DIRECTORATE OF DISTANCE EDUCATION





VIDYASAGAR UNIVERSITY
MIDNAPORE-721 102

M. Sc. in Mathematics
Part - II
Paper - VI
Module No. 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72

CONTENTS

Sl. No.	Module	Paper	Page No.
1.	Module No. 61	VI	1
2.	Module No. 62	VI	32
3.	Module No. 63	VI	64
4.	Module No. 64	VI	99
5.	Module No. 65	VI	123
6.	Module No. 66	VI	149
7.	Module No. 67	VI	176
8.	Module No. 68	VI	196
9.	Module No. 69	VI	232
10.	Module No. 70	VI	261
11.	Module No. 71	VI	281
12.	Module No. 72	VI	345

M.Sc. Course
in
Applied Mathematics with Oceanology
and
Computer Programming
Part - II
Paper - VI

Group - A **Marks - 50**

COMPUTER SCIENCE - I

COMPUTER ORGANIZATION

Module No. 61

STRUCTURE :

61.1 Introduction

61.2 Objectives

61.3 Key Words

61.4 Evaluation of Computers

61.5 Hardware, Software and Firmware

- 61.5.1 Introduction to computer programming
- 61.5.2 The operating system

61.6 Basics of Computer Architecture

61.7 Computer Structures

- 61.7.1 General register machines
- 61.7.2 Accumulator based machines
- 61.7.3 Stack machine

61.8 Flip-Flops

- 61.8.1 SR filp-flop
- 61.8.2 D flip-flop
- 61.8.3 JK flip-flop

61.9 Digital Components

- 61.9.1 Decoders
- 61.9.2 Multiplexers
- 61.9.3 Registers

61.10 Execution Unit

- 61.10.1 Adders

61.11 Memory Organization
61.12 Auxiliary Memory
61.12.1 Magnetic disks
61.12.2 Magnetic tape
61.12.3 Virtual memory
61.13 Multiprogramming
61.14 I/O organization
61.15 Direct Memory Access (DMA)
61.16 Module Summary
61.17 Self Assesment Questions
61.18 Further Suggested Readings

61.1 Introduction

Computer organization is a very essential topic in computer science. In this subject the logical configuration of computer structure, introduced by Von Neumann, are stndied. The fundamental concepts on computer architecture, the logical idea on different kind of memories, concept of multiprogramming are essential for every computer user. The knowledge on different digital components, such as flip-flops, registers, decoders, multiplexers, etc. are reguired. All these things are discussed in this module.

61.2 Objectives

Gone through this module the students will learn the following objectives.

- Generations of computer
- Von Neumann architecture
- Hardware, software and firmware
- Stored program computer system
- Execution of program
- Computer structures
- Flip-flops
- Digital components : Decoders, multiplexers, registers
- Adders
- Internal, primary and secondary memories
- Memory addressing

- Cache memory
- PROM and EPROM
- Magnetic disks and magnatic tape
- Virtual memory
- Multiprogramming
- I/O organization
- Exercise

61.3 Key words

Von Neumann architecture, Computer structures, Digital components, Adders, Memories, Magnetic disk and tape, Multiprogramming.

61.4 Evolution of Computers

In a computer, use of certain components to perform logical functions, repetitive functions and to store data and instructions, made a lots of difference in the capacity of the computer. The advancement of electronic computer technology is generally classified into four categories called the first, second, third and the four generations.

First generation computers (1945-1955)

The main component in first generation computer were the vacuum tubes, commonly known as valves. These vacuum tubes consumed a large amount of electric power, generated a huge amount of heat and were bulky and less reliable. The first computer used 17000 vacuum tubes, weighted 30 tons and needed a large area for its installation of a big hall. The computation speed was in the range of 3000-5000 additions per second. The first electronic computer, called ENIAC was invented in 1946 at the University of Pennsylvania. The additions and other operations were carried out by special wiring rather than by programming. The concept of programs stored in memory became feasible later.

Von Neumann designed the first conceptual architecture of a stored-program computer, called the EDVAC. Within a small duration M.V.Wilkes of Cambridge University implemented the EDSAC, the first operational stored-memory computer. Using the stored-program concept, Von Neumann designed a new, unique and feasible architecture called the Von Neumann architecture. All modern computers still use this architecture.

A typical Von Neumann computer is shown in the Figure 1.

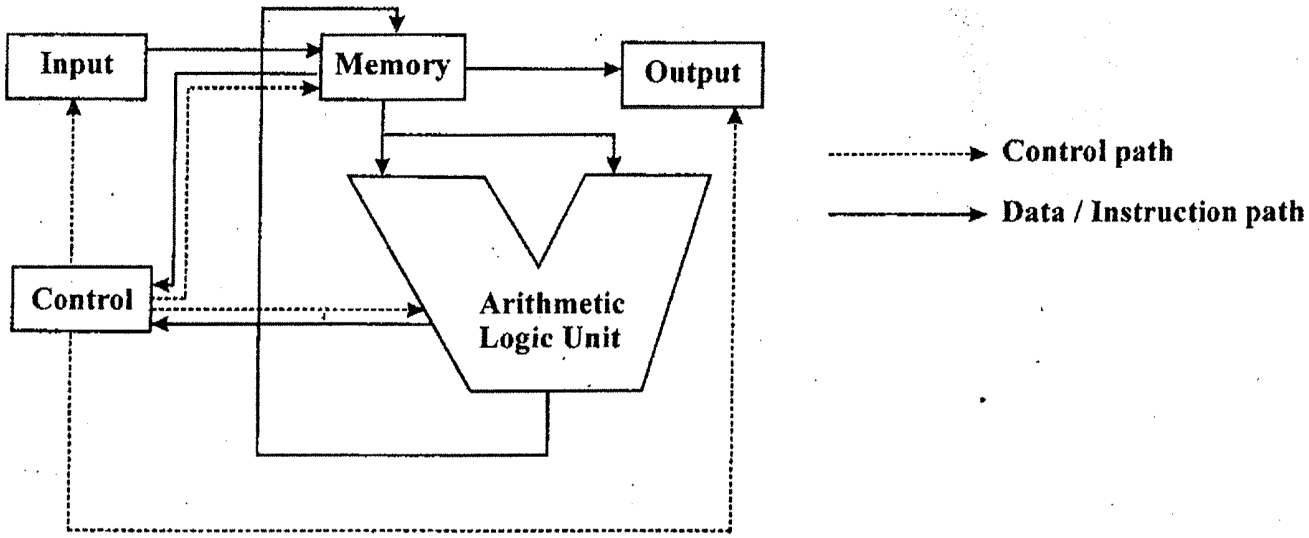


Figure 1. Von Neumann Architecture.

It consists of five basic elements :

- Input
- Memory
- Control Unit (CU)
- Arithmetic Logic Unit (ALU)
- Output.

The communication between human users and computer is done through the input and output units. The unit through which the computer reads data or code (program) is called input unit. The commonly used input device is keyboard. The modern computer uses mouse, joystick, magnetic pen, scanner, camera etc. as input devices.

The information received from input unit are stored in a unit which is known as memory. The memory is divided into two categories :

(i) main memory or primary memory, and (ii) secondary memory.

The primary memory is extremely fast and informations can be stored / retrieved at random in/from this memory. In modern systems, semiconductor devices are used as memories.

The common type of semiconductor memories are Rondom Access Memory (RAM) and Read Only Memory (ROM). RAM is also called read/write memory. It is volatile, i.e., all the information stored in it will be lost when the electrical power to it is cut off. Generally, user programs, data and a part of operating system are stored in RAM.

On the other hand, data can only be read from ROM but cannot be wirtten there. It is a non-volatile memory. ROM is generally used to store all information necessary for the operation of the computer. The secondary memory are floppy disk, hard disk, compact disk (CD), etc. The fundamental unit to measure memory is byte, 1 byte is equal to 8 bits. Other units are Kilobyte (KB), (1 KB = 1024 bytes), Megabyte (MB), (1 MB = 1024 KB), Giga byte (GB), (1 GB = 1024 MB), etc.

The arithmetic logic operations such as ADD, SUBTRACT, OR, AND, and NOT. The control unit (CU) acts as a supervisor in a computer system. It works in two phases, in a cyclic manner. Theses phases are (i) instruction phase, and (ii) execution phase. In instruction phase an instruction / command is received from the main memory. In the execution phase the command is executed.

All the results which are stored in the memory can be inscribed on an output unit. There are several output devices available with a modern computer. Commonly used putput devices are VDU terminal, printer (dot matrix, ink-jet, laser etc.), plotter etc.

To execute instructions, the computer performs the following steps :

- The CU reads or fetches an instruction from memory and decodes or translates it.
- For arithmetic or logic-type instructions, the CU generates enable signals for the ALU to perform the required operations.
- For I/O instructions, the CU generates enable signals for the I/O either to input data from or output data to external devices.

The Vow Neumann architecture was the first computer that allowed storing of instructions and data in the same memory. The memory of this computer was 4096 (2^{12}) words, which were 40 bits wide.

Second generation computers (1955-1965)

The discovery of transistor in 1948 at the Bell Laboratories brought in a great deal of changes in computer architecture. A transistor is a tiny semiconductor device of size 1/50th that of a valve and could perform most of the functions of a vacuum tube but was much faster with only a fraction of energy consumption and with the generation of only a small amount of heat. The use of transistors has reduced the size, manufacturing costs, and running costs and improved reliability and processing power of the computer. These computers had calculating speed ranging from 2000 to 50,000 additions per second. IBM, NCR, RCA and orther companies designed fully transistorized computers, and the concept of real-time computers evolved. Real-time computers are loosely defined as the computers that provide fast responses to process request.

The first computer for real-time application was designed at MIT in 1951. The computer was called WHIRLWIND-I (WWI), and its main areas of application were in process control and aircraft simulation. WWI had a 16-bit word length with 27 operation codes. It was capable of performing 20,000 instructions in 1 second. WWI was the first core-memory-based computer. Before this, electrostatic storage tubes had been used for memory.

No operating system (OS) had used in these computers and were obviously not user-friendly. The OS in the collection of programs, ueually written by the manufacturer, that controls the computer system. The OS allows the operator to communicate with the computer.

The computer industry made major strides in the 1950's. In 1951, the first tape-based system UNIVAC-I (universal automatic computer) was released by Eckert and Mauchly. This tape could be read forward or backward, and error-checking capabilities were included. UNIVAC-I used a mercury delay line for memory, which quickly made it obsolate. UNIVAC-II, a core-memory-based computer, was soon released. It was primarily designed for commercial applications.

In 1953, IBM developed a desk calculator called the IBM-701. The IBM-701 was designed using Von Neumann architecture with 2K 36-bit words of unreliable electrostatic memory. IBM's next computer, the 704, was announced in 1956 with 4K words of core memory, three index registers, and hardwared floating-point arithmetic.

In 1954, IBM announced its first drum-based computer, called the IBM 650. With the introduction of drum-based computers, the use of assemblers, compilers and other system-oriented programs become feasible.

Third generation computers (1965-1970)

In late 1950's, a large number of transistors were put together on a single thin chip of silicon. These are called Integrated Circuit (IC), commonly known as chip. These chips could be as small as $\frac{1}{16}$ th of an square inch, weighs a small fraction of an ounce only, requires only a bit of power and its production cost is very low. A small chip is equivalent to 200 to 2000 transistors. Some chips were also developed to store data. These constitute the memory chips. These computers were smaller in size and greater in capacity and performed as many as ten million additions per second.

IBM designed the IBM-360 scientific computer in the mid-1960's. This computer had more memory and I/O devices than earlier computers and was faster.

In 1970, IBM announced system 370, which is similar to the 360 except for a different circuit technology. Several other manufactures, such as RCA and UNIVAC, introduced computers based on the IBM-360 architecture.

Since the 1960's, Digital Equipment Corporation (DEC) has developed its program data processor (PDP) line of computers, such as the PDP 8 and PDP 11. The DEC computers provided the basis for future trends in real-time mini computers.

Fourth generation computers (1970 onwards)

From 1970 onwards, microprocessors were introduced, which are programmable, very large scale integrated circuit chips that can process the data.

In an attempt to create a general-purpose computational chip set, the Intel corporation designed the first microprocessor, the 4004, in 1969. The Intel 4004, control unit circuitry, and registers that define current microprocessors. The Intel 4004 was rapidly replaced by the Intel 8008 and then the Intel 8080/8085. Very quickly, other 8-bit microprocessors such as the Motorola 6800/6809, Zilog Z80 and MOS 6502 appeared.

Intel 8080/8085 was the microprocessor that provided the basis of the future trends in 8-bit microprocessors-Typical 8-bit microprocessors use an 8-bit data path and a 16-bit address range. The powerful instruction set (for its size, price and power consumption) of 8-bit microprocessors allowed the development of general-purpose micro computer systems. Advances in integrated circuit (IC) technology have led to the development of powerful 16-bit (Intel 8086 and Motorola 68000) and 32-bit (Intel 80386 and Motorola 68020) microprocessors. Micro computers designed using these microprocessors have the capability of processing programs written in high-level languages. In addition, a 16-bit or 32-bit microprocessors can directly address memory space in the order of 1MB to 4 GB.

Most of the first system monitors and application programs for micro computers were written in machine, or assembly language. As the power of micro computers grew and the price dropped, high-level languages were developed for use on microcomputers. One of the first microcomputer high-level languages was PL/M, which was developed for the Intel 8080. PL/M was a powerful block structured language, which was then used to create further system software such as the defacto standard for micro computer OS, CP/M.

The current trend for microprocessor and mini computer applications is towards distributed systems, where several microprocessors usually perform dedicated real-time activities. These microprocessors are connected to a host minicomputer via a data bus such a coaxial cable. High reliability is one of the main characteristics of such a system.

61.5 Hardward, Software and Firmware

Hardware refers to the physical components of a computer such as memory, CPU, transistors, nuts and bolts, and so on. When a computer is said to have the hardware capability to add, it has a hardwared electronic circuit that is capable of adding two numbers. Changes in the hardward cannot be accomplished by programmers.

Software consists of a collection of programs. Programs contain a sequence of instructions for performing a specified task. Firmware falls between hardware and software. Firmware consists of programs that are included in electronic circuits during their manufacture. Firmware is used when the programs must be retained in case of power failure and when programs are not expected to be changed.

In general, most computers are regarded as clocked sequential systems. In these computers, almost all activities pertaining to instruction execution are sychronized with master clocks.

Those operations that can be executed within one clock period are referred as microoperations or microinstructions. A program written using such micro-instructions is called a microprogram. This microprogram

is held in a separate memory called the control memory. It is executed in the same way as a conventional program is executed, by reading microinstructions from the read-only memory (ROM) inside the CU. This concept is called microprogramming and was originated by M.V. Wilkes. This technique allows greater flexibility and a structured approach in desigining CPU control algorithms, floating point arithematic routines and diagonostic routines.

61.5.1 Introduction to computer programming

A digital computer is capable of storing large amounts of information that include instructions and data. The computer executs these instruction and uses the data to produce results. A set of instructions and data specifying the solution of a particular problem is called a program.

The computer uses memory to store programs in binary. The computer can excute only programs that are represented in binary forms. A program written using binary numbers is called a machine language program. But, writting a program in machine language is difficult. Therefore, most computer manufacturers include a program called an assembler with the computer system. The assembler translates a program written in a symbolic programming language (assembly language) into machine language. Assembly language programming is used when a programmer needs exposure to the internal operation of a computer. Assembly language programming is chosen under the following situations :

- When memory space is critical
- For short to medium sized programs
- For more control operations than computations.

Assembly language is not suitable for writing well-structured, reliable programs. High-level languages, can be used efficiently to write structured programs. Most newer software is written in high-level languages. High-level languages require translators (compilers) for translating the high-level language program into machine language.

61.5.2. The operating system

The OS performs resource management and human-to-machine translation function. A resource may be the CPU, memory or an I/O device. Basically, an operating system is another computer program that tells the machine what to do under a variety of conditions. Major OS functions include efficient sharing of the memory, I/O peripherals and CPU among several users. In summary, an OS is

- An interface between hardware and users
- A manager of system resources in accordance with system policy to achieve system objectives.

61.6. Basics of computer Architecture

A computer is an electronic device capable of processing information. The computer processes information by executing a program consisting of a sequence of instructions stored in memory.

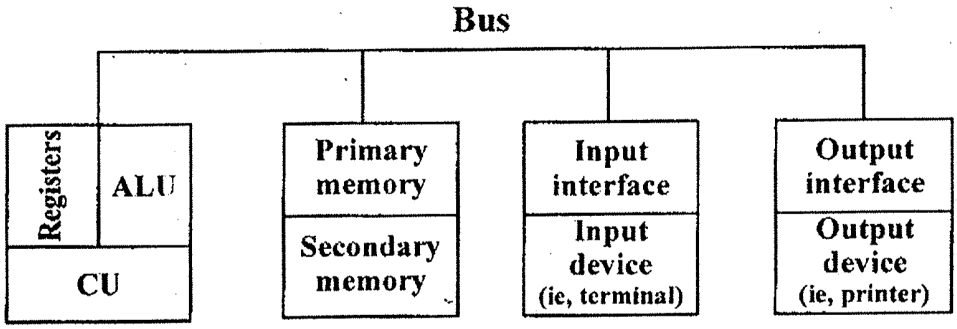


Figure 2. Organization of a stored-program computer system.

A computer system of this kind is called a stored-program computer because the instructions and data are held in the same memory.

Instruction execution takes place within the CPU and is confrolled by the CU. Typically, the CU has to perform the following takes to carry out the instruction execution process :

- instruction interpretation
- instruction sequencing.

In the instruction interpretation phase, the CU reads an instruction and recognizes the instruction operation code. It then retrieves the required operands or data items and performs the desired operation by activating the ALU. Finally, the result is routed to a specified destination. The CU determines the address of thenext instruction to be executed is stored in the memory word that immediately follows the word where the current instruction is held. Howerver, the following are samples of two exceptional situations to the sequential control flow mentioned earlier :

- Branch instruction
- Subroutine calls.

For the first case, the control unit jumps to an instruction stored in some arbitrary address B as the next instruction to be executed and continues from the jump address. The control unit begins executing a complete program starting at some address s for a subroutine call, and it returns to the next instruction after the subroutine-call instruction, when a return instruction is inserted in the body of the subroutine.

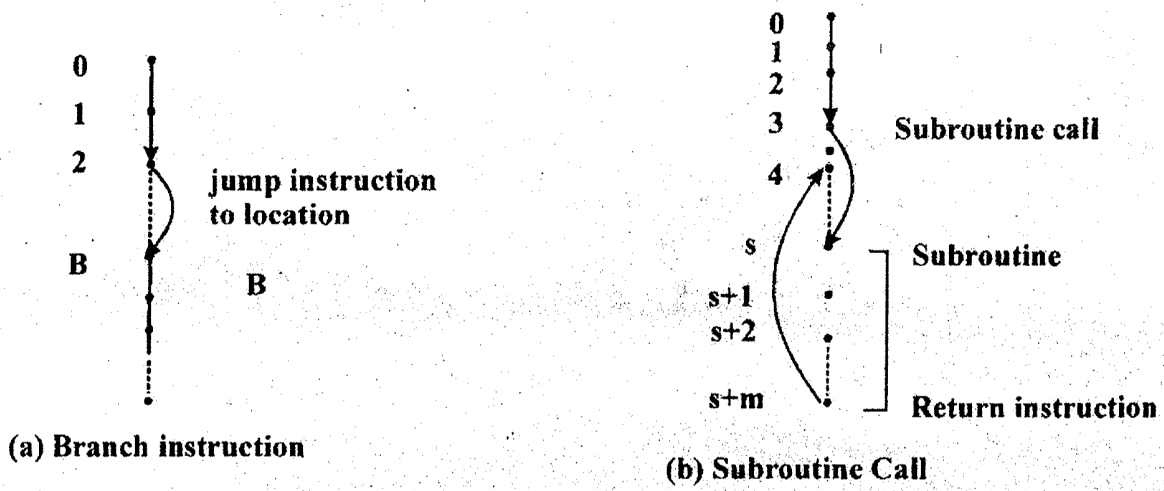


Figure 3 : Branch instruction and subroutine call.

The CPU of a computer includes a set of high-speed registers. These registers may hold data, temporary results, or memory address when a computation is in progress. The CU typically uses the following registers to accomplish a task :

Program counter (PC) : Holds address of the next instruction to be executed.

Instruction register (IR) : Holds the instruction currently being executed.

Effective address register (EAR) : Holds the address of the data to be retrieved from memory.

These registers are usually referred to as dedicated registers because they are available for the exclusive use of the control unit and cannot be accessed by a user program.

The key elements of a stored-program computer system are connected to each other by a bus. A bus consists of a set of wires carrying address, data and control information. The computer system bus may be divided into three subcategories, as follows :

- Address bus
- Data bus
- Control bus

61.7. Computer Structures

Most contemporary computers may be classified into three groups :

- General register machines
- Accumulator based machines
- Stack machines

61.7.1 General register machines

The organization of a general register processor is shown in Figure 4. There are eight general registers (R0 through R7). They are called general registers because any one of these registers can be resed to hold data, memory addresses, or the results of arithmetic or logic operations.

The stack pointer (SP) is a dedicated register that usually holds the address of the top element of the stack. The F register is known as flag register. It holds status flags such as the zero flag (z-flag) and the cary flag. These flags are usually updated after executing an arithmetic or logic instruction to indicate the status of a computation in progress. For example, if the z-flag is set to 1, the most recently executed instruction has produced a zero result.

General register processors support three and two address instructions. Some examples of three - address instructions are given below.

Instruction	Operation
ADD x, y, z	z ⟵ (x) + (y)
SUB y, x, z	z ⟵ (x) – (y)
MUL x, y, z	z ⟵ (x) * (y)
DIV y, x, z	z ⟵ (x) / (y)

Typical example of two-address instructions are as follows :

Instruction	Operation
MOV x, y	y ⟵ (x)
ADD x, y	y ⟵ (x) + (y)
SUB y, x	y ⟵ (x) – (y)
MUL y, x	x ⟵ (x) * (y)
DIV y, x	x ⟵ (x) / (y)

The symbols x, y, z refer to a memory word or a general register. z ⟵ (x) + (y) means the contents of x and y are added and the result is saved in the address z.

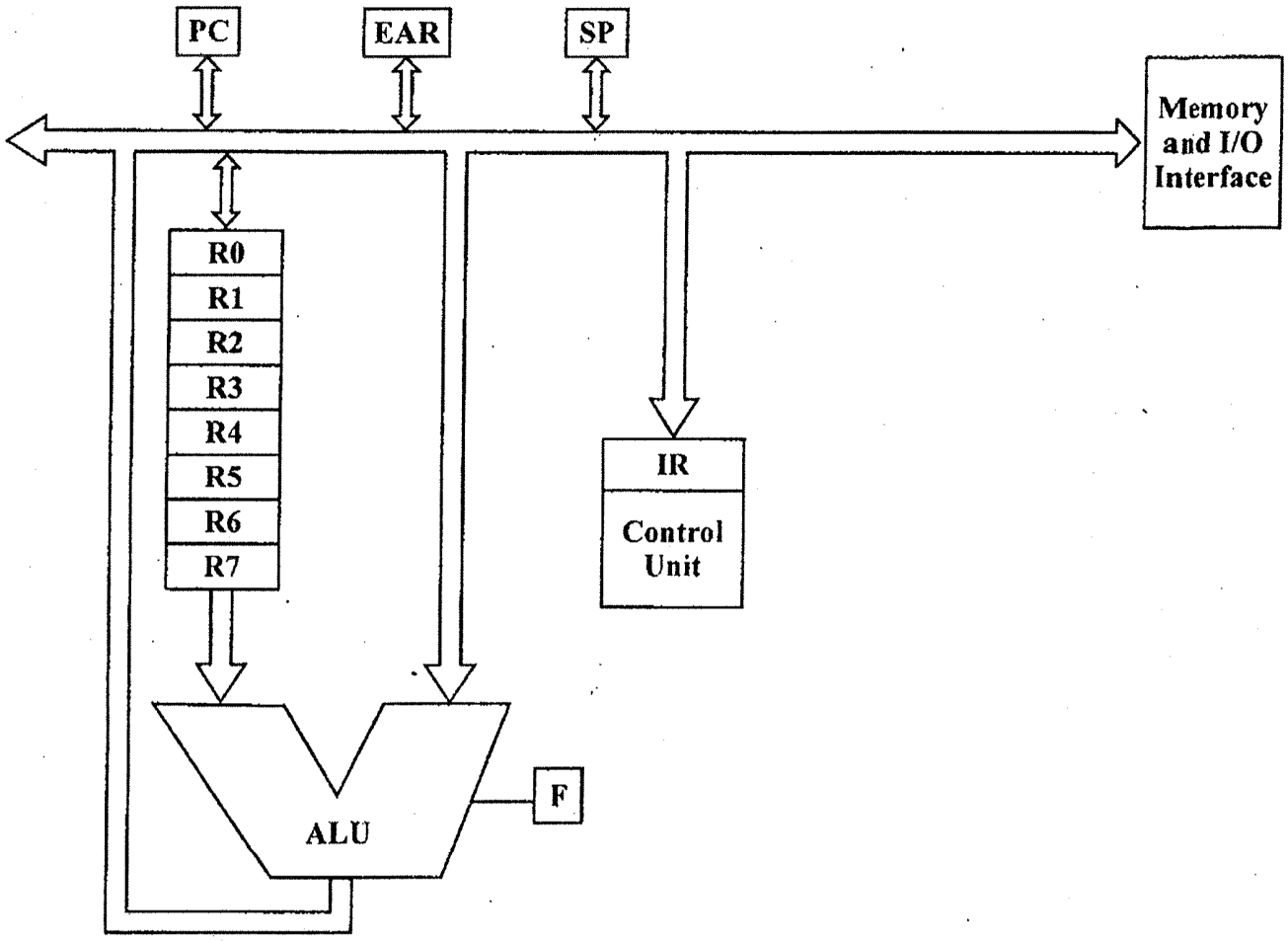


Figure 4 : Organization of a typical general register processor.

61.7.2 Accumulator based machines

The organization of a accumulator based processor is shown in Figure 5. One of the operands is assumed to held in the accumulator register (ACC) for arithmetic or logic operation. The results of all arithmetic and logic operations are routed to the ACC. One-address instructions are very predominant in this organization.

Some typical one-address instructions available in an accumulator-based machine are as follows :

Instruction		Operation
LDA	x	ACC ⟵ (x)
STA	x	x ⟵ (ACC)
ADD	x	ACC ⟵ (ACC) + (x)
SUB	x	ACC ⟵ (ACC) – (x)

Instruction		Operation
MUL	x	ACC ← (ACC) * (x)
DIV	x	ACC ← (ACC) / (x)

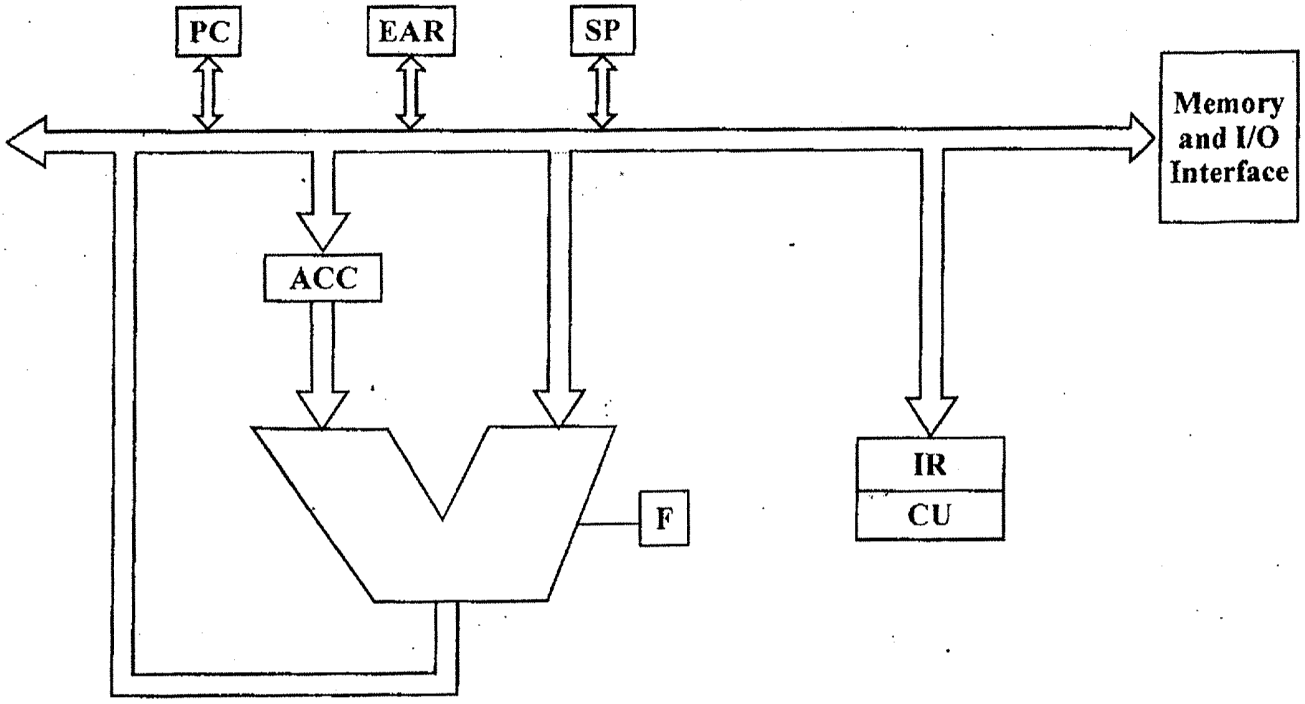


Figure 5 : Organization of a accumulator based processor.

In this system, these two registers may be combined to form one 16-bit accumulator for all 16-bit arithmetic and logic operations. The following example shows how the assignment statement D = A + B * C can be written using these instructions formats.

Three address

MUL B, C, D : D ← (B) * (C)

ADD A, D, D : D ← (A) + (D)

Two address

MOV B, D : D ← (B)

MUL C, D : D ← (C) * (D)

ADD C, D : D ← (A) + (D)

One address

LDA B : ACC ← (B)

MUL C : ACC ← (ACC) * (C)

ADD A : ACC ← (ACC) + (A)

STA D : D ⟵ (ACC)

61.7.3 Stack machine

A stack machine is a computer system is which the primary element of the CPU is a stack. The stack can be implemented by configuring some portion of the read-write memory as stack area. One of the CPU registers, called the stack pointer (SP), is used to hold the address of the most recently entered item into the stack. Hence, SP points to the top of the stack.

There are two basic operations associated with the stack. These are PUSH (writing into the stack), and POP (reading from the stack).

In this arrangement, the stack grows in the direction of lower addresses. This characteristic is typical of many processors, including the PDP-II, Intel 8085, Z80 and MC6809.

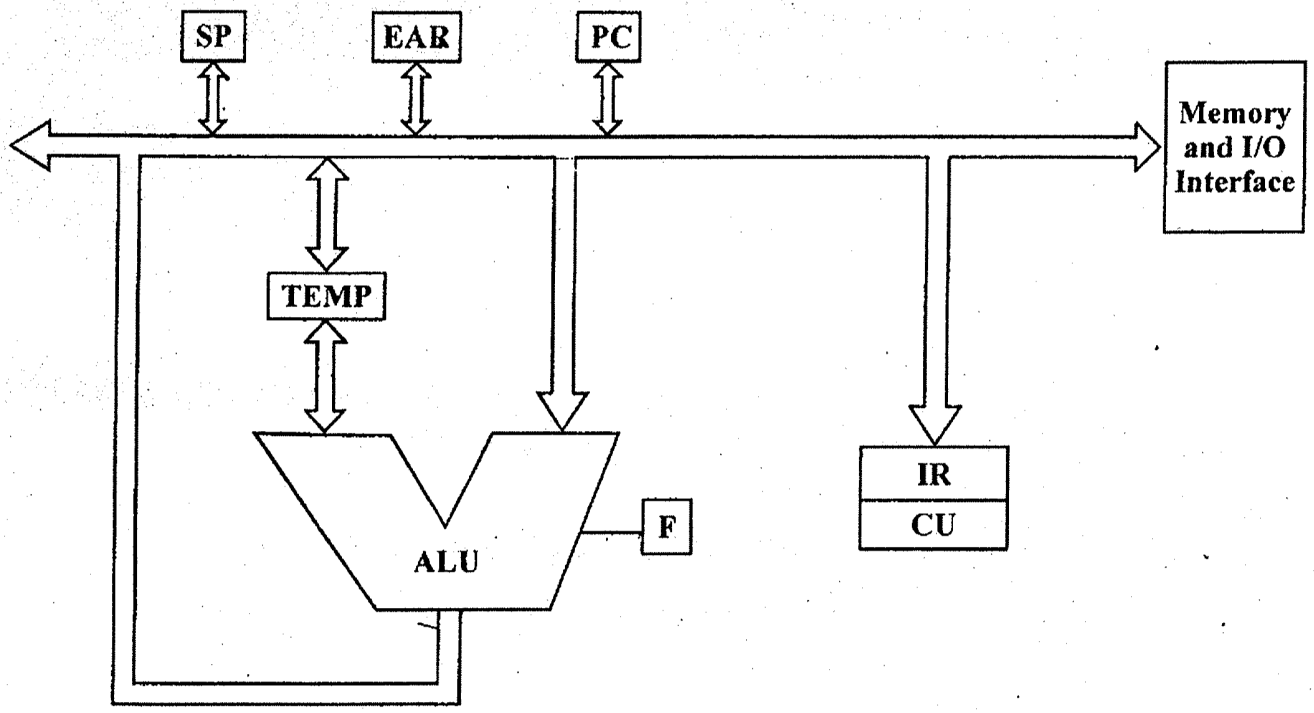


Figure 6 : Organization of a typical stack machine.

The following example illustrates how to model the assignment statement D = A + B*C using the instruction set.

PUSH A

PUSH B

```
PUSH   C
MUL        ;   Calculate   B * C
ADD        ;   Add         A to B * C
POP    D   ;   save result in D
```

One of the main advantages of the stack machine is its efficient evaluation of complicated arithmetic expressions. Typical examples of stack machines include HP 3000, B 1700 etc. General register machines, such as the PDP-II and MC 68000, possess the capabilities to perform some stack operations. A programme ran simulate the behavior of the stack machine on these processors by writting a suitable program :

Instruction	**Operation**
PUSH X	(TOS) ← (x)
POP Z	Z ← ((TOS))
ADD	(NOS) ← ((TOS) + (NOS))
SUB	(NOS) ← ((TOS) – (NOS))
MUL	(NOS) ← ((TOS) * (NOS))
DIV	(NOS) ← ((TOS) / (NOS))

TOS = top of the stack

NOS = next on the stack

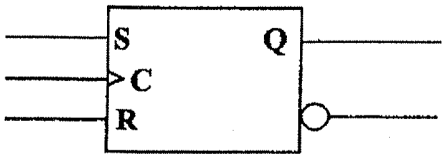
61.8 Flip- Flops

Every digital system in likely to have a combination circuit, most systems encountered in practice also include storage elements, which require that the system be described in terms of sequential circuits. The most common type of sequential circuit is the synchronous type. Synchronous sequential circuits employ signals that effect the storage elements only at discrete instants of time. Synchronization is achieved by a timing device called a clock pulse generator that produces a periodic train of clock pulses.

The storage elements employed in clocked sequential circuits are called flip-flops. A flip-flops is a binary cell capable of storing one bit of information. A flip-flop maintains a binary state until directed by a clock pulse to switch states.

61.8.1 SR flip-flop

It has three inputs, labeled S (for set), R (for reset) and C (for clock). It has an output Q and some times the flip-flop has a complemented output, which is indicated with a small circle at the other output terminal.



Graphical Symbol

S	R	Q(t+1)	
0	0	Q(t)	No change
0	1	0	Clear to 0
1	0	1	Set to 1
1	1	?	Indeterminate

Characteristic table

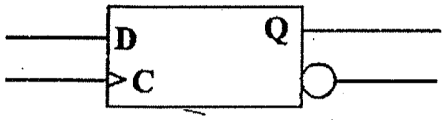
Figure 7 : SR flip-flop

There is an arrow head shaped symbol in front of the letter C to designate a dynamic input. The dynamic indicator symbol denotes the fact that the flip-flop responds to a positive transition (from 0 to 1) of the input clock signal.

The operation of the SR flip-flop is as follows. If there is no signal at the clock input C, the output of the circuit cannot change irrespective of the values at input S and R. Only when the clock signal changes from 0 to 1 can the output be affected according to the valnes in inputs S and R. If S=1 and R=0 when C changes from 0 to 1, output Q is set to 1, It S = 0 and R = 1 when C changes from 0 to 1, output Q is cleared to o. If both S and R are 0 during the clock transition, the output does not change when both S and R are equal to 1, the output is unpredictable and may go to either 0 or 1, depending on internal timing delays that occur within the circuit.

61.8.2 D flip-flop

The D (data) flip-flop is a slight modification of the SR flip-flop. An SR flip-flop is converted to a D flip-flop by inserting an inverter between S and R and assigning the symbol D to the single input. The D input is sampled during the occurrence of a clock transition from 0 to 1. If D = 1, the output of the flip-flop goes to the 1 state, but if D = 0, the output of the D flip-flop goes to the 0 state.



Graphic Symbol

D	Q(t+1)	
0	0	Clear to 0
1	1	Set to 1

Characteristic table

Figure 8 : D flip-flop

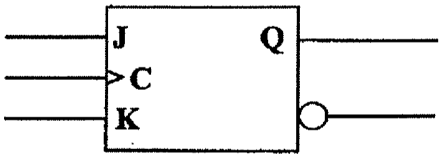
From the characteristic table we note that the next state Q(t+1) is determined from the D input. The relationship can be expressed by a characteristic equation

$$Q(t+1) = D.$$

This means that the Q output of the flip-flop receives its value from the D input every time that the clock signal goes through a transition from 0 to 1.

61.8.3 JK flip-flop

A JK flip-flop is a refinment of the SR flip-flop in that the indeterminate condition of the SR type is defined in the JK type. Input J and K behave like inputs S and R to set and clear the flip-flop respectively. When inputs J and K are both equal to 1, a clock transition switches the output of the flip-flop to their complement state.



Graphic Symbol

J	K	Q(t+1)	
0	0	Q(t)	No change
0	1	0	Clear to 0
1	0	1	Set to 1
1	1	Q′(t)	Complement

Characteristic table

Figure 9 : JK flip-flop

The J input is equivalent to the S(set) input of the SR flip-flop, and the K input is equivalent to the R (clear) input. Instead of the indeterminate condition, the JK flip-flop has a complement condition

$$Q(t+1) = Q'(t)$$

when both J and R are equal to 1.

61.9 Digital Components

61.9.1 Decoders

Discrete quantities of information are represented in digital computers with binary codes. A binary code of n bits is capable of representing upto 2^n distinct elements of the coded information. A decoder is a combinational circuit that converts binary information from the n coded inputs to a maximum of 2^n unique outputs. If the n-bit coded information has unused bit combinations, the decoder may have less than 2^n outputs.

The decoders with m outputs ($m \leq 2^n$) are called n–to–m line decoders. Their purpose is to generate the 2^n (or fewer) binary combinations of the n input variables. A decoder has n inputs and in outputs and is also referred to as n×m decoder.

The logic diagram of a 3–to–8 decoder is show in Figure 10.

The three data inputs A_0, A_1 and A_2 are decoded into eight outputs, each output representing are of the combinations of the three binary input variables. The three inverters provide the complement of the inputs, each of the eight AND gates generates one of the binary combination.

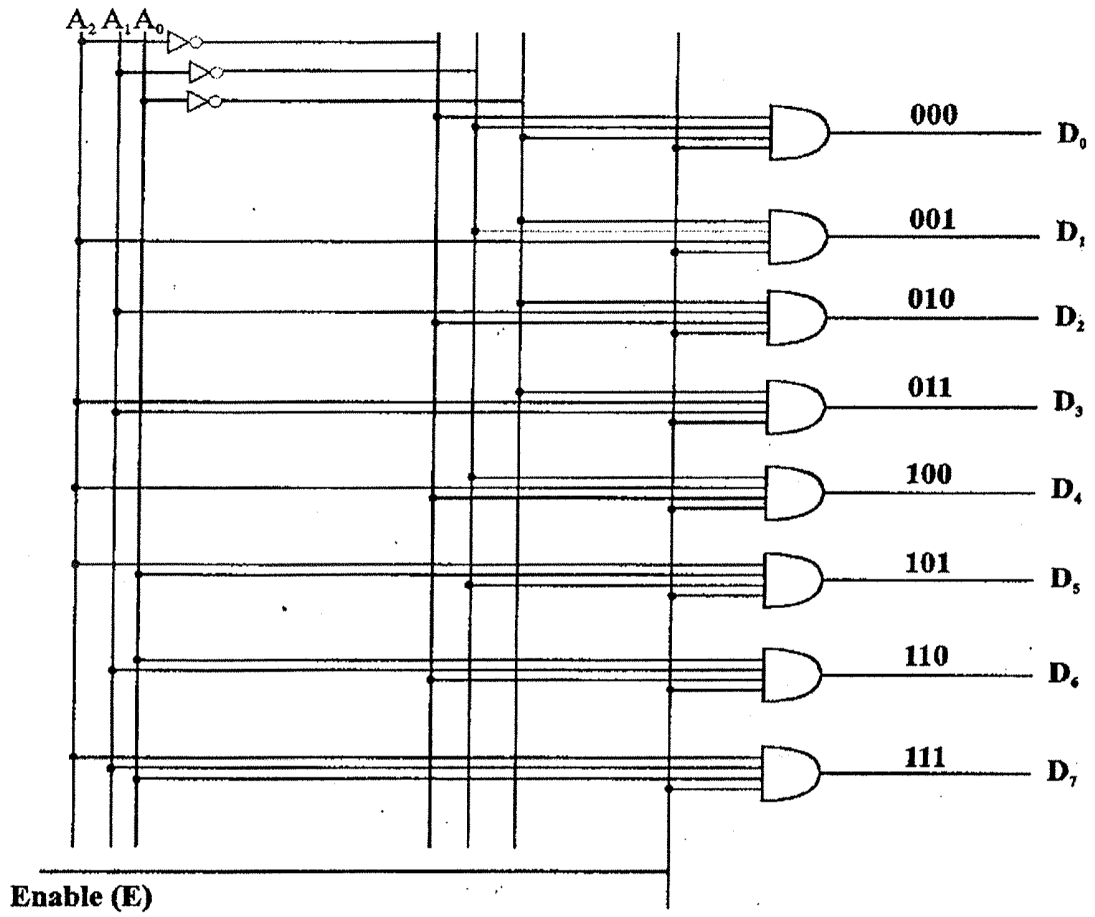


Figure 10 : 3–to–8 decoder.

A particular application of this decoder is a binary–to–octal conversion. The input variables represent a binary number and the outputs represent the eight digits of the octal number system. However, a 3–to–8 line decoder can be used for decoding only 3–bit code to provide eight outputs, one for each combination of the binary code.

Commercial decoders include one or more enable inputs to control the operations of the circuit. The decoder is enabled when E is equal to 1 and disabled when E is equal to 0.

Enable	Inputs			Outputs							
E	A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	×	×	×	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Figure 11 : Truth table for 3–to–8 line decoder.

