Example:

To convert a string to uppercase

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

main()
{
        char name[80];      /* declare an array of characters 0-79 */

        printf("Enter in a name in lowercase\n");
        scanf( "%s", name );
        strupr( name );
        printf("The name is uppercase is %s", name );
}
```

FUNCTIONS

A function in C can perform a particular task, and supports the concept of modular programming design techniques.

We have already been exposed to functions. The main body of a C program, identified by the keyword *main*, and enclosed by the left and right braces is a function. It is called by the operating system when the program is loaded, and when terminated, returns to the operating system.

Functions have a basic structure. Their format is

```
return_data_type  function_name  ( arguments, arguments )
data_type_declarations_of_arguments;
{
        function_body
}
```

It is worth noting that a return_data_type is assumed to be type *int* unless otherwise specified, thus the programs we have seen so far imply that *main()* returns an integer to the operating system.

```
return_data_type function_name (data_type variable_name, data_type variable_name, .. )
{
        function_body
}
```

A simple function is,

```
void print_message( void )
{
        printf("This is a module called print_message.\n");
}
```

Example:

Now lets incorporate this function into a program.

```
/* Program illustrating a simple function call */
#include <stdio.h>

void print_message( void );    /* ANSI C function prototype */

void print_message( void )     /* the function code */
{
        printf("This is a module called print_message.\n");
}

main()
{
        print_message();
}
```

To call a function, it is only necessary to write its name. The code associated with the function name is executed at that point in the program. When the function terminates, execution begins with the statement which follows the function name.

In the above program, execution begins at *main()*. The only statement inside the main body of the program is a call to the code of function *print_message()*. This code is executed, and when finished returns back to *main()*.

As there is no further statements inside the main body, the program terminates by returning to the operating system.

RETURNING FUNCTION RESULTS

This is done by the use of the keyword *return*, followed by a data variable or constant value, the data type of which must match that of the declared return_data_type for the function.

```
float add_numbers( float n1, float n2 )
{
        return n1 + n2;    /* legal */
        return 6;          /* illegal, not the same data type */
        return 6.0;        /* legal */
}
```

It is possible for a function to have multiple return statements.

```
int validate_input( char command )
```

```
{
        switch( command ) {
                case '+' :
                case '-' : return 1;
                case '*' :
                case '/' : return 2;
                default  : return 0;
        }
}
```

Example:

```
/* Simple multiply program using argument passing */
        #include <stdio.h>

        int calc_result( int, int );          /* ANSI function prototype */

        int calc_result( int numb1, int numb2 )
        {
                auto int result;
                result = numb1 * numb2;
                return result;
        }

        main()
        {
                int  digit1 = 10, digit2 = 30, answer = 0;
                answer = calc_result( digit1, digit2 );
                printf("%d multiplied by %d is %d\n", digit1, digit2, answer );
        }
```

RECURSION

This is where a function repeatedly calls itself to perform calculations. Typical applications are games and Sorting trees and lists.

Consider the calculation of 6! (6 factorial)

ie 6! = 6 * 5 * 4 * 3 * 2 * 1
6! = 6 * 5!
6! = 6 * (6 - 1)!
n! = n * (n - 1)!

Example:

```
        /* example for demonstrating recursion */
        #include <stdio.h>

        long int factorial( long int );        /* function prototype */

        long int  factorial( long int n )
        {
                long int result;
```

```
        if( n == 0L )
                result = 1L;
        else
                result = n * factorial( n - 1L );
        return ( result );
}

main()
{
        int j;

        for( j = 0; j < 11; ++j )
                printf("%2d! = %ld\n", factorial( (long) j) );
}
```

CALL BY VALUE:

When the value is passed directly to the function it is called call by value. In call by value only a copy of the variable is only passed so any changes made to the variable does not reflects in the calling function.

Example:

```
#include<stdio.h>
#include<conio.h>
swap(int,int);
void main()
{
int x,y;
printf("Enter two nos");
scanf("%d %d",&x,&y);
printf("\nBefore swapping : x=%d y=%d",x,y);
swap(x,y);
getch();
}
swap(int a,int b)
{
int t;
t=a;
a=b;
b=t;
printf("\nAfter swapping :x=%d y=%d",a,b);
}
```

SYSTEM OUTPUT:
Enter two nos 12 34
Before swapping :12 34
After swapping : 34 12

CALL BY REFERENCE:

When the address of the value is passed to the function it is called call by reference. In call by reference since the address of the value is passed any changes made to the value reflects in the calling function**.**

Example:

```
#include<stdio.h>
#include<conio.h>
swap(int *, int *);
void main()
{
int x,y;
printf("Enter two nos");
scanf("%d %d",&x,&y);
printf("\nBefore swapping:x=%d y=%d",x,y);
swap(&x,&y);
printf("\nAfter swapping :x=%d y=%d",x,y);
getch();
}
swap(int *a,int *b)
{
int t;
t=*a;
*a=*b;
*b=t;
}
```

SYSTEM OUTPUT:

```
Enter two nos 12 34
Before swapping :12 34
After swapping : 34 12
```

STRUCTURES AND UNIONS

STRUCTURES

A Structure is a data type suitable for grouping data elements together. Lets create a new data structure suitable for storing the date. The elements or fields which make up the structure use the four basic data types. As the storage requirements for a structure cannot be known by the compiler, a definition for the structure is first required. This allows the compiler to determine the storage allocation needed, and also identifies the various sub-fields of the structure.