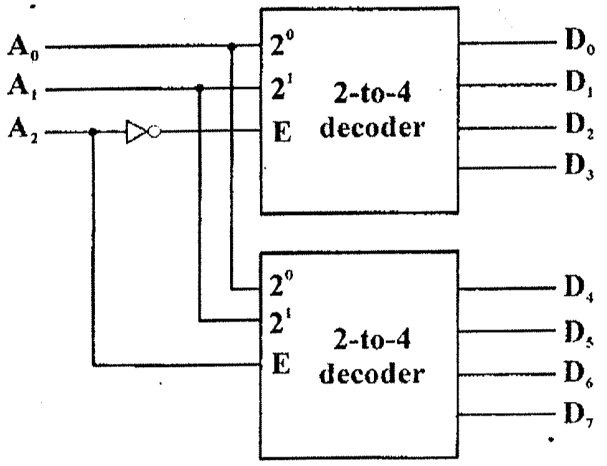


Figure 12 : A 3–to–8 decoder constructed with two 2–to–4 decoders.

61.9.2 Multiplexers

A multiplexer is a combinational circuit that receives binary information from one of 2^n input data lines and directs it to a single outpur line. The selection of a particular input data line for the output is determined by a set of selection inputs. A 2^n–to–1 multiplexer has 2^n input data lines and n input selection lines whose bit combinations determine which input data are selected for the output.

The 4–to–1 line multiplexer is shown in Figure 13.

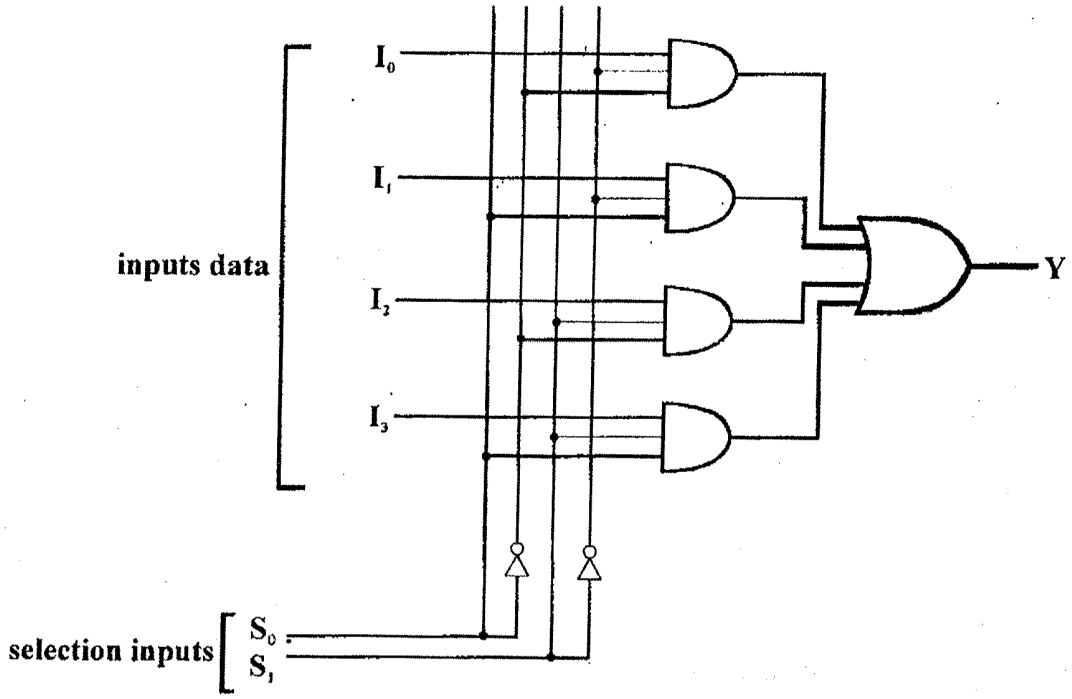


Figure 13 : 4–to–1 line multiplexer

The 4–to–1 line multiplexer has six inputs and one output. A truth table describing the circuit needs 64 rows since 6 input variables can have 2^6 binary combinations. A more convenient way to describe the operation of multiplexers is by means of a function table.

The table demonstrates the relationship between the four data input and the single output as a function of the selection inputs S_1 and S_0.

Select		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

61.9.3. Registers

A register is a group of flip-flops with each flip-flop capable of storing one bit of information. An n–bit register has a group of n flip-flops and is capable of storing any binary information of n bits. In addition to the flip-flops, a register may have combinational gates that perform certain data-processing tasks. Broadly, a register

consists of a group of flip-flops and gates that effect their transition. The flip-flops hold the binary information and the gates control when and how new information is transferred into the register.

Various types of registers are available commercially. The simplest register is one that consists only of flip-flops, with no external gates. Following is a register constructed with four D flip-flops.

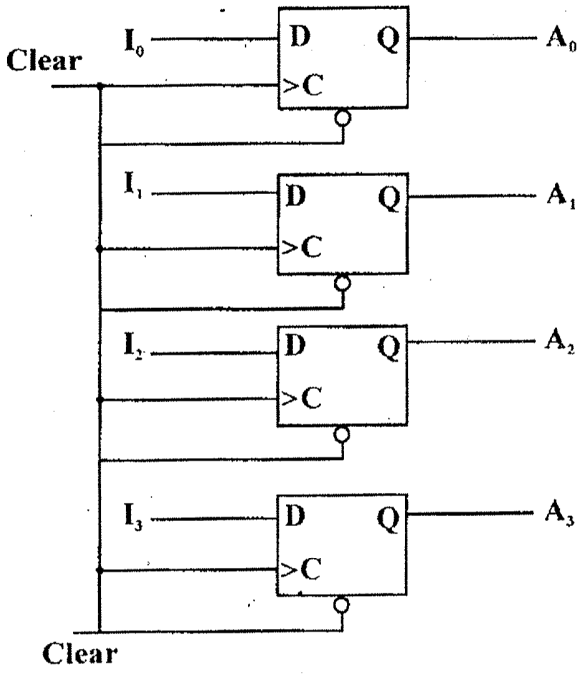


Figure 14 : 4-bit register.

61.10 Execution Unit

The typical components of a computer execution unit are

- General purpose registers
- ALU
- Dedicated hardware or firmware to perform special operations such as multiplication and division.

A conventional CPU consists of the following :

- General purpose registers
- Dedicated registers
- An ALU
- Dedicated hardware or firmware elements that perform a special operation such as multiplication or

division

- A control unit.

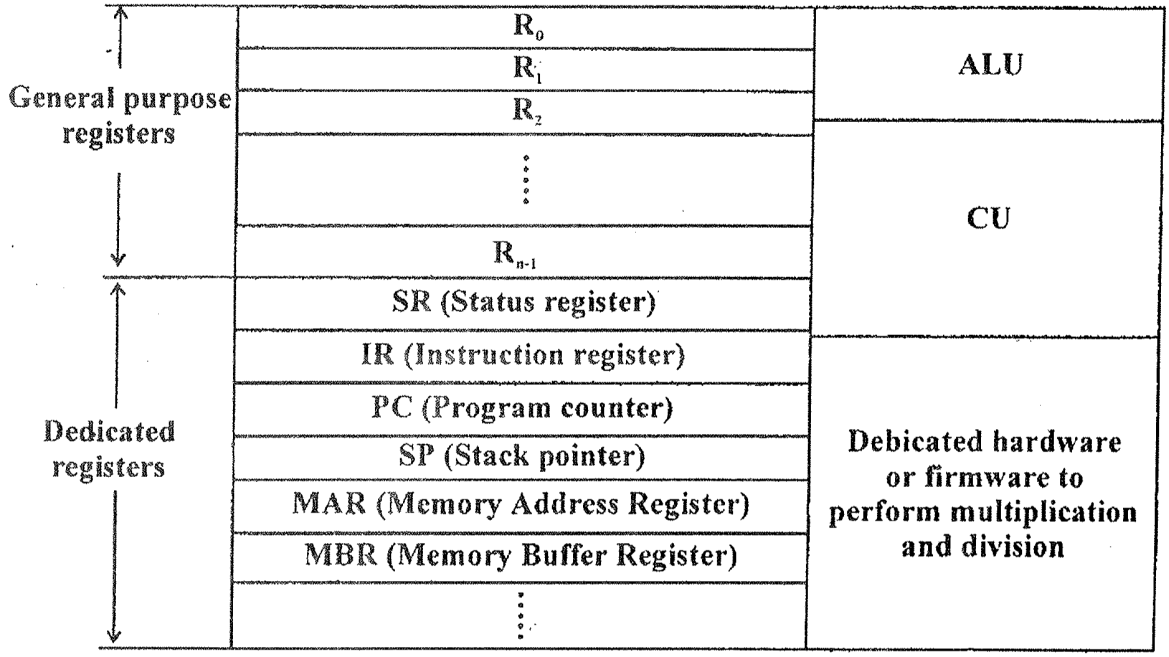


Figure 15 : A typical CPU model.

Some commonly known special-purpose registers and their tasks are listed below.

Register	Tasks
PC	Usually holds the address of the next instruction to be executed.
SP	Usually holds the address of the top element of the task.
IR	Holds the instruction code currently being executed.
MAR	Holds the address of the data item tobe retrieved from the main memory.
MBR	Holds the data item retrieved from the main memory.
SR of PSW	Holds the condition code flags and other information that describe the status of the running program.

61.10.1 Adders

Addition is the basic operation performed by an ALU. The operation is versatile, since A–B is equivalent to A+2's complement of B, and C*A may be obtained by adding A to itself for C–1 times. Thus the spced of the

adder unit is essential to the efficient operation of an execution unit.

A hardware unit that adds two single-bit numbers with a carry is called a full adder.

x_i	y_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Truth table

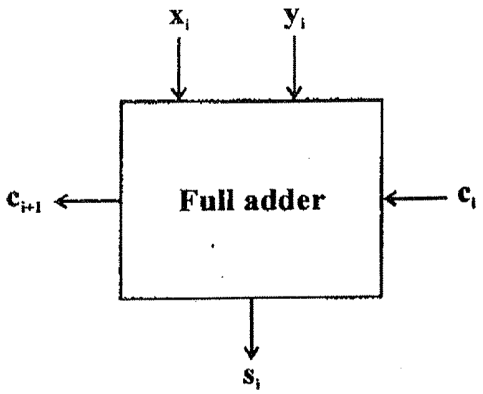
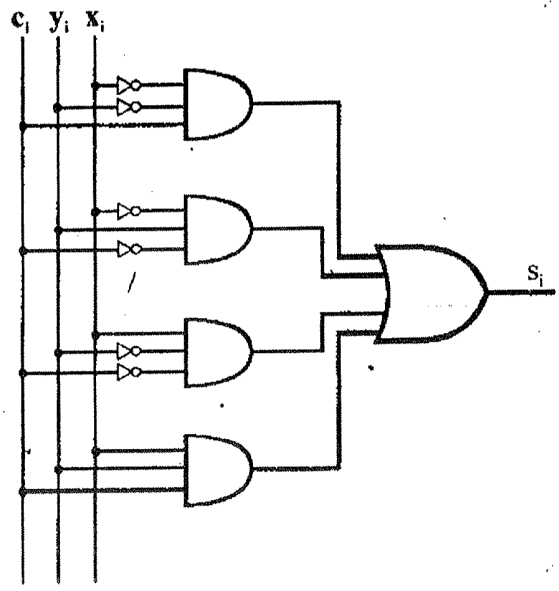
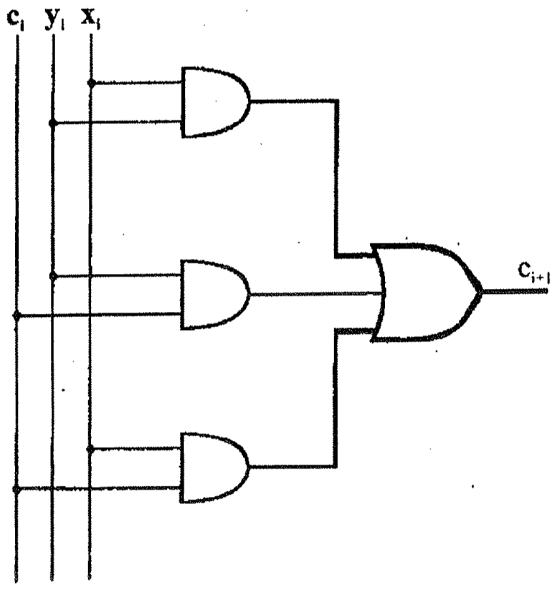


Figure 16 : Block diagram of full adder.



$s_i = x_i \oplus y_i \oplus c_i$



$c_{i+1} = x_iy_i + x_ic_i + y_ic_i$.

Figure 17 :

From the truth table

$$s_i = x_i' y_i' c_i + x_i' y_i c_i' + x_i y_i' c_i' + x_i y_i c_i$$

$$= c_i (x_i' y_i' + x_i y_i) + c_i' (x_i' y_i + x_i y_i')$$

$$= c_i z_i' + c_i' z_i \qquad \text{where } z_i = x_i' y_i + x_i y_i' = x_i \oplus y_i$$

$$= z_i \oplus c_i = x_i \oplus y_i \oplus c_i$$

also

$$c_{i+1} = x_i' y_i c_i + x_i y_i' c + x_i y_i c_i' + x_i y_i c_i$$

$$= x_i y_i + x_i c_i + y_i c_i.$$

From figures we see that to generate c_{i+1} from c_i, two gate delays (two stages) are required. In order to generate s_i from c_i, three gate delays are required, since c_i must be inverted to obtain c_i'. Note that no inverters are required to get x_i' or y_i' from x_i or y_i respectively, since the numbers tobe added are usually stored in a register that is a collection of flip-flops. The flip-flop often generates both normal and complemented outputs. Assume that the gate delay is Δ time units, and the actual value of Δ is decided by the technology. For example, if Transistor Translator Logic (TTL) circuits are used, the value of Δ will be 10ns.

By cascading n full adders, an n-bit parallel adder capable of handling two n-bit operands (x and y) can be designed. The implementation of a 4-bit ripple-carry adder is shown below.

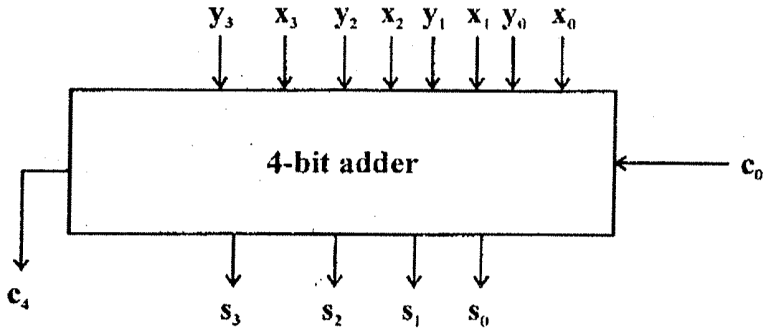


Figure 18 : Block diagram of a 4-bit Ripple-Carry-Adder.

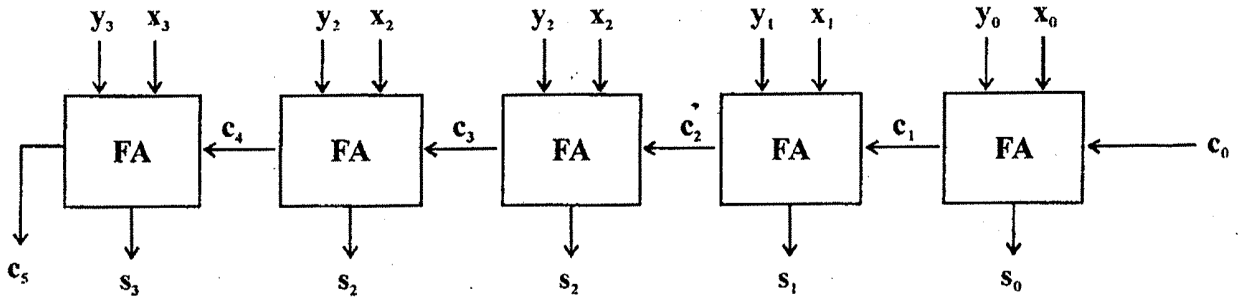


Figure 19 : Four full adders are cascaded to implement a 4-bit Ripple-Carry Adder.

When two unsigned integers are added, the input carry, c_0, is always zero. Yet, when this system is to operates on 2's complement numbers, the value of c_0 may be zero or one. The 4-bit adder is also called a carry-propagate adder (CPA), since the carry is propagated serially through each full adder. This hardware can be cascaded to obtain a 16-bit CPA.

Although the design of an n-bit CPA is straight forward, the carry propagation time limits the speed of operation, e.g., in 16-bit CPA, the addition operation is completed only when the sum bits so through s_{15} are available.

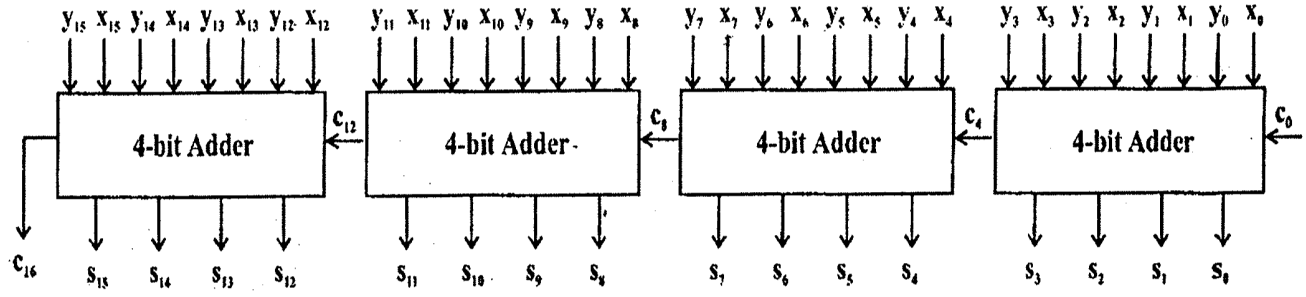


Figure 20 : Implementation of a 16-bit adder using 4-bit adders as building blocks.

To generate s_{15}, c_{15} must be available. The generation of c_{15} depends on the availability of c_{14}, which must wait for c_{13} to become available. In the worst case, the carry propagates through 15 full adders. Therefore, the worst-case add time of the 16-CPA can be estimated as follows :

Time taken for carry to propagate through 15 full adders (the delay involved in the path from c_0 to c_{15}) = 15 * 2Δ and the time taken to generate s_{15} from c_{15} = 3Δ. Thus the total time required is 33Δ.

If Δ = 10ns, then the worst-case add-time of a 16-bit CPA is 330ns. This delay is prohibitive for high-speed systems in which the expected add-time is typically less than 100ns.

61.11. Memory Organization

A memory unit is an integral part of a computer system. Its primary function is to store all information

needed by the system. Typically, a memory unit holds programs and data. The system performance is largely depends on the organization, storage capacity, and speed of operation of the memory system.

In a broad sense, a computer memory system can be logically divided into three groups :

- Internal memory
- Main memory, and
- Secondary memory (auxiliary memory).

Internal memory

This memory refers to the set of registers confined to the CPU. These registers hold temporary results when a computation is in progress. There is no speed disparity between these registers and the CPU because they are fabricated with the same technology. However, the cost involved in this approach forces a computer architect to include only a few registers (usually 8 or 16) in the CPU.

Primary memory

This is the storage area in which all programs are executed. The CPU can directly access only those items that are stored in primary memory. Therefore, all programs and data must be within the primary memory to speed up execution. In modern computers, MOS technology is employed in primary membory design. Usually, the size of the primary memory is much larger than processor memory and its operating speed is slower than the processor registers by a factor of 10.

Secondary memory

This memory refers to the storage medium comprised of slow devices such as magnatic tapes and drums. These devices are used to hold data files and programs, such as compilers and data base management systems, which are not frequently needed by the processor. Secondary memories are also referred to as auxiliary or backup store.

ROM

A read-only memory (ROM) is the simplest kind of memory. It is equivalent to a group of registers, each permanently storing a word. By applying control signals, we can read the word in any memory location.

Addresses

The address and contents of a memory location are two different things. As shown in the following :

Register	Address	Word
R_0	0	0111
R_1	1	1000
R_2	2	1011

Register	Address	Word
R_3	3	1100
R_4	4	0110
R_5	5	1001
R_6	6	0011
R_7	7	1110

The address of a memory location is the same as the subscript of the register storing the word. This is why register 0 has an address of 0 and contents of 0111; register 1 has an address of 1 and contents of 1000 and so on.

The idea of address applies to ROMs of any size. For example, a ROM with 256 locations has decimal addresses running from 0 to 255. A ROM with 1024 memory locations has decimal addresses from 0 to 1023.

The n address lines can select 2^n memory locations, e.g., 4 address lines can access 16 memory locations, 8 address lines can access 256 memory locations, and so on.

PROM and EPROM

A programmable ROM (PROM) is allows the user to store the data. An instrument called a PROM programmer does the storing by "burning in" (fusible links at the bit locations can be burned open by high current). With a ROM programmer, the user can burn in the programming is permanent. In other words, the stored contents connot be erased.

The erasable PROM (EPROM) user MOSFETs (Metal Oxide Semi-conductor Field Effect Transistor). Data is stored with a PROM programmer. Data can be erased with ultraviolet light. The light passes through a window in the IC package to the chip, where it releases stored charges. The EPROM is ultraviolet-light-erasable and electrically reprogrammable.

RAM

A random-access memory (RAM), also called read-write memory, is equivalent to a group of addressable registers. After supplying an address, one can either read the stored contents of the memory location or write new contents into the memory location.

Semiconductor RAM

Semiconductor RAMs may be static or dynamic. The static RAM uses bipolar or MOS flip-flops, data is retained indefinitely as long as power is applied to the flip-flops. On the other hand, a dynamic RAM uses MOSFETs and capacitors that store data. Because, the capacitor charge leaks off, the stored data must be refreshed (recharged) every few milliseconds. In either case, the RAMs are volatile.

Cache memory

The performance of a computer system will be severely affected if the speed disparity between the processor

and the main memory is significant, e.g., the cycle time of the CDC STAR 100 processor is 40 ns, which the cycle time of its main memory is 1280 ns. In such a situation, the system performance can be significantly improved by adding a small, expensive, but fast buffer memory between the processor and the main memory. The buffer memory is called cache memory and was first implemented in the IBM 360/85 computer. Latter, this concept became a part of minicomputers such as PDP-11. With the advent of VLSI technology, the cache concept is gaining acceptance in the microworld. The 32-bit VLSI microprocessors such as Motorola's MC 68020 and Intel's 80386 include an on board cache in their CPU.

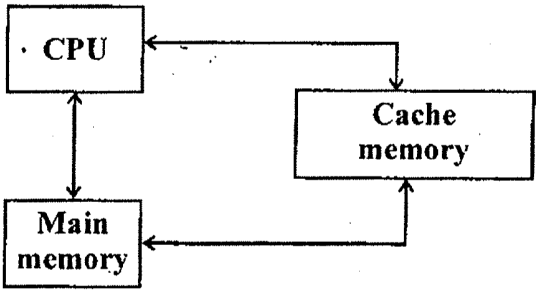


Figure 21 : Position of cache memory.

61.12. Auxiliary memory

61.12.1 Magnetic disks

A magnetic disk is a circular plate constructed of metal or plastic coated with magnetized material. Often both sides of the disk are used and several disks may be stacked on one spindle with read/write heads available on each surface. All disks rotate together at high speed and not stopped or started for access purposes. Bits are stored in the magnetized surface is spots along concentric circles called tracks. The tracks are commonly divided into sections called sectors. In most systems the minimum quantity of information which can be transferred is a sector.

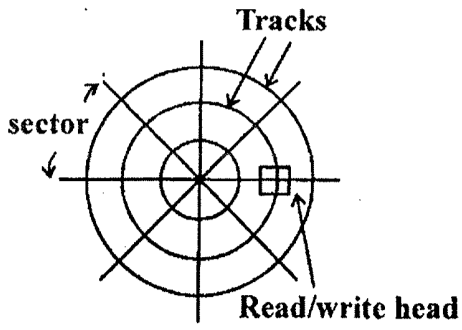


Figure 22 : Magnetic disk

Some units use a single read/write head for each disk surface. In this type of unit, the track address bits are used by a mechanical assembly to move the head into the specified track position before reading or writing. In other disk systems, separate read/write heads are provided for each track in each surface. A disk system is addressed by address bits that specify the disk number, the disk surface, the sector number and the track within the sector. After the read/write heads are positioned in the specified track, the system has to wait until the rotating disk reaches the specified sector under the read/write head. Information transfer is very fast once the beginning of sector has been reached. Disks may have multiple heads and simultaneous transfer of bits from several tracks at the same-time.

Disks that are permanently attached to the unit assembly and cannot be removed by the occasional user are called hard disk. A disk drive with removable disks is called a floppy disk or CD.

61.12.2 Magnetic tape

A magnetic tape transport consists of the electrical, mechanical and electronic components to provide the parts and control mechanism for a magnetic-tape unit. The tape itself is a strip of plastic coated with a magnetic recording medium.

Magnetic tape units can be stopped, started to move forward or in reverse, or can be rewound. However, they cannot be started or stopped fast enough between individual charactaristic.

61.12.3 Virtual memory

In a memory hierarchy system, programs and data are first stored in auxiliary memory. Portions of a program or data are brought into main memory as they are needed by the CPU. Virtual memory is a concept used is some large computer systems that permit the user to construct programs as through a large memory space were available, equal to the totality of auxiliary memory. A virtual memory system provides a mechanism for translating program-generated addresses into correct main memory locations. This is done dynamically, while programs are being executed in the CPU. The translation or mapping is handed automatically by the hardware by means of a mapping table.

61.13. Multiprogramming

Many operating systems are designed to enable the CPU to process a number of independent programs concurrently. This concept, called multiprogramming, refers to the existence of two or more programs in different parts of the memory hierarchy at the same time. In this may, it is possible to keep all parts of the computer busy by working with several programs in sequence. For example, suppose that a program is being executed in the CPU and an I/O transfer is required. The CPU initiates the I/O processor to start executing the transfer this leaves the CPU free to execute another program. In a multiprogramming system, when one program is waiting for input or output transfer, there is another program ready to utilize the CPU.

The part of the computer system that sxperaises the flow of information between anxiliary memory and main

memory is called the memory management system.

61.14. I/O Organization

The most common process to enter information into a computer is through a type writer like keyboard that allows a person to enter alphanumeric information directly. Every time a key is depressed, the terminal sends a binary coded character to the computer. The fastest possible speed for entering information this way depends on the person's typing speed. On the orther hand, CPU is an extremely fast divice capable of performing operations at very high speed. When input information is transferred to the processor via a slow keyboard, the processor will be idle most of the time while waiting for the information to arrive. To use a computer efficiently, a large amount of programs and data must be prepared in advance and transmitted into a storage medium such as magnetic tapes or disks. The information in the disk is then transferred into computer memory at a rapid rate. Results of programs are also transferred into a high-speed storage, such as disks, from which they can be transferred later into a printer to provide a printed output of results.

Devices that are under the direct control of the computer are said to be connected on-line. These devices are designed to read information into or out of the memory unit upon command from the CPU and are considered to be part of the total computer system. Input or output devices attached to the computer are also called peripherals. Among the most common peripherals are keyboards, display units, and printers. Peripherals that provide auxiliary storage for the system are magnetic disks and tapes. Peripherals are electromechanical and electromagnetic devices.

Video monitors are the most commonly used peripherals. They consist of a keyboard as the input device and a display unit as the output device. There are different types of video monitors, but the most popular use a cathode ray tube (CRT).

Printers provide a permanent record on paper of computer output data or text. There are three basic types of character printers : daisy wheel, dot matrix, and laser printers. The dot matrix printer contains a set of dots along the printing mechanism, e.g., a 5×7 dot matrix printer that prints 80 characters per line has seven horizontal lines, each consisting of 5×80=400 dots. Each dot can be printed or not, depending on the specific characters that are printed on the line. The laser printer uses a rotating photographic drum that is used to imprint the character imager. The pattern is then transferred onto paper in the same manner as a copying machine.

61.15. Direct Memory Access (DMA)

The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Remouing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called direct memory access (DMA). During DMA transfer, the CPU is idle and has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.

61.16. Module Summary

The basic concept of computer organization is briefly introduced. The technologies used in different generations of computer are discussed. The architecture proposed by Von Neumann with different components are briefly presented. Different kind of memory units and digital components are discussed. Adder to add two binary numbers is presented. A brief introduction of I/O organization is given here.

61.17. Self Assesment Questions

(i) Explain Von Neumann architecture of computer. Discuss all of its components.

(ii) Discuss about the technologies used in different generations of computer.

(iii) Dran a block diagram of organization of a stored program computer system.

(iv) Discuss about the branch instruction and subroutine call of program at the time of program execution.

(v) Write note about

(a) general register machines, (b) accumulator, and (c) stack machine.

(vi) What do you mean by flip-flop? Explain SR, JK and D flip-flops with block diagram and characteristic table.

(vii) Write short notes on

(a) Decoders, (b) multiplexers, and (c) registers.

(viii) Explain with block diagram the construction of a 4-bit registrer using D flip-flops.

(ix) Construct a 3-to-8 decoder using two 2-to-4 decorders.

(x) What is the use of adder? Explain adder with block diagram and logical expression for sum and carry.

(xi) Design a 4-bit adder using full-adder.

(xii) Construct a 16-bit adder using 4-bit adders as building blocks.

(xiii) Explain the followings :

(a) internal memory, (b) primary memory, (c) secondary memory, (d) different kind of ROMs, (e) cache memory, (f) virtual memory.

(xiv) Write short notes on the following allxiliary memory.

(a) magnetic disks, (b) magnetic tape.

(xv) What do you mean by multiprogramming? Explain it.

(xvi) Write a short note for I/O organization of a computer.

61.18. Further Suggested Readings

1. M. Morris Mano, Computer System Architecture, PHI.
2. Md. Rafiquzzaman and Rajan Chandra, Modern Computer Architecture, Galgotia.

M.Sc. Course
in
Applied Mathematics with Oceanology
and
Computer Programming

PART-II

Paper-VI

Group–A

Module No. - 62
DATA STRUCTURES

Structure :

62.1 Introduction

62.2 Objectives

62.3 Asymptotic Notarions

62.4 Time and Space Complexities

62.5 Data Structure

62.6 Arrays

62.7 Stacks

62.8 Evaluation of Expression

- 62.8.1 Evaluation of a postfix expression
- 62.8.2 Conversion of an infix expression to a postfix expression

62.9 Queues

62.10 Linked Lists

- 62.10.1 Circular Lists
- 62.10.2 Applications of linked lists
- 62.10.3 Addition of two polynomials

62.11 Module Summary

62.12 Self Assessment Questions

62.13 Suggested Further Readings

62.1 Introduction

Data structure is the main part to develop an algorithm. An efficient algorithm cannot be designed without appropriate use of data structure. Different kinds of data strucures are available to design an algorithmand each data structure has a special feature. So we have to learnt all the commonly used daa structures before development of an efficient algorith. In this module we study some commonly used data strucures, viz., arrays, stacks, queues and linked lists.

62.2 Objectives

Gone through this module the students will learn the following topics :

* Asymptotic Notations
* Arrays
* Stacks
* Application of stacks
* Queues
* Linked lists
* Applications of linked lists

62.3 Asymptotic Notations

To analysis the performance of an algorithm there are several kinds of mathematica notation are used. Some of them are presented here.

Def. 62.3.1 (O-notation) (big-oh). *To represent* **asymptotic upper bound** *the O-notation is used. For a given function g(n), we denote by O(g(n)) the set of functions*

$$O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and no such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0\}. \quad (62.1)$$

In other words, $f(n) = O(g(n))$ iff

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \text{a non-zero constant.}$$

Thus O-notation is used to represents an upper bound of a function, to within a constant factor.

The O-notation is used to express an upper bound, we might also wish to determine a function which is a lower bound.

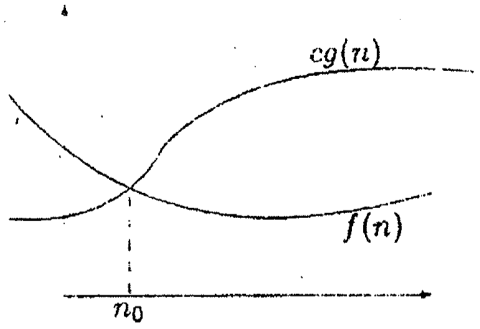


Figure 62.1: $f(n) = O(g(n))$.

Some properties of O-notation

(i) If $A(n) = a_k n_k + \ldots + a_1 n + a_0$ then $A(n) = O(n^k)$, k is independent of n.

(ii) $O\big(f(n)\big) + O\big(g(n)\big) = O\big(\max\{f(n), g(n)\}\big)$.

(iii) $O\big(f_1(n)\big) + O\big(f_2(n)\big) + \ldots + O\big(f_k(n)\big) = O\Big(\max_n\{f_1(n), f_2(n), \ldots, f_k(n)\}\Big)$, k is independent of n.

(iv) $cO\big(f(n)\big) = O\big(f(n)\big)$, c is independent of n.

(v) $O\big(f(n)\big) \times O\big(g(n)\big) = O\big(f(n) \times g(n)\big)$.

(vi) $n^k \times O\big(f(n)\big) = O\big(n^k f(n)\big)$, k is independent of n.

(vii) $O\big(n^k\big) \times O\big(n^l\big) = O\big(n^{k+l}\big)$, k and l are independent of n.

Def. 62.3.2 Ω-notation. The *Ω-notation provides an* **asymptotic lower bound.** *For a given function g(n), we denote by Ω(g(n)) the set of functions*

$$\Omega(g(n)) = \{f(n): \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0\}. \quad (62.2)$$

Def. 62.3.3 Ω-notaton (tight bound). *For a given function g(n), we denote by θ (g(n)) the set of functions*

$$\theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2 \text{ and } n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0\}. \quad (62.3)$$

In other words, for all $n \ge n_0$, the function f(n) is equal to g(n) to within a constant factor. We say that g(n) is an asymptotically tight bound for f(n).

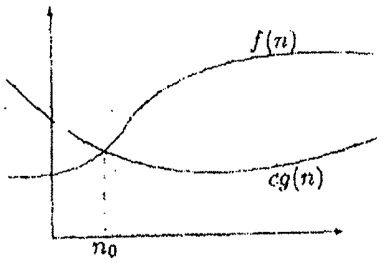


Figure 62.2: $f(n) = \Omega(g(n))$.

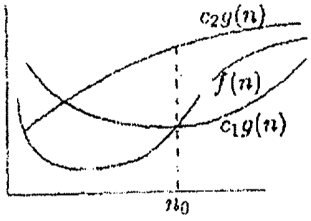


Figure 62.3: $f(n) = \theta(g(n))$.

Def. 62.3.4 o-notation (little-oh). *The o-notation is used to denote an upper bound that is not asymptoticaly tight. We define o(g(n)) as the set*

$$o(g(n)) = \{f(n) : \text{for any positive constants } c > 0, \text{ there exists a constant } n_0 > 0 \text{ such that } 0 \le f(n) < cg(n) \text{ for all } n \ge n_0\}. \quad (62.4)$$

For example, $2n = o(n^2)$, *but* $2n^2 \ne o(n^2)$.

The definitions of *O*-notation and *o*-notation are similar. The main difference is that in $f(n) = O(g(n))$, the bound $0 \le f(n) \le cg(n)$ holds for some constants $c > 0$ but in $f(n) = o(g(n))$, the bound $0 \le f(n) < cg(n)$ holds for all constants $c > 0$, i.e.,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

Def. 62.3.5 ω-notation. *Similar to O-notation and o-notation, ω-notation is defined in contrast of Ω-notation. The ω-notation is used to denote a lower bound that is not asymptotically tight.*

One way to define it is by

$$f(n) \in w(g(n)) \text{ iff } g(n) \in o(f(n)).$$

Formally,

$$\omega(g(n)) = \{f(n)\} \text{ : for any positive constants } c > 0, \text{ there exists a constant } n_{0\backslash>0} \text{ such that } 0 \le cg(n) < f(n) \text{ for all } n \ge n_0\}. \quad (62.6)$$

For example, $n^2/2 = \omega(n)$ *but* $n^2/2 \ne \omega(n^2)$. The relation $f(n) = \omega(g(n))$ implies that

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty.$$

Different kind of logarithms are used to analysis the algorithms. They are defined in the following.

$$\lg n = \text{og}_2 n \text{ (binary logarithm)}$$

$$\ln n = \log_e n \text{ (natural logarithm)}$$

$$\lg^k n = (\lg n)^k \text{ (exponentiation)}$$

$$\lg^{(2)} n = \lg \lg n = \lg (\lg n) \text{ (composition)}$$

$$\lg^* n = \min\{i \ge 0 : \lg^{(i)} n \le 1\}.$$

62.4 Time and Space Complexities

The **time complexity** of an algorithm is given by a function $T(n)$ which is the maximum over all input of size n, of the sum of the time required by each instruction executed. For a problem of size n, the maximum time required to solve the problem over all input sequences of that size is called the **worst case** time complexity. The space complexity of an algorithm is the number of memory units used. The space required by an algorithm is, therefore, the maximum number of memory required at any time during the course of execution of the algorithm.

If $T(n)$ is a polynomial function of n then the algorithm is said to take a polynomial time (space). If $T(n)$ is a polynomial of degree one then we say that the algorithm takes linear time (space).

An algorithm is considered to be **efficient** it it takes a time which is a polynomial function of the problem size. The problems for which such algorithms exist belong to the class *P.* Problems for which such algorithms do not seem to exist form the class of NP-complete problems. A problem which is NP-complete will have the property that it can be solved in polynomial time iff all other NP-complete probles can also be solved in polynomial time. If an NP-hard problem can be solved in polynomial time then all NP-complete problems can be solved in polynomial time. That is, al NP-complete problems are NP-hard but all NP-hard problems are not NP-complete.

62.5 Data Structure

Before presenting the definition of data structure we consider an example of a data structure, 'natural number'.

Suppose we want to define the data structure Natural Number (abbreviated by NatNo) where NatNo={0, 1, 2, ...} with the four operations being a test for zero, addition, equality and successor. The following notation can be used.

Structure NatNo

declare

1. ZERO ( ) → NatNo
2. ISZERO (NatNo) → NatNo
3. SUCC (NatNo) → NatNo
4. ADD (NatNo, NatNo) → NatNo
5. EQ (NatNo, NatNo) → boolean

6 for all $x, y \in$ NatNo let

7. ISZERO (ZERO) ::=true; ISZERO(SUCC(x))::=false;
8. ADD(ZERO, y)::=y; ADD(SUCC(x), y))::=SUCC(ADD(x, y));
9. EQ(x, ZERO)::=if ISZERO(x) then true else false;
10. EQ(ZERO, SUCC(y))::=false; EQ(SUCC(x), SUCC(y))::=EQ(x, y);
11. Endlfor;

End Natno.

In the declaration section five functions are defind by giving their names, inputs and outputs. ZERO is a constant function which means it takes no input arguments and its result is the natural number zero,written as ZERO. ISZERO is a boolean function whose result is either true or false. SUCC stands for successor. Using the functions ZERO and SUCC we can define all of the natural numbers as :

ZERO, 1=SUCC(ZERO), 2=SUCC (SUCC(ZERO)), and so on.

The rules on line 8 tell us exactly how addition operation works e.g., if we want to add two and three we would get the following sequence of expressions:

ADD(SUCC(SUCC(ZERO)), SUCC(SUCC(ZERO))) which, by line 8 equals

SUCC(ADD(SUCC(ZERO), SUCC(SUCC(SUCC(ZERO))) which by line 8 equals

SUCC(SUCC(SUCC(SUCC(SUCC(ZERO))))) of course, this is not the way to implement addition. In practice we use bit string which is a data structure that is usually provided on our computers. But, however the ADD operation is implemented, it must obey thee rules. Hopefully, this motivates the following definition of data structure.

Def. 62.5.1 A data structure *is a set of domains D, a designation domain $d \in D$, a set of functions F and a set of axioms A. The triplets (D, F, A) denotes the data structure d and it will usually be abbreviated by writing d.*

In previous example,

d = NatNo,

D = {NatNo, boolean},

F = {ZERO, ISZERO, SUCC, ADD},

A = {lines 7 through 10 of the structure NatNo}.

There are different kind of data structures are used to solve problems with the help of computer. The most useful data structures are array (one and two dimensions), stack, queue, linked lists etc.

62.6 Arrays

Oneof the simplest generalizations of a linear list is a two-dimensional or higher dimensional arrays of information e.g .. consider the case of an m x n matrix

$$\begin{bmatrix} A[1,1] & A[1,2] & \dots & A[1,n] \\ \vdots & \vdots & \vdots & \vdots \\ A[m,1] & A[m,2] & \dots & A[m,n] \end{bmatrix} \tag{62.7}$$

In this two-dimensional array, each node $A[j,k]$ belongs to two linear lists : the 'row j' list $A[j, 1], A[j, 2],...,$ $A[j,$ and the 'column k' list $A[1, k], A[2, k]..., A[m, k]$. These orthogonal row and column lists essentially account for the two-dimensional structure of a matrix. Similar, remarks apply to higher dimensional array of information.

Sequential allocation

When an array is stored is stored in sequential memory locations, storage is usually allocated so that,

$$LOC(A[j,k]) = a_0 + a_1 j + a_2 k \tag{62.8}$$

where a_0, a_1, a_2 are constants and $LOC(A[j,k])$ represents the location of the element $A[j,k]$ in the memory. Let us consider a more general case. Suppose we have a four dimensional array with one word elements $Q[i,j,k,l]$ *for* $0 \le i \le 2, 0 \le j \le 4, 0 \le k \le 10, 0 \le l \le 2$ we should like to allocate storage so that

$$LOC(Q[i,j,k,l]) = a_0 + a_1 i + a_2 j + a_3 k + a_4 l. \tag{62.9}$$

This means that a change in i, j, k or l leads to a readily calculated change in the location of $Q[i,j,k,l]$. The most natural natural (and most commonly used) way to allocate storage is to be the array applied in memory in the 'lexicographic order' of its indices, sometime called ' row major order'.

The elements of the array Q are

Q [0, 0, 0, 0], Q [0, 0, 0, 1], Q [0, 0, 0, 2], Q [0, 0, 1, 0], Q [0, 0, 1, 1], ..., Q [0, 0, 10, 2], Q [0, 1, 0, 0], ..., Q [0, 4, 10, 2], Q [1, 0, 0, 0], ..., Q [2, 4, 10, 2].

It is easy to see that this order satisfies the requirements of (62.9), and we have

$$LOC(Q[i,j,k,l]) = LOC(Q[0,0,0,0]) + 165i + 33j + 3k + l. \tag{62.10}$$

In general given a k-dimensional array with c-word elements $A[i_1, i_2, ..., i_k]$ for $0 \le i_1 \le d_1, 0 \le i_2 \le d_2, ..., 0 \le i_k \le d_k$, we can store it in memory as

$$\begin{aligned} LOC(A[i_1, i_2, ..., i_k]) &= LOC(A[0,0,...0]) + c(d_2+1)...(d_k+1)i_1 \\ &\quad +...+ c(d_k+1)i_{k-1} + ci_k \\ &= LOC(A[0,0,0,...0]) + \sum_{1 \le r \le k} a_r i_r \end{aligned} \tag{62.11}$$

where $a_r = c\prod_{r<s\le k}(d_s+1)$.

We want to store the triangular matrix $A[j, k]$ for say $0 \le k \le j \le n$, (column major order)

$$\begin{bmatrix} A[0,0] & & & \\ A[1,0] & A[1,1] & & \\ A[2,0] & A[2,1] & A[2,2] & \\ \cdots & \cdots & \cdots & \\ A[n,0] & A[n,1] & \cdots & A[n,n] \end{bmatrix} \tag{62.12}$$

We may know that all other entries are zero, or that $A[j, k] = A[k, j]$, so only half of the values need to be stored. If we want to store the lower triangular matrix of the form (62.12) we need only $\frac{1}{2}(n+1)((n+2)$ consecutive memory positions. We are forced to give up the possibility of linear allocation as in equation (62.8), but we can now ask instead for an llocation arrangement of the form

$$LOC(A[j,k]) = a_0 + f_1(j) + f_2(k) \tag{62.13}$$

where f_1 and f_2 are functions of one variable.

In turns out that lexicographic condition (62.13) and assuming one-word entries we have in fact the rather simple formula

$$LOC(A[j,k]) = LOC(A[0,0]) + \frac{j(j+1)}{2} + k. \tag{62.14}$$

If we store the matrix A in the array B then the element $A[j, k]$ is stored in $B[p]$, where

$$p = LOC(A[0,0]) + \frac{j(j+1)}{2} + k \text{ for given } j \text{ and } k. \tag{62.15}$$

There is a far better way to store two triangular matrices if we are fortunate enough to have two of them with the same size. If $A[j, k]$ and $B[j, k]$ are both to be stored for $0 \le k \le j \le n$ we can fit them both into a single matrix $C[i,j]$ for $0 \le j \le n, 0 \le k \le n+1$ using the vonvention

$$A[j,k] = C[j,k], B[j,k] = C[k, j+1]. \tag{62.16}$$

Thus

$$\begin{vmatrix} C_{00} & C_{01} & \cdots & C_{0\,n+1} \\ C_{10} & C_{11} & \cdots & C_{1\,n+1} \\ \vdots & \vdots & \vdots & \vdots \\ C_{n0} & C_{n1} & \cdots & C_{n\,n+1} \end{vmatrix} = \begin{vmatrix} A_{00} & B_{00} & B_{10} & \cdots & B_{n0} \\ A_{10} & A_{11} & B_{11} & \cdots & B_{n1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ A_{n0} & A_{n1} & A_{n2} & \cdots & A_{nn} \end{vmatrix}$$

The two triangular matrices are packed together tightly within the space of $(n+1)(n+2)$ locations and we have linear addressing as in (62.16). The generalization of triangular matrices to higher dmensions is called a tetrahedral array.

62.7 Stacks

A **stack** is an ordered list in which all insertions and deletions are made at one end, called the **top.** The stacks are sometimes referred to as Last In First Out (LIFO) lists.

Usually the following operations are performed in the stack.

create (S) which creates *S* as an empty stack.

push (i, S) which inserts the elements *i* onto the stack *S* and returns the modified stack.

pop (S) which removes the top elements of stack *S* and returns the modified stack.

top (S) which returns the top element of stack *S*.

isempty (S) which returns true if *S* is empty else false.

These five functions constitute a working definition of a stack. However, we choose to represents a stack, it must be possible to build these operations.

The simplest way to represent a stack is by using a one-dimensional array, say STACK (1: *n*), where *n* is the maximum number of allowable entries. Thus first or bottom elemet in the stack will be stored at STACK (1), the second at STACK(2) and *i*th at STACK(*i*).

Associated with the array there will be a variable 'top', which points to the top element in the stack. With this assumption, the functions used in stack are implemented below.

create ( ) declare the array STACK (1 : *n*) and set to ← 0;

isempty (STACK) = true (if toop=0) false otherwise.

top (STACK) = error (if top = 0) STACK (top) otherwise.

The algorithms for the functions push and pop are presented in the following.

Procedure push (*x*, S)

//Inserts an object *x* into the stack S of maximum size *n*; 'top' is the number of elements currently in the stack S.//

if top ≥ *n* then

print 'stack full', stop'

else

```
        top ← top + 1;
        S (top) ← x;
endif;
end push.
Proceudre pop (S)
//removes the top element from the stack S and stores it in x unless stack S is empty.//
if top ≤ 0 then
        print 'stack empty', stop;
else
        x ← S (top);
        top ← top – 1;
endif;
end pop.
```

One ideal example of application of stack is presented in the next section.

62.8 Evaluation of Expression

An expression is made up of operands and operators. The expresion $A/B\,\hat{}\,C+D*E-A*C$ has five operands A, B, C, D, E. Though these are all one letter variables, operands can be any legal variable name or constant in any programming language. In any expression the values that variables take must be consistent with the operations operformed on them. These operations are described by the operators. There are the five basic arithmetic operators viz., plus (+), minus (–), times (*), divide (/) and exponent (^).

The first problem with understanding the meaning of an expression is to decide in what order the operations are carried out.

To fix the order of evaluation, we assign to each operator a priority. Then within any pair of parentheses we understand that operators with the highest priority will be evaluated first. Operators (+, –, *, /, ^) the following is the order of precedence (highest to lowest) :

Operators	priority
^	3
* /	2
+ –	1
( )	0

Compiler can't proceed the mathematicsl expression or logical expression which is written in conventional way. Before evaluation of an expression the given (conventional) form is converted into another expression called

postfix expression. If E is an expression with operators and operands, the conventional way of writing E is called **infix**, because the operators come in-between the operands. The postfix (**profix**) form of an expression calls for each operators to appear after (before) its operands, e.g, the infix expression $A * B / C$ has postfix form $AB *C/$.

If we study the postfix form of $A * B /C$ we see that the multiplication comes immediately after its two operands A and B. Now imagine that $A * B$ is computed and stored in T. Then we have the division operator /, coming immediately after its two arguments T and C. That is,

$$(A * B)\ C = T / C \text{ where } T = AB *$$
$$= TC /$$
$$= AB * C/$$

Consider another expression

$$(A + B) * C = T * C \text{ where } T = AB +$$
$$= TC *$$
$$= AB + C*$$

Example 62.8.1 Convert the following infix expression to postfix expression :

$$A / B \wedge C + D * E - A * C.$$

Solution. In this expression ^ has highest priority, so $B \wedge C$ is computed first and stored it into T_1 as $T_1 = BC\wedge$. That is,

$A / B \wedge C + B * E - A * C$

$= A/T_1 + B * E - A * C$ where $T_1 = BC\wedge$

$= T_2 + T_3 - T_4$ $T_2 = AT_1/$, $T_3 = BE*$, $T_4 = AC*$

$= T_5 - T_4$ $T_5 = T_2T_3+$

$= T_5T_4-$

$= T_2T_3+ T_4 -$

$= AT_1 / BE * + AC * -$

$= ABC \wedge / BE * + AC * -$

Infix	Postfix
$A + B$	$AB +$
$A + B - C$	$AB + C -$
$(A + B) * (C - D)$	$AB + CD - *$
$A \wedge B * C - D + E/F/(G + H)$	$AB \wedge C * D - EF / GH + / +$
$((A + B) * C - (D - E) \wedge (F + G)$	$AB + C * DE - - FG + \wedge$
$A - B / (C * D \wedge E)$	$ABCDE \wedge * / -$

It may be noted that the parentheses presented in the infix expression are removed from the postfix expression. That is, the length (number of characters) of the postfix expression is less than the corresponding infix expression.

62.8.1 Evaluation of a postfix expression

Each operator in a postfix string refers to the previous two operands in the string. Obviously, one of these two operands may itself be the result of applying a previous operator. Suppose that each time we read an operand we push it onto a stack. When we reach an operator, its operands will be the top two elements on the stack. We can then pop these two elements, perform the indicated operation on them, and push the result on the stack so that it will be available for use as an operand of the next operator. The following algorithm evaluates an expression in postfil using this method.

Algorithm **Eval-Postfix**

```
//The array 'postfix[ ]' is the postfix expression and 'optstack' is the stack of operands.//
Initially opstack = empty;
for i = 1 to N // where N is the length of the array psotfix[]//
    symb=postfix[i];
    if(symb=operand)
        push(symb,opstack);
    else //symb-operator//
        opnd2=pop(opstack);
            //removes top element from the stack opstack and stored in opnd2//
        opnd1=pop(opstack);
            //removes next element from the stack opstack and stored in opnd1//
        value=result of applying symb to opnd1 and opnd2;
            //value = (opnd1) symb (opnd2) //
        push(value, opstack); //value placed on the top of the stack//
    end //else//
endif;
```

endfor;
value of expression = pop(opstack);
end Eval-postfix.

Limitations of the algorithm Eval-Postfix

Each character of postfix string assumed to be either an operator or an operand.

62.8.2 Conversion of an infix expression to a postfix expression

It is well known that expressions within innermost parentheses must first be converted to postfix so that they can then be treated as single operands. In this fashion parentheses can be successively eliminated until the entire expression is converted. The last pair of parentheses to be opened within a group of parentheses encloses the first expression within that group to be transformed. This last-in, first-out behavior should immediately suggest the use of a stack.

Since priority of operators plays an important role in transforming infix to postfix, let us assume the existence of a function P(op), where 'op' is a character representing operator.

Here we define the function P as follows.

P ('(') = P ('(') = 0
P ('+') = P ('–') = 1
P ('*') = P ('/') = 2
P ('^') = 3.

Algorithm **Infix2Postfix**

//Converts an infix expression to a postfix expression.//
Initially top = 0, i = 0, q = 0; // 'top' is the top of the stack //
N = length (infix[]);
while ($i < N$) do
    $i = i + 1$;
    case infix [i] of

        letter : $q + q + 1$; postfix[q]=infix[i];
            //puts infix character directly to postfix expression//
        '(' : push('(',stack); //puts left parenthesis directly to the stack//
        '(' : x=pop(stack);
            while $x \neq$ '(' do // removes elements from stack and
                puts tem to the postfix expression until
                x is a closing parenthesis//

```
            q = q + 1; postfix [q] = x;// puts to the postfix expression//
            x = pop(stack); //removes top element from the stack//
        endwhile;
    operator :    while {P(infix[i] ≤ P(stack(top))} and (top>0) do
                  x = pop(stack);//removes top element from the stack//
                  q = q + 1; postfix [q]=x; // puts to the postfix expression//
            endwhile;
            push(infix[i], stack); //puts to the stack//
    endcase;
endwhile;
while top > do // puts remaining elements to postfix expression//
    x=pop(stack);
    q=q+1; postfix[q]=x;
endwhile;
end Infix2Postfix.
```

62.9 Queues

A queue is an ordered list in which all insertions take place at one end, the **rear**, while all deletions take plae at the other end, the **front.**

A minimal set of useful operations on a queue includes the following :

createq(Q) which creates Q as empty queue;

addq(i, Q) which adds the element i to the rear of the Q and returns the new queue;

deleteq (Q) which removs the front element from the queue Q and returns the resulting queue;

front(Q) which returns the front element of Q;

isempty (Q) which returns true if Q is empty else false.

The representation of a finite queue in sequential locations is somewhat more difficult than a stack. In addition to a one dimensional array $Q(1 : n)$, we need two more variables, front and rear. The conventions we shall adopt for these two variables are the front is always 1 less than the actual front of the queue and rear always points to the last element in the queue. Thus, front = rear iff there are no elements in the queue. The initial condition then is front = rear = 0. The following implementation of the createq, isemptyq and front operations results for a queue with capacity n.

createq (Q) declares the array $Q(1 : n)$ and sets fron ← rear ← 0;

isempty (Q) is true if (front = rear) false otherwise;

front (Q) gives error if (isempty (Q)) is true else it will return Q(front +1).

The following algorithms for addq and deleteq result:

```
function addq (x, Q)
//Inserts an object x into the queue Q represented by Q(1: n) whose capacity n; rear is the pointer at the end of the
queue.//
if (rear=n) then
        print 'queue-full'; stop;
else
        rear ← rear + 1;
        Q(rear) ← x;
endif;
end addq.
```

```
function deleteq (Q)
//Deletes an element from the queue Q represented by Q(1: n); rear and front are respectively the pointers at the
end and front of the queue.//
if (rear = front) then
        print 'queue-empty'; stop;
else
        front ← front + 1;
        x ← Q (front);
endif;
end deleteq.
```

With this setup, notice that unless the front regularly catches up with the rear and both ponters are reset to zero, then the queue-full signal does not obtained by regarding the array $Q(1: n)$ as circular. It now becomes more convenient to declare the array as $Q(0: n-1)$. When $rear = n-1$, the next element is entered at $Q(0)$ in case that spot is free. *front* will always point one position counter-clockwise from the first element in the queue. Again, $front = rear$ iff the queue is empty. Initially, we have $front = rear = 0$. Figure 62.4 illustrates some of the possible configurations for a circular queue coontaining the four elements A, B, C, D with $n > 4$.

In order to add an element, it wil be necesary to move rear one position clockwise, i.e.,

if $rear = n-1$ then $rear \leftarrow 0$ else $rear \leftarrow rear + 1$.

Using the modulo operator which computes remainders, that is just

$$rear \leftarrow (rear + 1) \bmod n.$$

Similarly, it will be necessary to move front one position clockwise each time a deletion is made. Again,

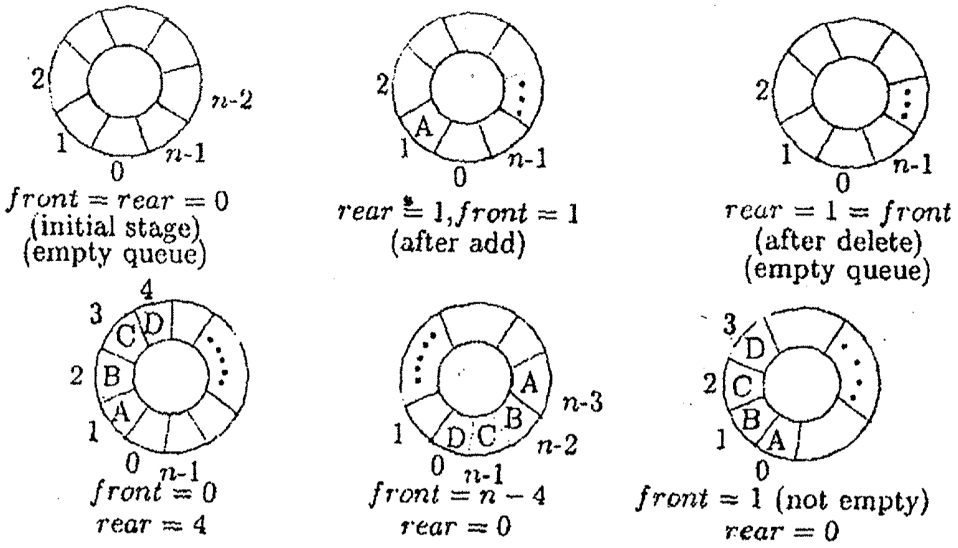


Figure 62.4: Circular representation of queue.

using the modulo operation, this can be accomplished by

$$front \leftarrow (front + 1) \bmod n.$$

An examination of the algorithms indicates that addition and delation can now be carried out in a fixed amount of time or $O(1)$.

function addcq(x, Q)

//Inserts an item x in the circular queue stored in $Q(0 : n-1)$ of size n; *rear* points to the last item and *front* is one position counterclockwise from the first item in Q.//

rear ← (*rear* + 1) mod n // advance rear clockwise

if *front* = *rear* then

    print 'queue-full'; stop;

else

    Q(*rear*) ← x;

endif;

end addcq.

function deletecq (Q)

//Removes the front element of the queue $Q(0 : n-1)$.//

if *front* = *rear* then

    print 'queue-empty'; stop;

else

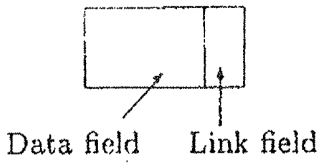


Figure 62.5: A node.

front ← (*front* + 1) mod *n* // advance front clockwise
x ← *Q* (*front*); //save front of queue as *x* for further use
endif;
end deletecq.

One surprising point in the two algorithms is that the test for queue full in addcq and the test for queue empty in deletecq are the same. In the case of addcq, however, when *front* = *rear* there is actually one space free, i.e., *Q* (*rear*), since the first element in the queue is not at *Q* (*front*) but is one position clockwise from this point. However, if we insert an item here, then we will not be able to distinguish between the cases full and empty, since this insertion would leave *front* = *rear*. To avoid this, we signal queue-full, thus permitting a maximu on *n* – 1 rather than *n* lements to be in the queue at any time.

62.10 Linked Lists

One major drawback of stack and queue in array implementation is that a fixed amount of storage remains allocated to the stack or queue even when the structure is actually using a smaller amount or possiblly no storage at all. Further, no more than that fixed amount of storage may be allocated, thus introducing the possibility of overflow.

In a sequential representation, the items of a stack or queue are implicitly ordered by the sequential order of storage. Thus, if *item*[*x*] represents an element of a queue, the next element will be *item*[*x*+1]. Suppose that the items of a stack or a queue were explicitly ordered, that is, each item contained within itself the address of the next item. Such an exlicit ordering gives rise to a data structure, which is known as a **linear linked list.** Each item in the list is caleld a **node** and contains two fields, an **data** field and a **link** field (address of the next node). The data field holds the actual element on the list.

The link field contains the address of the next node in the list. Such an address, which is used to access a particular node, is known as a **pointer.** The entire linked list is accessed from an external pointer **list,** which points to (contins the address of) the first node in the list. The link field of the last node in the list contains a special value, known as **null** or **nil** or 0, which is not a valid address. This **null pointer** is used to signal the end of a list. The list with no nodes on its called the **empty list** or the **null list.** The value of the external pointer **list** to such a list is the null pointer. Thus a list can be initialized to the empty list by the operation *list* = *null.*

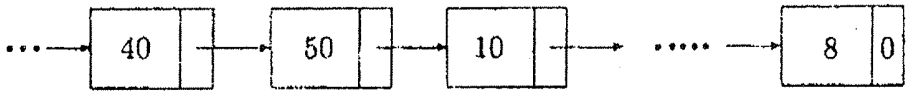


Figure 62.6: A portion of a list.

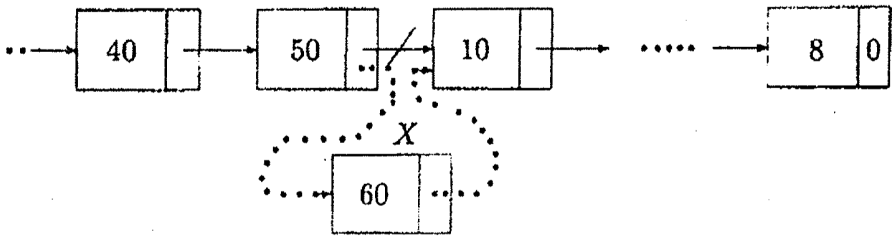


Figure 62.7 : The list after insertion of the node X after the node containing 50.

It is conventionally to draw linked lists as an ordered sequence of nodes with links being represented by arrows.

Notice that we do not explicitly put in the values of the pointers but simply draw arrows to indicate they are there. This is so that we reinforce in our own mind the facts that (i) the nodes do not actually reside in sequential locations, and that (ii) the locations of nodes may change on different runs. Therefore, when we write a program which works with lists, we almost never look for a specific address except when we test for null or 0.

Let us now see why it is easier to make arbitrary insertions and deletions using a linked list rather than a sequential list. To insert the data item 60 between 50 and 10 the following steps are needed.

(i) get a node which is currently unused, let its address be X.
(ii) set the data field of this node to 60.
(iii) set the link field of X to pont to the node after 50 which contains 10.

The Figure 62.7 shows how we can draw the insertion using our arrow notation. The new arrows are dashed. The important thing to notice is that when we insert 60 we do not have to move any other elements which are already in the list. We have overcome the need to move data at the expense of storage needed for the second field link. But, we will see that this is not too severe a penalty.

Now, suppose we want to delete 60 from the list. All we need to do is find the element which immediately precedes 60, which is 50. Again, there is no need to move the data around. Even though the link field of 60 still contains a pointer to 10, 60 is not longer in the list.

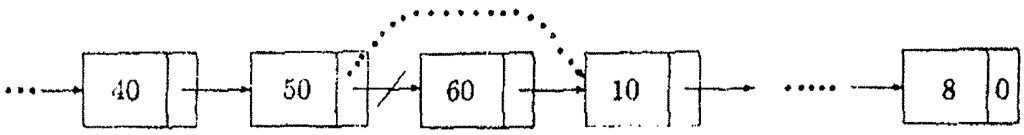


Figure 62.8: List after deletion of the node containing 60.

From our brief discussion of linked lists we see that the following capabilities are needed to make linked representations possible :

(i) A mechanism for dividing memory into nodes each having at least one link field.

(ii) A mechanism to determine which nodes are in use and which are free.

(iii) A mechanism to transfer nodes from the reserved pool to the free pool and vice-versa.

Though data and link look like conventional one dimensional arrays, it is not necesary to implement linked lists using them. For the time being let us assume that all free nodes are kept in a 'black box' called the storage pool and that there exist subalgorithms:

(i) GETNODE (X) which provides in X a pointer to a free node but if no node is free, it prints an error message and stops;

(ii) RET (X) which returns node X to the storage pool.

Example 62.10.1 Let T be a pointer to the first node to a linked list. T =0 if the list has no nodes. Let X be a pointer to some arbitrary node in the list T. The following algorithms inserts a node with data field Z following the node pointed at by X.

Procedure insertm(T, X) //inserts Z after the node X in the list T

```
    call GETNODE(I);
    data(I) ← Z;  //data(I) is the data field of the node I
    if T = 0 (null) then
        T ← I : // inserts into an empty list
        link(I) ← 0; // link (I) is the link field of the node I
    else
        link (I) ← link (X);
        link (X) ← I; //inserts after X
endif;
end insertm.
```

The following algorithm inserts an element at the beginning of the list T.

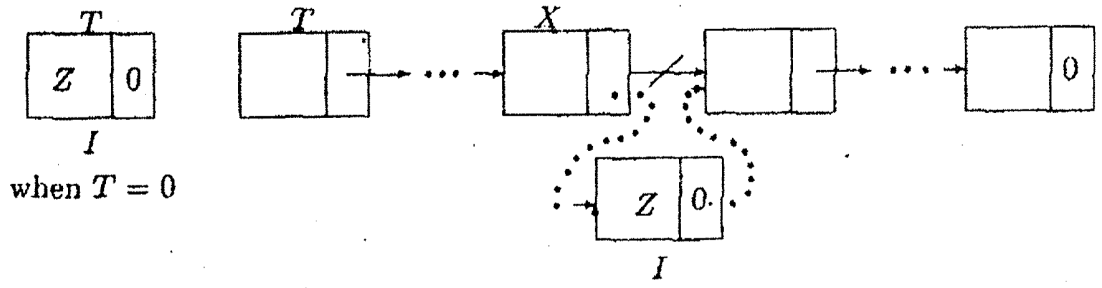


Figure 62.9: Insertion of a node into the list T.

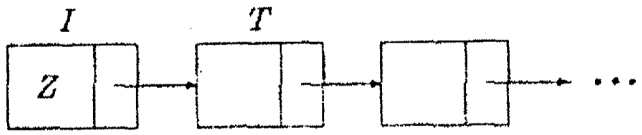


Figure 62.10: Insertion at the beginning of the list.

Procedure insertb(T)
//insert Z at the beginning of the list.//
call GETNODE (I);
data(I) ← Z;
link (I) ← T;
T ← I;
end insertb.

The following algorithm inserts an element Z at the end of the list T, where X is the last element of the list.

Procedure inserte (T, X)
//inserts Z at the end of the list T.//
call GENTNODE (I);
data (I) ← Z;
link (X) ← I;
link (I) ← 0;
end inserte.

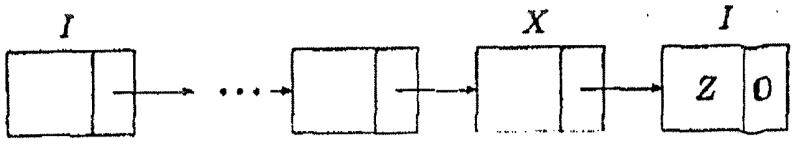


Figure 62.11: Insertion at the end of the list.

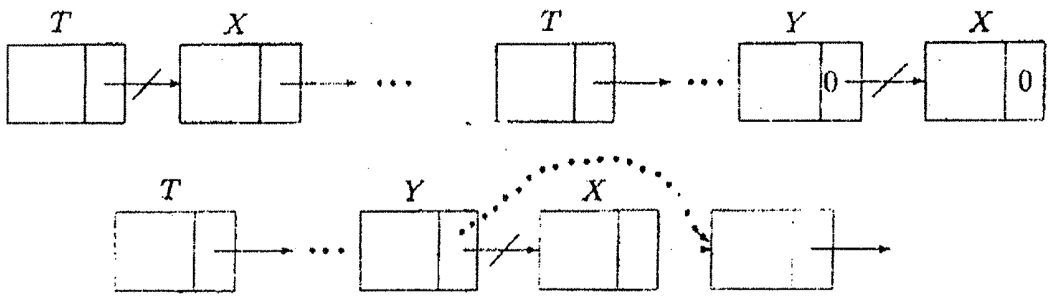


Figure 62.12: Deletion of the node from the list T.

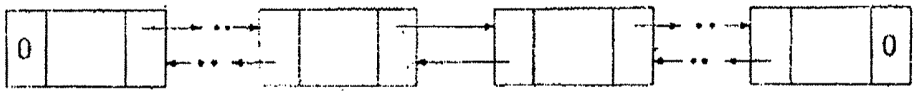


Figure 62.13: A double linked list.

Example 62.10.2 Let X be a pointer to some node in a linked list T. Let Y be the node preceding X. $Y = 0$ if X is the first node in T (i.e., if $X = T$). The following algorithm deletes node X from T.

```
Procedure delete (X, Y, T)
if Y = 0 then
        T ← link(T); //removes the first node
elseif
        link(X) = 0 then link(Y) ← 0; //removes last node
else
        link(Y) ← link(X); //removes an interior node
endif;
call RET(X); // returns node to storage poll.
end delete.
```

The single linked list cannot be traverse backward, nor can a node be deleted from a list, given only a pointer to that node. In cases where these facilities are required, the appropriate data structure is a double linked list. A node in a double linked list has three fields, say data, link (left link), rlink (right link). Merely add to the node an extra data member that stores a link to the previous node. The cost of this extra link is an increase in the space requirement and doubling of the cost of the insertions and deletions because there are more links to fix. On the other hand, it simplifies deletion, because it does not require to refer an item by using a pointer to the previous node; this information is now at hand.

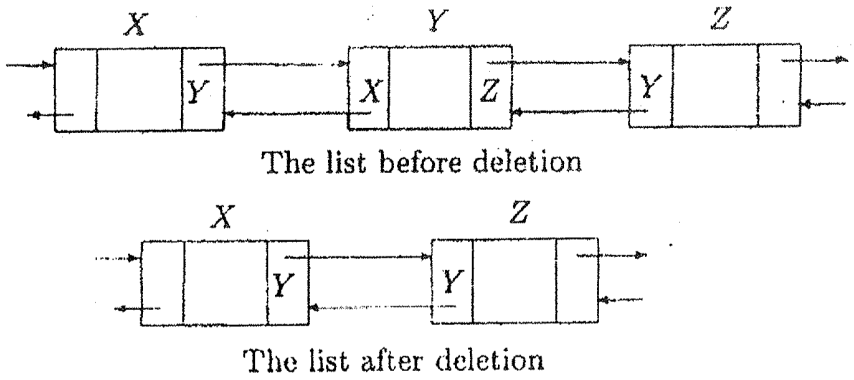


Figure 62.14: Deletion a node from a double linked list.

Deletion of a node from a double linked list.

We want to delete a node Y from a double linked list, using the following procedure.

Procedure delete (Y)

//deletes the node Y from a double linked list; value is the data stored in the node Y.//

value $\leftarrow$ *data* (Y);

$p \leftarrow$ *llink* (Y);

$n \leftarrow$ *rlink* (Y);

rlink (p) $\leftarrow n$;

llink (n) $\leftarrow p$;

RET (Y);

end delete.

The lines

$p \leftarrow$ *llink* (Y);

$n \leftarrow$ *rlink* (Y);

rlink (p) $\leftarrow n$;

llink (n) $\leftarrow p$;

can be replaced by

rlink(*llink*(Y)) $\leftarrow$ *rlink*(Y);

llink(*rlink*(Y)) $\leftarrow$ *llink*(Y);

Insertion of a node to the double linked list.

We want to insert a node Y after the node X in a double linked list.

Proceudre dinsert (Y, X)

llink(Y) $\leftarrow X$;

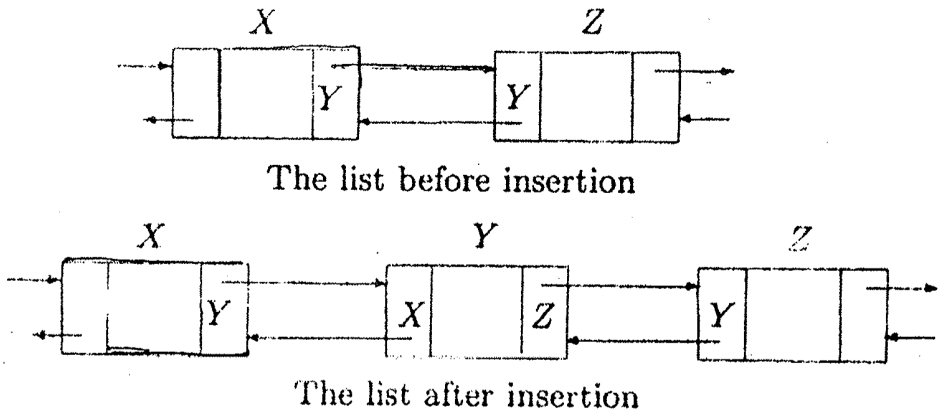


Figure 62.15: The double linked list before and after insertion.

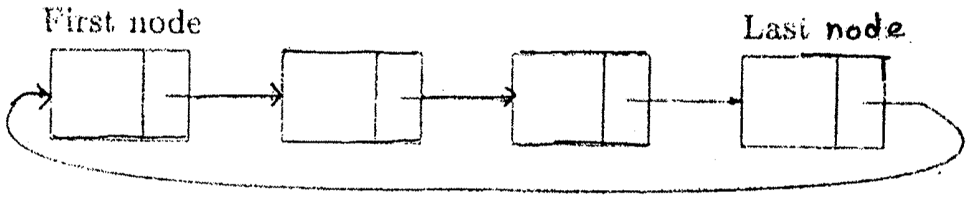


Figure 62.16: A circular list with first and last nodes.

rlink(*Y*) *rlink* (*X*);
llink(*rlink*(*X*)) ← *Y*;
rlink (*X*) ← *Y*;
data (*Y*) ← *value*; //value is the data of the node *Y*
end dinsert.

62.10.1 Circular lists

Given a pointer *p* to a node in a linear list, we cannot reach any of the nodes that precede *node*(*p*). If a list is traversed, the external pointer to the list must be preserved to be able to reference the list again.

Suppose that a small change is made to the structure of a linear list, so that the next field in the last node contains a pointer back to the first node raher than the null pointer. Such a list is called a circular list.

From any point in such a list it is possible to reach any other point in the list. If we begin at a given node and traverse the entire list, we ultimately end up at the starting point. A circular list does not have a natural 'first' or 'last' nodes.

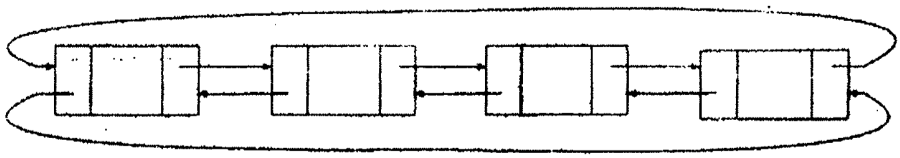


Figure 62.17: A double circular linked list.

62.10.2 Applications of linked lists

Example 62.10.3 Write an algorithm to create a single non-circular linked list containing n elements $x_1, x_2, ..., x_n$ and find the maximum among them.

Solution.

Algorithm MAX

//Creates a linked list of size n and finds maximum among them.//

```
        //creates the head node//
        getnode (p); //p is a node
        read x;
        data (p) = x;
        link (p) = 0;
        head = p; // 'head' is the head node of the list
        prev = p; // 'prev' is the prevoius node of the node p
        for i = 2, to n do
                getnode (p);
                read x;
                data (p) = x;
                link (p) = 0;
                link (prev) = p;
                prev = p;
        endfor;
//end of creation of the list.
        p = head;
        max = data (p); //max represents the maximum value of the data
        p = link (p); //moves to the next node
        fr i = 2 to n do
                if data (p) > max then max = data (p);
                p = link (p);
```

endfor;
end MAX.

Example 62.10.4 Write an algorithm for performing linear search over a linked list with one link field in each node.

Solution. We assumed that the linked list is already created containing certain data. Let *head* is the hed node of the list.

Algorithm Search-Linear
//*head* is the head node of the list.//
Input : a linked list and a key.
Output : true or false;
//if the key contain in the list then we set *found* = *true* otherwise *found*= *false*; found is a boolean variable.//
found = *false*;
p = *head*;
pos = 0;// *pos* represents the position of the node contains key in the list and *p* is the address of the node containing key//
while $p \neq 0$ do
pos = *pos* + 1;
if *data* (*p*) = *key* then *found* = *true* and exit;
p = *link* (*p*);
endwhile;
if *found* then write 'key is in the list at *pos* position of the list'.
end Search-Linear.

Example 62.10.5. Write an algorithm for insertion sort using single linked list (non-circular).

Solution. The basic step in this method is to insert a record *R* into a sequence of ordered records in such a way that the resulting sequence is also ordered. Here we sorting the data in ascending order.

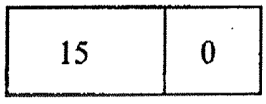
In insertion sort we always insert a node to the list in such a way that the resulting list becomes a sorted sequence. Insertions are made in two ways. If the data of the head node is less than the input value *x* (at a particular instant) then we insert a new node containing *x*, before the head node and this new node becomes the head of the appended list, otherwise we search the appropriate position for the input value and we insert a new node containing this value.

Algorithm Insort
//Sorts the data in ascending order by insertion sort using linked list.//
getnode (*p*);

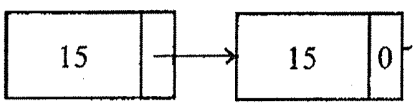
```
    read(x);  //x is the input value//
    head = p; // create a new head node//
    data(p) = x;
    link(p) = 0;
    prev = p;
    for i = 2 to n do // insert remaining n – 1 data //
        p = head;
        read (x);
        if data (p) ≥ x then //insert a node before the head node //
            getnode (p);
            data(p) = x;
            link (p) = head;
            head = p;
        else
            while (data(p)<x) and (p ≠ 0) do
                prev = p; // search appropriate position for x
                p = link (p);
            endwhile;
            getnode(q);
            data(q) = x;
            link(prev) = q;
            link(q) = p;
        endif;
    endfor;
end Insort.
```

For illustration, let us consider the data 15, 10, 25, 20.

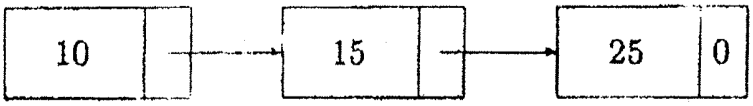
At the initial stage the list is as follows:



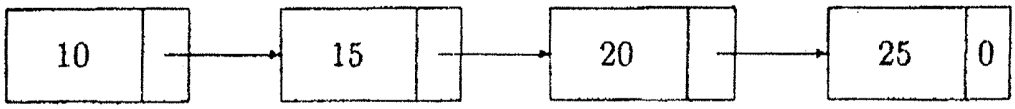
When $x = 10$ then a new node is inserted at the beginning of the list as follows:



When $x = 25$ then a new node containing $x = 25$ is inserted at the end of the list.



When $x = 20$ then a new node whose data filed 20 is inserted between the nodes containing 15 and 25 as follows:



Observed that the list is sorted sequence at each instant.

Time complexity of Insort (using linked list)

At any instant, assume that i elements are in the list. To insert a new item to this sorted list takes at most $O(i)$ time (while loop in else block). So for a fixed i the for loop takes $O(i)$ time. Hence the overall time complexity is

$$\sum_{i=2}^{n} O(i) = O(\sum_{i=2}^{n} Oi) = O(n^2).$$

62.10.3 Addition of two polynomials

The manipulation of symbolic polynomial has becomes a classical example of the use of list processing. In general, we want to represent the polynomial

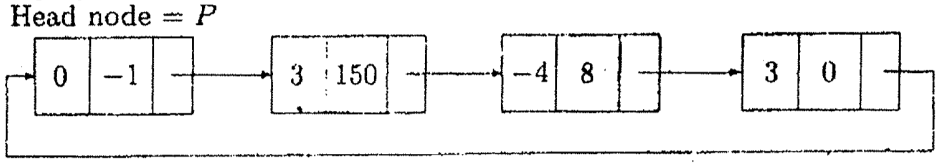
$$P(x) = a_m x^{e_m} + a_{m-1} x^{e_{m-1}} + \ldots + a_1 x^{e_1}$$

where the a_i are non-zero coefficients with exponents e_i such that $e_m > e_{m-1} > \ldots > e_2 > e_1 \geq 0$. Each term will be represented by a node. A node will be of fixed size having three fields which represent the coefficient and exponent of a term plus a pointer to the next term as shown below:



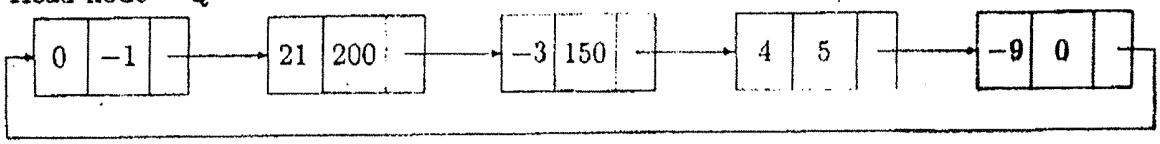
For instance, the polynomial $P(x) = 3x^{150} - 4x^8 + 3$ would be stored as

Head node = P



while $Q(x) = 21x^{200} - 3x^{150} + 4x^5 - 9$ would look like

Head node = Q

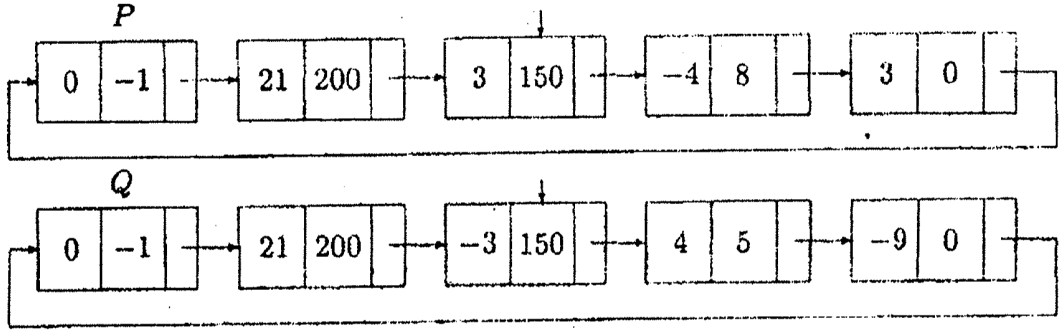


It may be noted that for each head node the dummy coefficient 0 and exponent −1 are initialized. Here two polynomials P and Q are added and replaced the value in the polynomial P.

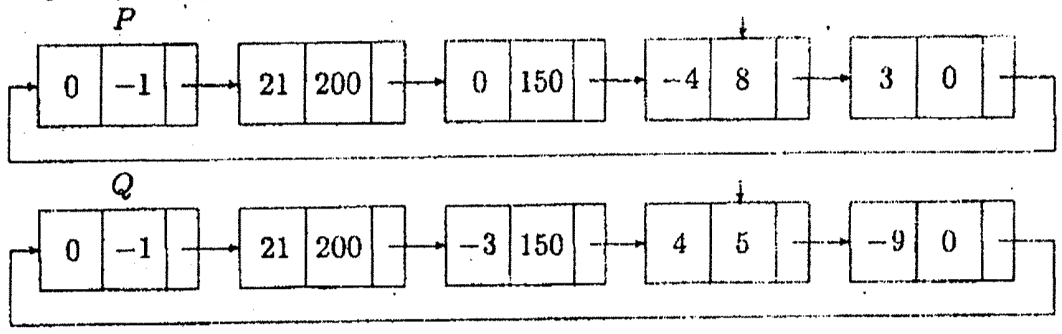
Addition of polynomials without third list

In order to add two polynomials together we examine their terms starting at the nodes after head nodes pointer by P and Q. Two pointers x and y are used to move along the nodes of P and Q. If the exponents f two terms are equal, then the coefficients are added and replace it to the node if P and pointers x and y are advanced to the next terms. If the exponent of the current term in P is less than the exponent of the current term of Q, then a duplicate term of Q is screated and attached to P before the current term of P. The pointer y is advanced to the current term of P. The pointer y is advanced to the next term. If the exponent of current term of P is greater than the exponent of the current term of Q, them advance the pointer Ix *to the next term.*

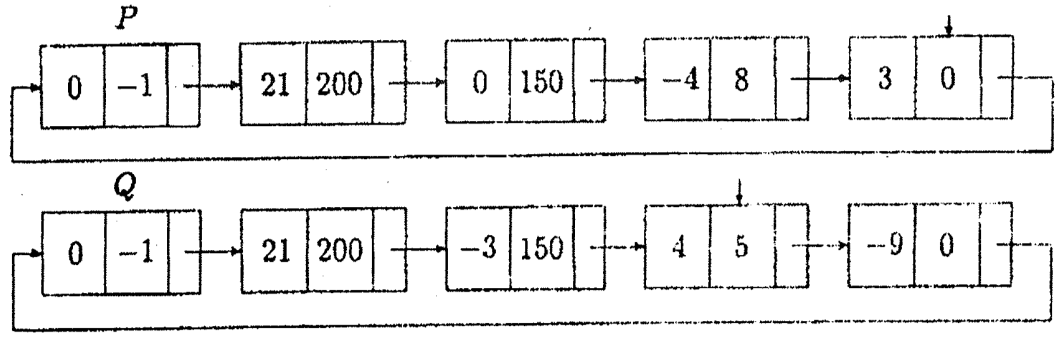
Step 1: $(exp(x) < exp(y))$



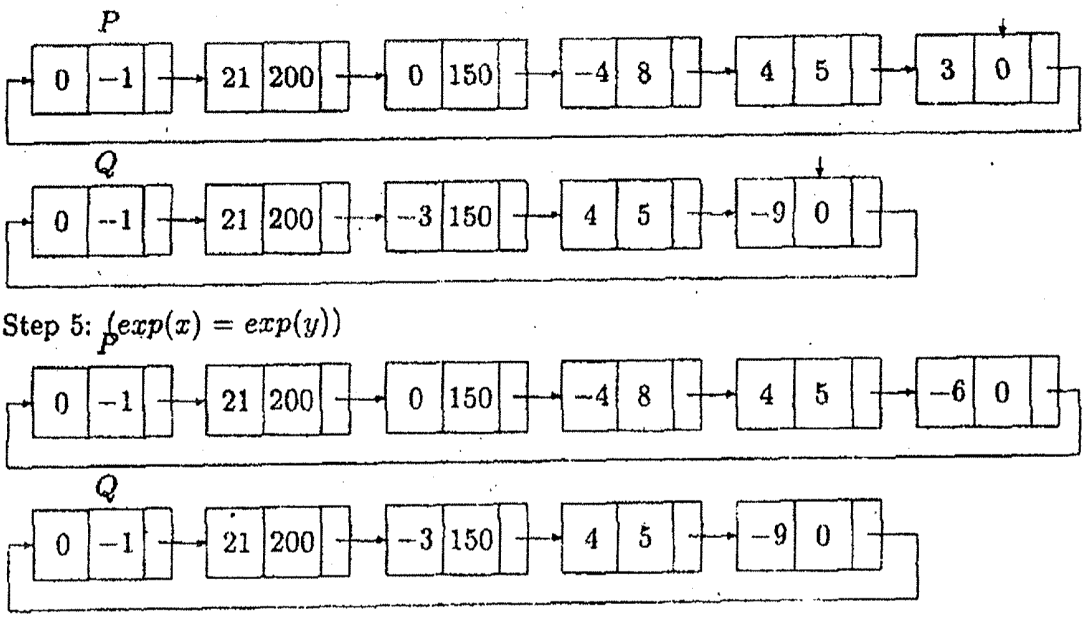
Step 2: $(exp(x) = exp(y))$



Step 3: $(exp(x) > exp(y))$



Step 4: $(exp(x) < exp(y))$



The sum of the polynomials P and Q is given by

$$P(x) = 21x^{200} + 0.x^{150} - 4x^8 + 4x^5 - 6.$$

Algorithm AddPoly ($P \leftarrow P + Q$)

// Addition of two polynomials P and Q; result stored in P; using circular linked list; P and Q are the heads of two lists.//

$X = link\ (P)$; $prex = P$;
$Y = link\ (Q)$; $prey = Q$;
while $Y \neq Q$ do
case $exp\ (X) = exp\ (Y)$:
$coef\ (X) = coef\ (X) + coef\ (Y)$;
if $coef\ (X) = 0$ then delete X from the list P;
$prex = X$; $prey = Y$;
$X = link\ (X)$; $Y = link\ (Y)$;
case $exp\ (X) < exp\ (Y)$:
$getnode\ (u)$;
$exp\ (u) = exp\ (Y)$; $coef\ (u) = coef\ (Y)$;
$link\ (u) = X$; $link\ (prex) = u$;
$prex = u$; $prey = Y$; $Y = link\ (Y)$;
case $exp\ (X) > exp\ (Y)$:
$prex = X$; $X = link\ (X)$;
endwhile;
end AddPoly

Polynomial addition without destroying input data, i.e., using third list

To add two polynomials together we examine their terms starting at the nodes pointed to by P and Q. Two pointers x and y are used to move along the terms of P and Q. Let R be the new list. If the exponents of two terms are equal, then the coefficients are added and a new term is created to the list R. If the exponent of the curent temr in P is less than the exponent of the current term of Q, then a duplicate of the term of Q is created and attached to R. The pointer y is advanced to the next term. Similar action is taken of P if $exp\ (x) > exp\ (y)$.