```
struct   date {
            int  month;
            int  day;
            int  year;
};
```

This declares a NEW data type called *date*. This date structure consists of three basic data elements, all of type integer. This is a definition to the compiler**.** It does not create any storage space and cannot be used as a variable.

In essence, it's a new data type keyword, like *int* and *char*, and can now be used to create variables. Other data structures may be defined as consisting of the same composition as the *date* structure,

```
struct  date   todays_date;
```

defines a variable called *todays_date* to be of the same data type as that of the newly defined data type struct *date*.

ASSIGNING VALUES TO STRUCTURE ELEMENTS

To assign todays date to the individual elements of the structure *todays_date*, the statement

```
todays_date.day = 21;
todays_date.month = 07;
todays_date.year = 1985;
```

is used. NOTE the use of the .element to reference the individual elements within *todays_date*.

Example:

```
/* Program to illustrate a structure */
        #include <stdio.h>

        struct date {                           /* global definition of type date */
                int month;
                int day;
                int year;
        };
        main()
        {

                struct date  today;

                today.month = 10;
                today.day = 14;
                today.year = 1995;

                printf("Todays date is %d/%d/%d.\n", \
                        today.month, today.day, today.year );
        }
```

UNIONS

This is a special data type which looks similar to a structure, but is very different. The declaration is,

```
union  mixed {
        char  letter;
        float radian;
        int   number;
};

union mixed all;
```

The first declaration consists of a union of type mixed, which consists of a char, float, or int variable. NOTE that it can be ONLY ONE of the variable types, they cannot coexist.

This is due to the provision of a single memory address which is used to store the largest variable, unlike the arrangement used for structures.

Thus the variable *all* can only be a character, a float or an integer at any one time. The C language keeps track of what *all* actually is at any given moment, but does not provide a check to prevent the programmer accessing it incorrectly.

ARRAYS

Arrays are data structures which hold multiple variables of the same data type. Consider the case where a programmer needs to keep track of a number of people within an organization. So far, our initial attempt will be to create a specific variable for each user. This might look like,

```
int name1 = 101;
int name2 = 232;
int name3 = 231;
```

It becomes increasingly more difficult to keep track of this as the number of variables increase. Arrays offer a solution to this problem.

An array is a multi-element box, a bit like a filing cabinet, and uses an indexing system to find each variable stored within it. In C, indexing starts at **zero**.

Arrays, like other variables in C, must be declared before they can be used.

The replacement of the above example using arrays looks like,

```
int names[4];
names[0] = 101;
names[1] = 232;
names[2] = 231;
names[3] = 0;
```

We created an array called *names*, which has space for four integer variables. You may also see that we stored 0 in the last space of the array. This is a common technique used by C programmers to signify the end of an array.

Arrays have the following syntax, using square brackets to access each indexed value (called an element).

```
x[i]
```

so that *x[5]* refers to the sixth element in an array called *x*. In C, array elements start with 0. Assigning values to array elements is done by,

```
x[10] = g;
```

and assigning array elements to a variable is done by,

g = x[10];

In the following example, a character based array named *word* is declared, and each element is assigned a character. The last element is filled with a zero value, to signify the end of the character string (in C, there is no string type, so character based arrays are used to hold strings). A printf statement is then used to print out all elements of the array.

```
/*  Introducing array's, 2  */
#include <stdio.h>

main()
{
        char word[20];

        word[0] = 'H';
        word[1] = 'e';
        word[2] = 'l';
        word[3] = 'l';
        word[4] = 'o';
        word[5] = 0;
        printf("The contents of word[] is -->%s\n", word );
}
```

DECLARING ARRAYS

Arrays may consist of any of the valid data types. Arrays are declared along with all other variables in the declaration section of the program.

```
/*  Introducing array's  */
#include <stdio.h>

main()
{
        int   numbers[100];
        float  averages[20];

        numbers[2] = 10;
        --numbers[2];
        printf("The 3rd element of array numbers is %d\n", numbers[2]);
}
```

The above program declares two arrays, assigns 10 to the value of the 3rd element of array *numbers*, decrements this value (*--numbers[2]*), and finally prints the value. The number of elements that each array is to have is included inside the square brackets

ASSIGNING INITIAL VALUES TO ARRAYS

The declaration is preceded by the word *static*. The initial values are enclosed in braces, eg,

Example:

```
#include <stdio.h>
main()
{
        int x;
        static int  values[] = { 1,2,3,4,5,6,7,8,9 };
        static char  word[] = { 'H','e','l','l','o' };
        for( x = 0; x < 9; ++x )
                printf("Values [%d] is %d\n", x, values[x]);
}
```

The previous program declares two arrays, *values* and *word*. Note that inside the squarebrackets there is no variable to indicate how big the array is to be. In this case, C initializes the array to the number of elements that appear within the initialize braces. So values consist of 9 elements (numbered 0 to 8) and the char array *word* has 5 elements.

MULTI DIMENSIONED ARRAYS

Multi-dimensioned arrays have two or more index values which specify the element in the array.

```
multi[i][j]
```

In the above example, the first index value *i* specifies a row index, whilst *j* specifies a column index.

DECLARATION

```
int       m1[10][10];
static int m2[2][2] = { {0,1}, {2,3} };

sum = m1[i][j] + m2[k][l];
```

NOTE the strange way that the initial values have been assigned to the two-dimensional array m2. Inside the braces are,

```
{ 0, 1 },
{ 2, 3 }
```

Remember that arrays are split up into row and columns. The first is the row, the second is the column. Looking at the initial values assigned to *m2*, they are,

```
m2[0][0] = 0
m2[0][1] = 1
m2[1][0] = 2
m2[1][1] = 3
```

Example:

```
#include <stdio.h>

        main()
        {
                static int m[][] = { {10,5,-3}, {9, 0, 0}, {32,20,1}, {0,0,8} };
                int row, column, sum;

                sum = 0;
                for( row = 0; row < 4; row++ )
                        for( column = 0; column < 3; column++ )
                                sum = sum + m[row][column];
                printf("The total is %d\n", sum );
        }
```

CHARACTER ARRAYS [STRINGS]

Consider the following program,

```
        #include <stdio.h>
        main()
        {
                static char name1[] = {'H','e','l','l','o'};
                static char name2[] = "Hello";
                printf("%s\n", name1);
                printf("%s\n", name2);
        }
```

The difference between the two arrays is that *name2* has a null placed at the end of the string, ie, in name2[5], whilst *name1* has not. To insert a null at the end of the name1 array, the initialization can be changed to,

```
        static char name1[] = {'H','e','l','l','o','\0'};
```

Consider the following program, which initialises the contents of the character based array *word* during the program, using the function *strcpy*, which necessitates using the include file *string.h*

Example:

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

main()
{
        char word[20];

        strcpy( word, "hi there." );
        printf("%s\n", word );
}
```

POINTERS

Pointers enable us to effectively represent complex data structures, to change values as arguments to functions, to work with memory which has been dynamically allocated, and to more concisely and efficiently deal with arrays. A pointer provides an indirect means of accessing the value of a particular data item. Lets see how pointers actually work with a simple example,

```
int  count = 10, *int_pointer;
```

declares an integer *count* with a value of 10, and also an integer pointer called *int_pointer*. Note that the prefix * defines the variable to be of type pointer. To set up an indirect reference between *int_pointer* and *count*, the & prefix is used, ie,

```
int_pointer = &count;
```

This assigns the memory address of *count* to int_pointer, not the actual value of *count* stored at that address.