Each time a new node is generated its $coef$ and exp fields are set and it is appended to the end of the list R. In order to avoid having to search for the last node in R each time a new node is added, we keep a pointer d which points to the current last node in R. The complete addition algorithm is specified by the proceudre PADD. PADD makes use of a subroutine JOIN which creates a new node and appends it to the end or R. To make things work out neatly, R is initially given a single node with no values which is deleted at the end of the algorithm.

Proceudre JOIN (C, E, d)

// Create a new term with $coef = C$ and $exp = E$ and attach it to the node pointed at by d.//

```
    getnode (z);
    exp (z) ← E;
    coef (z) ← C;
    link (d) ← z;
    d ← z;
end JOIN
Algorithm PADD (P, Q, R)
//Polynomials P and Q represented as single linked lists are summed to form the new list named R.//
    x ← P; y ← Q; // x, y points to next terms of P, Q//
    getnode (R);   d ← R; // initial node for R //
    while x ≠ 0 and y ≠ 0 do // while there are more terms in P and Q. //
        case exp (x) = exp (y) :
            t ← coef (x) + coef (y);
            if t ≠ 0 then JOIN (t, exp(x), d);
            x ← link (x); y ← link (y) //advance to next terms//
        case exp (x) < exp (y):
        JOIN (coef (y), exp (y); d);
        y ← link (y); //advanced to next term of Q//
        case exp (x) > exp(y):
            JOIN(coef (x), exp (x), d);
```

```
            x link (x); //advanced to next term of P//
        endcase;
        while x ≠ 0 do //copying remaining terms of P//
            JOIN (coe f (x), exp (x), d);
            x ← link (x);
        endwhile;
        while y ≠ 0 do //copying remaining terms of Q //
            JOIN (coe f(y), exp (y), d);
            y ← link (y);
        endwhile;
        link(d) ← 0; z ← R; R ← link (R) // delete extra initial node.//
        RET (z);
end PADD
```

62.11 Module Summary

At the beginning of this module we introduced some mathematical notations those are used to analysed the data structure and the algorithms. The definitions of time and space complexities are given. The formal definition of data structure is also provided. The very common data structures such as arrays, stacks and queues are discussed. The algorithm to convert an infix expression to postfix expression and evaluation of postfix expression are given. Some examples are also supplied to understand the infix to postfix conversion. A dynamic data structure, linked list and its variants is studied in detail. As applications of linked lists we considered some problems and they are solved. An exercise and references are also provided at the end of this module.

62.12 Self Assessment Questions

1. Explain the following terms : O (big oh), (ii) θ, (iii) Ω, (iv) ω, (v) o (little oh), (vi) time complexity, (vii) space complexity.
2. Write an algorithm to store a set of real numbers into a stack and find their sum.
3. Define infix, prefix and postfix expressions. Convert the following infix expressions into prefix and postfix expressions : (*a*) $A + B * C - (D + E) * C$, (b) $A * ((B + C) + G / (D + A)) - G$.
4. Write an algorithm to evaluate a postfix expression.
5. Write an algorithm to convert an infix expression to postfix expression.
6. Write an algorithm to create a queue. Also, write algorithms for insertion and deletion operations on queue.

7. Discuss about the implementation of queue as a circular array.
8. What do you mean by a linked list? Write algorithms to insert a node to the linked list and to remove a node from the linked list. (Here a single non-circular linked list is meant for linked list).
9. Write an algorithm to create a linked list containing n numbers and find maximum among them.
10. Suppose two linked lists L_1 and L_2 are given. Write an algorithm to concatenate these two lists.
11. Suppose a linked list (single or double) is given. Write a procedure to add a node containing the value, say, X after a given node.
12. Write an algorithm to add two polynomials without using a third list.
13. Write a algorithm to add two polynomials using a third list.

62.13 Suggested Further Readings

1. E. Horowitz, S.Sahni and D. Mehta, Fundamentals of Data Structures in C++, Galgotia.
2. N. Writh, Data Structures + Algorithms = Programs, Prentice Hall.
3. Adam Drozdek, Data Structures and algoithms in C++, PWS Publishing Company, Boston.
4. B.R. Preiss, Data Structurs and Algorithms with Object-Oriented Design Patterns in C++, John Wiley and Sons, Inc.
5. Y. Langsam, M.J. Augenstein and A.M. Tenenbaum, Data Strucutrs Using C and C++, Prentice-Hall of India.

--- 0 ---

M.Sc. Course
in
Applied Mathematics with Oceanology
and
Computer Programming

PART-II

Paper-VI Group–A

Module No. - 63

DESIGN AND ANALYSIS OF ALGORITHMS

Structure :

63.1 Introduction

63.2 Objectives

63.3 What is Algorithm?

63.4 The Basic Steps to Develop a Program

63.5 Searching

- 63.5.1 Linear search
- 63.5.2 Binary search

63.6 Sorting

- 63.6.1 Bubble sort
- 63.6.2 Straight selection sort
- 63.6.3 Simple insertion sort
- 63.6.4 Quick sort
- 63.6.5 Heap sort
- 63.6.6 Merge sort

63.7 Trees

- 63.7.1 Binary tree traversals

63.8 Representation of Graphs in Computer

63.9 Graph Searching

63.9.1 Depth First Search
63.9.2 Breadth First Search
63.10 Shortest Path Problem
63.10.1 Shortest path from a specified vertex to another specified vertex
63.11 Module Summary
63.12 Self Assessment Questions
63.13 Suggested Further Readings

63.1 Introduction

Development of an algorithm to solve a problem is the most complicated part to design a program/ software. To design an algorithm two issues should be kept in mind. These are time and space complexities. The time and space complexities are the part of the analysis of algorithm. The proper use of data structure and appropriate technique. an algorithm becomes efficient.

63.2 Objectives

Gone through this module the students will learn the following topic:

* Steps to develop a programs
* Searching
* Sorting
* Trees
* Traversal of binary trees
* Searching of graphs (DFS and BFS)
* Shortest path problem.

63.3 What is Algorithm?

The most central concepts in computer science is algorithm. The meaning of algorithm is 'any special method of solving a certain kind of problems'. The formal definition of algorithim is given below.

Def. 63.3.1 Algorithm. *An algorithm is a finite sequence of steps/instructions to solve a problem. The possible steps are :*

(a) Input,
(b) Computation and decision
(i) assignment, (ii) decision, and (iii) iteration or repetition,
(c) Output.

An algorithm should follow the following criteria.

(i) Input : An algorithm has precise inputs which are externally supplied.

(ii) Definiteness : Each steps must be clear, complete and unambiguous.

(iii) Finiteness : In all possible cases an algorithm will terminate after a finite number of steps.

(iv) Effectiveness : All the steps specified in an algorithm should be execute and in a finite amount of time.

(v) Generality : An algorithm should be designed in such a way that it can solve any problem of a particular type for which it is designed.

(vi) Output : It has at least one output.

An algorithm can be described in many ways. A natural language such as English may be used to design an algorithm but we must be careful that it is concise and satisfy all the conditions of the properites of algorithm and this code is known as **pseudocode**. There is no definite rules for describing algorithm but the main objective is it should be easily understandable description of the logicl to solve a problem. The algorithm is independent of any programming language.

The following is an algorithm to find the maximum among n numbers.

Algorithm MAXA

Step 1. Read n (number of elements) and $x_1, x_2, ..., x_n$;

Step 2. Set $max = x_1$;

Step 3. for $i = 2$ to n do

if $max < x_i$ then $max = x_i$ endif;

endfor;

Step 4. Print max;

Step 3. End.

63.4 The Basic Steps of Develop a Program

The basic steps to designed a program are ;

(i) Statement of the problem;

(ii) Development of a model,

(iii) Design of the algorithm,

(iv) Correctness of the algorithm,
(v) Implementaton,
(vi) Analysis and complexity of the algorithm,
(vii) Program testingl
(viii) Documentation.

Statement of the problem

Before we can understand a problem, we must be able to give it a precise statement. This condition is not sufficient for understanding a problem, but it is absolutely necessary.

Developing a precise problem statement is usually a matter of asking the right questions. Some important questions to ask upon a crudely formulated problem are :

* What information has been given?
* What do I want to find out?
* What assumptions have been made?
* How would I recognize a solution?
* Is any of the given information worthless?

Other questions may possible, depending on the problem.

Development of a model

Once a problem has been clearly stated, it should formulate as a mathematical model. This is a very important step in the overall solution process. The choice of a model has substantial influence on the remainder of the solution process. It is impossible to provide a set of rules which automates the modelling stage. Most problems must be given individual attention.

Design of an algorithm

Once a problem has been clearly stated and a model has been developed, we must get down to the business of designing an algorithm for solving the problem. The choice of a design technique, which is often highly dependent on the choice of model, can greatly influence the effectiveness of a solution algorithm. Two different algorithms may be correct, but, may differ tremendously in their effectiveness.

Correctness of the algorithm

One of the most difficult step in the design of an algorithm is providing or asserting that the algorithm is correct.

Perhaps the most common procedure followed to assert that a program is correct is to run it on a variety of test cases. If the answers produced by the program can be verified against hand calculations or known values, we are tempted to conclude that the program 'works'. However, this technique merely removes all doubts that there is some case for which the program will fail.

For now, we offer the following as a general guide for providing the correctness of an algorithm. Suppose that an algorithm is given in terms of a series of steps, say, step 0 through step m. Try to offer some justification for each step. In particular, this might involve a lemma about conditions that exist before and after the step is executed. Then try to offer some proof that the algorithm terminate and in doing so, will have examined all the appropriate input data and produced all the approximate output data.

Implementation

Once an algorithm has been stated in terms of a sequence of steps and one is convinced that it is correct, it is time to implement the algorithm, that is, to code it into a computer program.

This fundamental step can be quite hard. One reason for the difficulty is that all too often a particular step of an algorithm will be stated in a form that is not directly translatable into code, e.g., one of the steps in the algorithm may be stated in such a way as to require an entire subroutine for its implementation. Another reason why implementation can be a difficult process is that before we can even begin to write code, we must design an entire systems of computer data structures to represent important aspects of the model being used.

To do this, we must answer such questions as the following :

(i) What are the variables?

(ii) What are their types?

(iii) How many arrays and of what size are needed?

(iv) Would it be worthwhile to use linked lists?

(v) What subroutine are needed?

(vi) What programming language should be used?

The particular implementation used can significantly affect both the memory requirements and the speed of the algorithm.

Analysis and complexity of the algorithm

There are a number of important practical reasons for analyzing algorithms. One reasons is that we need to obtain estimats or bounds on the storage or run time which our algorithm will need to successfully process a particular

input. Computer time and memory are relatively expensive resources which are foten simultaneous sought by many users.

There are also important theoretical reasons for analyzing algorithms. One would like to have some quantitative standard for comparing two algorithms which claim to solve the same problem. The weaker algorithm should be improved or discarded.

to analyzed an algorithm we assume that a random access machine is available which permits one to either access or store any element in a mixed amount of time. Given an algorithm to be analyzed, the first task is to determine which operations are involved and what are their costs. These operations may include - addition, substraction, multiplication and division. Other basic operations might include arithmetic on floating point numbers, comparison, assigning values to variables and executing procedure calls. These operatons tpically take no more than a fixed amount of time and so we say that their time is bounded by a constant. This is not true for all operations. For example, a comparison of two character strings may use a character compare instruction which may use a shift and bit compare instruction. The total time for the comparison of two strings will depend upon their lengths, while the time for each character compare is bounded by a constant.

The second task is to determine a sufficient number of data sets which cause the algorithm to exhibit all possible patterns of behavior. This is one of the important and creative tasks of algorithm analysis. It requires us to understand the working of the algorithm well enough.

In producing a complete analysis of the computing time of an algorithm, we distinguish between two phases: a priori analysis and a posteriori testing. In a priori analysis we obtain a function which bounds the computing time of algorithm. In a posteriori testing we collect actual statistics about the algorithms consumption of time and space while it is executing. We ish to determine the total time that statement will spend executing given some initial state of input data. This requires essentially two items of information, the statement's **frequencey count**(i.e., the number of times of statement will be executed) and the time for one execution. The product of these two numbers is the total time. Sine the time per execution depends on both the machine being used and the programming language together with its compiler, and a priori analysis limits itself to determining the frequency count of each statement. This number can be determined directly from the algorithm, independent of the machine it will be executed on and the programming language the algorithm is written in.

Program testing

Once a program has been coded, it is time to run it. As we know all too well, running is preceded by debugging. After a variety of syntatic, key puncing and logical errors have been corrected, we finally get our program to run for a simple case.

Documentation

Documentation is not the last step in the complete development of an algorithm. In particular, it does not consists of adding comment when finished everything else. Thedocumentation process should be interwoven with the entire development of the algorithm and especially with the design and implementation steps.

It is hard to read some one, raw code. The most obvious reason for documentation is to enable individuals to understand programs which they did not write. Of course, the best way to do this is to write code that is so clear that it is virtually self-documenting.

63.5 Searching

In this section, different methods of searching on large amount of data to find a particular piece of information are discussed. Following the definition of some useful terms related to searching.

A **table** or a **file** is an array or a group of elements, each of which is called a **record**. Each record is associated with a **key**, which helps in distinguishing it from other records. For every file ther is at lest one set of unique keys (or more) (*i.e.*, no two records have the same key value). Such a key is called a **primary key.** 'Keys need not always be unique. For example, in a file containing names and addresses, if the state is used as the key for a particular search, it will be probably not unique, since there may be two records with the same state in the file. Such a key is called a **secondary key.**

A **search algorithm** is an algorithm that accepts an argument 'a' and tries to find a record whose key value is 'a'. The algorithm may return the entire record or it may return a position of that record in the file or array. It is possible that the search of a particular argument in a file or an array may be unsuccessful, *i.e.*, there may be no record in the table with that argument as its key. In such case the algorithm should return the message "not found". A successful search is sometime called a **retrieval.** A table of records in which a key is used for retrieval is often called a **search table** or a **dictionary.**

Searching is a very useful technique used in Computer Science and in other real life problems, i.e., to check a file in a subdirectory, a book in a library, or a particular telephone number in a telephone directory etc.

There are several searching techniques available in literature, namely linear search, binary search, fibonacci search, interpolation search, binary tree search etc.

In this section the following searching techniqus are discussed.

* Linear search.
* Binary search.

In every search it is assume that the key is equal to the record itself, *i.e.*, each record has only one field.

63.5.1 Linear search

Linear search is the most simple searching technique among all searching techniques. Let A be an array of records. Search a record with a certain key value K from the array A. If a has n records then one may carry out the search by examining the values $A_1, A_2, \ldots A_n$, in that order, until the correct record is located. Such a search is known as **sequential search** as the records are examined sequentially.

Algorithm of linear search is described below.

Algorithm LSEARCH

```
/* Search an array A for the key K. If A_i = K for some i
then K is in the array otherwise K is not in the array*/
    for i = 1 to n do
        if A_i = K then
            Print 'Key K is the array'
            exit
        endif
    endfor
    Print 'Key K is not in the array'
end LSEARCH
```

Complexity: Since the key value is compared with all the n elements (in worst case), therefore the time complexity of this algorithm is $O(n)$.

Advantage : In this search there is need not for the data to be in either ascending or descending order.

63.5.2 Binary search

Binary search is one of the better known method for searching 'ordered' elements. In this method, the search begins by examining the record at the middle of the array rather than going through one by one from ther either end as in linear search. Let us assume that the array to be searched is ordered in non-decreasing order (and in alphabetical order for strings). Then, based on the result of the comparison with the middle value, A_m, one can draw the following conclusions:

(i) if key $K = A_m$ then the middle record contains the required element, *i.e.*, key is the array and then the searching procedure should be halted.

(ii) if $K < A_m$ then the key K may be in the subarray left of A_m.

(iii) if $K < A_m$ then the key K may be in the subarray right of A_m.

So, after each comparison either the search terminats successfully or the size of the remaining array that is to be searched is reduced one half of the original size.

Algorithm BSEARCH

/*Search a record K from an ordered array $A, A_1 \leq A_2 \leq \ldots A_n$. l and r are left and right indices of the current search interval */

$l = 1; r = n, i = 0;$

while $l \leq r$ do

$mid = [(l + r)/2]$ /* middle index */

if $K = A_{mid}$ then $i = mid$ and exit;

if $K < A_{mid}$ then $r = mid - 1$; /* search in left half */

if $K > A_{mid}$ then $l = mid + 1$; /*search right half*/

endwhile;

if $i \neq 0$ then Print 'key found at', *mid*

else Print 'key not found'

end BSEARCG

Complexity : After each comparison either the search terminates successfully or the size of the remaining array to be searched is reduced to one half of the original size. So after x key comparison the size of the remaining array to be examined will be atmost $[n/2^x]$ (where n is the total number of records in the array).In worst case, this search requires $O(\log_2 n)$ comparisons to search a key.

Merits and Demerits : In this search the record should be in order. If the records are not in order then the search produced an abnormal result (recall that the order is not required in linear search) but its main advantage is that it takes only $O(\log n)$ time, *i.e.*, for $n = 2^{10} = 1024$, this search compares only 10 records while the linear search comparers 1024 records.

The following example shows the value of l, r, mid and subarray to be searched at each step. Lt the array be 5 15 23 30 42 45 50 65 70 72 and the key element be 30.

```
  5   15   23    30    42    45   50   65   70   72
                     ↑
l = 1              mid=5                r = 10   key<A_mid
  5   15   23    30    42
            ↑
l = 1    mid=3        r = 5                     key>A_mid
            23    30    42
                  ↑
          l = 3  mid=4  r = 5                   key=A_mid=30
```

63.6 Sorting

Sorting is the technique used to arrange the objects or records in a particular order. It is an useful technique used in Computer Science to solve real life problems. Sorted records are obviously easier to search, update, delete etc. It is also easier to find the intersection between two sets. Sorting is also used in compiling computer programs when symbol tables are constructed. Using sorting one can compute the rank of the students in an examination.

Sorting can be broadly calssified into two categories :

* Internal sorting
* External sorting.

Here we shall dicuss only internal sorting.

Unfortunately, no sorting algorithm is best for every situation. Even it become difficult to decide which algorithm is best for a given situation, as there are many factors which creates influence on the effectiveness of a sorting algorithm.

The internal sorting techniques are classified into the following categories.

* Exchange sorts
  * Bubble sort
  * Quick sort
* Selection sorts
  * Straight selection sort
  * Heap sort
  * Binary tree sort
* Insertion sorts
  * Simple insertion sort
  * Shell sort
  * Address calculation sort
* Merge sort
* Radix sort

In this section we shall discuss the following internal sorting methods.

* Bubble sort
* Simple insertion sort
* Straight selection sort
* Quick sort
* Heap sort

Merge sort

63.6.1 Bubble sort

Bubble sort is basically an exchange sort with the simple idea of exchanging the values of the elements at two consecutive locations. At first, two consecutive recors (starting from first record) are compared and if the first one is greater than the second one (assuming that the sort is carred out in ascending order) then there is an exchange between the records and then the next two records are compared. This process is continue until all the records are compared. By this process, the record having largest value is placed at the end. After that the same technique is repeated from first to last but one record. This method is repeated for $n-1$ (assuming the size of the array to be n) times and after each step the total number of comparison is reduced by one.

The algorithm of bubble sort is given below.

Algorithm BUBBLE

/* sort n records of the array A in non-decreasing order */

```
    for i = 1 to n – 1 do
        for j = 1 to n – i do
            if (A_j > A_j+1) then call SWAP (A_j, A_j+1)
              /* interchange between A_j and A_j+1 */
        endfor;
    endfor;
end BUBBLE
```

The interchange between two elements can be performed using any of the following two ways.

First method (Using a third variable)

Algorithm SWAP (X, Y)

```
/* T is a temporary variable which stores the value of X */
    T = X
    X = Y
    Y = T
end SWAP
```

Second method (Without using a third variable)

Algorithm SWAP (X, Y)

```
    X = X – Y /* difference is stored in X */
    Y = X + Y + Y /* set to Y */
    X = Y – X /* Y set to X */
end SWAP
```

Let us consider the array A containing 8 records as 5 2 8 10 3 14 7 9.

The different situations are shown below:

For $i = 1, j$ varies from 1 to $n - 1 = 7$

$j = 1$	compare(5,2)	SWAP(5,2)	2 5
$j = 2$	compare(5,8)		2 5 8
$j = 3$	compare(8,10)		2 5 8 10
$j = 4$	compare(10,3)	SWAP(10,3)	2 5 8 3 10
$j = 5$	compare(10,14)		2 5 8 3 10 14
$j = 6$	compare(14,7)	SWAP(14,7)	2 5 8 3 10 7 14
$j = 7$	compare(14,9)	SWAP(14,9)	2 5 8 3 10 7 9 14

Observe that the last record is the maximum.

For $i = 2, j$ varies from 1 to $n - 2 = 6$.

$j = 1$	compare(2,5)		2 5
$j = 2$	compare(5,8)		2 5 8
$j = 3$	compare(8,3)	SWAP(8,3)	2 5 3 8
$j = 4$	compare(8,10)		2 5 3 8 10
$j = 5$	compare(10,7)	SWAP(10,7)	2 5 3 8 7 10
$j = 6$	compare(10,9)	SWAP(10,9)	2 5 3 8 7 9 10

After second step the array becomes 2 5 3 8 7 9 10 14 and at the end of the algorithm BUBBLE the array becomes 2 3 5 7 8 9 10 14 which is sorted in non-decreasing order.

Complexity : The inner loop (*i.e.*, j loop) is repeated for $n - i$ times, where $i = 1, 2, ..., n - 1$. Hence the total time complexity is $O\left(\sum_{i=1}^{n-1}(n-i)\right) = O(n^2)$.

63.6.2 Straight selection sort

The idea of this sort is also simple. In the first step, the first record is compared with all the remaining records and the minimum recorrd is plaed at the first position. In the second step, the second record is compared wiht all the remaining records, except the first one, and the record containing minimum value is placed at the second position. This process is carried out til the last but one record is compared with the last record. The algorithm is given below.

Algorithm SSORT

/* sort the array A of size n in non-decreasing order * /

```
for i = 1 to n – 1 do
    for Ij = i + 1 to n do
        if (A_j > A_j) then /* interchange between A_i and A_j */
            t = A_i;
            A_i = A_j;
            A_j = t;
        endif;
        endfor;
    endflor;
```

end SSORT

Let 5 7 3 20 10 8 7 2 6 be an array of records. The different situations of straight selection sort are shown below.

For $i = 1, j$ varies from 2 to $n = 9$

$j = 2$	compare(5,7)		5 7
$j = 3$	compare(5,3)	SWAP(5,3)	3 7 5
$j = 4$	compare(3,20)		3 7 5 20
$j = 5$	compare(3,10)		3 7 5 20 10
$j = 6$	compare(3,8)		3 7 5 20 10 8
$j = 7$	compare(3,7)		3 7 5 20 10 8 7
$j = 8$	compare(3,2)	SWAP(2,3)	2 7 5 20 10 8 7 3
$j = 9$	compare(2,6)		2 7 5 20 10 8 7 3 6

For $i = 2, j$ varies from 3 to $n = 9$

$j = 3$	compare(7,5)	SWAP(7,5)	2| 5 7
$j = 4$	compare(5,20)		2| 5 7 20
$j = 5$	compare(5,10)		2| 5 7 20 10
$j = 6$	compare(5,8)		2| 5 7 20 10 8
$j = 7$	compare(5,7)		2| 5 7 20 10 8 7
$j = 8$	compare(5,3)	SWAP(5,3)	2| 3 7 20 10 8 7 5
$j = 9$	compare(3,6)		2| 3 7 20 10 8 7 5 6

and so on for $= 3, 4, ..., n-1$ (=8).

At the end of the algorithm SSORT the array becomes 2 3 5 6 7 7 8 10 20.

Complexity : The inner loop (*i.e.*, the j loop) varies for $n-i$ times for $i = 1, 2, ..., n-1$. Hence the total time complexity is $O\left(\sum_{i=1}^{n-1}(n-i)\right) = O(n^2)$.

63.6.3 Simple insertion sort

The basic principle of insertion sort is to insert a record R into a sequence of ordered records $A_1, A_2, ..., A_i (A_1 \leq A_2 \leq .... \leq A_i)$ in such a way that the resulting sequence is also a sorted array of size $i+1$. The algorithm for insertion sort is described below. In the algorithm a fictitious record A_0 with valued $-\infty$ (*i.e.*, all records $\geq A_0$) is considered, which simplify the while loop, avoiding a test for the end of the array.

Algorithm INSERT (R, i)

/* insert a record R into the ordered sequence $A_0, A_1, ..., A_i$ in such a way that the resulting sequence is also ordered. We assume that A_0 is a record such that $R > A_0$ */

```
j ← i
while R < A_j do
    A_{j+1} ← A_j;
        /* move A_j one space right as R is to be inserted to the left of A_j */
    j ← j − 1;
endwhile;
A_{j+1} ← R
```

end INSERT

Insertion sort is now performed by beginning with ordered sequence A_0, A_1 and then successively inserting the records $A_2, A_3, ..., A_n$ into the sequence. The main algorithm for insertion sort is given below.

Algorithm INSORT

/* sort the records $A_1, A_2, ..., A_n$ in non-decreasing order, assuming $n > 1$ */

```
A_0 ← − ∞
for j = 2 to n do
    t ← A_j
    call INSERT (t, j − 1) /* insert records A_j, j =2, 3, ..., n */
endfor;
```

end INSORT

Let 8 3 10 12 7 6 13 9 be an array. Each step of insertion sort is ilustrated below.

A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8		
$-\infty$	8	3	10	12	7	6	13	9	initial	stage
$-\infty$	3	8							$i = 2$	(R=8)
$-\infty$	3	8	10						$i = 3$	(R=10)
$-\infty$	3	8	10	12					$i = 4$	(R=12)
$-\infty$	3	7	8	10	12				$i = 5$	(R=7)
$-\infty$	3	6	7	8	10	12			$i = 6$	(R=6)
$-\infty$	3	6	7	8	10	12	13		$i = 7$	(R=13)
$-\infty$	3	6	7	8	9	10	12	13	$i = 8$	(R=9)

Complexity : In worst case the algorithm INSERT (R, i) compare i+1 records before making the insertion. Hence, the complexity of algorithm INSERT is of $O(i)$. The NOSORT algorithm calls the INSERT algorithm for i = 2, 3, ..., n. So, in worst case the resulting time as of $O\left(\sum_{i=1}^{n-1}(n-i)\right) = O(n^2)$.

Note 63.6.1 The implementation of bubble sort and stragiht selection sort are very easy, but their time complexities are high. Some sorting methods are preented in the subsequent sections whose time complexities are of $O(n \log n)$, but the implementation is complicated compared to the previous two algorithms.

63.6.4 Quick sort

The quick sort scheme was developed by C.A. Hoare has the best average behaviour among all other sorting techniques. This sosrt is also referred as **partition exchanged sort.** Let A be an array of n records to be sorted. A record R is chosen from a specific position within the array (usually, the first record is chosen as R). Suppose that the records of A are partitioned in such a way that R is plaed at position j where $A_i \le R$, $i = 1, 2, ..., j-1$ and $A_j \le R$, $i = j+1, ..., n$. It can be proved that R is the jth smalelst record of A, and R remains at the jth position when the array is completely sorted. So, after the positioning has been made, the original array is now partitioned into two subarrays, one consisting of records $(A_1, ..., A_{j-1})$ and the other of records $(A_{j-1}, ..., A_n)$. Now, the two subarrays thus formed can be again sorted independently using same approach discussed above, finally resulting into a sorted array.

Before presenting an algorithm; let us illustrate the quick sort with an example. Let the initial array be

20 60 50 5, 10 90 88 32

and the first element 20 after being is placed at its prper position, the resulting array becomes

10 20 60 50 35 90 88 32

Observe that 20 is in its proper position in the array, each record to the left of 20 is less than 20 and each record to the right of 20 is greater than 20. As 20 is in its final position the original problem has now been divided into two subproblems of sorting the two subarrays

(10) and (60 50 35 10 90 88 32)

i.e., the entire array may now be viewed as

(10) 20 (60 50 35 10 90 88 32).

The first subarray *i.e.*, (10 has only one element and it is sorted. To sort the second subarray the process is repeated and the subarray is further subdivided. After, placing 60 at its proper position using same technique, the second subarray yields

10 20 (50 35 32) 60 (90 88)

and further repetitions yield

10	20	(35	32)	50	60	(90	88)
10	20	(32)	35	50	60	(90	88)
10	20	32	35	50	60	(90	88)
10	20	32	35	50	60	88	(90)
10	20	32	35	50	60	88	90

The final array is sorted.

The quick sort is best stated recursively as below where SWAP (X, Y) performs T ← X, X ← Y, Y ← T.

Algorithm QSORT (p, q)

/* sort the records $A_p, ..., A_q$ into non-decreasing order.

First record A_p is chosen to be placed in its proper position */

Initially, $p = 1, q = n$ /* n is the size of the aray A */

```
if (p < q) then
    i ← p; j ← q + 1; R ← A_p
    repeat
        repeat i ← i + 1 until A_i ≥ R;
        repeat j ← j − 1 until A_j ≥ R;
        if (i < j) then SWAP (A_i, A_j);
    until (i ≥ j);
    SWAP (A_p, A_j);
    if p < j − 1 then QSORT (p, j − 1);
    if j + 1 < q then QSORT (j + 1, q);
endif;
```

end QSORT

Complexity : Computation of time complexity is complicated. In worst case, the time complexity is of $O(n^2)$, but the average case time complexity is of $O(n \log n)$, even then this sort is faster than any other sort.

63.6.5 Heap sort

There are relatively a few sorting algorithms available whose worst case time complexity is of $O(n \log n)$. One of these is a heap sort, due to Williams, which makes use of a special tree structure called a **heap.** A heap a labelled, rooted binary tre of a given height h which has the following properties :

(i) Every leaf node has height h or $h-1$.

(ii) Every leaf node of height h appears to the left of every leaf node of height $h-1$.

(iii) Value at each node is greater than the values of its children, if they exist.

From these properties it follows that the largest element is at the root of the heap.

The tree T_1, T_2, T_3 in Figure 63.1 are not heap while the tree T_4 is a heap.

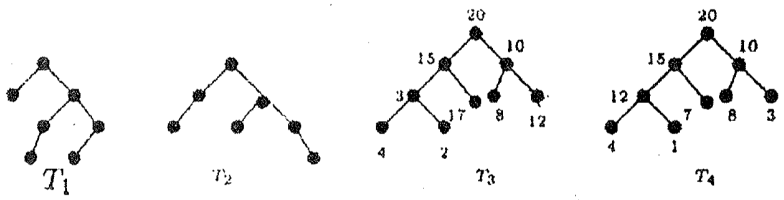


Figure 63.1: T_1, T_2, T_3 are not heap, T_4 is a heap.

The tree T_1 vilates the property (i), T_2 violates the property (ii) and T_3 violates the property (iii). A heap of n nodes can be stored in any array A where the two children of the node A_i are sotred at A_{2i} and A_{2i+1}. If $2i > n$, then A_i has no children and it is a leaf node of the tree, and if $2i \leq n$ but $2i + 1 > n$ then A_i has only a left child. The heap T_4 can be stored in any array A with size 9 as follows:

node:	1	2	3	4	5	6	7	8	9
A:	20	15	10	12	7	8	3	4	1

This sorting requirs a heap as an input. An algorithm to convert an arbitrary array of records into a heap is also needed.

The algorithm of heap sort depends on another sub-algorithm that takes a binary three T whose left and right subtree satisfy the heap property but whose root may not and the tree T is adjusted such that the entire binary tree now becomes a heap.

Let the original records be

20 50 45 32 10 88 84 30.

Figure 63.2 illustrates the adjustment of the heap as A_1 is repeatedly selected and placed into its proper position and the heap is readjusted, until all the elements are processed.

This construction is done by the repeated use of the algorithm ADJUST. The algorithm ADJUST adjusts the binary tree with root i to satisfy the heap property, where the left and the right subtrees of i, already satisfy the heap property.

Algorithm ADJUST (A, i, n)

/*ith element of the array A (of size n) is placed at its proper position in the heap */

$j \leftarrow 2i$; $R \leftarrow A(i)$;
while $j \leq n$ do
if $j < n$ and $A(j) < A(j+1)$ then /* compare left and right child */
$j \leftarrow j+1$ /* j is the position of largest child */
endif;
if $R < A(j)$ then $A(\lfloor j/2 \rfloor) \leftarrow A(j)$ and $j \leftarrow 2j$
/* largest child put up */
else exit from while loop;
endif;
endwhile;
$A(\lfloor j/2 \rfloor) \leftarrow R$;

end ADJUST

now the heap sort algorithm is presented below.

Algorithm HSORT

/* The array $A(1), A(2), ..., A(n)$ is sorted into non-decreasing order */

for $i \leftarrow \lfloor n/2 \rfloor$ to 1 step –1 do
cal ADJUST (A, i, n) /* construction of initial heap*/
endfor;
for $i \leftarrow n-1$ to 1 step -1 do /* placed $A_{n-1}, A_{n-2}, ..., A_1$ */
$T \leftarrow A(i+1)$; $A(i+1) \leftarrow A(1)$; $A(1) \leftarrow T$
/* interchange $A(1)$ and $A(i+1)$ */
call ADJUST ($A, 1, i$) /* adjust heap of i nodes */
endfor;

end HSORT

Figure 63.3 illustrates the heap after re-adjusting and the corresponding sorted part of the array.

Complexity : The algorithm ADJUST takes $O(h)$ time where h is the height of the heap and height of a heap of n nodes is $\log_2 n$. When the heap contains i elements then algorithm ADJUST takes $O(\log i)$ time. The algorithm HSORT calls ADJUST for $i = 1$ to $n-1$. Hence the time complexity of HSORT is of $O\left(\sum_{i=1}^{n-1} \log i\right) = O(n \log n)$..

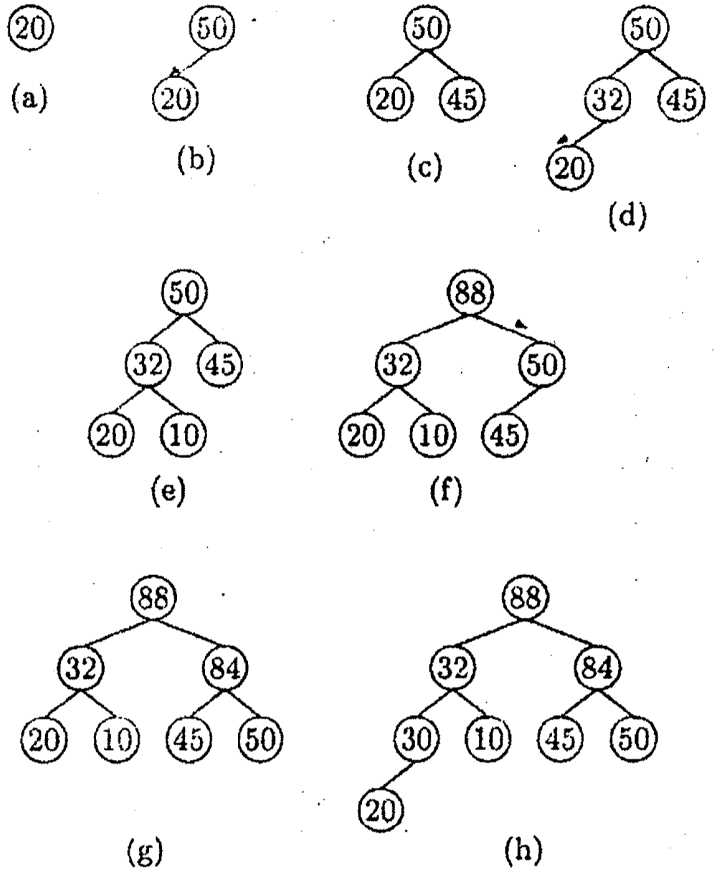


Figure 63.2:

63.6.6 Merge sort

Another sorting algorithm whose worst case time complexity of $O(n \log n)$ is merge sort or a 2-way merge sort. The merge sort begins by interpreting the input array as n sorted subarrays each of length 1. Consecutive two subarrays are merged to obtain $n/2$ subarrays each of size 2 (if n is odd then one of the subarrays is of size 1). These $n/2$ subarrays are then merged pairwise (two consecutive subarrays) and this process is continue till the number of subarrays becomes one. Then the resulting array becomes a sorted one.

Suppose there is an array A of 10 elements as

20 5 3 80 50 35 7 22 5 20

which is to be sorted in non-decreasing order.

The subarrays at different stages are shown below.

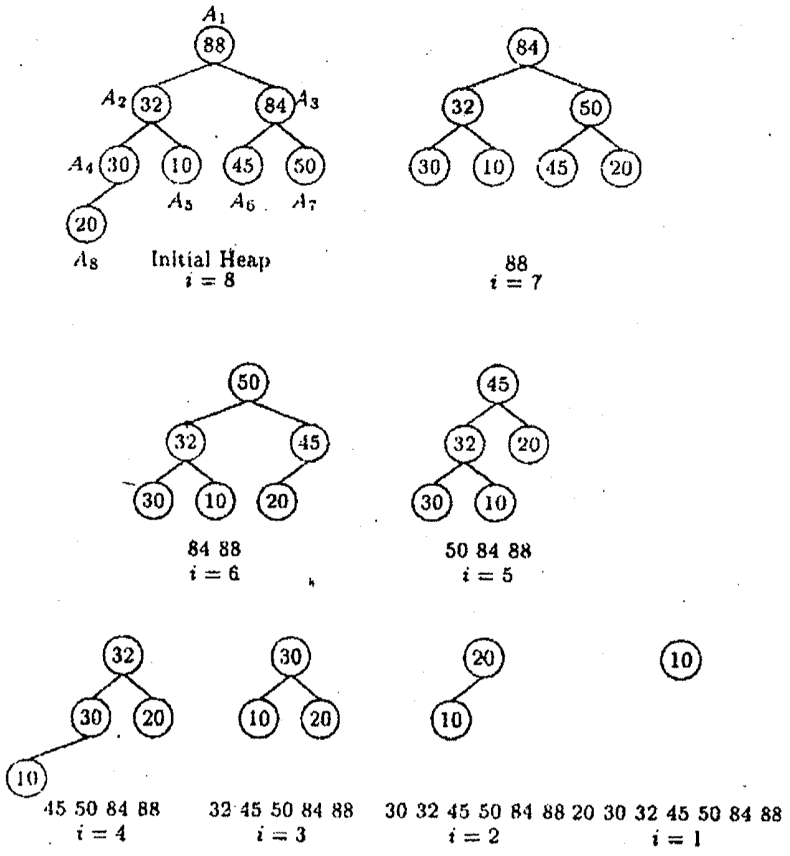


Figure 63.3:

Step-1	(20)	(5)	(3)	(80)	(50)	(35)	(7)	(22)	(5)	(20)	no. of subarrays 10
Step-2	(5	20)	(3	80)	(35	50)	(7	22)	(5	20)	no. of subarrays 5
Step-3	(3	5	20	80)	(7	22	35	50)	(5	20)	no. of subarrays 3
Step-4	(3	5	7	20	22	35	50	80)	(5	20)	no. of subarrays 2
Step-5	(3	5	5	7	20	20	22	35	50	80)	no. of subarray 1

The last subarray is a sorted one. In the following an efficient algorithm is presented to merge two sorted arrays in such a way that the resulting array becomes sorted. The algorithm MERGE does this.

Let $X = (A_i, ..., A_j)$ and $Y = (A_{j+1}, ..., A_k)$ be two sorted arrays with $A_i \le ... \le A_j$ and $A_{j+1} ... \le A_k$ and they are merged into a sorted array $A_i \le ... \le A_k$.

Algorithm MERGE (*A, i, j, k*)

/* $X = (A_i, ..., A_j)$ and $Y = (A_{j+1}, ..., A_k)$ are two soted arrays with $A_i \le ... \le A_j$ and $A_{j+1} ... \le A_k$. They are merged into an array $B_0, ..., B_{k-i}$ with $B_0 \le ... \le B_{k-i}$ and then the array B is copied into the array A from A_i to A_k such that $A_i \le ... \le A_k$ */

$p \leftarrow i; q \leftarrow j + 1; r \leftarrow 0$ /* p, q, r are the positions of X, Y and B */

```
while p ≤ j and q ≤ k do
        if A_p ≤ A_q then B_r ← A_p; p ← p + 1;
        else B_r ← A_q; q ← q + 1;
        endif;
    r ← r + 1;
    endwhile;
    if p > j then (B_r ..., B_k-l) ← (A_q ..., A_k)
        /* copying remaining elements of Y to B */
    else (B_r ..., B_k-l) ← (A_p ..., A_j)
        /* coying remaining elements of X to B */
    endif;
    for p = 0 to k – i do
        A_i+p ← B_p
    endfor;
endMERGE
```

he merge sort algorithm takes the form :

```
burithm MSORT
bort an array A in non-decreasing order, l representing the size of the subarray */
    l ← 1; initial length of each subarray is 1 */
    i ← 1; /* i is the starting position of a subarray */
    repeat
        while (i ≤ n – 2l + 1) do
            call MERGE (A, i, i + 1, i + 2l – 1);
            i ← i + 2l;
        endwhile;
        if (i + l – 1 < n) call MERGE (A, i, i + l – 1, n);
        l ← 2l;
    until (l < n);
end MSORT
```

Complexity : Time complexity of the algorithm MERGE (A, i, j, k) is $O(k-i+1) = O(n)$. The repeatuntil looop repeats $\lceil \log_n n \rceil$ times, because, at the beginning the size of subarray is 1, at the second time size is 2 and at the ith time the subarrays being merged are of size $2^i - 1$. Hence, the time complexity of merge sort is of $O(n \log_2 n)$.