POINTERS CONTAIN MEMORY ADDRESSES, NOT VALUES!

To reference the value of *count* using *int_pointer*, the * is used in an assignment, eg,

```
x = *int_pointer;
```

Since *int_pointer* is set to the memory address of *count*, this operation has the effect of assigning the contents of the memory address pointed to by *int_pointer* to the variable *x*, so that after the operation variable *x* has a value of 10.

Example:

```
#include <stdio.h>

main()
{
        int count  = 10, x, *int_pointer;

        /* this assigns the memory address of count to int_pointer  */
        int_pointer = &count;

        /* assigns the value stored at the address specified by int_pointer to x */
        x = *int_pointer;

        printf("count = %d, x = %d\n", count, x);
}
```

This however, does not illustrate a good use for pointers.

Example:

The following program illustrates another way to use pointers, this time with characters,

```
#include <stdio.h>

main()
{
        char c = 'Q';
        char *char_pointer = &c;

        printf("%c %c\n", c, *char_pointer);

        c = 'Z';
        printf("%c %c\n", c, *char_pointer);
        *char_pointer = 'Y';
        /* assigns Y as the contents of the memory address specified by char_pointer  */

        printf("%c %c\n", c, *char_pointer);
}
```

POINTERS AND STRUCTURES

Consider the following,

```
struct date {
        int month, day, year;
};

struct date  todays_date, *date_pointer;

date_pointer = &todays_date;

(*date_pointer).day = 21;
(*date_pointer).year = 1985;
(*date_pointer).month = 07;

++(*date_pointer).month;
if((*date_pointer).month == 08 )
        ......
```

Pointers to structures are so often used in C that a special operator exists. The structure pointer operator, the ->, permits expressions that would otherwise be written as,

```
(*x).y
```

to be more clearly expressed as

```
x->y
```

making the if statement from above program

```
if( date_pointer->month == 08 )
```

Example:

```
/* Program to illustrate structure pointers */
#include <stdio.h>

main()
{
        struct date { int month, day, year; };
        struct date today, *date_ptr;

        date_ptr = &today;
        date_ptr->month = 9;
        date_ptr->day = 25;
        date_ptr->year = 1983;

        printf("Todays date is %d/%d/%d.\n", date_ptr->month, \
                date_ptr->day, date_ptr->year % 100);
}
```

STRUCTURES CONTAINING POINTERS

Naturally, a pointer can also be a member of a structure.

```
struct  int_pointers {
        int  *ptr1;
        int  *ptr2;
};
```

In the above, the structure *int_pointers* is defined as containing two integer pointers, *ptr1* and *ptr2*. A variable of type struct int_pointers can be defined in the normal way, eg,

```
struct  int_pointers  ptrs;
```

The variable *ptrs* can be used normally, eg, consider the following program,

Example:

```
#include <stdio.h>
main()   /* Illustrating structures containing pointers */
{
        struct  int_pointers {  int  *ptr1, *ptr2;  };
        struct int_pointers  ptrs;
        int  i1 = 154, i2;

        ptrs.ptr1 =  &i1;
        ptrs.ptr2 =  &i2;
        (*ptrs).ptr2 =  -97;
        printf("i1 = %d, *ptrs.ptr1 = %d\n", i1, *ptrs.ptr1);
        printf("i2 = %d, *ptrs.ptr2 = %d\n", i2, *ptrs.ptr2);
}
```

POINTERS AND CHARACTER STRINGS

A pointer may be defined as pointing to a character string.

```
#include <stdio.h>

main()
{
        char *text_pointer = "Good morning!";

        for( ; *text_pointer != '\0'; ++text_pointer)
                printf("%c", *text_pointer);
}
```

or another program illustrating pointers to text strings,

Example:

```
#include <stdio.h>

        main()
        {
                static char *days[] = {"Sunday", "Monday", "Tuesday", "Wednesday", Thursday", "Friday",
"Saturday"};
                int i;

                for( i = 0; i < 6; ++i )
                        printf( "%s\n", days[i]);
        }
```

Remember that if the declaration is,

```
char *pointer = "Sunday";
```

then the null character { '\0' } is automatically appended to the end of the text string. This means that %s may be used in a *printf* statement, rather than using a *for* loop and %c to print out the contents of the pointer. The %s will print out all characters till it finds the null terminator.

POINTERS AND FUNCTION ARGUMENTS

Since C passes arguments to functions by value, there is no direct way for the called function to alter a variable in the calling function. For instance, a sorting routine might exchange two out-of-order arguments with a function called swap. It is not enough to write

```
swap(a, b);
```

where the swap function is defined as

```
void swap(int x, int y)  /* WRONG */
{
    int temp;

    temp = x;
    x = y;
    y = temp;
}
```

Because of call by value, swap can't affect the arguments a and b in the routine that called it. The function above

swaps *copies* of a and b.

The way to obtain the desired effect is for the calling program to pass *pointers* to the values to be changed:

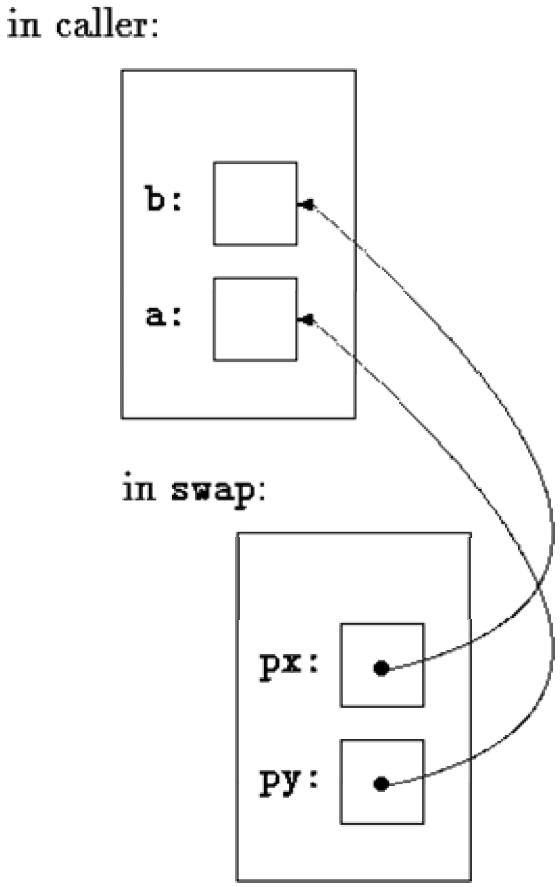
```
swap(&a, &b);
```

Since the operator & produces the address of a variable, &a is a pointer to a. In swap itself, the parameters are declared as pointers, and the operands are accessed indirectly through them.

```
void swap(int *px, int *py)  /* interchange *px and *py */
{
    int temp;

    temp = *px;
    *px = *py;
    *py = temp;   }
```

Pictorially:



Pointer arguments enable a function to access and change objects in the function that called it. As an example, consider a function getint that performs free-format input conversion by breaking a stream of characters into integer values, one integer per call. getint has to return the value it found and also signal end of file when there is no more input. These values have to be passed back by separate paths, for no matter what value is used for EOF, that could also be the value of an input integer.

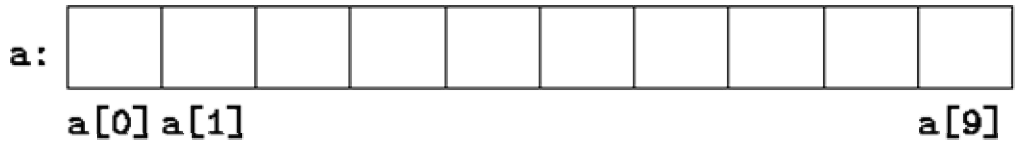
POINTERS AND ARRAYS

In C, there is a strong relationship between pointers and arrays, strong enough that pointers and arrays should be discussed simultaneously. Any operation that can be achieved by array subscripting can also be done with pointers. The pointer version will in general be faster but, at least to the uninitiated, somewhat harder to understand.

The declaration

```
int a[10];
```

defines an array of size 10, that is, a block of 10 consecutive objects named a[0], a[1], ...,a[9].



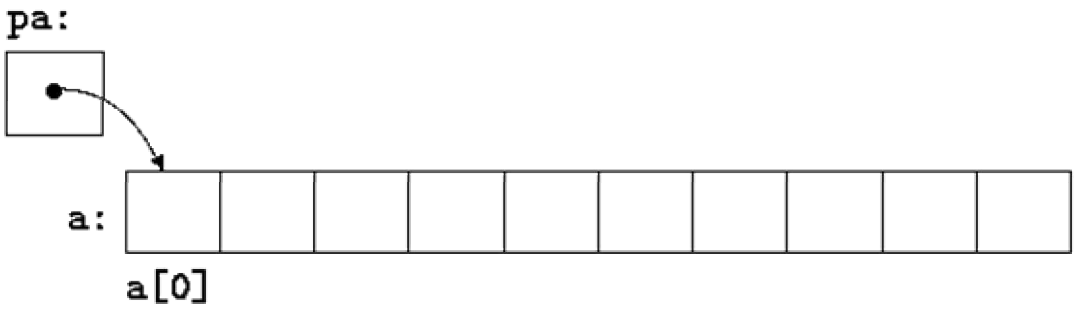
The notation a[i] refers to the i-th element of the array. If pa is a pointer to an integer, declared as

```
int *pa;
```

then the assignment

```
pa = &a[0];
```

sets pa to point to element zero of a; that is, pa contains the address of a[0].



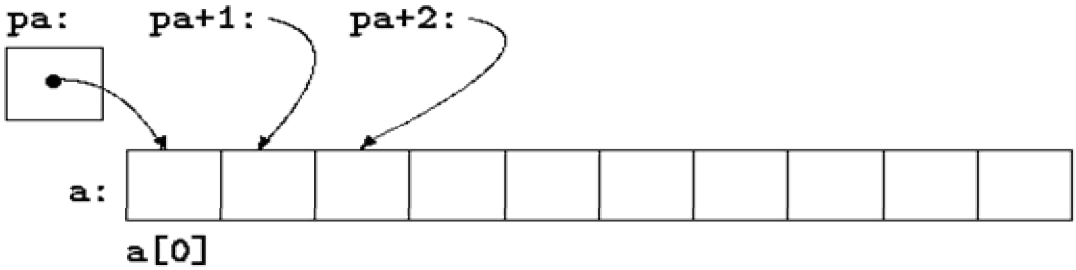
Now the assignment

```
x = *pa;
```

will copy the contents of a[0] into x.

If pa points to a particular element of an array, then by definition pa+1 points to the next element, pa+i points i elements after pa, and pa-i points i elements before. Thus, if pa points to a[0],

*(pa+1) refers to the contents of a[1], pa+i is the address of a[i], and *(pa+i) is the contents of a[i].



These remarks are true regardless of the type or size of the variables in the array a. The meaning of ``adding 1 to a pointer," and by extension, all pointer arithmetic, is that pa+1 points to the next object, and pa+i points to the i-th object beyond pa.

The correspondence between indexing and pointer arithmetic is very close. By definition, the value of a variable or expression of type array is the address of element zero of the array. Thus after the assignment

```
pa = &a[0];
```

pa and a have identical values. Since the name of an array is a synonym for the location of the initial element, the assignment pa=&a[0] can also be written as

```
pa = a;
```

Rather more surprising, at first sight, is the fact that a reference to a[i] can also be written as *(a+i). In evaluating a[i], C converts it to *(a+i) immediately; the two forms are equivalent. Applying the operator & to both parts of this equivalence, it follows that &a[i] and a+i are also identical: a+i is the address of the i-th element beyond a. As the other side of this coin, if pa is a pointer, expressions might use it with a subscript; pa[i] is identical to *(pa+i). In short, an array-and-index expression is equivalent to one written as a pointer and offset.

There is one difference between an array name and a pointer that must be kept in mind. A pointer is a variable, so pa=a and pa++ are legal. But an array name is not a variable; constructions like a=pa and a++ are illegal.

When an array name is passed to a function, what is passed is the location of the initial element. Within the called function, this argument is a local variable, and so an array name parameter is a pointer, that is, a variable containing an address. We can use this fact to write another version of strlen, which computes the length of a string.

```
/* strlen:  return length of string s */
int strlen(char *s)
{
    int n;

    for (n = 0; *s != '\0', s++)
        n++;
    return n;
}
```

Since s is a pointer, incrementing it is perfectly legal; s++ has no effect on the character string in the function

that called strlen, but merely increments strlen's private copy of the pointer. That means that calls like

```
strlen("hello, world");   /* string constant */
strlen(array);            /* char array[100]; */
strlen(ptr);              /* char *ptr; */
```

all work.

As formal parameters in a function definition,

```
char s[];
```

and

```
char *s;
```

are equivalent; we prefer the latter because it says more explicitly that the variable is a pointer. When an array name is passed to a function, the function can at its convenience believe that it has been handed either an array or a pointer, and manipulate it accordingly. It can even use both notations if it seems appropriate and clear.

It is possible to pass part of an array to a function, by passing a pointer to the beginning of the subarray. For example, if a is an array,

```
f(&a[2])
```

and

```
f(a+2)
```

both pass to the function f the address of the subarray that starts at a[2]. Within f, the parameter declaration can read

```
f(int arr[]) { ... }
```

or

```
f(int *arr) { ... }
```

CHARACTER POINTERS AND FUNCTIONS

A *string constant*, written as

```
"I am a string"
```

is an array of characters. In the internal representation, the array is terminated with the null character '\0' so that programs can find the end. The length in storage is thus one more than the number of characters between the double quotes.

Perhaps the most common occurrence of string constants is as arguments to functions, as in

```
printf("hello, world\n");
```

When a character string like this appears in a program, access to it is through a character pointer; printf receives a pointer to the beginning of the character array. That is, a string constant is accessed by a pointer to its first element.

String constants need not be function arguments. If pmessage is declared as

```
char *pmessage;
```

then the statement

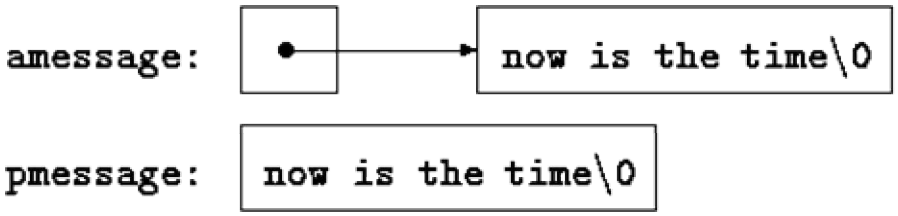
```
pmessage = "now is the time";
```

assigns to pmessage a pointer to the character array. This is *not* a string copy; only pointers are involved. C does not provide any operators for processing an entire string of characters as a unit.

There is an important difference between these definitions:

```
char amessage[] = "now is the time"; /* an array */
char *pmessage = "now is the time"; /* a pointer */
```

amessage is an array, just big enough to hold the sequence of characters and '\0' that initializes it. Individual characters within the array may be changed but amessage will always refer to the same storage. On the other hand, pmessage is a pointer, initialized to point to a string constant; the pointer may subsequently be modified to point elsewhere, but the result is undefined if you try to modify the string contents.



We will illustrate more aspects of pointers and arrays by studying versions of two useful functions adapted from the standard library. The first function is strcpy(s,t), which copies the string t to the string s. It would be nice just to say s=t but this copies the pointer, not the characters. To copy the characters, we need a loop. The array version first:

```
/* strcpy:  copy t to s; array subscript version */
void strcpy(char *s, char *t)
{
    int i;

    i = 0;
    while ((s[i] = t[i]) != '\0')
        i++;
}
```

For contrast, here is a version of strcpy with pointers:

```
/* strcpy:  copy t to s; pointer version */
void strcpy(char *s, char *t)
{
    int i;

    i = 0;
    while ((*s = *t) != '\0') {
        s++;
        t++;
    }
}
```

Because arguments are passed by value, strcpy can use the parameters s and t in any way it pleases. Here they are conveniently initialized pointers, which are marched along the arrays a character at a time, until the '\0' that

terminates t has been copied into s

POINTERS VS. MULTI-DIMENSIONAL ARRAYS

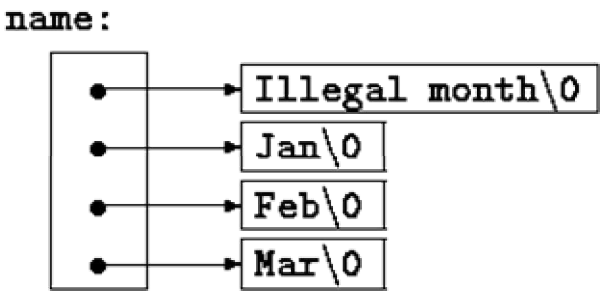
Newcomers to C are sometimes confused about the difference between a two-dimensional array and an array of pointers, such as name in the example above. Given the definitions

```
int a[10][20];
int *b[10];
```

then a[3][4] and b[3][4] are both syntactically legal references to a single int. But a is a true two-dimensional array: 200 int-sized locations have been set aside, and the conventional rectangular subscript calculation 20 * *row* +*col* is used to find the element a[row,col]. For b, however, the definition only allocates 10 pointers and does not initialize them; initialization must be done explicitly, either statically or with code. Assuming that each element of b does point to a twenty-element array, then there will be 200 ints set aside, plus ten cells for the pointers. The important advantage of the pointer array is that the rows of the array may be of different lengths. That is, each element of b need not point to a twenty-element vector; some may point to two elements, some to fifty, and some to none at all.