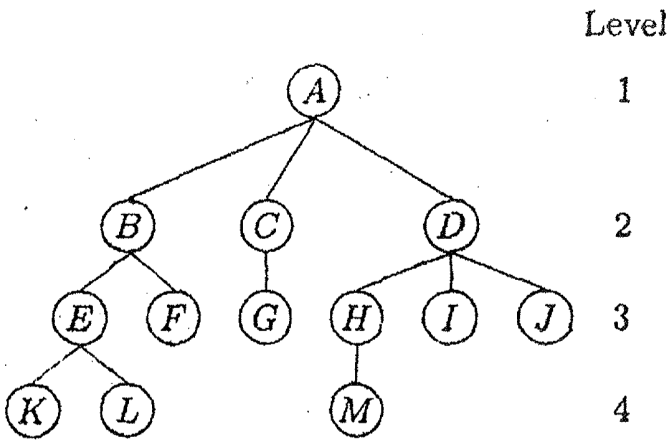


Figure 63.4:

63.7 Trees

A tress is a connected graph without circuits. A graph G with n vertices is calld a tree if

(i) G is connected and is circuitless,

(ii) G is connected and has $n-1$ edges,

(iii) G is circuitless and has $n-1$ edges,

(iv) There is exactly one path between every pair of verticeis in G,

(v) G is a minimally connected graph.

A tree can also be defined, recursively, in the following way:

A **tree** is a finite set of one or more nodes such that (i) there is a specially designed node called the roote; (ii) the remaining nodes are partitioned into $n \geq 0$ disjoint sets $T_1, T_2, ..., T_n$ where each of these sets is a tree $T_1, T_2, ..., T_n$ are called the subtress of the root.

A node stands for an item of information and the branches to other items.

The number of subtrees of a node is called its degree. The gree A (see Figure 63.4) is 3, of C is 1 and of F is zero. Nodes that have degree zero are called leaf or terminal nodes. $\{K, L, F, G, M, I, J\}$ is the set of leaf nodes.

The roots of the subrrees of a node X are children of X. X is te parent of its children. Thus, the children of D are H, I, J. The parent of D is A.

Children of the same parent are said to be siblings. H, I, J are siblings. In this manner we can define grandparent of M which is D, etc.

The degree of a tree is the maximum degree of the nodes in the tree. The tree of Figure 63.4 has degree 3.

The ancestors of a node are all the nodes along the path from the root to that node. The ancestors of M are A, D, H.

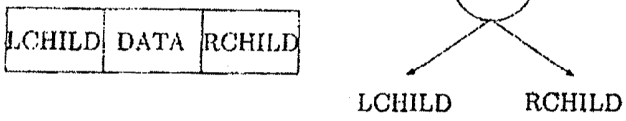
The level of a node is defined by initially letting the root be at level one. If a node is at level l, then its children are at leve $l + 1$. The height or depth of a tree is defined to be the maximum level of any node in the tree.

A binary tree is a very important data structure which occur very often. It is characterized by the fact that any node can have at most two branches, i.e., there is no node with degee greater than two. For binary trees we distinguish between the subtrees on the left and on the right, whereas for trees the order of the subtrees was irrelevant. Also, a binary tree may have zero nodes. Thus a binary tree is really a different object than a tre. The definition of a binary tree is given below.

Def. 63.7.1 (Binary tree). *A binary tree is a finite set of nodes which is either empty or consists of a root and two disjoint binary trees called the left subtrace and the right subtree.*

***A full binary tree** of depth k is a binary tree of depth k having $2^k - 1$ nodes.*

A binary tree can be represented by an array. But, it has some difficultires when a node is deleted from or inserted to the tree. These problems can be easily overcome through the use of a linked representation. Each node will have three fields LCHILD, DATA and RCHILD as below:



The binary tree may be used to store an infix expression, which is illustrated in the following example.

Example 63.7.1 Construct a binary tree to store the following infix expression.

$$A * (B + C) * ((D + E) * (F - G) + H).$$

Solution. The subexpression $D + E$ forms a binary tree with root + and the left and the right children are respectively D and E. Similarly, the root of the expression $(F - G) + H$ is + and its left and the right children are – and H. The children of the node – are F and G.

The full binary tree is hwon Figure 63.5.

It may be noted tha all the operands lie on the leaf nodes and the operators are on the internal nodes.

63.7.1 Binary tree traversals

There are many operations that we often want to perform on trees. One notion that arises frequently is the idea of traversing a tree or visiting each node in the tree exactly once. A full traversal produces a linear order for the

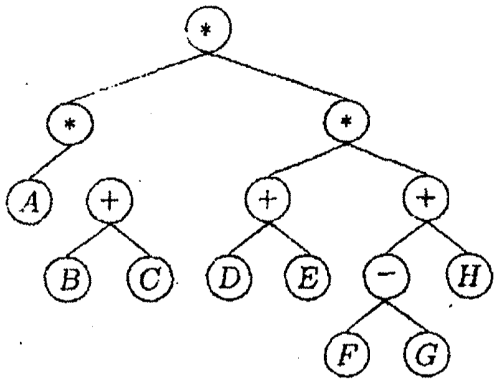


Figure 63.5:

information of a tree. This linear order may be familiar and useful. When traversing a binary tree we want to treat each node and its subtrees in the same fashion. If we let *L*, *D*, *R* stand for moving left, printing of the daa, and moving right when at a node then there are six possible combinations of traversal *LDR*, *LRD*, *DLR*, *DRL*, *RDL*, *RLD*. If we adopt the convention that we traverse left before right then only three traversals remain : *LDR*, *LRD* and *DLR*. To these we assign the names inorder, postorder and precorder because there is a natural correspondence between these traversals and producing the infix, postfix and prefix forms of an expression.

Inorder traversal

Informally, this calls for moving down the tree towards the left until we can not go further. Then 'visit' the node, move one node to the right and continue again. If we cannot move to the right, go back one more node. A precise way of describing this traversal is to write it as a recrusive procedure.

```
Proceudre Indorder (T) //recursion form//
//T is a binary tree where each node has three fields LCHILD, DATA, RCHILD//
    if T is not-empty then
        Inorder (LCHILD(T));
        visit (root);
        Inorder (RCHILD(T));
end Inorder.
```

To demonstrate the binary tree traversal we consider a binary tree of Figure 63.6. This tree contains an arithmetic expression with binary operators : add (+), multiply (*), divide (/), exponentiation (**) and variables *A*, *B*, *C*, *D* and *E*.

The elements get visits in the order

$$A/B ** C * D + E$$

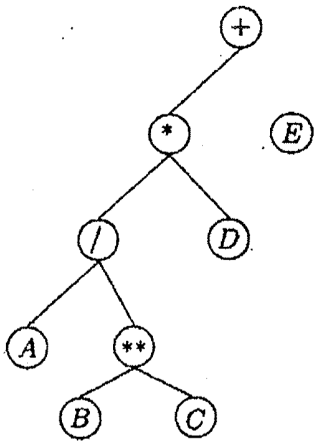


Figure 63.6: Binary tree with arithmetic expression.

which is the infix form of the expression.

A second form of traversal is preorder.

Preorder traversal

We would say 'visit a node, traverse left and continue again. When we cannot continue, move right and begin again or move back until we can move right and resume'.

The recursion algorithm is depicted below:

```
Proceudre Preorder (T)
//T is a binary tree where each node has three fields LCHILD, DATA, RCHILD//
if T ≠ 0 then
    visit (T);
    Proorder (LCHILD(T);
    Preorder(RCHILD(T);
End Preorder.
```

The nods of Figure 63.6 would be visited in preorder as

$$+ * / A ** BCDE$$

which is the prefix form of the expression.

Postorder traversal

We would say 'visit a node, trverse right and continue again. When we cannot continue move left and begin again or move back until we can move left and resume'. The recursive algorithm is given below:

```
Procedure Postorder (T)
//T is a binary tree where each node has three fields LCHILD, DATA, RCHILD//
if T ≠ 0 then
    Postorder (LCHILD(T);
    Postorder (RCHILD(T);
    visit (root);
end Postorder.
```

The output produced by Postorder for the tree of the tree of Figure 63.6 is

ABC ** /D * E +

which is the postfix form of our expression.

Though we have written these three algorithms using recursion, it is very easy to produce an equivalent nonrecursive algorithm. Let us take inorder as example.

```
Procedure Inorder 1 (T)
//a nonrecursive algorithm//
i ← 0; // initialize stack
loop
    while T ≠ 0 do // move down LCHILD fields
        i ← i + 1;
        if i > n then cal STACK_FULL;
        STACK(i) ← T; T ← LCHILD(T);
    endwhile;
    if i = 0 then return;
    T ← STACK (i), i ← i –1;
    visit (root); T ← RCHILD (T);
forever;
end Inorder1.
```

63.8 Representation of Graphs in Computer

Two commonly used techniques to represent a graph into computer are

* adjacency matrix and
* adjacency list.

Let G be a graph whose vertices, listed as $1, 2, ..., n$. The adjacency matrix A or G is the n x n matrix with entries.

$$A(u,v) = \begin{cases} 1, \text{if } (u,v) \in E, \\ 0, \text{otherwise.} \end{cases}$$

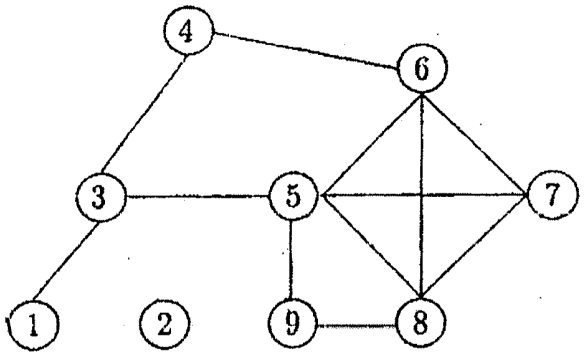


Figure 63.7: An example graph.

The main diagonal of A is all zeros if G has no loops. If G is an undirected graph then the matrix. A is symmetric about the main diagonal. If A is stored in a computer as a two dimensional array, then only one step (more precisely $O(1)$ time) is required for the statement "Is $(u, v) \in E$?" or "erase the edge (u, v)". An instruction such as "mark each vertex which is adjacent of v" is required for scanning the entire column corresponding to the vertex v and hence it takes n steps. Similarly, "mark each edge" takes n^2 steps. An $O(n^2)$ space is required to store then matrix A. A graph whose edges are weighted can be represented in the same fashion. In this more general case, $A(u, v)$ will be equal to the weight fo (u, v); a non-edge will have weight either zero or infinity depending upon the application.

A graph can also be represented by an adjacency list. A list is a sequence of zero or more elements. For a graph $G = (V, E)$, the adjacency list of a vertex $v \in V$ is is list of all the vertices adjacent to v. An adjacency list representation of G is the set of all the adjacency lists for all vertices of G. Each element in the adjacency list of a vertex consists of two memory locatons. The first one contains the item itself and second one contains a pointer to the next location. The space requirement for the adjacency list representation of a graph with n vertices and m edges is $O(n + m)$. Thus, from storage considerations, it is usally more advantageous to use adjacency lists than the adjacency matrix to store a sparse graph. Adjacency matrix representation has the advantage that we can determine whether there is any edge between two vertices u and v in $O(1)$ time while an $O(n)$ time is required if the graph is represented by an adjacency list representation. However, as discussed below, interval graphs, circular-arc graphs and permutation graphs may be stored using less space.

For illustration of adjacent matrix and adjacent list we consider the graph of Figure 63.7.

The adjacency matrix and adjacency list of the graph of Figure 63.7 are shown in Figure 63.8 and Figure 63.9 respectively.

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

Figure 63.8: Adjacency matrix of the graph of Figure 63.7.

63.9 Graph Searching

Given the root node of a binary tree, one of the most common things one wishes to do is to traverse the tree and visit the nodes in some order. Given an undirected graph $G=(V, E)$ and vertex v in V we are reachable from v (i.e., all vertices connected to v). We shall look at two ways of doing this : Depth First Search (DFS) and Breath First Search (BFS).

63.9.1 Depth First Search

Depth first search of an undirected graph proceeds as follows. The start vertex v is visited. Next an unvisited vertex v adjacent to v is selected and a depth search from w initiated. When a vertex u is reached such that all its adjacent vertices have visited, we back up to the last vertex visited which has an unvisited vertex w adjacent to it and initiate a depth first search from u. The search terminates when no unvisited vertex can be reached from any of the visited vertex. This procedure is best described recursively as in the following.

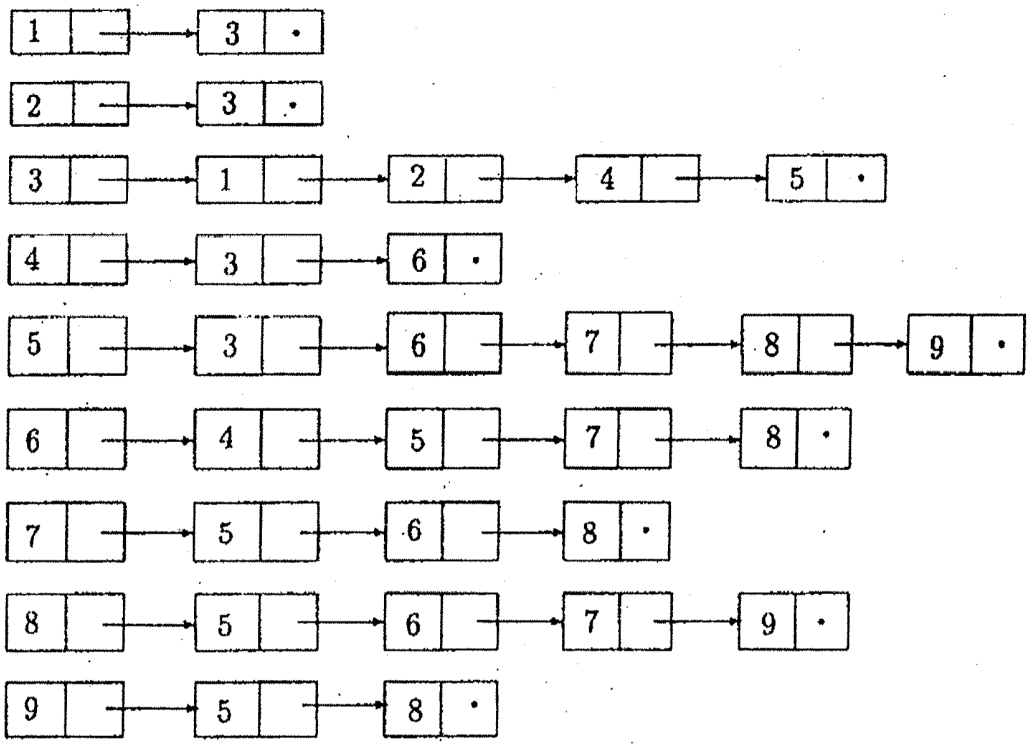


Figure 63.9: Adjacency list of the graph of Figure 63.7.

Procedure DFS(v)
//Given an undirected graph $G = (V, E)$ with n vertices and an array VISITED (n) initially set to zero, this algorithm visits all vertices reachable from v. The graph G and the array VISITED are global.//
    VISITED (v) ← 1;
    for each vetex w adjacent to v do
        if VISITED (w) = 0 then call DFS (w);
    end;
end DFS.

Analysis of the algorithm

In case G is represented by its adjacent lists then the vertices w adjacent to v can be determined by following a chain of links. Since the algorithm DFS would examine each node in the adjacency lists at most once and there are $2m$ (m is the total number of edges) list nodes, the time to complete the search is $O(m)$. If G is represented by its adjacency matrix, then the time to determine all vertices adjacent to v is $O(n)$. Since at most n vertices are visited, thte total time is $O(n^2)$.

Example 63.9.1 Find a DFS on the graph whose successor list is given below.

Vertex	x	a	b	c	d
Adjacent vertices	a	b	x, c, d	x, a	x

Solution. The graph for the given successor list is shown in Figure 63.10.

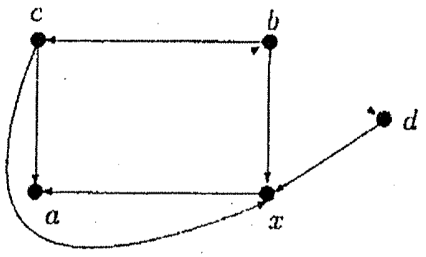


Figure 63.10: The graph G before DFS.

We start the DFS from the vertex x. We scan the edge (x, a) and draw a forward edge in the graph G. Then we traverse the edge (a, b) as this edge was not traversed previously. We seek the edge (b, x), but the vertex x already is in the graph $\overline{G}$.. So in this case we draw a back edge (dotted edge) from the vertex b to the vertex x. Then we return back to the vertex b and seek the non-visited vertex adjacent to b. The next non-visited vertex and adjacent to b is c. So we draw a forward edge (b, c). Continue this process until all the edges are scanned.

The order in which edges of G were scanned are (x, a), (a, b), (b, x), (b, x), (c, a), (b, d), (d, x).

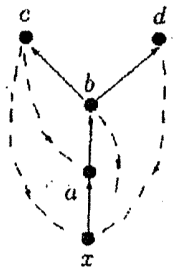


Figure 63.11: The graph $\overline{G}$ after DFS.

The DFS tree of the graph of Figure 63.10 is shown in Figure 63.10.

63.9.2 Breadth First Search

Starting at vertex v and marking it as visited, breadth first search differes from depth first search in that all unvisited vertices adjacent to v are visited next. Then unvisited vertices adjacent to these vertices are visited and so on. A

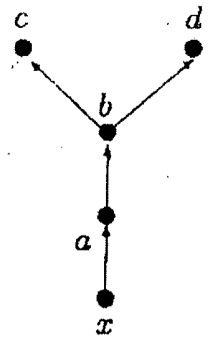


Figure 63.12: The DFS tree of the graph of Figure 63.10.

breadh first search beginning at vertex v_1 would first visit v_1 and then adjacent vertices to v_1. An algorithm is given below.

```
Procedure BFS(v)
// A breadth first search of G is carried out beginning at vertex v. All vertex visited are marked as VISITED(i)=1.
The graph G are array VISITED are global and VISITED is initialized to zero.//
    VISITED(v) ← 1;
    initialized Q to be and empty queue;
    loop
        for all vertices w adjacent to v do
            if VISITED(w) = 0 then          //add wI to queue//
              call ADDQ(w, Q); // add w to the queue Q//
              VISITED(w) ← 1; //mark w as VISITED//
            endlor;
            if Q is empty then return;
            call DELETEQ(v, Q); // delete the vertex v from Q//
        foreover;
end BFS.
```

Analysis of algorithm

Each vertex visitd gets into queue exactly once, so the loop forever is iterated at most n times. If an adjacency matrix is used, then the for loop takes $O(n)$ time for each vertex visited. The total time is, therefore, $O(n^2)$. In case adjacency lists are used the for loop has vertices visited, together with all edges incident to them form a connected component of G.

The graphy traversal techniques have many applications, such as finding the connected components of a graph, finding a spanning tree of a connected graph, etc.

63.10 Shortest Path Problem

There are different types of shortest path problems available in books. Most frequently encountered among these are the following.

(i) Shortest path between two specified vertices.

(ii) Shortest paths between all pairs of vertices.

(iii) Sortest paths from a specified vertex to all other vertices.

(iv) Shortest paths between two specified vertices that passes through some other specified vertics.

(v) The second, third and so on, shortest paths.

63.10.1 Shortest path from a specified vertex to another specified vertex

The problem of finding the shortest path from a specified vertex s to another specified vertex t, can be stated as follows:

A simple weighted digraph G of n vertices is described by an $n \times n$ matrix $D = [d_{ij}]$, where

d_{ij} = length (or distance or weight) of the directed edge from vertex i to vertex j, $d_{ij} \geq 0$;
$d_{ij} = 0$;
$d_{ij} = \infty$, if there is no edge from i to j.

The distance of a directed path P is defined to be the sum fo the lengths of the edges of P. The problem is to find the shortest possible path and its length from a starting vertex s to a terminal vertex t.

Among several algorithms that have been proposed for the shortest path between a specified vertex pair, perhaps the most efficient one is an algorithm due to Dijkstra.

Dijkstra's algorithm labels the vertices of the given digraph. At each stage in the algorithm some vertices have permanent labels and others temporary labels. The algorithm begins by assigninga permanent label 0 to the starting vertex s, and a temporary label ∞ to the remaining $(n-1)$ vertices. From then on, in each iteration another vertex gets a permanent label, according to the following rules:

1. (Computation of temporary label). Every vertex j that is not yet permanently labelled gets a new temporary label whose value is given by

   $$\min \{\text{old label of } j, (\text{old label of } i + d_{ij})\},$$

   where i is the latest permanently labelled, in the previous iteration, and d_{ij} is the direct distance between vertices i and j. If i and j are not joined by an edge, then $d_{ij} = \infty$

2. (Computation of permanent label). The smallest value among all the temporary labels is found, and this becomes the permanent label of the corresponding vertex. In case a tie, select any one of the candidates for permanent labelling.

Steps 1 and 2 are repeated alternately until the destination vertex t gets a permanent label.

The first vertex to be permanently labelled is at a distance of zero from s. The second vertex to get a permanent label (out of the remaining $(n-1)$ vertices) is the vertex closet to s. From the remaining $(n-2)$ vertices, the next one to be permanently labelled is the second closet vertex to s, and so on. The permanent label of each vertex is the shortest distance of that vertex from s. This statement can be proved by induction.

Analysis of algorithm

Steps 1 and 2 take only $O(n)$ time. But, these steps repeat at most n times. Hence the total time complexity is $O(n^2)$.

Example 63.10.1 Find the shortest distance from the vertex B to the vertex G of the graph of Figure 63.13. The numbers adjacent to the edges represent the edge weights.

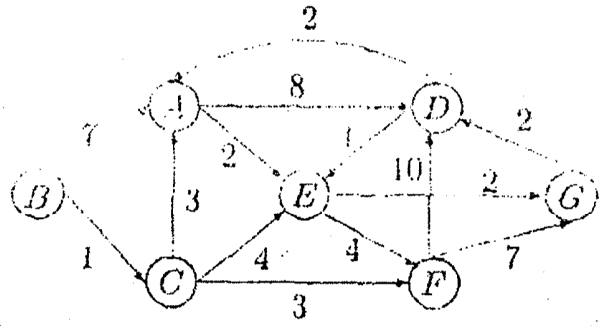


Figure 63.13:

Solution. We shall use a vector of length seven to show the temporary and permanent labels of the vertices as we go through the solution. The most recently assigned permanent label in the vector is indicated by a tick square (□√). The labelling proceeds as follows:

A	B	C	D	E	F	G
∞	0 √	∞	∞	∞	∞	∞
7	—	1 √	∞	∞	∞	∞
4	—	—	∞	5	4 √	∞
4 √	—	—	14	5	—	11
—	—	—	12	5 √	—	11
—	—	—	12	—	—	7 √

In this step the destination vertex G gets permanent label. Thus the shortest distance between the vertices B and G is 7.

Note 63.10.1 If the Steps 1 and 2 are repeated alternately until all the vertices get permanent label then this algorithm finds the shortest distances from a specified vertex to all other vertices. The time complexity remain unchanged, i.e., $O(n^2)$.

63.11 Module Summary

In this module we tried to give an idea about the design and analysis of algorithms. The basic steps to design a computer program are discussed. Two most commonly used searching methods, viz., linear and binary search are incorporated here. It is well known that the sorting is used very frequently in different problems. The bubble sort, straight selection sort, simple insertion sort, quick sort, heap sor and merge sort algorithms are presented and their complexiiteis are calculated. The tree data structure is introduced and binary tree traversals algorithms are presented. The methods to represent a graph into computer are discussed there. The graph searching methods such as DFS and BFS explained with examples. The famous Dijkstra algorithm for finding shortest distances among vertices are designe and its complexities are calculated. At the end of this module an exercise and references are incorporated.

63.12 Self Assessment Questions

1. What are the basic steps for developing an algorithm? Describe the steps briefly.
2. Write algorithms for linear search and binary search. Compare them.
3. Write an algorithm for quick sort.
4. Write an algorithm to sort n records using merge sort. Calculate time complexity of the proposed algorithm.
5. Write an algorithm to sort a set of numbers using insertion sort.
6. Write an algorithm to sort a set of numbers using straight selection sort.
7. Define heap. Write an algorithm to arrange a set of data using heap sort.
8. Define a tree and binary tree. Define height of a tree.
9. Write iterative algotithms to traverse a binary tree in postorder, inorder and preorder.
10. How a graph can be represented in a computer? Illustrate your answer using one of the representation stated by you.
11. Explain DFS and BFS.
12. Write Dijkstra algorithm to find the shortest distances from a specified vertex to all other vertices.

13. Write an algorithm to find the shortest distance between two given vertices on a weighted directed graph. Calculate time and space complexities.

63.13 Suggested Further Readings

1. E. Horowitz and S. Sahni, Fundamental of Computer Algorithms, Galgotia.
2. T.H. Cormen, C.E. Leiserson and R.L. Rivest, Introduction to Algorithms, Prentice Hall of India.
3. S.E. Goodman and S.Te. Hedetniemi, Introduction to the Design and Analysis of Algorithms, McGaw-Hill.
4. D.E. Knuth, Fundamental Algorithsm (The art of computer progamming, vol 1), 2ed, Narosa.
5. D.E. Knuth, Sorting and Searching, (The art of computer programming, vo. 2) 2ed, Addison-Wesley, Reading.
6. A.V. Aho, J.E. Hopcroft and J.D. Ullman, The Design and Analysis of Algorithms, Addison-Wesley.
7. N.Writh, Data structures + Algorithms = Programs, Prentice Hall.
8. Adam Drozdek, Data Structures and Algorithms in C++, PWS Publishing Company, Boston.

M.Sc. Course
in
Applied Mathematics with Oceanology
and
Computer Programming

PART-II

Paper-VI Group–A

Module No. - 64
COMPUTER NETWORK

Structure :

64.1 Introduction

64.2 Objectives

64.3 Keywords

64.4 Centralized and Distributive Computing

64.5 What is Internet?

64.6 Advantages of Networks

64.7 History and Development of the Internet

64.8 Structure of the Communications Network

- 64.8.1 Point-to-Point and Multidrop Circuits
- 64.8.2 Data Flow and Physical Circuits

64.9 Network Topologies and Design Goals

64.10 Fundamentals of Communications Theory

64.11 Local Ara Network

- 64.11.1 What a LAN can do?
- 64.11.2 Components of a LAN
- 64.11.3 Workstations
- 64.11.4 File Servers
- 64.11.5 LAN cables

64.11.6 Network Adapters
64.11.7 LAN Software
64.12 Wide Area Network (WAN)
64.13 How Does An Internet Works
64.13.1 By number
64.13.2 By name
64.14 Some Standard Internet Servicesw
64.14.1 Electronic mail
64.14.2 Telnet
64.14.3 File Transfer Protocol (FTP)
64.14.4 World Wide Web (WWW)
64.15 How to Get Connected to the Internet?
64.15.1 Software and Hardware Requirements
64.16 Internet Service Provider
64.17 Module Summary
64.18 Self Assessment Questions
64.19 Suggested Further Readings

64.1 Introduction

The Internet is the world's largest network. A computer network is bacially a group of computers connected together using cables. Generally, computer networks consist of a central server and few terminals that are connected to the server. For example, in the railway reservation system, all terminals can access the information of the server. So, the status of the reservation details, at any point of time, kept in the server can be accessed by any of the terminals connected to it.

64.2 Objectives

At the end of this module the students will learn the following topics.

* Local area and wide area network
* History of Internet
* Structure of the communication system
* Network topologies
* Components of LAN
* Working principle of Internet
* Internet services

* Connection of local computer with remote host

64.3 Keywords

LAN, WAN, Internet, Network topologies, internet services.

64.4 Centralized and Distributive Computing

With the increasing availability of inexpensive yet powerful personal computers and minicomputers, there has been an increasing trend toward distributed data processing (DDP). The use of DDP allows processors, data, and other aspects of a data processing system to be dispersed within an organization. Dispersal provides a system that is more responsive to user needs, is able to provide better response times, and may minimize communications costs by comparison with a centralized approach. A DDP system involves a partitioning of the computing function and may also involve a distributed organization of data bases, device control, and interaction (network) control. In many organizations, there is heavy reliance on personal computers coupled with a large central facility for data processing. Personal computers are used to support a variety of user-friendly applications, such as word processing, spreadsheets, and presentation graphics. The mainframe houses the corporate database plus sophisticated software for database management and information systems. Linkages are needed among the personal computers and between each personal computer and the mainframe. Various approches for the micro-mainframe link are in common use, ranging from treating the personal computer as a simple terminal to a high degree of integration between personal computer applications and the database. These application trends have been supported by the evolution of distributed capabilities in the operating system and support utilities. A spectrum of capabilities have been explored, as follows:

* Communications architecture : Software that supports a network of independent computers. It provides support for distributed applications, such as electronic mail, file transfer, and remote terminal access. However, the computers retain a distinct identity to the user and to the applications, which must communicate with other computers by explicit reference. EAch computer has its own separate operating system, and a heterogeneous mix of computers and operating systems is possible, as long as all machines support the same communications architecture.
* Network operating system : A configuration in which there is a network of application machines, usually single user workstations and one or more 'server' machines. The server machiens provide networkwide services or application such as file storage and printer management. Each computer has its own private operating system. The network operating system is simply an adjunct to the local

operating system that allows application machines to interact with server machines. The user is aware that there are multiple independent computers and deals with them explicitly. Typically, a common communications architecture is used to support these network applications.

* Distributed operating system : A common operating system shared by a network of computers. It looks to its users like an ordinary centralized operating system but provides the user with transparent access to the resources of a number of machines. A distributed operating system may rely on a communications architecture for basic communications functions; more commonly, a stripped-down set of communications functions is incorporated into the operating system to provide efficiency.

The technology of the communications architecture is well developed and is supported by all vendors. The first commercially available communications architecture was IBM's System Network Architecture (SNA), introduced in 1974. Network operating systems are a much more recent phenomenon, but a number of commercial products exist.

64.5 What is Internet?

What is a computer network? Several definitions are available in literature. Perhaps the most simplest definition is the following:

A computer network is a number of computers (and usually terminals) interconnected by one or more transmission paths. The very commonly used transmission path is telephone line, due to its convenience and universal presence. The network exists to meet one goal – the transfer and exchange of data between the compuers and terminals. This data exchange provides for many computer-based services. We often take for granted in our daily lives, such as bank teller machines, point-of-sale terminals, check-verification devices, and even the guidance of the space shuttle, etc.

64.6 Advantages of Networks

Computer networks provide several important advantages to organizations and individuals. These are listed below:

* Modern organizations today are widely dispersed, with offices located in diverse parts of a country and the world. Many of the computers and terminals at the sites need to exchange information and data, often daily. A network provides the means to exchange data among these computers and to make programs and data available to the people of the organization.
* The networking of computers permits the sharing of resources. For instance, if a computer becomes saturated with too much work at one site, the work can be reloaded through the network onto another computer in the network. Such load sharing permits a better, more even utilization of resources.

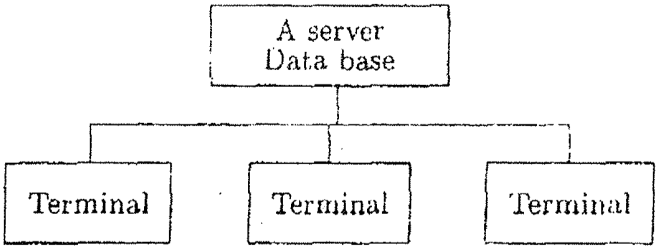


Figure 64.1: A simple network.

* Networking also supports the critical function of backup. In the event a computer fails, its counterpart can assume its functions and workload. Backup capability is especially important in systems such as those used for air traffic control.
* The use of networking allows a very flexible working environment. Employees can work at home by using terminals tied through networks into the computer at the office. Many people now carry terminals or portable personal computers (laptop) on trips and connect into their networks through hotel room telephones. Other people travel to remote offices and use telephones and networks to transmit and receive critical sales, administrative, and research data from computers at company headquarters.

Most of the Internet services come free of cost. But to get connected to the Internet we have to pay to the Internet Service Provider.

Some of the Internet services are :

* Electronic Mail (e-mail) : Sending and receiving mail electronically.
* File Transfer (FTP) : Transferring flles from one computer to another.
* Telnet : Logging in to any computer to use its resources, provided you have permission, as a terminal.
* Information Retrieval (Searching) : Viewing information residing in any other computer. The information can be anything from research paper on any field to games.

The Internet links universities, publishers, companies, governments, cities, countries, organisations, students, researchers, personal computers, super computers, weather stations, and even pizza huts.

64.7 History and Development of the Internet

The first work on Internet was started by Advanced Research Projects Agency Network (ARPANET). This project was started by the Department of Defence (DOD) of USA in 1969 to experiment reliability of networking

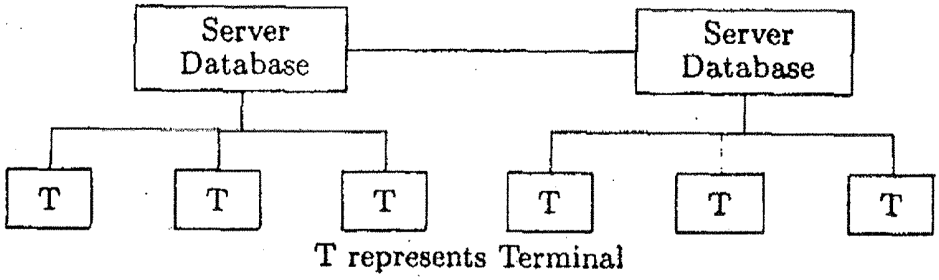


Figure 64.2: A network of networks.

to link DOD and military research conractors, including the large number of universities doing military-funded research. The ARPANET started as a small network and connected three computers, but it grew to span the entire world. The ARPANET was successful and many of the universities joined the net.

Later, it was divided into two parts, MILNET, which was used for military related sites, and ARPANET, which had nonmilitary sites. The two networks remained connected using IP (Internet Protocol), a technology which allows network communication from one network to another.

In the early 1970s the ARPANET allowed various universities to link to the net. In a very short time, students began linking their own campus networks into ARPANET, using a well-defined protocol called TCP/IP (Transmission Control Protocol/ Internet Protocl), thereby linking the various networks together. This joining of networks was called inter-networking, and soon the entire setup was called the Internet.

In 1973, ARPANET allowed internatonal bodies to link to the net. ARPANET broke the military tie-up and moved into the civilian domain. In 1977, ARPANET was handed over to the National Science Foundation (NSF). Then NSF arranged to set up regional networks to connect the users in each region, with the NSFNET connecting all the regional networks.

By 1994, several large commercial Internet networks grew within the Internet, and some were run by large organisations. From four hosts in 1969 to over five million hosts in 1995. Today there are more than 30,000 networks on the Internet.

64.8 Structure of the Communications Network

Figure 64.3 illustrates a simple data communications system. The **application process (AP)** is the end-user application. It usually consists of software such as a computer program. Typical examples are an accounts receivable program, a payroll program, an railway reservation system, an inventory control package, or a personnel system, etc.

In Figure 64.3, site A could execute an application process (AP_{A1}) in the form of a software program to access an application process at site B 9which is, in this case, a program (AP_{B1}) and a data base). Figure 64.3 also

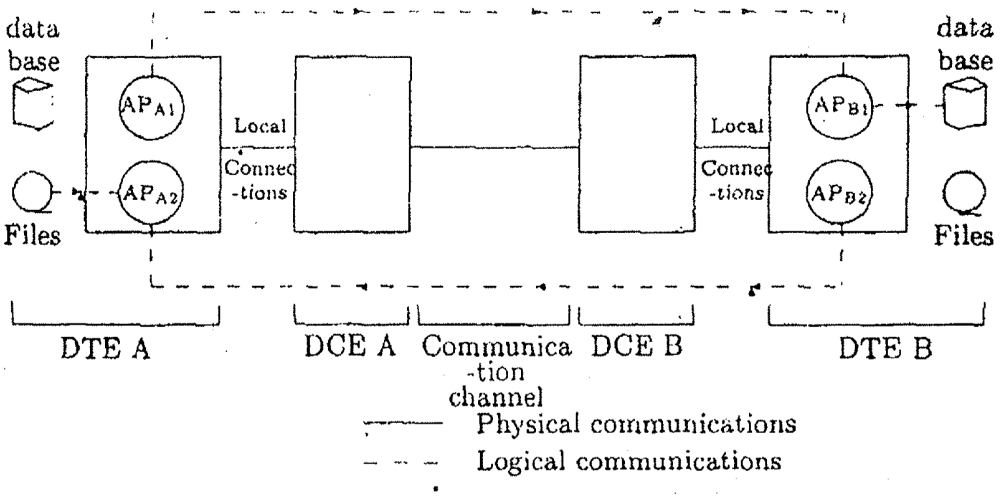


Figure 64.3: A communication system.

shows a site B program (AP_{B2}) accessing a file at site A through an application program (AP_{A2}). The application process here we mean end-user application. The application resides in the **data terminal equuipment or DTE.** The DTE is a generic term used to describe the end-user machine, which is usually a computer or terminal. The DTE could be a large mainframe computer, such as alrage IBM or ICL machine, or it could be a smaller machine, such as a terminal or a personal computer.

The function of a communications network is to interconnect DTEs so they can share resources, exchange data, provide backup for each other, and allow employees and individuals to perform their work from any location.

Figure 64.3 shows that a network provides logical and physical communications for the computers and terminals to be connected. The applications and files use the physical channel to effect logical communications. Logical, in this context, means the DTEs are not concerned with the physical aspects of the communications process. Application A1 need only issue a logical 'Read' request with an identification of the data. In turn, the communication system is responsible for sending the 'Read' request across the physical channels to application B1.

Figure 64.3 also shows the **data circuit-terminating equipment or DCE** (also called data communications equipment). Its function is to connect the DTEs into the communication line or channel. The DCEs designed in the 1960s and 1970s were strictly communications devices. However, in the last decade the DCEs have incorporated more user functions, and today some DCEs contain a portion of an application process. The primary function of the DCE remains to provide an interface of the DTE into the communications network. The familiar modem is an example of a DCE.

The interfaces are specified and established through **protocols.** Protocols are agreements on how communications components and DTEs are to communicate with each other. Tehy may include actual regulations

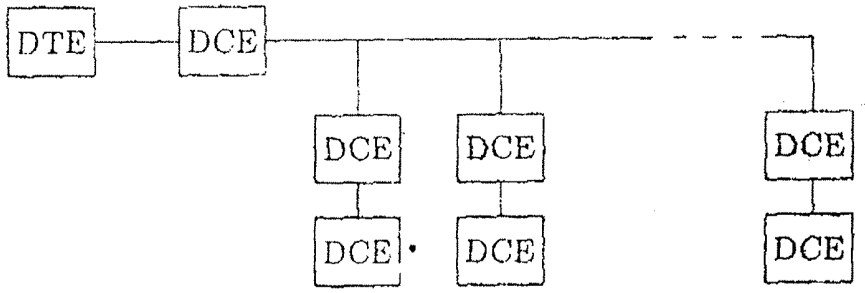


Figure 64.4: Multidrop circuit.

which stipulate a required or recommended convention or technique. Typically, several levels of interfaces and protocols are required to support an end-user application.

64.8.1 Point to-Point and Multidrop Circuits

DCE's and DTE's are connected in one of two ways. As illustrated in Figure 64.3, they are connected as a point-to-point configuration in which only two DTE devices are on the line or channel. Illustrated in Figure 64.4 is another approach called a multidrop configuration. In this configuration, more than two devices are connected to the same channel.

64.8.2 Data Flow

The DTEs and DCEs send communication traffic to each other in one of three methods :

* Simplex : Transmission in one direction only.
* Half-Duplex : Transmission in both directions, but only one direction at a time (also called two-way alternate (TWA).
* Full-Duplex (or Duplex) : Transmission simultaneously in both directions (also called two-way simultaneous(TWS).

Simplex

Simplex transmission is common in television and radio. It is not as common in data communications because of the one-way nature of the process, but simplex systems are found in some applications, such as telemetry.

Half-duplex

Half-duplex transmission is found in many systems, such as inquiry/ response aplications wherein a DTE sends a query to another DTE and waits for the applications process to access and/or compute the answer and transmit the

response back. Terminal based systems (keyboard terminals and terminals with CRT screens) often use half-duplex techniques.

Full-duplex (or simply duplex)

Full-duplex provides for simultaneous two way transmission, without intervening stop-and-wait aspect of half-duplex. Full duplex is widely used in applications requiring continuous channel usage, high throughput, and fast response time.

64.9 Network Topologies and Design Goals

A network configuration is called a **network topology.** A network topology is the shape (or the physical connectivity) of the network. The term **topology** comes from geometry to describe the form of something. The network designer has three major goals when establishing the topology of a network :

* Provide maximum possible reliability to assure proper receipt of all traffic (alternate routing).
* Route the traffic across the least-cost path within the network between the sending and receiving DTEs. If other factors (such as reliability) become more important then least-cost route may not be chosen.
* Give the end user the best possible response time and throughput.

The reliability of a network means the ability to delivery user data correctly (without errors) from one DTE to another DTE. It entails the ability to recover from errors or lost data in the network, including channel, DTE, DCE, or DSE failure. Reliability also refers to the maintenance of the system, which includes day-to-day testing; relieving faulty or failing components of their tasks; and fault isolation in the event of problems. When a component creates problems, the network diagnostic system should pinpoint the error readily, isolate the fault, and perhaps isolate the component from the network.

The second major goal to developed a topology for the network is to provide the least-cost path between the application processes residing on the DTEs. This involves :

(i) Minimizing the actual length of the channel between the components, and this is usually done by selecting fewest number of intermediate components.

(ii) Support of least expensive channel option for a particular application. For example, transmitting low-priority data over a relatively inexpensive dial-up, low-speed telephone line and in contast to transmitting the same data over an expensive high-speed satellite channel.

The third major goal in develop a network topology is to provide the best possible response time and throughput. Short response time gives minimizing delay between the transmission and the receipt of the data

between the DTEs, and is especially important for interactive sessions between user applications. Throughput entails the transmission of the maximum amount of end-user data in a given period. The more common network topologies are listed below:

* Hierarchical topology (tree),
* Horizontal topology (bus),
* Star topology,
* Ring topology (hub),
* Mesh topology.

Hierarchical Topology

The hierarchical topology is one of the more common networks found today. The software to control the network is relatively simple and the topology provides a concentration point for control and error resolution. In most cases, the DTE at the highest order of the hierarchy is in control of the network. In Figure 64.5(a), traffic flow among the between the DTEs is initiated by DTE A. Many vendors implement a distributed aspect to the hierarchical network by providing methods for the subordinate DTEs to directly control the DTEs below them in the hierarchy. This reduces the workload of the central host at site A.

While the hierarchical topology is attractive from the standpoint of simplicity of control, it presents significant potential bottleneck problems. In some instances, the uppermost DTE, typically a large-scale mainframe computer, controls all traffic between DTEs. Not only can this create bottlenecks, it also presents reliability problems. In the event the upper-level machine fails, the network capabilities are lost completely if this DTE is not fully backed up by another computer. The hierarchical topologies have been used widely in the past and will continue to be used in the future. They permit a graceful evolution toward a more complex network, because subordinate DTEs can be added relatively easily.

The hierarchical topology is also called a "vertical network" or a "tree network". The word "tree" is appropriate because a hierarchical network often resembles a tree with branches stemming from the top of the tree down to the lower level. Indeed, the advantages and disadvantages of a vertical data communicationsn network are much the same as those of hierarchically structured business-clear lines of authority with frequent bottlenecks at the upper levels and often insufficient delegation of responsibility.