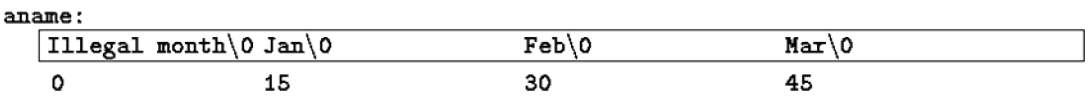
Although we have phrased this discussion in terms of integers, by far the most frequent use of arrays of pointers is to store character strings of diverse lengths, as in the function month_name. Compare the declaration and picture for an array of pointers:

```
char *name[] = { "Illegal month", "Jan", "Feb", "Mar" };
```



with those for a two-dimensional array:

```
char aname[][15] = { "Illegal month", "Jan", "Feb", "Mar" };
```



STORAGE MANAGEMENT

The functions malloc and calloc obtain blocks of memory dynamically.

void *malloc(size_t n)

returns a pointer to n bytes of uninitialized storage, or NULL if the request cannot be satisfied.

void *calloc(size_t n, size_t size)

returns a pointer to enough free space for an array of n objects of the specified size, or NULL if the request cannot be satisfied. The storage is initialized to zero.

The pointer returned by malloc or calloc has the proper alignment for the object in question, but it must be cast into the appropriate type, as in

```
int *ip;

ip = (int *) calloc(n, sizeof(int));
```

free(p) frees the space pointed to by p, where p was originally obtained by a call to malloc or calloc. There are no restrictions on the order in which space is freed, but it is a ghastly error to free something not obtained by calling malloc or calloc.

It is also an error to use something after it has been freed. A typical but incorrect piece of code is this loop that frees items from a list:

```
for (p = head; p != NULL; p = p->next) /* WRONG */
    free(p);
```

The right way is to save whatever is needed before freeing:

```
for (p = head; p != NULL; p = q) {
    q = p->next;
    free(p);
}
```

Example:

```
/* linked list example, pr101, 1994 */
#include <string.h>
#include <alloc.h>
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <conio.h>

/* definition of a node */
struct node {
   char data[20];
   struct node *next;
};


struct node * initialise( void );
void freenodes( struct node * );
int insert( struct node * );
void delete( struct node *, struct node * );
void list( struct node * );
void menu( struct node *, struct node * );
void readline( char [] );
```

```
void readline( char buff[] )
{
   int ch, loop = 0;

   ch = getche();
   while( ch != '\r' ) {
     buff[loop] = ch;
     loop++;
     ch = getche();
   }
   buff[loop] = 0;
}

struct node * initialise( void )
{
   return( (struct node *) calloc(1, sizeof( struct node *) ));
}
```

SIZEOF

The sizeof() function returns the memory size of the requested variable. This call should be used in conjunction with the *calloc()* function call, so that only the necessary memory is allocated, rather than a fixed size. Consider the following,

```
struct date {
        int hour, minute, second;
};

int x;

x = sizeof( struct date );
```

x now contains the information required by *calloc()* so that it can allocate enough memory to contain another structure of type *date*.

THE PREPROCESSOR

The *define* statement is used to make programs more readable, and allow the inclusion of macros. Consider the following examples,

```
#define TRUE    1    /* Do not use a semi-colon , # must be first character on line */
#define FALSE   0
#define NULL    0
#define AND     &
#define OR      |
#define EQUALS  ==

                    game_over = TRUE;
                    while( list_pointer != NULL )
                              .....
```

MACROS

Macros are inline code which are substituted at compile time. The definition of a macro, which accepts an argument when referenced,

```
#define  SQUARE(x)  (x)*(x)

y = SQUARE(v);
```

In this case, *v* is equated with *x* in the macro definition of *square*, so the variable *y* is assigned the square of *v*. The brackets in the macro definition of *square* are necessary for correct evaluation. The expansion of the macro becomes

```
y = (v) * (v);
```

Naturally, macro definitions can also contain other macro definitions,

```
#define IS_LOWERCASE(x)  (( (x)>='a') && ( (x) <='z') )
#define TO_UPPERCASE(x)  (IS_LOWERCASE (x)?(x)-'a'+'A':(x))

while(*string) {
          *string = TO_UPPERCASE(*string);
          ++string;
}
```

CONDITIONAL COMPILATIONS

These are used to direct the compiler to compile/or not compile the lines that follow

```
#ifdef  NULL
#define NL 10
#define SP 32
#endif
```

In the preceding case, the definition of NL and SP will only occur if NULL has been defined prior to the compiler encountering the #ifdef NULL statement. The scope of a definition may be limited by

```
#undef NULL
```

This renders the identification of NULL invalid from that point onwards in the source file.

Typedef

This statement is used to classify existing C data types, eg,

```
typedef  int counter;  /* redefines counter as an integer */
counter j, n;          /* counter now used to define j and n as integers */

typedef struct {
            int month, day, year;
} DATE;

DATE  todays_date;/* same as struct date todays_date */
```

ENUMERATED DATA TYPES

Enumerated data type variables can only assume values which have been previously declared.

```
enum month { jan = 1, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec };
enum month this_month;

this_month = feb;
```

In the above declaration, *month* is declared as an enumerated data type. It consists of a set of values, jan to dec. Numerically, jan is given the value 1, feb the value 2, and so on. The variable *this_month* is declared to be of the same type as month, then is assigned the value associated with feb. This_month cannot be assigned any values outside those specified in the initialization list for the declaration of month.

Example:

```
#include <stdio.h>

        main()
        {
                char *pwest = "west",*pnorth = "north", *peast="east", *psouth = "south";
                enum location { east=1, west=2, south=3, north=4};
                enum location direction;

                direction = east;

                if( direction == east )
                        printf("Cannot go %s\n", peast);
        }
```

The variables defined in the enumerated variable *location* should be assigned initial values.

DECLARING VARIABLES TO BE REGISTER BASED

Some routines may be time or space critical. Variables can be defined as being register based by the following declaration,

```
register int index;
```

DECLARING VARIABLES TO BE EXTERNAL

Here variables may exist in separately compiled modules, and to declare that the variable is external,

```
extern  int  move_number;
```

This means that the data storage for the variable *move_number* resides in another source module, which will be linked with this module to form an executable program. In using a variable across a number of independently compiled modules, space should be allocated in only one module, whilst all other modules use the extern directive to access the variable.

NULL STATEMENTS

These are statements which do not have any body associated with them.

```
/* sums all integers in array a containing n elements and initializes */
/* two variables at the start of the for loop */
for( sum = 0, i = 0; i < n; sum += a[i++] )
        ;


/* Copies characters from standard input to standard output until EOF is reached  */
for( ; (c = getchar ()) != EOF; putchar (c));
```

COMMAND LINE ARGUMENTS

It is possible to pass arguments to C programs when they are executed. The brackets which follow main are used for this purpose. *argc* refers to the number of arguments passed, and *argv[]* is a pointer array which points to each argument which is passed to main. A simple example follows, which checks to see if a single argument is supplied on the command line when the program is invoked.

```
#include <stdio.h>

main( int argc, char *argv[] )
{
        if( argc == 2 )
                printf("The argument supplied is %s\n", argv[1]);
        else if( argc > 2 )
                printf("Too many arguments supplied.\n");
        else
                printf("One argument expected.\n");
}
```

Note that **argv[0]* is the name of the program invoked, which means that **argv[1]* is a pointer to the first argument supplied, and **argv[n]* is the last argument. If no arguments are supplied, *argc* will be one. Thus for n arguments, *argc* will be equal to n + 1. The program is called by the command line,

```
myprog  argument1
```

DEVELOPING A C PROGRAM: SOME GUIDELINES

In C, lowercase and uppercase characters are very important! All commands in C must be lowercase. The C programs starting point is identified by the word

```
main()
```

This informs the computer as to where the program actually starts. The brackets that follow the keyword *main* indicate that there are no arguments supplied to this program (this will be examined later on).

The two braces, { and }, signify the begin and end segments of the program. The purpose of the statment

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

is to allow the use of the *printf* statement to provide program output. Text to be displayed by *printf()* must be enclosed in double quotes. The program has only one statement

```
printf("Programming in C is easy.\n");
```

printf() is actually a function (procedure) in C that is used for printing variables and text. Where text appears in double quotes "", it is printed without modification. There are some exceptions however. This has to do with the \ and % characters. These characters are modifier's, and for the present the \ followed by the n character represents a newline character. Thus the program prints

```
Programming in C is easy.
```

and the cursor is set to the beginning of the next line. As we shall see later on, what follows the \ character will determine what is printed, ie, a tab, clear screen, clear line etc. Another important thing to remember is that all C statements are terminated by a semi-colon;

Summary of major points:

- program execution begins at *main()*
- keywords are written in lower-case
- statements are terminated with a semi-colon
- text strings are enclosed in double quotes
- C is case sensitive, use lower-case and try not to capitalise variable names
- \n means position the cursor on the beginning of the next line
- *printf()* can be used to display text to the screen
- The curly braces {} define the beginning and end of a program block.

CIA3 important questions:

2 marks:

1. Differentiate structures and unions.
2. Write the advantages of pointers.
3. Distinguish between printf and scanf.
4. What is a null pointer?
5. List any 4 dynamic memory management functions in C.
6. What is an escape sequence?
7. Define problem solving.
8. Draw any 4 the symbols of Flowchart.
9. Distinguish between Flowchart and Algorithm.
10. Distinguish between malloc and calloc
11. What is an union? Give syntax for defining union.
12. State any 4 library functions in C.

14 marks:

1. Define Pointer. Explain Call by Value and Call by Reference using a pointer
2. Define structures. Explain it with suitable examples
3. What are the 4 major file operations in C programming? Explain each of them in detail.
4. Describe the Basic structure of C Program
5. Draw a flowchart for Fibonacci series.
6. Write the Algorithms and Pseudo code for Fibonacci series
7. Draw flowchart to find whether the given number is even or odd
8. Write the Algorithm, Pseudo code and Program to find whether the given number is even or odd.

1. A Technique for algorithm design that tries to accommodate this human limitation is known as

a) Top-Down Design
b) Bottom up Design
c) Normal Design
d) Abnormal Design

2. Which one of this is not a Dynamic Programming Approach?

a) Greedy Search
b) Backtracking
c) Branch and Bound
d) Divide and Conquer Strategy

3. Finding the kth smallest element repeatedly reduces the problem of

a) Size
b) Element
c) Node
d) Thickness

4. Order of a notation is denoted by

a) Oh(n)
b) o(n)
c) O(n)
d) Ω(n)

5.The Applications of Sine functions are

a) Statistical Computations
b) Trigonometric Computations
c) Miniature problems
d) Allied Computations

6. The exponential growth constant e is characterized by the expression

a) e = 1/0! +1/1! +1/2!+ 1/3! + 0……..

b) e = 1/1! +1/2! +1/3! + 1/4! + 0……..

c) e = 1/1! +2/3! +3/4! + 4/5! + 0……..

d) e = 1! + 2! +3! + 4! + 5! + 0……..

7. Base conversion applications are

a) Interpretation
b) Compilation
c) Deduction
d) Analysis

8. When we convert a character to a number conversion then it proves for
   a) Tape processing
   b) Number processing
   c) Allied processing
   d) Input/output processing

9. Encryption is an application of
   a) Raising a number to a larger power
   b) Pseudo Random Numbers
   c) Prime Factor denotation
   d) Fibonacci Series

10. Prime Numbers should be considered as
   a) Candidate Divisors
   b) Greatest Common Divisors
   c) Lowest Common Factors
   d) Polynomial Divisors

11. Analysis of Algorithms, Simulation Problems and Games are applied by
   a) Generation of Pseudo Random Numbers
   b) Raising a number to a larger power
   c) Pseudo Random Numbers
   d) Prime Factor denotation

12.The factoring up of numbers can be made upto
   a) 6 digits
   b) 5 digits
   c) 4 digits
   d) 3 digits

13 . More advanced computer Applications arrays can be used to build
   a) Finite State Automata
   b) Pseudo Random Number
   c) Prime Factor denotation
   d) Fibonacci Series

14. The Application of Array Order Reversal is
   a) Vector and Matrix Processing
   b) Determinant Processing
   c) In – Order Traversal Processing
   d) Post – Order Traversal Processing

15. amod47 = 48, then a is
   a) 94
   b) 95
   c) 96

d) 97

16.Array counting is otherwise known as

a) Histogramming
b) Masking
c) Array processing
d) Determinant Processing

17. For small data sets, which one is most purposefully useful?

a) Quicksort Algorithm
b) Bubblesort Algorithm
c) Heapsort Algorithm
d) Radixsort Algorithm

18. The randomly ordered set of n numbers sort them into non –decreasing order using

a) Shells diminishing decrement insertion method
b) Shells diminishing supplement insertion method
c) Shells diminishing increment insertion method
d) Shells diminishing min-max insertion method

19. Works well on using this method of sorting_____ with more advanced methods

a) Large data sets
b) Small data sets
c) Complex data sets
d) Hybrid data sets

20. Two way merge processing is applied in

a) Allied processing
b) Number processing
c) Tape processing
d) Input/output processing

PART – B (5x2 = 10 Marks)
Answer ALL the questions
ALL THE QUESTIONS CARRY EQUAL MARKS

21. Write the advantages of pointer.
22. Write the general form of structure of C functions.
23. What is an escape sequence?
24. What is symbolic constant (pre-processor constant)?
25. What is string constant?