Horizontal Topology (Bus)

The horizontal topology or bus topology is illustrated in Figure 64.5 (b). This arrangement is quite popular in local area networks. It is relatively simple to control traffic flow between and among the DTEs because the bus permits all stations to receive every transmission. That is, a single station broadcasts to multiple stations. The main drawback

of a horizontal topology stems from the fact that usually only one communicatons channel exists to service all the devices on the network. Consequently, in the event of a failure of the communications channel, the entire network is lost. Some vendors provide for fully redundant channels in the event of the loss of a primary channel. Others provide bypass switches around failed nodes. Another problem with this particular configuraton is the difficulty in isolating faults to anyone particular component tied into the bus. The absence of concentration points makes problem resolution difficult.

Star Topology

The star topoogy is another widely used structure for data communications systems. One of the major reasons for its continued use is based on istorical precedence. The star network was used in the 1960s and early 1970s because it was easy to control the software and the traffic flow is simple. All traffic control by the hub (labelled A, the central site in Figure 64.5(c)). Site A, typically a computer, is in complete control of the DTEs attached to it. Consequently, it is quite similar to the hierarchical topology, except that the start topology has limited distributed processing capabilities. Site A is responsible for routing traffic to the other components; it is responsible for fault isolation as well. Fault isolation is relatively simple in a start network because the lines can be isolated to identify the problem. However, like the hierarchical structure, the star network has also failure problems at the central site. Several star network built in the 1970s experienced serious reliability problems because of the centralized aspect of the network. Other systems established a redundant backup of the hub node, which provided considerably more reliability to the system.

Ring Topology

The another popular approach to network configuration is ring topology, as illustrated in Figure 64.5.(d). The name comes from the circular aspect of the data flow. In most instances, data flow in one direction only, with one single station receiving the signal and relaying it to the next station on the ring. The ring topology is attractive because bottlenecks, such as those found in the hierarchical or start systems, are very uncommon. Moreover, the logic to implement a ring network is relatively simple. Each component is tasked with a straightforward job of accepting the data, sending it to the DTE attached to it, or sending it out on the ring to the next intermediate component. However, like all networks, the ring network does have its deficiencies. The primary problem is the one channel tying in all the components in the ring. If a channel between two nodes fails, then the entire network is lost. Consequently, some vendors have established designs which provide for backup channels in the event a channel is lost.

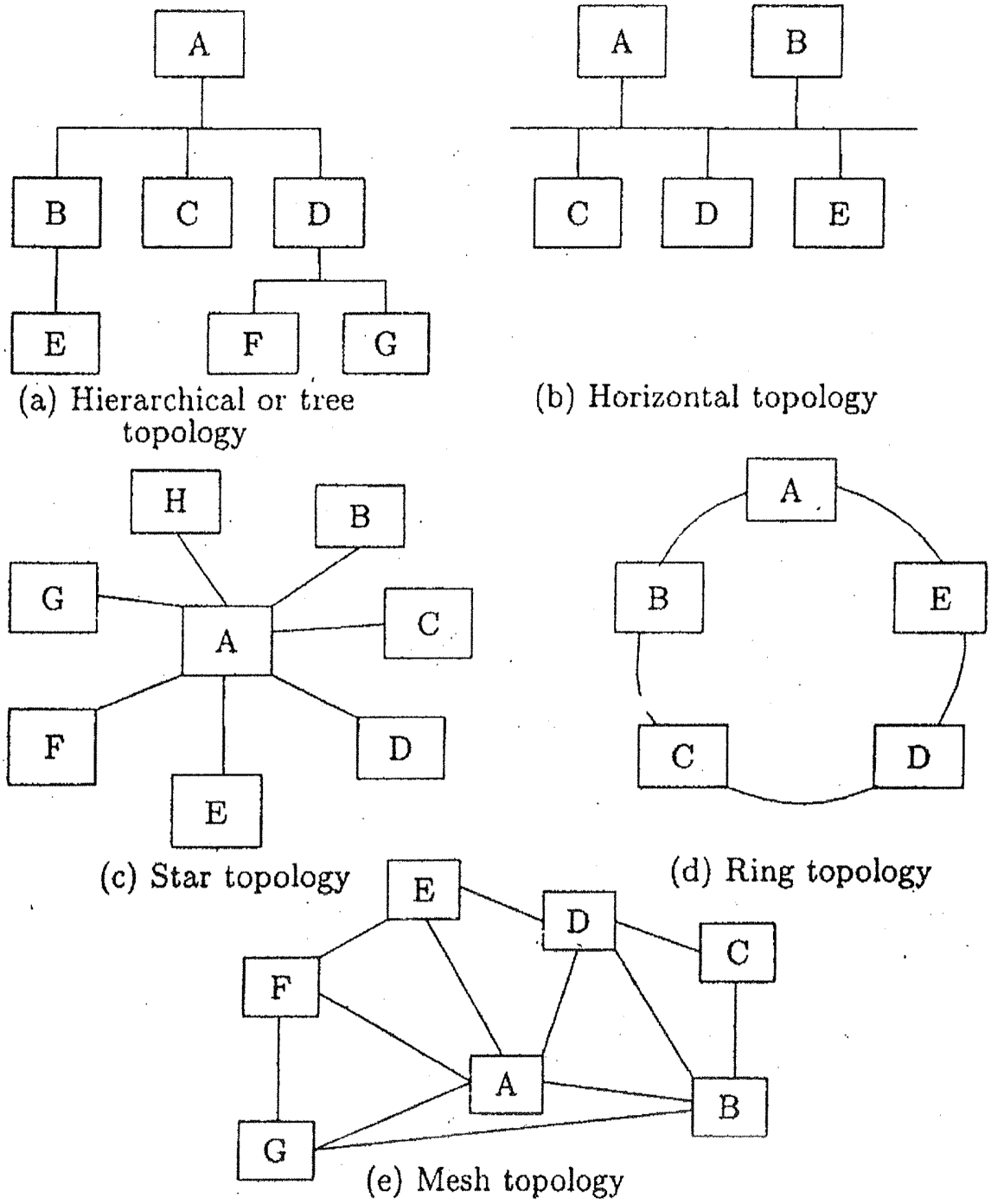


Figure 64.5: Network Topologies.

Mesh topology

The mesh topology has been used somewhat in the last few years (Figure 64.5(e)). Its attraction is its relative immunity to bottleneck and failure problems. Due to the multiplicity of paths from the DTEs and DCEs, traffic can be routed around failed components or busy nodes. Even though this approach is an expensive undertaking, some users prefer the reliability of the mesh network to that of the others.

64.10 Fundamentsl of Communications Theory

Channel speed and bit rate

The most common method to send a binary number on a communications path is to switch the signal on and off electrically, or to provide high or low voltages on the line to represent the 1s and 0s. Regardless of how the data are represented on the path-in the form of on/off states, levels of voltage, or directions of current flow the communications channel is described by its capacity in the number of bits per second transmitted. Abbreviations for bits per second are bit/s, bps, or bs. When one speaks of a 4800 bit/s line, it means a device sends 4800 bits per second through the channel. A bit is simply the representation of the electrical, optical, or electromagnetic state of the line: voltages, current, or some form of radio or optical signal. Seven or eight bits are generally used for user coded character, or byte. A data communications channel utilizing conventional telephone lines is very slow. Below are some examples. For purposes of comparison, a channel is classified by categories of low speed, medium speed, and igh speed:

Low Speed : 0-600 bits per second
Medium Speed : 600-4800 bits per second
High Speed : 4800-9600 bits per second.

Only recently, in the last few years, has the industry successfully moved to 9.6 kilobytes per second (kbit/s) on telephone channels. The typical speeds found beyond 9600 bits per second are 14,400, 19,200, 56,000 and 64,000 bit/s. The 1.544 megabits per second channel is the well-known T1 carrier.

One might reasonably ask, why the slow speed? The answer is that DTEs and DCEs usually communicate through the telephone line. It was the most convenient and readily available path when the industry developed computers and began to interface them with terminals and other computers in the 1960s. The telephone channel is not designed for fast transmission between high-speed computers, but for voice transmission between people, which does not require the speed associated with data transmission. Moreover, the majority of DTEs remaining today are connected through the telephone line.

Voice communications and analog waveforms

Voice communications generate acoustical waveforms which propagate through the air. In effect, voice communications are physical energy. When one speaks, oscillating waveforms of high and low air pressure sequences

are generated. These waveforms are called analog waveforms. They are so named because they exhibit a continuous, repeating occurrence and they are non-dicrete, gradually changing from high to low pressure. Of course, one cannot see waveforms in the air because the voice transmissions are air pressure variations. The telephone handset translates the physical oscillations of the air to electrical energy with similar waveform characteristics. The wave form exhibits three primary characteristics that are very important to data communications : amplitude, frequency, and phase. The amplitude of the signal is a measurement in relation to its voltage, which can be zero or a plus or minus value. Notice the analog aspect of the signal – it gradually increases in positive voltage, then traverses the zero voltage to the negative voltage, and returns to zero again. This complete oscillation is called a cycle. The second major aspect of the signal is its frequency. The frequency describes the number of complete cycles per second or the number of oscillations per second. The number of oscillations per second or frequency is also called Hertz (Hz). Hertz describes the number of complete waveforms that pass a reference oint in a second.

The term **baud** refers to the rate of signal change on the channel, regardless of the information content of those signals. For example, an 1800 Hz signal can be changed at 1200 times per second. The 1800 Hz signal describes the "carrier" frequency and the 1200 changes describe the baud (rate).

The third major component fo the analog signal is **phase**. The phase of the signal represents the point the signal has reached in the cycle. When the cycle has gone one-fourth of its phase at point A, it is said to have traversed through 90 percent of its cycle, just as traversing onefourth the way around a circle represents a distance of 90 degrees.

Digital Signals

The communication between DTEs through telephone path is carried out by voice-oriented analog signal. However, DTEs work in digital forms. As illustrated in Figure 64.6, the digital waveform looks considerably different from the analog waveform. It is similar in that it is continuous, repeats itself, and is periodic, but it is very different from the standpoint that it is discrete-it has very abrupt changes in its voltage state. Computers and terminals use digital, binary images because semiconductor transistors are basically two-state, discrete devices. (It should be noted that Figure 64.6 shows an ideal digial signal. Due to the electrical properties of the channel, the signal is actually less discrete and square shaped). Digital transmission is available in many systems today. It has several distinct advantages over the analog channel. However, analog channels still dominate the local connections of DTEs into the telephone companies channels.

The Modem

A method is essential to communicate between two digital devices through the dissimilar analog environment. The modem provides this digital/analog interface. It alters either the amplitude, the frequency, or the phase to represent

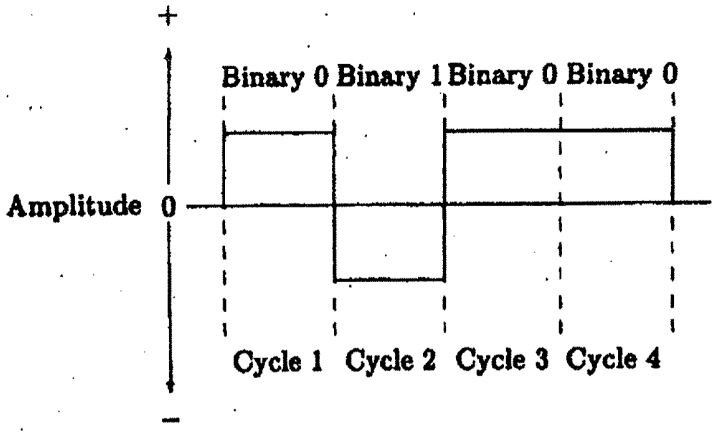


Figure 64.6: The digital signal.

the binary data as an analog signal. The modem is our first example of a DCE. It provides the interface between the digital and analog worlds, as well as the capability to transmit from a digital DTE across the analog channel to a receiving digital DTE. The word "modem" is a shortened term for modulation/demodulation. The process modulates the signal at the transmitting modem and demodulates the transmission at the receiving modem. To be precise, the exact definition of modulation is : the modification of a frequency to carry data. This frequency is called the carrier frequency. The data that modulates the carrier (i.e., the data coming from the terminal or computer) is called the baseband signal. The term "baseband" usually refers to an unmodulated signal. The modem modifies the carrier signal (either its amplitude, frequency, or phase) to carry the baseband signal.

64.11 Local Area Network

Local area network is abbreviatd as LAN. Within a single building or small geographical area, a LAN enables to connect a group of personal computers. People using these networked computes can share information. A LAN ives the simultaneous access of some application softwares those are designed for multiple users. In addition to easily sharing files, people on a LAN can share a printer, a CD-ROM drive, a modem, a scanner, a fax machine, etc.

64.11.1 What a LAN can do?

A LAN can do virtually everything like a mainframe computer or minicomputer, at a much lower cost. In each environment, people can share computer resources and information, and they can work together on projects and tasks that require coordination and communication even though they may not be physically close. If a LAN crashes,

though, each person may be able to continue working on his or her personal computer. A mainframe or min-computer crach, on the other hand, brings work to a half for an entire department or organization.

The following seven operations can be performed with a LAN that we cannot easily do with non-networked, stand-alone personal computers:

(i) **Share files** : A LAN allows many users to share a single copy of a file stored on a central file server, which helps the organization keep its records, documents, and other files consistent.

(ii) **Transfer files :** A LAN allows to copy files from machine to machine without having to exchange floppy diskettes. This is especially useful for large files that don't fit on a single floppy disk.

(iii) **Access information and files :** A LAN allows anyone to run accounting software or other application software from any of its workstations. People can access software tools from any LAN-connected desktop computer.

(iv) **Share applications :** A LAN allows to use the same copy of any application software, for example, the Microsoft Word word processing program. But, two people cannot edit the same document simultaneously.

(v) **Simultaneously key data into an application :** A LAN based application program allows two people to key into it at once. Two people can key general ledger transactions at the same time, for example, with the program coordinating their work so they don't interfere with ech other. Note that only special LAN based versions of programs enable simultaneous data entry. Ordinary computer programs enable only one person at a time to use the program on a given set of files.

(vi) **Printer sharing :** We can share one or more expensive printers among several workstations through LAN.

(vii) **Electronic mail :** We can use a LAN as a post office to send memos, reports, and typed messages to other people in other parts of the building. The telephone is faster and more convenient, but with this system two people should be present in the path simultaneously. A LAN e-mail system takes messages when people are away from their phones or out of station.

64.11.2 Components of a LAN

A Lan constitute by computers, LAN cables, network adapter cards, network operating system software, and LAN application software. On a LAN, each personal computer is called a workstation, except for one or more computers designated as file servers. Each workstation and file server contains a network adapter card. LAN cables (or other media) connect all workstations and file servers. In addition to DOS, Windows, UNIX, OS/2, or other operating system, each workstation runs network software that enables it to communicate with the file

servers. In turn, the file servers run network software that communicats with the workstations and serves up files to those workstations. LAN based application software runs at each workstation, communicating with the file server when it needs to read and write files.

64.11.3 Workstations

In a LAN two types of computer are available, viz., workstations, usually operated by users; and file servers, usually located in a separate room or closet or a remot place.

The workstations are he various PCs or terminals connected to the file server. People use these PCs on their own or, with network software, link to the file server to access programs, data, printers, etc. Workstations, sometimes referred to as clients, are usually intermediate-speed machines and low RAM. Workstations often have good quality colour or, less often, grayscale VGA monitors as well as high quality keyboards, but these characteristics simply make them easy to use and are not required to make the LAN work. Also, a workstation usually has a smaller, slower hard disk. The diskless workstations may also be available.

64.11.4 File Servers

A file server is a computer that serves all the workstations primarily by sotring and retrieving data from files shared on its disks. File servers are usually fast Pentium or Pentium Pro-based computers with high speed and with 15MB or more or RAM. They usually have monochrome monitors and inexpensive keyboard because people don't use them directly. They almost always have one or more large, fast, expensive hard disks. Servers must be high-quality, heavy-duty machines because, in serving the whole network, they do many times the work of an ordinary workstation computer. The server stores all the data of an entire organization. In particular, the file server's hard disk(s) must be durable and reliable. On smaller LANs the file server may double as a workstation.

If a LAN has 20 workstations and one server, under a heavy load each workstation can use only 1/20th of the server's resources. In practice, though, most workstations are idle most of the time, at least from a "disk file access" point of view. If at any moment only one workstation is using the server then this workstation can use 100% of the server's resources.

64.11.4 LAN cables

LAN cables are the nervous system of the whole LAN. These cables connect everyone to the file server and peripherals such as printers. A LAN can use one of several transmission media: twisted-wire cable, coaxial cable, fiber-optic cable, radio waves, and infrared waves.

A few companies, such as Motorolla, are pioneering a type of LAN that does not require cables at all. Such a wireless LAN uses infrared or radio waves to carry network signals from computer to computer.

64.11.6 Network Adapters

A network adapter card fits in a slot in each workstation and file server. The workstation sends requests through the network adapter to the file server. The workstation receives responses through the network adapter when the file server wants to deliver a portion of a file to an user. People often refer to network adapters as Network Interface Cards (NICs).

Only two network adapters may communicate with each other at the same time on a LAN. That is, other workstations have to wait for a negligible time if the network is in use. The LAN gives the appearance of many workstations accessing the file server simultaneously.

64.11.7 LAN Software

A special type software called network operating system is required in a LAN. The operating system by itself cannot use a network adapter to connect to a file server. The most popular network operating system software is Net Ware designed by Novell. The population desktop computer operating systems are Windows for Workgroups, Windows 95, Windows NT, and OS/2.

Following is a list of network OSs with their manufacturer.

Network OS	Manufacturer
Apple Talk	Apple
LANastic	Artisoft
Net Ware	Novell
Network File System (NFS)	Sun Micro-systems
Windows NT Server	Microsoft
Windows 95	Microsoft
Windows for Workgroups	Microsoft
Warp Server	IBM
Warp Connect	IBM

64.12 Wide Area Network (WAN)

In previous section we have seen that a local area network can encompass a small geographic area, such as a department-sized office or small building. We also observed that there are distance limitations of popular local area networks (Ethernet, Token Ring, and ARCnet). Someday our organization may need to exceed these distance limitations to allow people in different locations to share files. The WAN is the composition of the several local area networks.

Exploring WAN basics within an office space, the office own office's LAN. We can purchase new workstations,

cables, repeaters, and hubs, and add these components to the LAN. We can diagnose and correct the problems that happen on the LAN. With a WAN, the situation is different. We must contract with a communications services provider (say, telephone company, for instance) to connect two LANs that are across the street or across the country from one another. The telephone company will lease a special kind of phone line. We shall probably purchase the WAN hardware that connects our LANs to either end of the phone line, but we rely on the phone company to make sure the phone line is continuously available and noise-free. WAN hardware consists of routers, bridges, and gateways. Basically, the WAN hardware provides a path between LANs, over which two or more LANs can share frames and packets. This communication of frames and packets between LANs allows a workstation on one LAN to access a file server or print server on another LAN.

WANs don't use Ethernet, Token Ring, or ARCnet. At the physical layer, WANs use protocols that typically encapsulate the Ethernet, Token Ring, or ARCnet frames. An encapsulated frame travels across the WAN link to the other LAN, where the WAN turns the frame back into a LAN packet. The WAN hardware decides which frames need to travel to another LAN, and the type and configuration of the WAN hardware determines which WAN protocol is used. Different WAN protocols carry information at different speeds (and at different costs). In general, the WAN equipment (router, bridge, or gateway) uses one or more of these protocols to transport frames and packets between LANs. Because these WAN protocols typically operate more slowly than the LAN protocols described we shall notice that accessing files across a WAN takes more time than accessing the same files on our local area network.

Two types of connection are required to connect different LANs into one network. First, we must create a physical path between the LANs. Next, we must enable the flow of information between the LANs. This means that we must establish a common protocol the LANs can use to exchange information. Most vendor products for connecting LANs use a common high-level protocol such as TCP/IP, IPX, or NetBIOS. At a layer below the high-level protocol, the WAN uses a protocol designed for use over highs speed telephone lines to transport the TCP/IP, IPX or NetBIOS packets. We can think of interconnected LANs in terms of layers connected by these common protocols. The bottom layer consists of individual LANs, each with its own file servers and workstations. The common protocol ties the bottom layers through special communications devices. Such a device does not change the individual LANs; the device simply transports message packets between the LANs.

64.13 How Does An Internet Works

A huge number of computers are connected in the Internet. Now, the question is how we can identify a particular computer among these large number of computers (servers). There are two ways to identify a particular computer on the Internet. One is by a number associated to a computer and other is by the name assigned to the computer.

64.13.1 By number

Each computer (host) (Host is a computer of any kind, smaller or larger, providing service to the Internet) on the Internet has a number assigned to identify it from other hosts, like telephone numbers. The numbers are divided into four parts, and separated by dots. For example, 192.168.1.5 may be possible number of a host.

Each host is assigned a host number. Internet host numbers are divided into two parts; the first part is the network number, since the Internet is composed of many different but interconnected networks. The second part is the local part which is the host number on that particular local network. In the case of the host number 192.168.1.5, the network number is 192.168, and the local host number is 1.5

64.13.2 By name

Internet hosts are usually referred to by names, not by the numbers, because remembering a four part number is a difficult task. In the earliest days of ARPANET, computers had simple one-part names, and there was a master list of names. But with millions of computers on the net, it would be very hard to come up with diffrent names for all the computers on the Internet.

This problem was solved by adapting multi-part names, a naming scheme generally known as the Domain Name System (DNS). Host names are a string of words separated by dots like lycos.com. The last part com means that it is a commercial site. Other commonly used sites are given below:

Type of Organization	Name
Commercial	com
Educational	edu
Government Bodies and Departments	gov
Military Sites	mil
Networking Organisations	net

It is common to use geographic names outside USA. Two-letters zone names are organized geographically. Each zone corresponds to a country or apolitical entity. There is an official list of two-letter country codes.

Country Name	Code
India	in
Australia	au
Canada	ca
France	fr
Japan	jp
Korea	kr

64.14 Some Standard Internet Services

64.14.1 Electronic mail

Electronic mail is the most widely used Internet service. Every system on the Internet supports at least one type of mail system, using which we can send and receive mail electronically.

If the e-mail address of a pupil is known then anybody can send e-mail to that pupil at any time from any place of the world.

An e-mail address has two parts, they are separated by @ (the at sign). The first part before the @ is the mailbox, which may be the personal names, and the part after the @ is the domain, usually the name of the computer.

The mailbox is usually the username, so the e-mail address may be like

gpb@math.iitkgp.ernet.in

64.14.2 Telnet

The another widely used interactive form of Internet service is Remote Login or Telnet. We can log on to a remote computer as if our computer is attched directly ont eh net. In the Internet we can log on to a host that is in the other part of the world as easily as we can log on to the one in the office. The only difference being that the connection to the distant computer may be slower. Though most of the computers are open, buyt, for some computers we have to take permission to log on.

By logging on to á remote computer, we can use its resources, such as, computing speed, softwares, lots of information, and even some special devices like printer, scanner, camera, etc.

Once we log on to the remote computer, we have to log out at the end of the work. When the remote computer is log out then that computer will the Telnet connection and will inform the local Telnet program.

64.14.3 File Trnasfer Protocol (FTP)

Another service of Internet is the file transfer facility from one computer to anotehr remote computer. In some cases we may need to copy a file from remote host to our service providers host and then from the service providers computer to our local computer. We can also send files to the remote computer.

64.14.4 World Wide Web (WWW)

This the most striking and mostly used service in the Internet. World Wide Web (WWW) or the Web is a major step towards advancement in making the process of looking for the information fast, powerful, and user-friendly. WWW is based on the technology called hypertext. In this system text, sound and video are used so it is more appropriate to call it hypermedia.

Hypertext organises data to help in information retrieval. It is organized like a library catalogue. It keeps one finger in one drawer and another finger in another drawer to help us go through one path and then go back and try a different path. And hypertext can have hundreds of fingers in drawers all over the world.

The World Wife Web is a hypertext-based page that links a page to the other pages throughout the global Internet. A page is simply a title, a collection of information, which serves as a pointer or hyperlink to other information. We can view information using a browse program which is linked to a page in a remote host. Depending on the system we are using, we can view text, graphics, and hear sound clippings from recorded music and even we can see movie clippings.

64.15 How to Get Connected to the Internet?

To work in Internet we have to connect the local computer to the Internet followed by some specific steps. The connection procedure is discussed below.

64.15.1 Software and Hardware Requirements

Using the communication software we can connect the local computer to another computer available on the Internet. Communication software is commonly used with a modem. The software offers a way to give commands to a modem, to dial the other computer's modem, and then to issue commands and receive responses from the remote computer. Today many different softwares for different services are available.

The selection of the software is depends on how we want to use the computer and the services available from the service provider. TCP/IP based communication software uses either SLIP (Serial Line Internet Protocol) or PPP (point-to-Point Protocol).

The hardware could be a PC or a mainframe, depending on the software and service we need and the operating system which is used in the PC.

64.16 Internet Service Provider

The Internet Service Provider is basically a server and it is directly connected to the Internet. The connection of a local computer to the Interenet is done through a service provider. In India, Bharat Sanchar Nigam Limited (BSNL) is an Internet Service Provider and their service is available in many cities over the country.

Two basic types of the dial-up access are available:

* Dialling in as a terminal (Shell Account)
* Dialling in as a host (TCP/IP Account).

If we dial in as a network host, the computer becomes an Internet host as long as the computer is connected and acts like the other Internet hosts. If a computer is connected to the Internet then we can run any network application software we want. If the software is multitasking system like Microsoft Windows, Windows NT and Windows 95, then we can run several programs simultaneously.

To connect to the Internet we need to know all about connection and the Service Provider. We should know the following details to configure the software.

* User name,
* Password,
* The phone number to dial to connect to the Internet,
* Domain Name Server (DNS) address,
* Authentication procedure (whether or not your provider requires a terminal window to log on).

64.17 Module Summary

A very brief description of centralized and distributive computing is given at the ebggining of the module. The concept of Internet and its advantages are discussed. A brief history and development of Internet are introduced. The structure of communication network and different types of data flow and physical circuits are studied. Different kinds of network topologies and their aims are explained with diagram.s The LAN and its components with working methodologies are discussed in this module. The some standard Internet services such e-mail, telnet, FTP and WWW are discussed here. The software and hardware requirements including Internet service provider are all studied. The module ends with an exercise and references.

64.18 Self Assessment Questions

1. What do you mean by LAN, WAN and Internet?
2. What services are provided by the Internet? Describe them briefly.
3. What are the advantages of the networks?
4. Write short notes on e-mail, FTP, Telnet, WWW.
5. How two remote computers are connected to the Internet through communication systems? Explain with a diagram.
6. Explain the following terms in connection iwth data flow : simplex, half-duplex and full-duplex.
7. Explain with diagrams the Pont-to-Point and Multidro circuits.
8. Explain about the data flow techniques and physical circuits available on the Internet.
9. Write a note on network topologies.

10. What are LAN and WAN? What are their goals? Explain network protocols.
11. Write a procedure to connect your ocal computer to the Internet.
12. When a computer is connected to the Internet then what extra works are available to this computer compared to a stand-alone (the computer is not connected to the net) PC. Describe them briefly.
13. Which services are available by a PC when it is connected on the Internet. Describe them briefly.
14. What do you mean by WAN? Write a short note on WAN.
15. What do you mean by Internet Service Provider? What is the role of it in the Internet?
16. Write short notes on workstaton, server, LAN cables, LAN adaptor card.
17. What are the rules to numbering and naming a computer in the Internet. What is the rule to assign an e-mail address of a pupil.

64.19 Suggested Further Readings

1. Barry Nance, Introduction to Networking, Prentice-Hall of India.
2. Uyless Black, Computer Networks, Prentice-Hall of India.
3. Christian Crumlish, The ABCs of the Internet, BPB Publications.

--- 0 ----

M.Sc. Course
in
Applied Mathematics with Oceanology
and
Computer Programming

PART-II

Paper-VI Group–A

Module No. - 65
OPERATING SYSTEM-I

Structure :

65.1 Introduction
65.2 Objectives
65.3 Keywords
65.4 Structure of Operating System
65.5 What is an Operating System?
- 65.5.1 The operating system as an extended machine
- 65.5.2 The operating system as a resource manager

65.6 Operating System Concepts
- 65.6.1 Processes
- 65.6.2 Files
- 65.6.3 System Calls
- 65.6.4 The Shell

65.7 Operating System Structure
- 65.7.1 Monolithic systems
- 65.7.2 Layered systems
- 65.7.3 Virtual machines
- 65.7.4 Client-Server Model

65.8 Process Management

65.8.1 The process model
65.8.2 Implementation of processes
65.9 Interprocess Communication
65.9.1 Race Conditions
65.9.2 Sleep and Wakeup
65.9.3 The Producer-Consumer problem
65.9.4 Semaphores
65.10 Process Scheduling
65.10.1 Round Robin Scheduling
65.10.2 Priority scheduling
65.11 Module Summary
65.12 Self Assessment Questions
65.13 Suggested Further Readings

65.1 Introduction

Today we can't think a computer without operating system. Operating system have earned the reputation for being the most critical software in a computer system. Performance and functionality are key to the usefulness of an operating system. The operating system's performance sets the stages for the performance of all software on a computer. Perhaps one of the most important reasons for studying operating systems is to learn how to extract the performance from them. Also, the operating system provide a wide range of functions to assist in processing a program.

In our course, we introduce briefly the operating system, into two modules.

65.2 Objectives

The students will learn the following topics when finished this module.

* Definition of operating system
* Operating system as an extended machine and resoruce manager
* Processes
* Files
* System Calls
* Shell
* Structure of operating system
* Client-Server Model
* Process Management

* Race Conditions
* Sleep and Wakeup
* The Producer-Consumer problem
* Semaphores
* Round Robin Scheduling
* Priority scheduling

65.3 Keywords

Concepts of operating system, structure of operating system, process management, interprocess communication, process scheduling.

65.4 Structure of Operating System

Now a days it is well known that a computer is basically a useless instrument of metal. With software, a computer can store, process, and retrieve information and can do many other valuable activities. Computer software can be roughly divided into two kinds: the system programs which manage the operation of the computer itself, and the application programs, which solve problems for the users. The most fundamental of all the system programs is the operating system, which controls all the computer's resources and provides the base upon which the application programs can be written.

The present computer system consists of one or more processors, some main memory (often known as "core memory"), clock, terminals, disks, network interfaces, and other input/output devices. Writing programs that keep track of all these components and use them correctly is an extremely difficult job. If every programmer had to be concerned with how disk drives work, and with all the dozens of things that could go wrong when reading a disk block, it is unlikely that many programs could be written at all. Many years ago it became abundantly clear that some way had to be found to shield programmers from the complexity of the hardware. The way that has gradually evolved is to put a layer of software on top of the bare hardware, to manager all parts of the system, and present the user with an interface or virtual machine that is easier to understand and program. This layer of software is the operating system. This system is shown in Figure 65.1. At the bottom is the hardware, which is many case is itself composed of two or more layers. The lowest layer contains physical devices, consisting of integrated circuit chips, wires, power supplies, cathode ray tubes, and similar physical devices.

The next is a software that directly controls these devices and provides a cleaner interface tot he next layer. This software, called the microprogram and it is generally located in read-only memory. It is actually an interpreter, fetching the machine language instructions such as ADD, MOVE, and JUMP. To carry out an ADD instruction the microprogram must determine where the numbers to be added are located, fetch them, add them, and store the

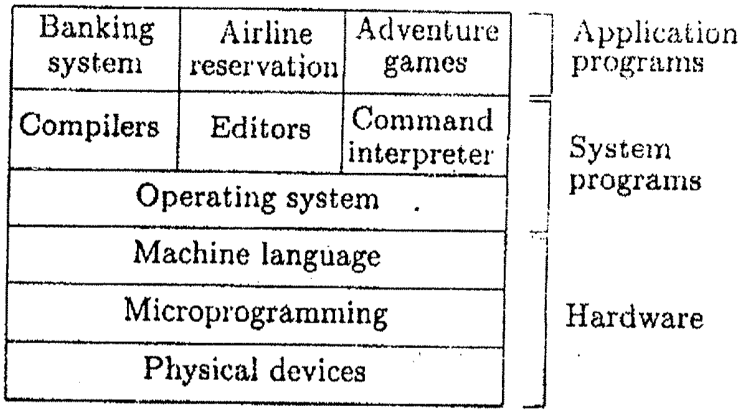


Figure 65.1 : A computer system consists of hardware, system programs and application programs.

result somewhere. The set of instructions that the microprogram interprets defines the machine language, which is not really part of the hard machine at all. On some machines the microprogram is implemented in hardware, and is not really a distinct layer. The machine language typically has between 50 and 300 instructions, mostly for moving data around the machine, doing arithmetic, and comparing values. In this layer, the input/output devices are controlled by loading values into special device registers. For example, a disk can be commanded to read by loading the values of the disk address, main memory address, byte count, and direction (READ or WRITE) into its registers. In practice, many more parameters are needed, and the status returned by the drive after an operation is highly complex. Furthermore, for many I/O devices, timing plays an important role in the programming. A major function of the operating system is to hide all this complexity and give the programmer a more convenient set of instructions to work with. For example, read block from file is conceptually simpler than having to worry about the details of moving disk heads, waiting for them to settle down, and so on. On top of the operating system is the rest of the system software. Here we find the command interpreter (shell), compilers, eidtors and similar application independent programs. It is important to realize that these programs are definitely not part of the operating system. The operating system is that portion of the software that runs in kernel mode or supervisor mode. It is protected from user tampering by the hardware (ignoring for the moment some of the older microprocessors that do not have hardware protection at all). Compilers and editors run in user mode.

65.5 What is an Operating System :

Most computer users have some experience with an operating system, but it is difficult for tehm accurately what an operating system is. Part of the problem is that operating systems perform two basically unrelated functions, these are described below:

65.5.1 The operating system as an extended machine

The architecture (instruction set, memory organization, I/O and bus structure) of most computers at the machine language level is primitive and awkward to program, especially for input/output. The program that hides the truth about the hardware from the programmer and presents a nice, simple view of named files that can be read and written is the operating system. Just as the operating system shields the programmer from the disk hardware and presents a simple file-oriented interface, such as timers, memory management, and other low-level features. In each case, the abstraction presented to the user of the operating system is simpler and easier to use than the underlying hardware. Int his view, the function of the operating system is to present the user with the equivalent of an extended machine or virtual machine that is easier to program than the underlying hardware.

65.5.2 The operating system as a resource manager

The concept of the operating system as primarily providing its users with a convenient interface is topdown view. An alternative, bottom-up, view holds that the opering system is there to manage all the pieces of a complex system. Modern computers consist of processors, memories, timers, disks, terminals, magnetic tape drives, network interfaces, laser printers, and a wife variety of other devices. The job of the operating system is to provide for an orderly and controlled allocation of the processors, memories, and I/O devices among the various programs competing for them.

When a computer has multiple users, the need for managing and protecting the memory, I/O devices, and other resources is even more apparent. This need arises because it is frequently necessary for users to share expensive resources such as tape drives and phototypesetters. Economic issues aside, it is also often necessary for users who are working together to share information. In short, this view of the operating system holds that its primary taks is to keep track of who is using which resource, to grant resource requests, to account for usage, and to mediate conflicting requests from different programs and users.

65.6 Operating System Concepts

The interface between the operating system and the user programs is defined by the set of "extended instructions" and this instructions are provided by operating system. These extended instructions are known as **system calls.** The system calls create, delete, and use various software objects managed by the operating system. The most important of these are *processes* and *files.*

65.6.1 Processes

The main concept of every operating system is the process. A process is basically a program in execution. It consists of the executable program, the program's data and stack, its program counter, stack pointer, and other

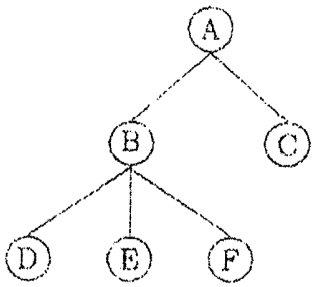


Figure 65.2 : A process tree. Process A created two child processes; B and C. Process B created three child processes, D, E, and F.

registers, and all the other information needed to run the program. We will discuss about the process for the timesharing systems. Periodically, the operating system decides to stop running one process and start running another.

When a process is temporarily suspended, it must later be restarted in exactly the same state when it was stopped. This means that all information about the process must be explicitly saved somewhere during the suspension. For example, if the process has several files open, the exact position in the files where the process was must be recorded somewhere, so that a subsequent READ given after the process is restarted will read the proper data. In many operating systems, all the information about each process, other than the contents of its own address space, is stored in an operating system table called the **process table,** which is an array (or linked list) of structures, one for each process currently in existence.

Thus, a suspended process consists of its address space, usually called the **core image** and its process table entry, which contains its registers, among other things. The key process management system calls are those dealing with the creation and termination of processes. If a process can create one or more other processes (referred to as child processes) and these processes in turn can create child processes. The relationship among the child processes is shown in Figure 65.2.

In a multiprogramming system, it is important to keep track of which user owns which process. In such a system, each authorized user is assigned a **uid** (user identification), typically a 16-bit or 32-bit integer. Each process is assigned the auid of its owner. When a process sends a signal to another process, a check can be made to see that the sender and receiver have the same uid. Similarly, people can be divided into groups (project teams, departments, etc.), each with its own **gid** (group identification). The uids and gids also play a role in protecting information in the computer.

65.6.2 Files

The other broad category of system calls relates to the file system. A major function of the operating system is to hid the peculiarites of the disks and otehr I/O devices, and present the programmer with a nice, clean abstract model

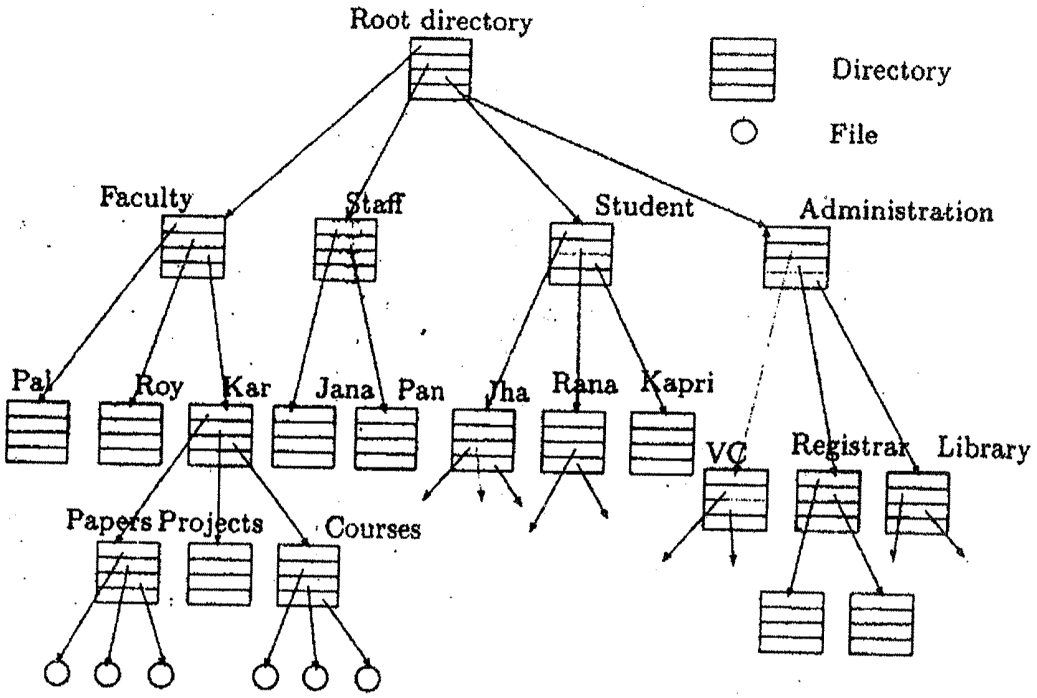


Figure 65.3: File system of a university.

of device-independent files. System calls are obviously needed to create files, remove files, read files, and write files. Before a file can be read, it must be opened, and it should be closed after reading. In order to provide a place to keep files, most operating systems support the concept of a directory (folder) as a way of grouping files together. System calls are then needed to create and remove directories. Calls are also provided to put an existing file in a directory, and to remove a file from a directory. Directory entries may be either files or other directories. This model also gives rise to a hierarchy-the file system, as shown in Figure 65.3.

The process and file hierarchies both are organized as trees, but the similarity stops there. Process hierarchies usually are not very deep (usually not more than three), whereas file hierarchies are commonly four, five, or even more levels deep. Process hierarchies are typically short-lived, generally a few minutes at most, whereas the directory hierarchy may exist for years. Ownershhip and protection also differ for processes and files. Typically, only a parent process may control or even access a child process, but mechanisms nearly always exist to allow files and directories to be read by a wider group than just the owner. Every file within the directory hierarchy can be specified by giving its path name from the top of the directory hierarchy, the root directory. Such absolute path names consist of the list of directories that must be traversed from the root directory to get to the file, with slashes separating the components. At every instant, each process has a current working directory, in which path names

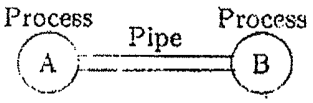


Figure 65.4: Two processes connected by a pipe.

not beginning with a slash are looked for. Processes can change their working directory by issuing a system call specifying the new working directory. As an aside, some systems use/ instead of /as the separator in path names.

Before a file can be read or written, it must be opened, at which time the permissions are checked. If the acces is permitted, the system returns a small integer called a **file descriptor** or **handle** to use in subsequent operations. If the access is prohibited, an error code is returned.

Many operating systems, including UNIX and MS-DOS, provide an abstraction to allow users to perform input/output without getting buried in all the details of the hardware. This abstraction represents each I/O device as a special file. Special files are provided in order to make I/O devices look like files. That way, they can be read and wrtie using the same system calls as are used for reading and writing files. Two kinds of special files exist, viz., block special files and character special files. Block special files are used to model devices that consist of a collection of randomly addressable blocks, such as disks. By opening a block special file and reading, say, block 4, a program can directly access the fourth block on the device, without regard to the structure of the file system contained on it. Programs that do system maintenance often need this facility. Access to special files is controlled by the same protection mechanism used to protect other files. Character special files are used to model devices that consist of character streams, rather than fixed-size randomly addressable blocks. Terminals, line printers, and network interfaces are typical examples of character special devices. The normal way for a program to read and write on the user's terminal is to read and write the corresponding character special file. In UNIX and MS-DOS, when a process is started up, file descriptor 0, called **standard input,** is normally arranged to refer to the terminal for the purpose of reading. File descriptor 1, called **standard output,** refers to the terminal for writing. File descriptor 2, called **standard error,** also refers to the terminal for output, but normally is used only for writing error messages. The last feature we will discuss in this overview is one that relates to both processes and files called pipes. A **pipe** is a sort of pseudo-file that can be used to connect two processes together, as shwon in Figure 65.4. When process A wants to send data to process B, it writes on the pipe as though it were an output file. Process B can read the data by reading from the pipe as though it were an input file. Thus, communication between processes looks very much like ordinary file reads and writes.

65.6.3 System Calls

User programs communicate with the operating system and request services from it by making system calls. Corresponding to each system call is a library procedure that user programs can call. This procedure puts the parameters of the system call in a specified place, such as the machine registers, and then issues a TRAP instruction (a kind of protected procedure call) to start the operating system. The purpose of the library procedure is to hide the details of the TRAP instruction and make system calls look like ordinary procedure calls. When the operating system gets control after the TRAP, it examines the parameters to see if they are valid, and if so performs the work requested. When it is finished, the operating system puts a status code in a register, telling whether it succeeded or failed, and executes a RETURN FROM TRAP instruction, to return control back to the library procedure. The library procedure then returns to the caller int he usual way, returning the status code as a function value. Sometimes additional values are returned in the parameters.

To make the system call mechanism clear, we a simple example the READ system call used in UNIX and MS-DOS. It has three parameters, the first one specifying the file to be read, the second one specifying a buffer to put the file's data, and the third one telling how many bytes to read. A call to READ from a C program might look like

count = read (file, buffer, nbytes);

Note that there is a subtle difference between the user-callable library procedure, *read*, and the actual system call, READ, which is invoked by *read*. To try to make these concepts clear, system call names will be printed in caps (e.g., READ) while procedure names will be printed in italics (e.g., *read*). In many contexts they are effectively interchangeable.

The effect of calling the *read* proceudre (and thus making the READ system call) is to cause data from the specified file to be copied into the buffer, where the program can get at it. The procedure returns the number of bytes actually read in count. This value is normally the same as *nbytes*, but may be smaller, for example, the end-of-file is encountered while reading. If the system call cannot be carried out, either due to an invalid parameter or a disk error, *count* is set to-1, and the error number is put in a global variable, where the program can inspect it. Programs should always check the results of their system calls to see if errors occurred.

65.6.4 The Shell

The operating system is the code that carries out the system calls. Editors, compilers, assemblers, linkers, and command interpreters are not part of the operating system, even though they are important and useful. To avoid confusing, here we consider the UNIX command interpreter, called the shell, which, although not part of the operating system, makes heavy use of many operating system features and thus serves as a good example of how

the system calls can be used. It is also the primary interface between a user sitting at his terminal and the operating system.

When any user logs in, a shell is started up. The shell has the terminal as standard input and standard output. It starts out by typing the prompt, a character such as a dollar sign, which tells the user that the shell is waiting to accept a command. If the user types

```
$date
```

the shell creates a child process and runs the *date* program as the child. While the child process is running, the shell waits for it to terminate. When the child finishes, the shell types the prompt again and tries to read the next input line.

The user can specify that standard output be redirected to a file by typing, for example,

```
$sort < file1 > file2
```

which invokes the sort program with input taken from *file1* and output sent to *file2*. The output of one program can be used as the input for another program by connecting them with a pipe. Thus

```
$cat file1 file2 file3 / sort > /dev/1p
```

invokes the *cat* program to concatenate three files *file1, file2, file3* and send the output to *sort* to arrangeall the lines in alphabetical order. The output of *sort* is redirected to the file */dev/lp*, which is a typical name for the special character file for the line printer.

If a user puts an ampersand after a command, the shell does not wait for it to complete. Instead it just gives a prompt immediately. Consequently,

```
$cat file1 file2 file3 | sort >/dev/1p &
```

and starts up the sort as a background job, allowing the user to continue working normally while the sort is going on.

65.7 Operating System Structure

Here we will examine four different structures to get some idea of the inside of the operating system.

65.7.1 Monolithic systems

The most common organisation, this approach referred as "The Big Mess". The operating system is written as a collection of procedures, each of which can call any of the other ones whenever it needs to. When this technique is used, each procedure in the system has a well-defined interface in terms of parameters and results, and each one is free to call any other one, if the latter provides some useful computation that the former needs.

To construct the actual object program of the operating system when this approach is used, one compiles all the individual procedures, or files containing the procedures, and then binds them all together into a single object

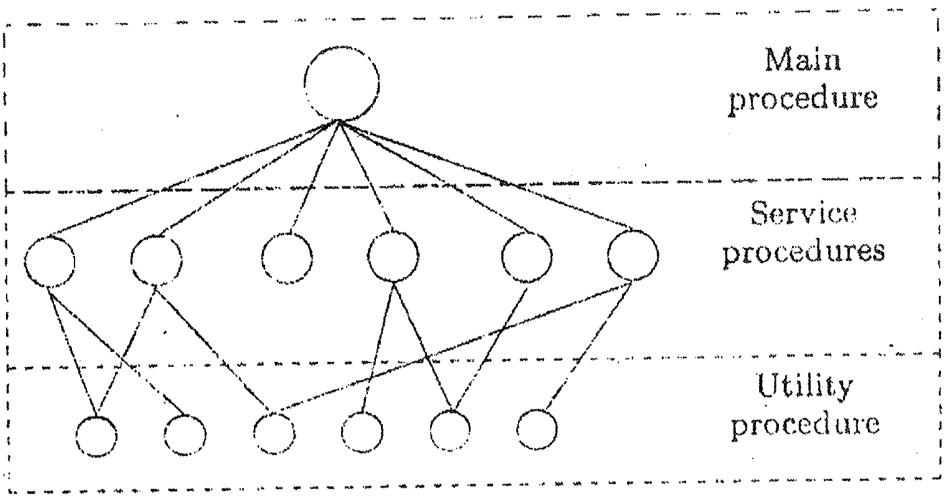


Figure 65.5: A simple structuring model for a monolithic system.

file with the linker. In terms of information hiding, there is essentially none-every procedure is visible to every other one as opposed to a structure containing modules or packages, in which much of the information is local to a module, and only officially designated entry points can be called from outside the module. Even in monolithic systems, however, it is possible to have at least a little structure. The services (system calls) provided by the operating system are requested by putting the parameters in well-defined places, such as in registers or on the stack, and then executing a special TRAP instruction known as a **kernel call** or **supervisor call.** This instruction switches the machine from **user mode to kernel mode** (also known as supervisor mode), and transfers control to the operating system.

The operating system then examines the parameters of the call to determine which system call is to be carried out. Next, the operating system indexes into a table that contains in slot *k* a pointer to the procedure that carries out system call *k*. This operation identifies the service procedure, which is then called. Finally, the system call is finished and control is given back to the user program. This organization suggests a basic structure for the operating system:

* A main program that invokes the requested service procedure.
* A set of service procedures that carry out the system calls.
* A set of utility procedures that help the service procedures.

In this model, for each system call there is one service procedure that takes care of it. The utility procedures do things that are needed by several service procedures, such as fetching data from user programs. This division of the procedures into three layers is shown in Figure 65.5.

5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Figure 65.6: Structure of the THE operating system.

65.7.2 Layered systems

The above approach suggest a generalization of operating system as a hierarchy of layers, each one constructed upon the one below it. The first system constructed in this way was the THE system built by E.W. Dijkstra (1968) and his students. The THE system was a simple batch system. The system had 6 layers, as shown in Figure 65.6.

Layer 0 dealt with allocation of the processor, switching between processes when interrupts occurred or timers expired. Above layer 0, the system consisted of sequential processes, each of which could be programmed without having to worry about the fact that multiple processes were running on a single processor. In other words, layer 0 provided the basic multiprogramming of the CPU. Layer 1 did the memory management. It allocated space for processes in main memory and on a 512K word drum used for holding parts of processes (pags) for which there was no room in main memory. Above layer 1, processes did not have to worry about whether they were in memory or on the drum; the layer 1 software took care of making sure pages were brought into memory whenever they were needed. Layer 2 handled communication between each process and the operator console. Above this layer each process effectively had its own operator console. Layer 3 took care of managing the I/O devices and buffering the information streams to and from them. Above layer 3 each process could deal with abstract I/O devices with nice properties, instead of real devices with many peculiarities. Layer 4 was where the user programs were found. They did not have to worry about process, memory, console, or I/O management. The system operator process was located in layer 5.

65.7.3 Virtual machines

The virtual machines originally called CP/CMS and now called VM/370, was based on an astute observation : a timesharing system provides (i) multiprogramming and (ii) and extended machine with a more convenient interface than the abre hardware. The essence of VM/370 is to completely separate these two functions. The heart of the system, known as the virtual machine monitor, runs on the bare hardware and does the multiprogramming, providing

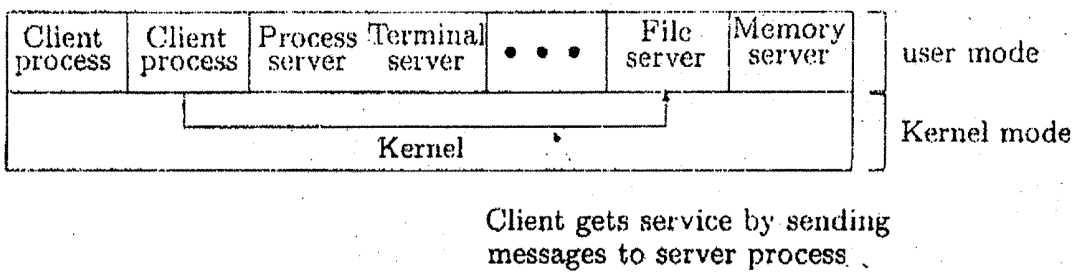


Figure 65.7: The client-server model.

not one, but several virtual machines to the next layer up. However, unlike all other operating systems, these virtual machines are not extended machines, with files and other nice features. Instead, they are exact copies of the bare hardware, including kernel/user mode, I/O, interrupts, and everything else the real machine has. Because each virtual machine is identical to the true hardware, each one can run any operating system that will run directly on the hardware. Different virtual machines can, and usually do, run different operating system. Some run one of the descendants of OS/360 for batch processing, while other ones run a single-user, interactive system called CMS (Conversational Monitor System) for timesharing users. Whena CMS program executes a system call, the call is trapped to the operating system in its own virtue machine, not to VM/370, just as it would if it were running on a real machine instead of a virtual one. CMS then issues the normal hardware I/O instructions for reading its virtual disk or whatever is needed to carry out the call. These I/O instructions are trapped by VM/370, which then performs them as part of tis simulation of the real hardware. By making a complete separation of the functions of multiprogramming and providing an extended machine, each of the pieces can be much simpler, more flexible, and easier to maintain.

65.7.4 Client-Server Model

A trend in modern operating systems is to take this idea of moving code up into higher layers even further, and remove as much as possible from the operating system, leaving a minimal kernel. The usual approach is to implement most of the operating system functions in user processes. To request a service, such as reading a block of a file, a user process (now knonw as the **client process**) sends the request to a **server process,** which then does the work and sends back the answer.

In this model (see Figure 65.7) all the kernel does is handle the communication between clients and servers. By splitting the operating system up into parts, each of which only handles one facet of the system, such as file service, process service, terminal service, or memory service, each part becomes small and manageable. Furthermore, because all the servers run as user-mode processes, and not in kernel mode, they do not have direct access to the hardware. As a consequence, if a bug in the file server is triggered, the file service may crash, but this will not usually

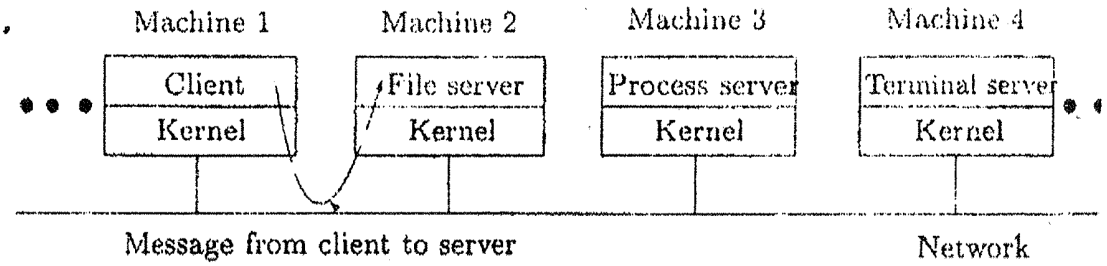


Figure 65.8: The client-server model in a distributed system.

bring the whole machine down. Another advantage of the client-server model is its adaptability to use in distributed systems (see Figure 65.8). If a client communicates with a server by sending it messages, the client need not know whether the message is handled locally in its own machine, or whether it was sent across a network to a server on a remote machine. A far as the client is concerned, the same thing happens in both cases: a request was sent and a reply came back.

In Figure 65.8, we shown only the passing of message from clients to servers and back is not completely realistic. Some operating system functions (such as loading commands into the physical I/O device registers) are difficult, if not impossible, to do from user-space programs. There are two ways of dealing with this problem. One way is to have some critical server processes (e.g., I/O device drivers) actually run in kernel mode, with complete access to all the hardware, but still communicate with other processes using the normal message mechanism. The other way is to build a minimal amount of mechanism into the kernel, but leave the policy decisions up to servers in user space. For example, the kernel might recognize that a message sent to a certain special address means to take the contents of that message and load it into the I/O device registers for some disk, to start a disk read. In this example, the kernel would not even inspect the bytes in the message to see if they were valid or meaningful; it would just blindly copy them into the disk's device registers. The split between mechanism and policy is an important conceptl it occurs again and again in operating systems in various contexts.

65.8 Process Management

The main concept in operating system is the **process:** an abstraction of a running program. All modern computers can performed several things at the same moment. While running a user program, a computer can also be reading from a disk and printing on a terminal or printer. In a multiprogramming system, the CPU also switches from program to progam, running each for tens or hundreds of milliseconds. But, practically, at any instant of time, the CPU is executing only one program, in the course of 1 second, it may work on several programs, thus giving the users the illusion of parallelism. Sometimes people speak of pseudo parallelism to mean this rapid switching back and forth of the CPU between programs, to contrast it with the true hardware parallelism of the CPU computing while one or more I/O devices are running.

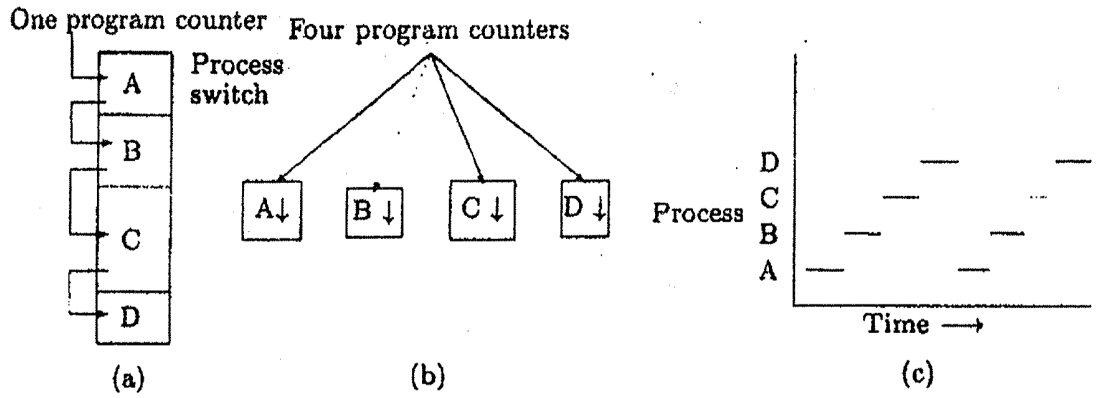


Figure 65.9: (a) Multiprogramming of four programs. (b) Conceptual model of four independent, sequential processes. (c) Only one program is active at any instant.

Keeping track of multiple, parallel activities is hard to do. Therefore, operating system designers over the years have evolved a model that makes parallelism easier to deal with.

65.8.1 The process model

In this model, all the runnable software on the computer, even operating system, is organized into a number of **sequential processes,** or just **processes** for short. A process is just an executing program, including the current values of the program counter, registers, and variables. Conceptually, each process has its own virtual CPU. In reality, the real CPU switches back and forth from process to process, but to understand the system, it is much easier to think about a collection of processes running in (pseudo) parallel, than to try to keep track of how the CPU switches from program to program. This rapid switching back and forth is called **multiprogramming.**

In Figure 65.9(a) we see a computer multiprogramming four programs in memory. In Figure 65.9(b) we see how this is abstracted into four processes, each with its own flow of control (i.e., its own program counter), and each one running independent of the other ones. In Figure 65.9(c) we observed that viewed over a long enough time interval, all the processes have made progress, but at any given instant only one process is actually running.

With the CPU switching back and forth among the processes, the rate at which a process performs its computation will not be uniform, and probably not even reproducible if the same processes are run again. Thus, processes must not be programmed with built-in assumptions about timing.

For example, an I/O process that starts a magnetic tape in motion, executes an idle loop 1000 times to let the tape get up to speed, and then issues a command to read the first record. If the CPU decides to switch to another process during the idle loop, the tape process might not run again until after the first record was already past the read head. When a process has critical real-time requirements like this, that is, certain events absolutely must occur

within a specified number of milliseconds, special measures must be taken to ensure that they do occur. Normally, however, most processes are not affected by the unerlying multiporgramming of the CPU or the relative speeds of different processes. The difference between a process and a program is subtle, but crucial.

Process hierarchies

In most systems, some way is needed to create and destroy processes as needed during operation. In UNIX, processes are created by the FORK system call, which creates an identical copy of the calling process. After teh FORK call, the parent continues running, in parallel with the child. The parent can then fork off more children, so at any instant it may have several executing children. The children can also execute FORK, so it is possible to get a whole tree of processes with arbitrary height.

Process states

Each process is an independent entity, with its own program counter and internal state, even then it needs to interact with other processes. One process may generate some output that another process uses as input. In the shell command

$cat art1 art2 art3 | grep tree

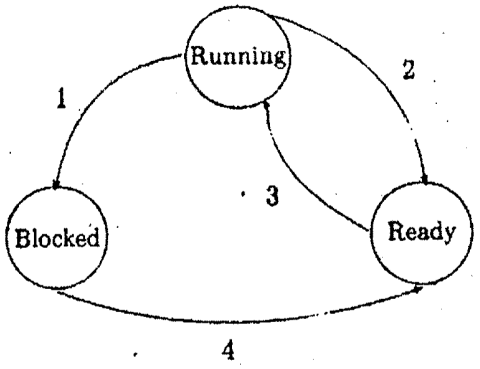
the first process, running *cat*, concatenates and outputs three files. The second process running *grep*, selects all lines containing the word "tree". Depending on the relative speeds of the two processes, it may happen that grep is ready to run, but there is no input waiting for it. It must then block until some input is available. It is also possible for a process that is conceptually ready and unable to run because the operating system has allocate the CPU to another process for a while. These two conditions are completely different. In the first case, the suspension is inherent in the problem. In the second case, it is a technical problem of the system (not enough CPUs to give each process its own private processor). In Figure 65.10 we see a state diagram showing the three states a process may in:

(i) Running (actually using the CPU at that instant).
(ii) Ready (runnable; temporarily stopped to let another process run).
(iii) Blocked (unable to run until some external event happens).

In both cases the process is willing to run, only the second one, there is temporarily no CPU available for it. The third state is different from the first two in that the process cannot run, even if the CPU is idle.

65.8.2 Implementation of processes

To implement the process model, the operating system maintains a table (an array of structures), called the **process table**, with one entry per process. This entry contains information about the process state, its program counter,



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Figure 65.10: Transitions between three states running, blocked and ready of a process are shown here.

stack pointer, memory allocation, the status of its open files, its accounting and scheduling information, and everything else about the process that must be saved when the process is switched from *running* state so that it can be restarted later as if it had never been stopped.

65.9 Interprocess Communication

There is a need for communication between processes, preferably in a well-structured way not using interrupts. In the following sections we will look at some of the issues related to this **InterProcess Communication or IPC.**

65.9.1 Race Conditions

In some operating systems, the processes share some common storage that each one can read and write. The shared storage may be in main memory or it may be a shared file; the location of the shared memory does not change the nature of the communication or the problems that arise.

To explain the interprocess communication, let us consider a simple but common example, a print spooler. When a process wants to print a file, it enters the file name in a special spooler directory. Another process, the printer daemon, periodically checks to see if there are any files to be printed, and if there are it prints them and then removes their names from the directory. Suppose that our spooler directory has a large number of slots, numbered 0, 1, 2, ..., each one capable of holding a file name. Again, we assume that there are two shared variables, one of them points to the next file to be printed, and other points to the next free slot in the directory. At a certain instant, slots 0 to 3 are empty (the files have already been printed) and slots 4 to 6 are full (with the names of files queued for printing). More or less simultaneously, processes A and B decide they want to queue a file for printing. This situation is shown in Figure 65.11.

Process A reads *in* and stores the value 7 in a local variable say, *next-slot*. If a clock interrupt occurs and the CPU decides that process A has run, so it switches to process B. Process B also reads *in* and also gets a 7, so it

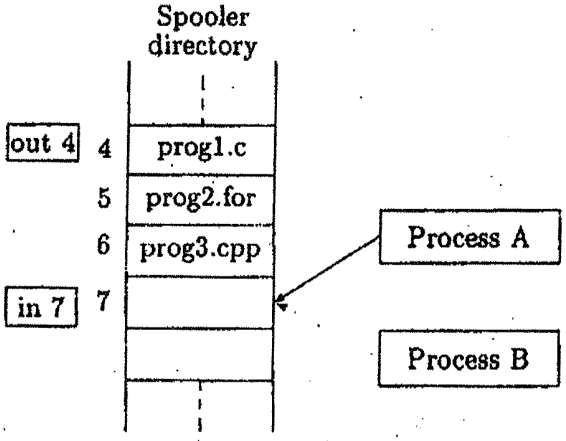


Figure 65.11: Two processes want to access shared memory at the same time.

stores the name of its file in slot 7 and updates as 8. Then it goes off and does other things.

Eventually process A runs again, starting from the place where it was stoped. It finds the *next-free-slot*, finds a 7 there, and writes its file name in slot 7, erasing the name that process B just put there. Then it computes *next-free-slot*+1, which is 8, and sets in to 8. The spooler directory is now internally consistent, so the printr daemon will not notice anything wrong, but process B will never get any output. Situations like this, where two or more processes are reading or writing some shared data and the final result depends on who runs precisely when, are called **race conditions** Debugging programs containing race conditions is no fun at all.

65.9.2 Sleep and Wakeup

Consider a computer with two processes, H, with high priority and L, with low priority. The scheduling rules are such that H is run whenever it is in ready state. At a certain moment, with L in its critical region, H becomes ready to run (e.g., an I/O operation completes). H now begins busy waiting, but since L is never scheduled while H is running, L never gets the chance to leave its critical region, so H loops forever. this situations is sometimes referred to as the **priority inversion problem.**

Now let us look at some interprocess communication primitives that block instead of wasting CPU time when they are not allowed to enter their critical sections. One of the simplest is the pair SLEEP and WAKEUP. SLEEP is a system call that causes the caller to block, that is, be suspended until another process wakes it up. The WAKEUP call has one parameter, the process to be awakened. Alternatively, both SLEEP and WAKEUP each have one parameter, a memory address used to match up SLEEPs with WAKEUPSs. These two prmitives are used in the following problem.

65.9.3 The Producer-Consumer problem

Now, we consider the producerconsumer problem, this problem is also known as **bounded buffer** problem. Two processes share a common, fixed-size buffer. One of them, the producer, puts information into the buffer and the other one, the consumer, takes it out. Problem arises when the producer wants to put a new ite4ms in the buffer, but it is already full. The solution is for the producer to go to sleep, to be awakened when the consumer has removed one or more items. Similarly, if the consumer wants to remove an item from the buffer and sees that the buffer is empty, it goes to sleep until the producer puts something in the buffer and waks it up.

This approach sounds simple enough, but it leads to the same kinds of race conditions. To count the number of items in the buffer, we need a variable, say 'count'. If the maximum number of items the buffer can hold is *N*, the producer's code will first test to see if count is *N*. If it is, the producer will go to sleep; if it is not, the producer will add an item and increment count. The consumer's code is similar : first test count to see if it is 0. If it is, go to sleep; if it is nonzero, remove an item and decrement the counter. Each of the processes also tests to see if the other should be sleeping, and if not, wakes it up. The code for both producer and consumer is shown in the following program.

```
#include <prototypes.h>
#define N 100          /*capacity of the buffer*/
#define TRUE 1
int count = 0        /*numebr of items in the buffer*
void producer(void)
      {
      int item;
      while (TRUE)
      {
      producer_item(&item);        /*generate a new item*/
      if(count= =N) sleep ( );      /* if buffer is full, go to sleep*/
      enter_item(item);            /*put tem in buffer*/
      count=count+1;               /*increment count of items in buffer*/
      if (count= =1) wakeup (consumer); /*buffer was empty*/
      }
}
void consumer ( )
{
      int item;
      while (TRUE)
```

```
    {
    if(count= =0) sleep ( );        /*if buffer is empty go to sleep*/
    remove_item(&item);        /*remove an item from buffer*
    count=count-1;              /*decrement count of items*/
    if(count = =N–1) wakeupt(producer);  /*buffer was full*/
    consume_item(item);        /*print item*/
    }
}
```

To express sytem calls such as SLEEP and WAKEUP in C, we wills how them as calls to library routines. They are not part of the standard C library, but we asume that these are available on any system that actually had these system calls. The procedures *enter-item* and *remove_item*, which are not shown, handle the bookkeeping of putting items into and taking items out of the buffer.

Now we discuss about the race condition. It can occur because access to *count* is unconstrained. The following situation may occur. The buffer is empty and the consumer has just read *count* is as 0. At that instant, the scheduler decides to stop running the consumer temporarily and start running the producer. The producer enters an item in the buffer, increments count, and notices that it is now 1. Reasoning that count was just 0, and thus the consumer must be sleeping, the producer calls wakeup to wake the consumer up. Unfortunately, the consumer is not yet logically asleep, so the wakeup signal is lost. When the consumer next runs, it will test the value of *count* it previously read, find it to be 0, and go to sleep. Sooner or later the producer will fill up the buffer and also go to sleep. Both will sleep forever.

65.9.4 Semaphores

The **semaphore** is an integer variable which count the number of wakeups saved for future use and this variable is suggested by E.W. Dijkstra in 1965. A semaphore could have the value 0, indicating that no wakeups were saved, or some positive value if one or more wakeups were pending. Dikstra proposed having two operatons, DOWN and UP (generalizatons of SLEEP and WAKEUP, respectively). The DOWN operation on a semaphore checks to see if the value is greater than 0. If so, it decrements the value (i.e., uses up one stored wakeup) and just continues. If the value is 0, the process is put to sleep. Checking the value, chaning it, and possibly going to sleep is all done as a single, indivisible, atomic action. It is guaranteed that once a semaphore operation has started, no other process can access the sempahore until the operation has completed or blocked. This atomicity is absolutely essential to solving synchronization problems and avoiding race conditions. The UP operation increments the value of the seaphore addressed. If one or more processes were sleeping on that semaphore, unable to complete an earlier DOWN operaton, one of them is chosen by the system (e.g., at random), and is allowed to complete its DOWN. Thus, after an UP on a sempahore with processes sleeping on it, the semaphore will still be 0, but there

will be one fewer process sleeping on it. the operation of incrementing the semaphore and waking up one process is also invidisible. No process ever blocks doing an UP, just as no process ever blocks doing a WAKEUP in the earlier model.

Solving the Producer-Consumer Problem using Semaphores

Semaphores solve the lost-wakeup problem. It is essential that they be implemented in an indivisible way. The normal way is to implement UP and DOWN as system calls, with the operating system briefly disabling all interrupts, while it is testing the semaphore, updating it, and putting the process to sleep, if necessary. If multiple CPUs are being used, each semaphore should be protected by a lock variable, with the TSL instruction used to make sure that only one CPU at a time examines the semaphore. Be sure you understand that using TSL to prevent several CPUs from accessing the semaphore at the same time is quite different from busy waiting by the producer or consumer waiting for the other to empty or full the buffer. The semaphore operation will only take a few microseconds, whereas the producer or consumer might take arbitrarily long. This solution uses three sempahores : one caleld *full* for counting the number of slots that are full, one called *empty* for counting the number of slots that are empty, and one called *mutex* to make sure the producer and consumer do not access the buffer at the same time. The variable *full* is initially 0, *empty* is initially equal to the number of slots in the buffer, and *mutex* is initially 1. Semaphores that are initialized to 1 and used by two or more processes to ensure that only one of them can enter its critical region at the same time are called **binary semaphores.** If each process does a DOWN just before entering its critical region and an UP just after leaving, it mutual exclusion is guaranteed.

```
#include <prototypes.h>
#define N 100          /*number of slots in the buffer*/
#define TRUE 1
typedef int sempahore;
semaphore mutex = 1        /*controls access to critical region*/
semaphore empty = n        /*counts empty buffer slots*/
sempahore full = 0         /*counts full buffer slots*/

void producer ( )
      {
      int item;
      while (TRUE)
      {
      produce_item(&item);          /*generate a new item*/
      down(&empty);                 /*decrement empty count*/
```

```
        down (&mutex);          /*enter critical region*/
        enter_item(item);       /*put new item in buffer*/
        up(&mutex);             /*leave critical region*/
        up(&full);              /*increment count*/
        }
}
void consumer ( )
        {
        int item;
        while (TRUE)
        {
        down(&full);            /*decremet full count*/
        down(&mutex);           /*enter critical region*/
        remove_item(&item);     /*remove an item from buffer*/
        up(&mutex);             /*leave critical region*/
        up*&empty);             /*increment count of empty slots*/
        consume_item (item);    /*same item for further use*/
        }
}
```

In the program the semaphores are used in two different wasy. This difference is important enough to make explicit. The *mutex* semaphore is used for mutual exclusion. It is designed to guarantee that only one process at a time will be reading or writing the buffer and the associated variables. This mutual exclusion is required to prevent chaos.

65.10 Process Scheduling

In the previous section we have seen that two or more processes (e.g., producer and consumer) may logically runnable. When more than one process is runnable, the operating system must decide which one to run first. That part of the operating system which takes decision is called the **scheduler,** and the algorithm it uses is called the **scheduling algorithm.** Back in the old days of batch systems with input in the form of card images on a magnetic tape, the scheduling algorithm was simple : just run the next job on the tape. With multiuser timesharing systems, often combined with batch jobs in the back ground, the scheduling algorithm is more complex. There are multiple users waiting for service, and there may be batch or other background jobs as well. Even in pure timesharing systems there are often background jobs. Such as the electronic mail system, which often is running all the time, sending or receiving mail or network news. Before looking at specific scheduling algorithms, we should think about what the scheduler is trying to achieve. After all the scheduler is concerned with deciding on policy, not providing

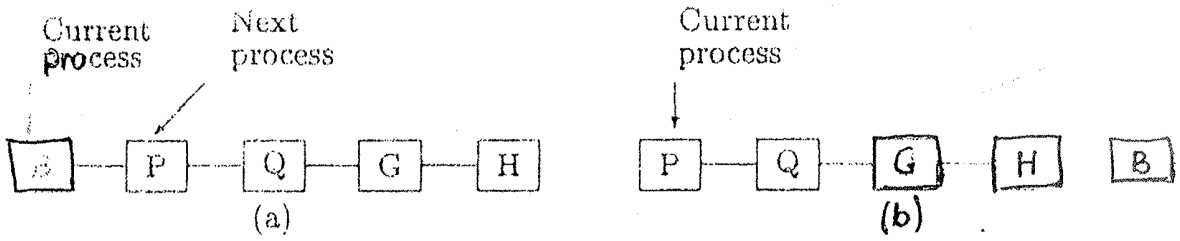


Figure 65.12 : Round robin scheduling. (a) The list of runnable processes. (b) The list of runnable processes after completion of B's quantum.

a mechanism. Various criteria constitutes a good scheduling algorithm, some of the more obvious possibilities are:

(i) Fairness : make sure each process gets its fair share of the CPU.
(ii) Efficiency : keep the CPU busy 100% percent of the time.
(iii) Response time : minimize response time for interactive users.
(iv) Turnaround : minimize the time batch users must wait for output.
(v) Throughput : maximize the number of jobs processed per hour.

It may be noted that some of these goals are contradictory. The strategy of allowing processes that are really runnable to be temporarily suspended is called **preemptive scheduling**, and is in contrast to the run to completion method of the early batch systems. Run to completion is also called **nonpreemptive scheduling**.

65.10.1 Round Robin Scheduling

One of the oldest, simplest, fairest and most widely acceptable algorithms is round robin. Each process is assigned a time interval, called its **quantum**, which it is allowed to run. If the process still running at the end of the quantum, the CPU is preempted and given to another process. If the process has blocked or finished before the quantum has elapsed, the CPU swithcing is done when the process blocks. Round robin is easy to implement. All the scheduler needs to do is maintain a list of runnable process, shown in Figure 65.12(a). When the quantum runs out on a process, it is place on the end of the list, shown in Figure 65.12(b).

The only interesting issue with round robin is the length of the quantum. Swithing from one process to another requires a certain amount of time for doing other tasks, such as, saving and loading registers and memory maps, updating various tables and lists, etc. Suppose this process switch or context swithc, as it is some times called, takes 5 msec. Also suppose that the quantum is set at 20 msec. With these parameters, after doing 20 msec of useful work, the CPU will have to spend 5 msec on process switching. Twenty percent of the CPU time will be

wasted on other (saving, loading etc.) works. To improve the CPU efficiency, we can choose the quantum to, say, 500 msec. now the wasted time is less than 1 percent. But consider what happens if ten interactive users hit the carriage return key at roughly the same time. Ten processes will be put on the list of runnable processes. If the CPU is idle, the first one will start immediately, the second one may not start until about 1/2 sec later, and so on. The unlucky last one may have to wait 5 sec before getting a chance, assuming all the others use their full quanta. Most users will perceive a 5-sec response to a short command as terrible.

The conclusion can be formulated as follows: setting the quantum too short causes too many process switches and lowers the CPU efficiency, but setting it too long may cause poor response to shan interactive requests. A quantum around 100 msec is often a reasonable value.

65.10.2 Priority scheduling

In round robin scheduling it is assume that all processes are equally important. But it is not happened always. For example, at a university computer center, the pecking order may be deans first, then professors, secretaries, and finally students. The need to take external factors into account leads to priority scheduling. The basic idea is straightforward : each process is assigned a priority, and the runnable process with the highest priority is allowed to run. To prevent high priority processes from running indefinitely, the scheduler may decrease the priority of the currently running process at each clock tick (i.e., at each clock interrupt). If this action causes its priority to drop below that of the next highest process, a process switch occurs. Priorities can be assigned to processes statistically or dynamically.

The UNIX system has a command, nice, while allows a user to voluntarily reduce the priority of his process, in order to be nice to the other users. Nobody ever uses it. Priorities can also be assigned dynamically by the system to achieve certain system goals. For example, some processes are highly I/O bound and spend most of their time waiting for I/O to complete. Whenever such a process wants the CPU, it should be given the CPU immediately, to let it start its next I/O request, which can then proceed in parallel with another process actually computing. Making the I/O bound process wait a long time for the CPU will just mean having it around occupying memory for an unnecessarily long time. A simple algorithm for giving good service to I/O bound processes is to set the priority to $1/f$, where f is the fraction of the last quantum that a process used. A process that used only 2 msec of its 100 msec quantum would get priority 50, while a process that run 50 msec before blocking would get priority 2, and a process that used the whole quantum would get priority 1. It is often convenient to group processes into priority classes and use priority scheduling among the classes but round robin scheduling within each class.

Figure 65.13 shows a system with four priority classes. The scheduling algorithm is as follows: as ong as there are runnable processes in priority class 4, just run each one for one quantum, round robin fashion, and never bother with lower priority classes. If priority class 4 is empty, then run the class 3 processes round robin. If classes

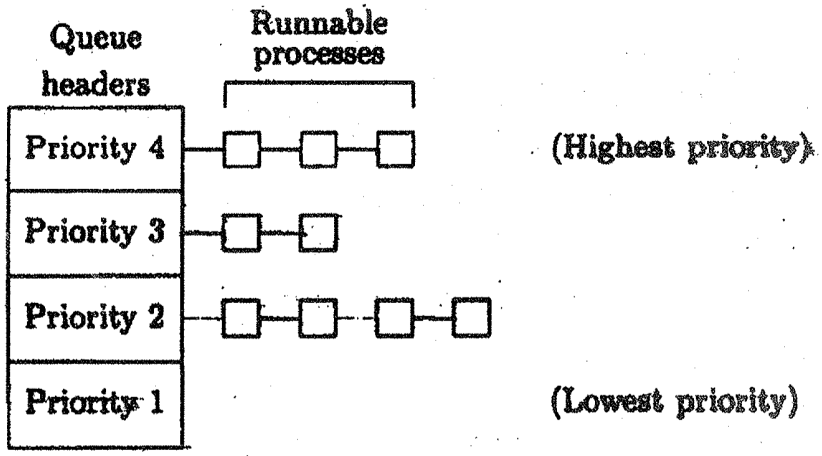


Figure 65.13: A scheduling algorithm with four priority classes.

4 and 3 are both empty, then run clas 2 round robin, and so on. If priorities are not adjusted from time to time, lower priority classes may all starve to death.

65.11 Module Summary

At the beginning of this module, the definition of operating systems and their components are given. It is well known that the operating system has many roles, that is, operating system is used as an extended machine, as a resource manager, etc. The operating system concepts such as processes, files, system calls, shell, etc. are discussed here. Different types of structures, viz., monolithic systems, layered systems, virtual machines, client-server model are introduced here. The process management and interprocess communication are briefly introduced. The producer-consumer problem is defined and its solution using sempahore is provided. Two scheduling methods such as round robin and priority are provided with this module. The module is end with an exercise and references.

65.12 Self Assessment Questions

1. What are the two main functions of an operating system?
2. What is multiprogramming? Give two reasons for having it.
3. Explain client-server model in operating system.
4. The client-server model is popular in distributed systems. Can it also be used in a single-computer system?
5. Why is the process table needed in a timesharing system? Is it also needed in personal computer

systems in which only one process exists, that process taking over the entire machine until it is finished?

6. What is a race condition?
7. Explain operating system as an extended machine.
8. Explain operating system as resource manager.
9. Explain operating system as a virtual machine.
10. Write short notes on monolithic system and layered system.
11. Write short notes on files, system calls, processes, shell.
12. Explain round robin scheduling.
13. Most round robin scheduling use a fixed size quantum. Give an argument in favour of a small quantum. Now, give one in favour of a large quantum.
14. Five batch jobs *A* through *E*, arrive at a computer centre at almost the same time. They have estimated running times of 10, 6, 2, 4 and 8 minutes. Their (externally determined) priorities are 3, 5, 2, 1 and 4 respecitively, with 5 being the highest priority. For the priority scheduling algorithm, determine the mean process turnaround time. Ignore process overhead.
15. Explain producer-consumer problem.
16. Solve the producer-consumer problem using semaphores.

65.13 Suggested Further Readings

1. A.S. Tenenbaum, Modern Operating Systems, Prentic Hall of India.
2. Gary Nutt, Operating Systems : A modern perspective, Addison-Wesley.
3. A.S. Godbole, Operating Systems, Tata McGraw-Hill, New Delhi.
4. H.M. Deitel, An Introduction to Operating Systems, 2e, Pearson Educaton Asia.
5. Charles Crowley, Operating Systems, Tata McGraw-Hill Edition.
6. William Stallings, Operating Systems, 2e, Prentice Hall of India.

--- 0 ---

M.Sc. Course
in
Applied Mathematics with Oceanology
and
Computer Programming

PART-II

Paper-VI

Group--A

Module No. - 66

OPERATING SYSTEM-II

Structure :

66.1 Introduction

66.2 Objectives

66.3 Keywords

66.4 Device Management

- 66.4.1 Devices
- 66.4.2 Device controllers
- 66.4.3 Device drivers
- 66.4.4 Interrupts

66.5 Memory Management

- 66.5.1 Memory management without swapping or paging
- 66.5.2 Multiprogramming and memory usage
- 66.5.3 Swapping
- 66.5.4 Multiprogramming with variable partitions
- 66.5.5 Memory management with linked lists
- 66.5.6 Memory management with the buddy system
- 66.5.7 Virtual memory

66.6 File Systems

- 66.6.1 File naming

66.6.2 Structure of a file
66.6.3 File types
66.6.4 File access
66.6.5 File attributes
66.6.6 File operations
66.7 Directories
66.7.1 Hierarchical directory systems
66.7.2 Path names
66.7.3 Directory operations
66.8 Security
66.8.1 The security environment
66.8.2 The Internet worm
66.8.3 Generic security attacks
66.8.4 Viruses
66.9 Module Summary
66.10 Self Assessment Questions
66.11 Suggested Further Readings

66.1 Introduction

This is the last module of the course operating system. In this module we discuss the five major issues of operating system, viz., device management, memory management, file systems, directories and security.

66.2 Objectives

Gone through this module the students will learn the following topics.

* Devices
* Device controllers and derivers
* Interrupts
* Memory management with and without swapping
* Memory management with linked lists
* Memory management with the buddy system
* Virtual memory
* File : naming, structure, types, access, attribute and operations
* Directories
* Security
* Viruses

66.3 Keywords

Device management, memory management, files and directories systems, security.

66.4 Device Management

Device management is an important but relatively simple part of the overall operating system design. Most operating systems treat all devices such as disks, tapes, terminals, and printers in the same general manner, with special management for the processor and memory. Device management refers to the way these generic devices are handled. Generally, the operating system manage the allocation, isolation, and sharing of the devices according to desired policies. Even operating systems that do not support multiple processes incorporate device management. Normally, the source code for an operating system is provided with the hardware. If an end user organization adds a physical device to the computer, it must also add a device driver to the operating system. It is does not access to source code, it cannot recompile the operating system to include the new driver.

This limitation has led to the development of reconfigurable device drivers in modern operating systems. The device driver can be written (or acquired) by the end user, compiled, and added to the operating system without the user's having to recompile the operating system.

66.4.1 Devices

The I/O-devices of the von Neumann machine are used to place data into the primary memory and to store its ontents on a more permanent medium. Each I/O device is made up of logic to control detailed operation and the physical device itself. The device may be a storage device such as a magnetic disk or tape; a character device such as a terminal display, a mouse, or a keyboard; or a communication device such as a serial port connected to a modem or a network interface. There are many types of devices of each class, ranging from slow and inexpensive ones to fast and expensive ones. As a result, there is a wide range of interfaces to the devices used by software.

Every device uses a device controller to connect it to the computer's address and data bus. The controller provides a set of physical components that CPU instructions can manipulate in order to perform I/O operations. While the details of the interface differ among controllers, each controller provides the same basic interface. The operating system is required to hide the differences among controllers behind an interface common to all types of devics. Then, even though the speeds capacities, and operation details differ among controllers, the programmer need not know the details about each devicein order to use the device effectively. Programmers use an abstract

I/O paradigm implemented by the high-level difinition to write I/O code for a broad spectrum devices without knowing the details for any of the devices. The high-level machine provides operations to open/close the device, to read/write information. It also provides a general-purpose operation by which the programmer can invoke device-specific operatons. In turn, the high-level machine software implementation encapsulates device-specific aspects of each type of controller behind the common interface.

66.4.2 Device controllers

A device may require constant attention during its operation. It software were to control the device directly, it would need to ontinuously monitor the device's detailed operating status during the operation. Much of this attention is simply to observe status, provide detailed commands, and correct for minor errors. It is easy to incorporate into hardware algorithms to manage some of this mundane housekeeping. This is done as a device controller. The interface between devices and controllers is important to the industry that manufactures devices but not to software designers. This device-controller interface is concerned with the means by which devices manufactured by one vendor can be connected to a controller manufactured by another. The SCSI (small computer serial interface) is an example of such an interface. The interface between the bus and the controller is important to anyone wishing to attack a device to a computer so that the device can intemperate with other facilities in the machine. However, like the device-controller interface, this interface is conceptually transparent to software.

A higher-level interface defines the interaction between software and the controller via the controller's registers. This interface defines how software manipulates the conroller to cause the device to perform I/O operatons.