PART– C (5x14 = 70 Marks)
Answer ALL the questions
ALL THE QUESTIONS CARRY EQUAL MARKS

26. (a) Explain the basic structure of a C program with example (14)

[OR]

(b)Write a C program to compute Simple Interest. Draw the flow chart for the same (14)

27.(a) Write short notes on

i) If Else Statement iv) For loop vii) Continue & Break statement
ii) While Loop v) Switch statement
iii) Do While Loop vi)Nested If Else Statement (14)

[OR]

(b)i)Write a C program to find the Factorial of a given number using do .. While loop (7)
ii) Write a C program to find the Fibonacci of a given number (7)

28.(a) What is Array? Explain 1D and 2D with suitable Examples? (14)

[OR]

(b)Write Short notes on

i) StrCmp()
ii) StrCpy()
iii) StrCat()
iv) StrLen()
v) Strtev() (14)

29.(a) Define function. Illustrate the working function with suitable examples (14)

[OR]

(b)Explain the concept of pass by value and pass by reference. Write a C program to swap the content of two variables using pass by reference (14)

30. (a)Define structures. Explain it with suitable examples (14)

[OR]

(b) Explain about files and with it types of file processing. (14)

Answer ALL the questions
ALL THE QUESTIONS CARRY EQUAL MARKS

1. A program will handle all variations of the entire problem in an effort to be sure both the limiting and unusual cases is
   a**)** Verification
   b) Validation
   c) **Testing**
   d) Debugging

2. Two measures of performance behaviors considered are
a) Best and Average
   b)Average and Worst
   c) Worst and Good
   d) Good and Best

3. In P∩Q , P is called as…..
   a) **Assumption**
   b) Declaration
   c) Execution
   d) Implication

4.A false set of instructions which demands an informal notation is
   a**) Pseudo – Code**
   b) Algorithm
   c) Flow Chart
d) Data flow

5. The application of Mathematical proof techniques to establish the results is
   a**) Program Verification**
   b) Program Deduction
   c) Program Execution
   d) Program Assertion

6. The maximum problem complexity corresponds for a given problem is
a) **Worst Case behavior**
b) Average Case behavior
c) Best Case behavior
d) Good Case behavior

7. Reversing the digits of an integer applies on
a**) Hashing**
b) Counting
c) Summation
d) Computer Data

8. Choose the correct one

	Character	8 bit code	Decimal Value
a)	**0**	**00110000**	**48**
b)	3	0011110053	
c)	5	0011110057	
d)	7	0011001164	

9. 60mod48 =

a) 12
b) 18
c) 24
d) 30

10. The original doubling method is not attractive as the final algorithm because it requires

a) Separate Storage
b) Adequate Storage
c) Excessive Storage
d) Proper Storage

11. The sequences that are predictable are called as

a) **Pseudo Random Sequences**
b) Pseudo Generator Sequences
c) Pseudo Number Sequences
d) Pseudo code Sequences

12. x^23 is generated_____of the power

a) less than one
b) greater than one
c) multiple of one
d) conjugate of the above

13. Statistical Analyses is applied by

a) Histogramming
b)Dynamic Programming
c) Greedy Technique
d) Branch and Bound

14.Data Compression and Text processing problems are implications of

a) Removal of Duplicates from ordered arrays
b) Removal of Duplicates from unordered arrays
c) Adding of Duplicates from ordered arrays
d) Adding of Duplicates from unordered arrays

15.A monotone increasing subsequence is strictly increasing from

a) Left to right
b) Right to left

c) Always from left to left
d) Always form right to left

16.Non linear data structures are
a) Trees and Graphs
b) Trees
c) Graphs
d) None of the above

17. Which one has got a better complexity?
a) n
b) n^2
c) n log n
d) n^2 / n log n

18.In Quick sort algorithm by nature is
a) Non - Recursive
b) Procedural
c) Recursive
d) Non – Procedural

19. Finding the median and percentiles are inferred by
a) k smallest element
b) Monotone substance
c) Array partition
d) Maximum number of a set

20. In hash searching, the performance of a hash searching algorithm can be characterized by the n, where the function of the fraction of occupied locations and load factor is denoted by
a) β
b) γ
c) α
d) θ

PART – B (5x2 = 10 Marks) (2 ½ hours)
Answer ALL the questions
ALL THE QUESTIONS CARRY EQUAL MARKS

21. Given an array int a[10]={101,012,103,104,105,106,107,108,109,110}. Show the memory representation and calculate its length.
22. Define string.
23. What is the basic file operation in C?
24. Is it possible to create your own header files?
25. Define Dynamic memory allocation

PART– C (5x14 = 70 Marks)
Answer ALL the questions
ALL THE QUESTIONS CARRY EQUAL MARKS

26. (a) Write the Basic Anatomy of computer system (14)

[OR]

(b) Write the Algorithm, Pseudocode and Program for weather the given number is even or odd. (14)

27. (a) i) Write a C program to find the Factorial of a given number using do .. While loop (7)
ii) Write a C program to find the Fibonacci of a given number (7)

[OR]

(b) i) Write a C program to find wheather given number is Even or Odd (7)
ii) Write a C program given number is Prime or Not (7)

28. (a) i) Write a C program for Inseration Sort (7)
ii) Write a C program for Selection Sort (7)

[OR]

(b) What is Array? Explain 1D and 2D with suitable Examples? (14)

29. (a) Define function. Illustrate the working function with suitable examples (14)

[OR]

(b) i) Write a C program to Factorial using recursion (7)
ii) Write a C program to Fibonacci using recursion (7)

30. (a) Define Pointer . Explain Call by Value and Call by Reference using a pointer (14)

[OR]

(b) Explain about files and with it types of file processing. (14)

****