Figure 66.1 shows a software interface to a hardware controller. Devices effectively incorporate two flags as part of their status register interface, viz., busy and done. If both flags are set to 0 (or FALSE), the software can place a command in the command register to activate the device. After software has put data in one or more of the data registers for an output operation, the device is available for use. The presence of a new I/O command causes the controller to set the busy flag to TRUE and to begin the operation. An output operation causes the data in the data register to be written to the device, and an input operation causes a read operation to be sent to the device. The process can detect the status of the operation by checking the status register. When the I/O operation has completed (successfully or unsuccessfully), the controller clears the busy flag and sets the done flag. On completion of the read operation, data will have been copied into the data register from the device. On completion of a write operation, data will have been copied from the controller data register to the device. If the device flags have both been set to FALSE after a write operation, it is safe to write new data to the controller data register. When the software reads the data from the controller, the controller clears the done register to indicate the device is again ready for use. If the controller encountered an unrecoverable error, then when it terminated, it would set the error code field in the status register. Controllers must incorporate a small amount of memory to temporarily hold data

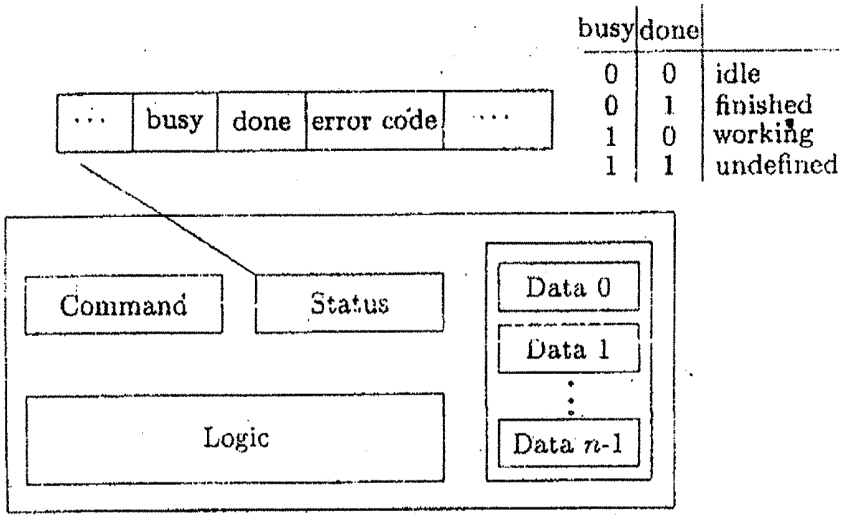


Figure 66.1: The device controller interface.

after it is read from the device but before it is retrieved, by the program in the CPU. Conversely, the memory must hold data waiting to be written to the device. This memory is called a (hardware) buffer and is sometimes used to increase the chances of overlapping the operation of the device and the CPU.

66.4.3 Device drivers

The device controller provides an interface to be used by the high-level machine software shown in Figure 66.2. This software implements the device management part of the operating system as a colection of device drivers. The device driver to controller interface may be relatively complex in the sense that there can be many parameters to set prior to starting the controller and many aspects of the status to check when each operation has completed. However, because of the "standardized" device driver to application software interface, application software can deal uniformly with different kinds of devices. Drivers for different devices implement a similar interface, so a higher-level machine need not be especially concerned with the details of the particular device (see Figure 66.2). The goal is to simplify the software interface to devices as much as possible, the reason being that differences between controllers produced by different manufacturers will be transparent to application software and even differences between different types of devices will be as transparent as possible. The presence of devices, controllers, and drivers enables two parts of a program to operate at the same time-one part using the CPU and the other part using the devices.

One goal of an operating system is to provide facilities that enable application programmes to take advantage of the physically separate devices so as to execute logically seprate operations simultaneously. Unfortunately, most programming langauges discourage such overlap through the semantics of high-level I/O instructions. The operating

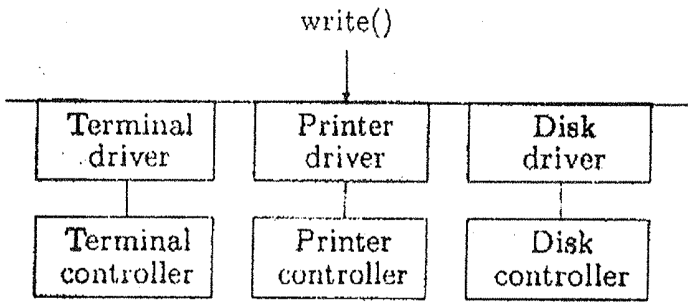


Figure 66.2: The device driver interface.

system can manage the processes so that CPU utilization by oen process overlaps the I/O operation of another. While this overlap does not increase the performance of either process, it makes the use of the hardware more effective, since two or more parts are used simultaneously rather than sitting idle.

The dervice driver is the operating system entity that controls CPU-I/O parallelism. A device can be started by the device driver, and the application program can continue operation in parallel with the device operation. A problem arises in providing some operating system mechanism to notify the part of the program that is executing on the CPU that the I/O operation has completed. Contemporary computer systems use interrupts, discussed next, to accomplish this notification.

66.4.4 Interrupts

The 'busy-done' flags must be tested by the driver software that determines when the device has completed its operation. When a process is using the CPU but is repeatedly testing the flags, a 'busy-wait' situation occurs. The process uses the CPU but is logically waiting for the device to complete its operation. A busy-wait wastes processor cycles that might be better used by other processes.

If the process does not use the processor to check the controller status continually, there will be a time period after which the device has completed operation but before the process checks and detects the completion. This period adds to the time the process is blocked waiting for the I/O operation to conclude. Clearly, the most effficient overlap between the device and the CPU could be obtained if the device would signal the processor as soon as it has compited the I/O operation. This would eliminate the need for busy-waits and minimize the idle time. The von Neumann architecture can be modified to implement this approach by incorporating device interrupts through which a device notifies the processor when it has completed an I/O operation. First, an interrupt request flag is incorporated into the hardware and the control unit is modified so that it checks the flag during each instruction fetch execute cycle. Conceptually, the hardware connects all device done flags to the interrupt request flag in the control unit using inclusive-OR logic. The signal from the hardware to the software, the interrupt, causes the processor to cease executing the sequence of instructions addressed by the PC and to branch to a new instruction

sequence whose address is stored in. When the branch occurs, the hardware saves the old PC location (address of the next instruction to be executed) of the interrupted process. The address of the interrupt handler is stored as an indirect address in memory when the machine is started. The interrupt handler is a part of the operating system that will be executed when any device completes its operation. So the application software need not continuously poll the device to detect when it has completed.

66.5 Memory Management

Memory management is a very important part of operating system. The part of the operating system that manages memory is called the **memory manager.** Its task is to keep track of which parts of memory are in use and which parts are not in use, to allocate memory to processes when they need it and deallocate it when they are done, and to manage swapping between main memory and disk when main memory is not big enough to hold all the processes.

66.5.1 Memory management without swapping or paging

Memory management systems can be divided into two classes, such as (i) the processes move back and forth between main memory and disk during execution (swapping and paging); and (ii) the processes that do not follow this technique. The latter are simpler, so we will study them first and then we examine swapping and paging.

Monoprogramming without swapping or paging

The simplest possible memory management scheme is to have just one process in memory at a time, and to allow that process to use all of memory, and this is called **monoprogramming.** The user loads the entire memory with a program from disk or tape, and it takes over the whole machine. Although this approach was common up until about 1960, it is not used any more, not even on inexpensive home computers, mostly because it implies that every process must contain within it a device driver for each I/O device it uses.

The common used technique on simple microcomputers is shown in Figure 66.3. The memory is divided up between the between the operating system and a single user process. The operating system may be at the bottom of memory in RAM (shown in Figure 66.3(a)), or it may be in ROM at the top memory (shown in Figure 66.3(b)) and the rest of the operating system in RAM at the bottom of memory (see Figure 66.3(c)). The program in the ROM is called the BIOS (Basic Input Output System).

When the system is organized in this manner, only one process at a time can be running. The user types a command on the terminal, and the operating system loads the requested program from disk into memory and executes it. When the process finishes, the operating sysem types of prompt character on the terminal and then waits for a command from the terminal to load another process, overwriting the first one.

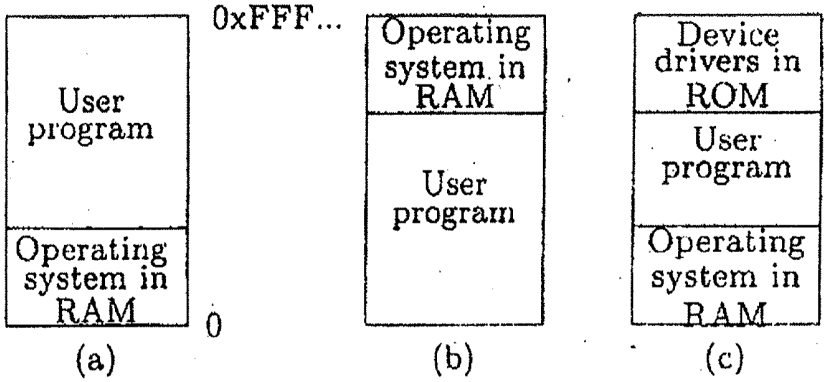


Figure 66.3 : Three ways of Operating memory with an operating system and one user process.

66.5.2 Multiprogramming and memory usage

The monoprogramming is sometimes used on small computers and also it is used on larger computer with multiple users. The multiprogramming technique is easier to program an application by splitting up into two or more processes. Another motivation is that large computers often vide interactive service to several people simultaneously, which requires the ability to have more than one process in memory at once in order to get reasonable performance. Loading a process, running it for 100 msec, and then spending a few hundred milliseconds swapping it to disk is inefficient. But if the quantum is set too much above 100 msec, the response time will be poor.

Another reason for multiprogramming a computer (also applicable to batch system) is that most processes spend a substantial fraction of their time waiting for disk I/O to complete. It is common for a process to sit in a loop reading data blocks from a disk file and then doing some computation on the contents of the blocks read. If it takes 40 msec to read a block, and the computation takes 10 msec, with monoprogramming the CPU will be idle waiting for the disk 80 percent of the time.

66.5.3 Swapping

In a batch system the organization of memory into fixed partitions is simple and effective. As long as enough jobs can be kept in memory to keep the CPU busy all the time, there is no reason to use anything more complicated. With timesharing, the situation is different: there are normally more users than there is memory to hold all their processes, so it is necessary to keep excess processes on disk. To run these processes, they must be brought into main memory. Moving processes from main memory to disk and back is called **swapping.**

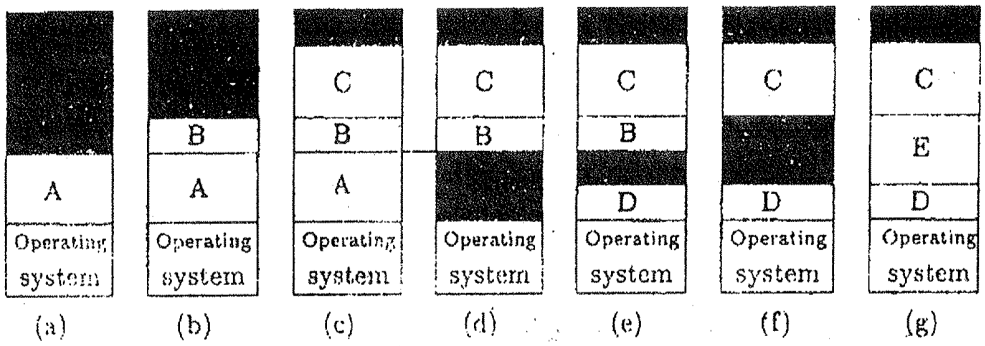


Figure 66.4: Memory allocation changes as process come into memory and leave it. The shaded regions are unused memory.

66.5.4 Multiprogramming with variable partitions

Actually, a swapping system is based on fixed partitons. Whenever a process blocked, it could be moved to the disk and another process brought into its partition from the disk. In practice, fixed partitions are unattractive when memory is scarce because too much of it is wasted by programs that are smaller than their partitions. A different memory management algorithm is used instead. It is known as **variable partitions.**

When variable partitions are used, the number and size of the processes in memory vary dynamically. Figure 66.4 shows how variable partitions work. Initially only process *A* is in memory. Then processes *B* and *C* are created or swapped in from disk. In Figure 66.4(d) *A* terminates or is swapped out to disk. Then *D* comes in and *B* goes out. Finally *E* comes.

The main difference between the fixed partitions and the variable partitions is that the number, location, and size of the partitions vary dynamically in the latter as processes come and go, whereas they are fixed in the former. The flexibility of not being tied to a fixed number of partitions that may be too large or too small improves memory utilization but it also complicates allocating and deallocating memory, as well as keeping track of it.

It is possible to combine all the holes into one big one day moving all the processes downward as far as possible. This technique is known as **memory compaction.** It is usually not done because it requires a lot of CPU time.

If, however, processes data segments can grow, for example, by dynamically allocating memory from a heap, as in many programming languages, a problem occurs whenever a process tries to grow. If a holes is adjacent to the process, it can be allocated and the process allowed to grow into the hole. On the other hand, if the process is adjacent to another process, the growing process will either have to be moved to a hole in memory large

at 128K and one at 192K. The block at 128 K is allocated to the process, marked as B. The third request is for 80K.

Now let us see what happens when a block is returned. Imagine that 128K block A (of which only 70K is used) is freed at this point. It just goes on the free list for 128K blocks at this point. Now a 60K block is needed, so a check is made to see if any sufficiently large blocks are available. Thc 64K block located at address 192K will do so it is allocated. Now block B is returned. At this point we have a 128K block at 0 and a 64K at 128K that are free. No merging is possible yet. Note that even if the 128K at 0 had been split into a 64K block at 0 that was in usc and a free block at 64K no merging could occur. When block D is returned, we can reconstruct the 256K block at address 0. Finally, when block C is returned, we return to the initial configuraton of a single hole of 1M.

Buddy systems have an advantage over algorithms that sort blocks by size but not necessarily at addresses that are multiples of the block size. The advantage is that when a block of size 2^k bytes is freed, the memory manager has to search only the list of 2^k holes to see if a merge is possible. With other algorithms that allow memory blocks to be split in arbitrary ways, all the hole lists must be searched. The result is that the buddy system is fast.

But, it has a disadvantages in terms of memory utilization. The problem comes from the fact that all requests must be rounded up to a power of 2. A 35K process must be allocated 64K. The extra 29K is just wasted. This form of overhead is known as **internal fragmentation** because the wasted memory is internal to the allocated segments. In Figure 66.4 we have holes between the segements, but no wasted space within the segments. This form of waste is called **external fragmentation or checkerboarding.**

55.5.7 Virtual memory

In old days the programmers were fetched some problems with programs that were too big to fit in the available memory. The solution usually adopted was to split into pieces, called **overlays.** Overlay 0 would start running first. When it was done, it would call another overlay. Some overlay systems were highly complex, allowing multiple overlays in memory at once. The overlays were kept on the disk and swapped in and out of memory by the operating system.

Although the actual work of swapping overlays in and out was done by the system, the work of splitting the program into pieces had to be done by the programmer. Splitting up large programs into small, modular pieces was time consuming and boring. It did not take long before someone thought of way to turn the whole job over the computer. The method that was devised has come to be known as **virtual memory.** The basic idea behind virtual memory is that the combined size of the program, data, and stack may exceed the amount of physical memory availble for it. The operating system keeps those parts of the program currently in use in main memory, and the rest on the disk. For example, a 1M program can run on a 256K machine by carefully choosing which 256K to keep in memory at each instant, with pieces of the program being swapped between disk and memory as needed.

Virtual memory can also work in a multiprogramming system. For example, eight 1M programs can each be allocated a 256K partition in a 2M memory, with each program operating as though it has its own, private virtual machine. In fact, virtual memory and multiprogramming fit together very well. While a program is waiting for part of itself to be swapped in, it is waiting for I/O and cannot run, so the CPU can be given to another process.

66.6 File Systems

When an application software is running then its need to store and retrieve information. While a program is running, it can store a limited amount of information within its own address. However, the storage capacity is restricted to the size of the virtual address. For some applications this size is adequate, but for others, such as airline, banking, or record keeping in large companies, it is far too small.

The three essential requirements for long-term information storage are :

(i) It must be possible to store a very large amount of information.

(ii) The information must survive the termination of the process using it.

(iii) Multiple processes must be able to access the information simultaneously.

The solution to all these problems is to store information on disks and other external media in units called **files.** Processes can then read them and write ones if need be. Information stored in files must be **persistent,** that is, not be affected by process creation and termination. A file should only disappear when its owner explicitly removes it. Files are manages by the operating system. The part of the operating system dealing with files is known as the **file system.**

66.6.1 File naming

Files are an abstraction mechanism. They provide a way to store information on the disk and read it back later. This must be done in such a way as to shield the user from the details of how and where the information is stored, and how the disks actually work. Probably the most important characteristic of any abstraction mechanism is the way the objects being managed are named, so we will start our examination of file systems with the subject of file naming. When a process creates a file, it gives the file a name. When the process terminates, the file continues to exist, and can be accessed by other processes using its name. The exact rules for file naming vary from system to system, but all operating systems allow strings of one to eight letters as legal file names.

Some operating system distinguish between upper case letters and lower case letters. For example, UNIX differentiate between upper case and lower case letters as file name while MS-DOS do not. Thus in a UNIX system, the files firstprog, FirstProg, FIRSTprog are consider as distinct files.

Many operating systems support two-part file names, these two parts are separated by a period, e.g., prog1.c. The part following the period is called the **file extension,** and usually indicates the type of the file. In MS-DOS, file names are 1 to 8 characters, plus an optional extension of 1 to 3 characters. In UNIX, the size of the extension depends on the user, and a file may have two or more extensions, e.g., prog1.c.zip, where.zip is commonly used to indicate that the file (prog1.c) has been compressed. Some typical file extensions and their meanings are shown in Table 66.1.

Extension	Meaning
prog1.c	C source program
prog1.for	Fortran source program
prog1.bac	Backup file
prog1.bas	BASIC source program
prog1.bin	Executable binary program
prog1.dat	Data file
prog1.doc	MS-Word document file
prog1.hlp	Text for HELP command
prog1.lib	Library of.obj files used by the linker
prog1.obj	Object file (compiler output, not yet linked)
prog1.pas	Pascal source program
prog1.tex	Input text for the TEX formatting program
prog1.txt	General text file
prog1.pdf	Document file prepared by Acrobat Writer
prog1.htm	Hyper text file

Tale 66.1 : Some typical file extensions.

In some, cases, the file extensions are just conventions, and are not enforced in any way. A file named prog1.text is probably some kind of text file, but that name is more to remind the owner than to convey any specific information to the computer. On teh other hand, a C compiler may actually insist that the files it is to compile in c, and it may refuse to compile them if they do not. Conventions like this are especially useful when the same program can handle several different kinds of files. The C compiler, can be given a list several files to compile and link together, some of them C files and some of assembly language files. The extension then becomes essential for the compiler tell which are which.

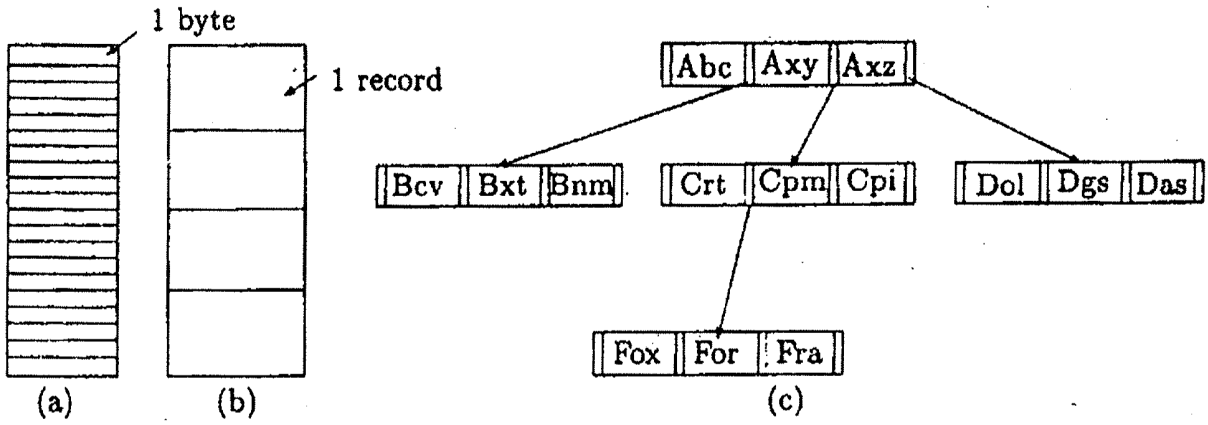


Figure 66.7: Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.

66.6.2 Structure f a file

The files are structured in many different ways. Three common structured are shown in Figure 66.7. The file in Figure 66.7 (a) is an unstructured sequence of bytes. In effect, the operating system does not know or care waht is in the file. All it sees bytes. Any meaning must be imposed by user-level programs. Both UNIX and DOS use this approach.

The byte sequences technique provides the maximum flexibility. The programs can put anything in files, and name them in any convenient way. The operating system does not help, but it also does not get in the way.

The first step up in structure is shown in Figure 66.7(b). In this model, a file is a sequence of fixed-length records, each with some internal structure. Central to the idea of a file being a sequence of records is the idea that the read operation returns one record and the write operations overwrites a appends one record. In the age of punched card, many operating systems organized their file systems by considering of 80-character records. These systems also supported files of 132-character records, which wre intended for the line printer (which in those days had 132 columns). Programs read input in units of 80 characters and wrote it in units of 132 characters and remanning 52 could be spaces. The most recent system that uses files as fixed-length records is CP/M. It uses a 128-character record. With the advent of CRT terminals, which may variable length lines, the concept of fixed-length records is lost.

The third kind of file structure is shown in Figure 66.7 (c). In this organizaton, a file consists of a tree of records, not necessarily all the same length, each containing key field in a fixed position in the record. The tree is sorted on the key field, to allow rapid searching for a particular key. The new records can be added to the file, with the operating system, and not the user, deciding where to place them. This type of file is clearly quite different from the unstructured byte streams used in UNIX and MS-DOS. It is widely used on the large mainframe computers used in commercial data processing.

66.6.3 File types

The operating systems support several types of files. UNIX and MS-DOS, have regular files, directories, character special files, and block special files, and block special files. **Regular files** are the ones that contain user information. All the files of Figure 66.7 are regular files. **Directories** are system files for maintaining the structure of the file system. **Character special files** are related to input/output and used to model serial I/O devices such as terminals, printers, and networks. **Block special files** are used to model disks.

Regular files are generally either ASCII files or binary files. ASCII files consist of lines of text. In some systems each line is terminated by a carriage return character. In others, the line feed character is used. Some times both are required. Lines need not all be of the same length. The great advantage of ASCII files is that they can be displayed and printed as is, and they can be edited with an ordinary text editor. Furthermore, if large numbers of programes use ASCII files for input and output, it is easy to connect the output of one program to the input of another, as in shell pipelines.

The files other than ASCII are binary files. Listing them on teh printer gives an incomprehensible listing full of what is apparently random junk. Usually they have some internal structure. For example, we consider a simple executable binary file of UNIX operating system (see Figure 66.8(a)). Technically the file is just a sequence of bytes, the operating system will only execute a file if it has the proper format. It has five sections such as header, text, data, relocation bits and symbol table. The header starts with a **magic number,** identifying the file as an executable file. Then come 16-bit integers giving the sizes of the various pieces of the file, the address it starts execution at and some bits. Following the header are the text and the data of the program itself. These are loaded into memory and relocated using the relocated bits. The symbol table is used for debugging.

Another type of binary file is archieve. It consists of a collection of library procedures (modules) compiled but not linked. Each one is prefaced by a header telling its name, creation date, onwer, protection code and size. Like the executable file, the module headers are full of binary numbers. All operating system must recognize one file type, their own executable file, but some recognize more.

66.6.4 File access

The old operating systems provided only one kind of file access viz., sequential access. In these systems, a process could read all the bytes or records in a file in order, starting at the beginning, but could not skip around and read them out of order. Sequential files can be rewound, however, so they can be read as often as needed. Sequential files are convenient when the storage medium is magnetic tape, rahter than disk. When disks came into use for storing files, it became possible to read the bytes or records of a file out ot order, or to access records by key, rather than by position. Files whose bytes or records can be read in any order are called **random access files.** Random access files are used in many applications, e.g., data base systems. If an airline customer calls up and wants to reserve a seat on a particular flight, the reservation program must be able to access the record for the flight

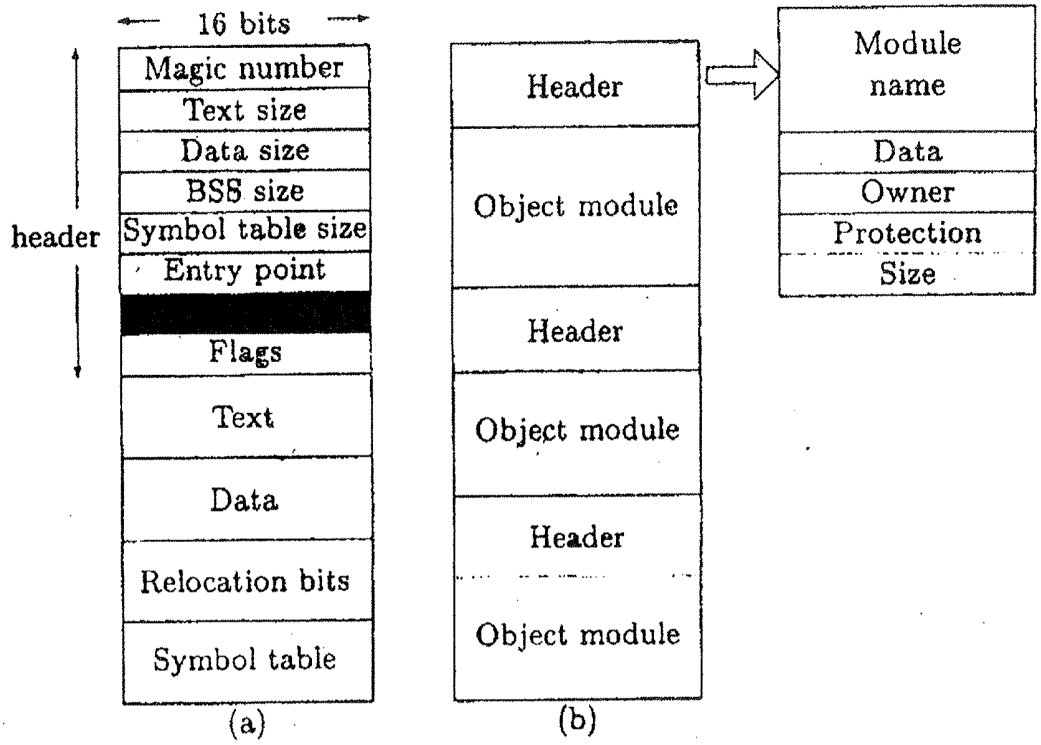


Figure 66.8: (a) An executable file, (b) An archive.

without having to read the records for thousands of other flights first. In some older mainframe operating systems, files are classified as either sequential or random access when they are created. This allows the operating system to use different storage techniques for the two classes. Modem operating systems do not make this distinction. All their files are automatically random access.

66.6.5 File attributes

Every file has a name and its data. Also, all operating systems associate other information with each file, for example, the date and time the file was created and the file's size. We will call these extra items the file's attributes. The list of attributes varies considerably from system to system. The Table 66.2 shows some of the possibilities, but other ones also exist. No existing system has all of these, but each one is present in some system. The first four attributes relate to the file's protection and tell who may access it and who may not.

In some systems the user must present a password to access a file, in which case the password must be one of the attributes. The flags are bits or short fields that control or enable some specific property. For example, hidden files are not visible in listings of all the files. The archive flag is a bit that keeps track of whether the file has been backed up. The backup program clears it, and the operating system sets it whenever a file is changed. In this

Field	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	Id of person who created the file
Owner	Current owner
Read-only flag	0 for read/write, 1 for read only
Hidden flag	0 for normal, 1 for do not display in listings
System flag	0 for normal file, 1 for system file
Archieve flag	0 has been backed up, 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file, 1 for binary file
Random access flag	0 for sequential access only, 1 for random access
Temporary flag	0 for normal, 1 for delete on process exit
Lock flags	0 for unlocked, nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time file was created
Time of last access	Date and time file was last accessed
Time of last change	Date and time file was last changed
Current size	Number of bytes in the file
Maximum size	Maximum size file may grow to

Table 66.2 : Some possible file attributes.

way, the backup program can tell which files need backing up. The temporary flag allows a file to be marked for automatic deletion when the process that crated it terminates. The record length, key position, and key length fields are only present in files whose records can be looked up using a key. They provide the information required to find the keys. The various times keep track of when the file was created, most recently accessed and most recently modified. These are useful for a variety of purposes. For example, a source file that has been modified after the creation of the corresponding object file needs to be recompiled. These fields provide the necessary information.

The current size tells how big the file is at present. Some mainframe operating systems require the maximum size to be specified when the file is created. Minicomputer and personal computer systems are clever enough to do without this item.

66.6.6 File operations

Different systems provide different operations to allow storage and retrieval. Below is a discussion of the most common system calls relating to files.

* Create. The file is created with no data. The purpose of the call is to announce that the file is coming and to set some of the attributes.
* Delete. If the file has no need in future then, it has to be deleted to free the disk space. There is always a system call for thsi purpose. In addition, some operating systems automatically delete any file that has not been used in some specified days.
* Open. Before using a file, a process must open it. The purpose of the open call is to allow the system to fetch the attributes and list of disk addresses into main memory for rapid access on subsequent calls.
* Close. When all the accesses are finished, the attributs and disk addresses are no longer needed, so the file should be closed to free up internal table space. Many systems encourage this by imposing a maximum number of open files on processes.
* Read. Data are read from file. Usually the bytes come from the current position. The caller must specify how much data is needed and must also provide a buffer to put it in.
* Write. Data are written to the file, usually at the current position. If the current position is the end of the file, the file's size increases. If the current position is in the middle of the file, existing data are overwritten and lost foreever.
* Append. This calls is a restricted form of write. It can only add data to the end of the file. Systems that provide a minimal set of system calls do not generally have append, but many systems provide multiple ways of doing the same thing, and these systems sometimes have append.
* Seek. For random access files, a method is needed to specify from where to take the data. One common approach is a system call, seek, that repositions the pointer to the current position to a specific place in the file. After this call has completed, data can be read from, or written to, that position.
* Get Attributes. Processes often need to read file atrributes to do their work. For example, the UNIX make program is commonly used to manage software development projects consisting of multiple source files. When make is called, it examines the modification times of all the source and object files, and arranges for the minimum number of compilations required to bring everything up to date. To do its job, it needs to look at the attributes, specifically, the modification times.

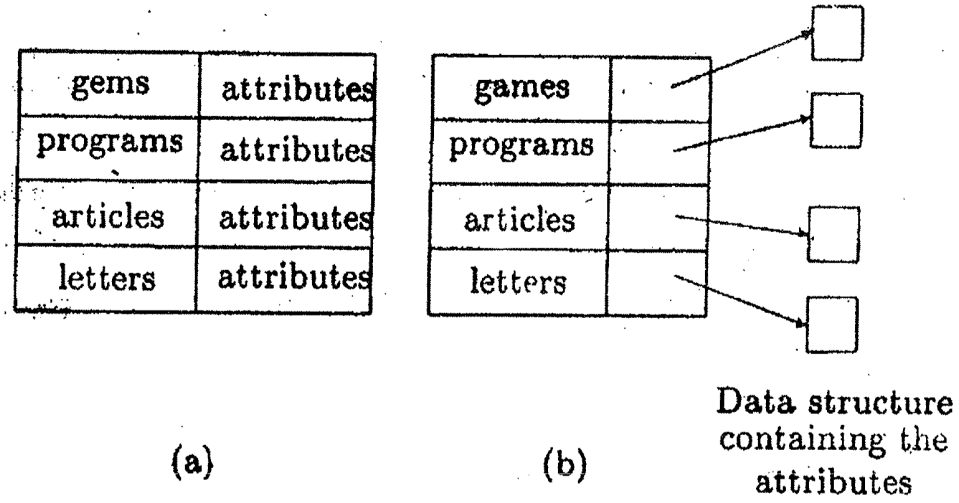


Figure 66.9 : Directories, (a) Attributes in the directory entry, (b) Attributes elsewhere.

* Set Attributes. Some of the attributes are user-sellable and can be changed. After the file has been created. This system call makes that possible. The protection mode information is an obvious example. Most of the flags also fall in this category.

* Rename. It frequently happens that a user needs to chagne the name of an existing file. This sytem call makes that possible. It is not always strictly necessary, because the file can always be copied to a new file with the new name, and the old file then deleted. Usually this is good enough.

66.7 Directories

To properly organized the files, the file system normally provides directories, which in many systems, are themselves files. Here we discuss directories, their organization, their properties, and the operations that can be performed on them.

66.7.1 Hierarchical directory systems

A directory typically contains a number of entries. One possible way is shown in Figure 66.9(a), in which each entry contains the file name, the file attributes and the disk addresses where the data re stored. The other possible is shown in Figure 66.9(b). Here a directory entry holds the file name and a pointer to another data strucutre where the attributes and disk addresses are found. Both of these systems are commonly used.

When a file is opened, the operating system searches its directory until it finds the name of the file to be opened. It then extracts the attributes and disk addresses, either directly from the directory entry or from the ddata structure pointed to, and puts them in a table in main memory. All subsequent references to the file use the information

in main memory. The number of directories varies from system to system. The simplest design is for the system to maintain a single directory containing all the files of all the users. If there are many users, and they choose the same file names, conflicts and confusion will quickly make the system unworkable.

66.7.2 Path names

When the file system is organized as a directory tree, some way is needed for specifying file names. Two different methods are commonly used. In the first method, each file is given an absolute path name consisting of the path from the root directory to the file. As an example, the path/ usr/software/cpp means that he root directory contains a subdirectory usr, which in turn contains a subdirectory software, which contains the file cpp. Absolute path names always start at the root directory and are unique. In UNIX the components of the path are separated by /. In MSDOS the separator is \. No matter which character is used, fi the first character of the path name is the separator, then the path is absolute.

The other kind of name is the relative path name. This is used in conjunction with the concept of the working directory (also called the current directory). A user can designate one directory as the current working directory, in which case all path names not beginning at the root directory are taken relative to the working directory. For example, if the current working directory is/usr/software, then the file whose absolute path is/urs/software/cpp can be referenced simply as cpp.

Most operating systems that support a hierarchical directory system have two special entries in every directory, "." and "..", generally pronounced "dot" and "dotbot". Dot refers to the current directory; dotdot refers to its parent.

66.7.3 Directory operations

The allowed system calls for managing directories exhibit more variation from system to system than those relating to files do.

* Create. A directory is created. It is empty except for dot and dotdot, which are put there automatically by the system.
* Delete. A directory is deleted. Only an empty directory can be deleted. A directory containing only dot and dotdot is considered empty as these usually cannot be deleted.
* Opendir. Directories can be read. For example, to list all the files in a directory, a listing program opens the directory to read out the names of all the files it contains. Before a directory can be read, it must be opened.
* Closedir. When a directory has been read, it should be closed to free up internal table space.

* Readdir. This call returns the next entry in an open directory. Formerly, it was possible to read directories using the usual read system call, but that approach has the disadvantae of forcing the programmer to know and deal with the internal strucure of directories. In contrast, readdir always returns one entry in a standard format, no matter which of the possible directory structures is being used.
* Rename. In many respects, directories are just like files, and can be renamed the same way files can be.
* Link. Linking a technique that allows a file to appear in more than one directory. This system call specifies an existing file and a path name, and creates a link from the existing file to the name specified by the path. In this way, the same file may appear in multple directories.
* Unlink. A directory entry is removed. If the file being unlinked is only present in one directory (the normal case), it is removed from the file system. If it is present in multiple directories, only the path name specified is removed. The others remain.

66.8 Security

Some files may contain very important / confidential document. Protecting this information against unauthorized usage is therefore a major concern of all file system. Here we discuss a variety of issues concerning security and protection.

66.8.1 The security environment

The terms "security" and "protection" often used interchangeably. It is frequently useful to make a distinction between the general problems involved in making sure that files are not read or modified by unauthorized persons, which include technical, managerial, legal, and political issues on the one hand, and the specific operating system mechanisms used to provide security, on the other. To avoid confusion, we will use the term security to refer to the overall problem, and the term protection mechanisms to refer to the specific operating system mechanisms used to safeguard information in the computer. The boundary between them is not well defined.

Security has many facets. Two of the more important ones are data loss and intruders. Some of the common causes of data loss are :

(i) *Acts of God.* Fires, floods, earthquakes, wars, riots, or rats gnawing tapes of floppy disks.

(ii) *Hardware or software errors.* CPU malfunctions, unreadable disks or tapes, telecomunication errors, program bugs.

(iii) *Human errors.* Incorrect data entry, wrong tape or disk mounted, wrong program run, lost disk or tape.

Most of these can be dealt with by maintaining adequate backups, preferably far away from the original data.

Intruders come in two varieties. Passive intruders just want to read files they are not authorized to read. Active intruders are more malicous; they want to make unauthorized changes to data. When designing a system to be secure against intruders, it is important to keep in mind the kind of intruder one is trying to protect against. Some common categories are :

(i) Casual use by nontechnical users. Many people have terminals to timesharing systems on their desks, and human nature being what it is, some of them will read other people's electronic mail and other files if no barriers are placed in the way.

(ii) Snooping by insiders. Students, system programmers, operators, and other technical opersonnel often consider it to be a personal challenge to break the security of the local computer system. They often are highly skilled and are willing to devote a substantial amount of time to the effort.

(iii) Determined attempt to make money. Some bank programmers have attempted to break into a banking system to steal from the bank. Schemes have varied from changing the software to truncate rather than round interest, keeping the fraction of a cent for themselves, to siphoning off accounts not used in years, to blackmail.

(iv) Commercial or military espionage. Espionage refers to a serious and well-funded attempt by a competitor or a foreign country to steal programs, trade secrets, patents, technology, circuit designs, marketing plans, and so forth. Often this attempt will involve wiretapping or even erecting antennas directed at the computer to pick up its electromagnetic radiation.

Another aspect of the security problem is **privacy**, that is, protecting individuals from misuse of information about them. This quickly gets into many legal and moral issues.

66.8.2 The Internet worm

The greatest computer security violation of all time began in the evening of November 2, 1988 when Robert Tappan Morris (a student of Cornell University) released a worm program into the Internet. This action brought down thousands of computers at universities, corporations and government laboratories all over the world before it was tracked down and removed. It also started a controversy that has not yet died down.

Technically, the worm consisted of two programs, the bottstrap and the worm proper. The bootstrap was 99 lines of C program named 11.c. It was compiled and executed on the system under attack. Once running, it connected to the machine from which it came, uploaded the main worm, and executed it. After going to some

trouble to hide its existence, the worm then looked through its new host' routing tables to see what machines that host was connected to, and attempted to spread the bootstrap to those machines.

66.8.3 Generic security attacks

The usual way to test a system's security is to hire a group of experts, known as **tiger teams** or **penetration teams,** to see if they can break in. Below we have a list of some common attacks that are often successful.

(i) Request memory pages, disk space or tapes and just read them. Many systems do not erase them before allocating them, and they may be full of interesting information written by the previous owner.

(ii) Try illegal system calls or legal system calls with illegal parameters or even legal system calls with legal but unreasonable parameters.

(iii) Start logging in and then hit del, rubout or break halfway through the login sequence. In some systems, the password checking program can easily be confused.

(iv) Try modification complex operating system structures kept in user space. In many systems, to open a file, the program builds a large data structure containing the file name and many other parameters and passes it to the system. As the file is read and written, the system sometimes updates the structure itself. Changing these fields can wreak havoc with the security.

(v) Spoof the user by writing a program that types "login" on the screen and go away. Many users will walk up to the terminal and willingly tell it their login name and password, which the program carefully records for its evil master.

(vi) Convince a system programmer to change the system to skip certain vital security checks for any user with your login name. This attack is known as a **trapdoor.**

66.8.4 Viruses

A special category of attack is the computer virus, which has become a major problem for many computer users. A virus is a program fragment that is attached to legitimate program with the intention of infecting other programs. It differes from a worm only in that a virus piggybacks on an existing program, whereas a worm is a complete program in itself. Viruses and worms both attempt to spread themselves and both can do severe damage.

A typical virus works as follows. The person writing the virus first produces a useful new program, often a game for MS-DOS. This program contains the virus code hidden away in it. The game is then uploaded to a public bulletin board system or offered for free or for a modest price on floppy disk. The program is then adverised, and

people begin downloading and using it. Constructing a virus is not easy, so the people doing this are invariably quite bright, and the quality of the game or other program is often excellent.

When the program is started up, it immediately begins examining all the binary programs on the hard disk to see if they are already infected. When an uninfected program is found, it is infected by attaching the virus code to the end of the file, and replacing the first instruction with a jump to the virus. When the virus code is finished executing, it executes the instruction that had previously been first, and then jumps to the second instruction. In this way, every time an infected program runs, it tries to infect more programs.

In addition to just infecting other programs, a virus can do other things, such as erasing, modifying, or encrypting files. One virus even displayed an extortion note on the screen, telling the user to send 500 dollars ill cash to a post office box in Panama or face the permanent loss of his data and damage to the hardware. It is also possible for a virus to inject the hard disk's boot sector, making it impossible to boot the computer. Such a virus may ask for a password, which virus writer will supply in exchange for money.

Virus problems are easier to prevent than to cure. The safest course is only buy shrink-wrapped software from respectable sotres. Uploading free software from bulletin boards or getting pirated copies on floppy disk is asking for trouble. Commercial antivirus packages exist, but some of these work by just looking for specific known viruses.

A more general approach is to first reformat the hard disk completely, including the boot sector. Next, install all the trusted software and compute a checksum for each file. The algorithm does not matter, as long as it has enough bits. Store the list of (file, checksum) pairs in a safe place, either offline on a floppy disk, or online but encrypted. Starting at that point, whenever the system is booted, all the checksums should be recomputed and compared to the secure list of original checksums. Any file whose current checksum differs from the original one is immediately suspect. While this approach does not prevent infection, it at least allows early detection. Infection can be made more difficult if the directory where binary programs reside is made unwritable for ordinary users. This technique makes it difficult for the virus to modify other binaries. Although it can be used in UNIX, it is not applicable to MS-DOS because the latter's directories cannot be made unwritable at all.

66.9 Module Summary

In this module we discussed briefly about the some important features of operating systems. At the beginning we introduced devices and their management, device controllers, device drivers and interrupts. Next we discussed about memory management techniques without swapping or paging. A very brief introduction is provided about multiprogramming and memory usage. Two memory management methods, viz., linked lists and buddy system are discussed here. In the next section we introduced the concept of files, their naming rules, structure, types, attributes,

etc. The different types of file operations are also provided here. An introduction of the directory systems, path names and file operations are given. A very brief concept of file security is introduced in this module. An exercise and references are given at the end of this module.

66.10 Self Assessment Questions

1. What do you mean by devices? How the devices are managed by computer? Explain.
2. Explain about the device driver and the interrupts.
3. What do you mean by device controller? Explain device controller interface with diagram.
4. Explain memory management technique without swapping or paging.
5. What do you mean by multiprogramming? How memory is organized in a multiprogramming system.
6. What is swapping? What is the use of it?
7. Explain memory management technique with linked lists. What are the advantages and disadvantages of this method.
8. Explain memory management technique with buddy system. What are the advantages and disadvantages of buddy system.
9. A minicomputer uses the buddy system for memory management. Initially it has one block of 256K at address 0. After successive requests for 5K, 25K, 35K and 20K come in, how many blocks are left and what are their sizes and addresses?
10. Write a short note on virtual memory.
11. What are the rules for naming a file? Which types of files are used in an operating system?
12. How a file can be organized in compuer memory? Explain the methods.
13. Write short notes on file access and file attributes.
14. Explain the following terms in connection with files :

    open, close, append, create, delete, read, write, seek, rename, get attributes and set attributes.
15. What do you mean by directory? What are the use of it? Explain about the hierarchical directory systems.
16. What do you mean by absolute path name, relative path name and working directory.

17. Explain the following directory operations :

    create, delete, opendir, closedir, readdir, rename, link, unlink.

18. What do you mean by security of a file. What are the causes for losing data from a file.

19. What do you mean by intruders? Explain the different kinds of intruders.

20. Explain about different types of security attacks.

21. Write a note on computer viruses.

66.11 Suggested Further Readings

1. A.S. Tanenbaum, Modern Operating System, Prentice Hall of India.
2. Gary Nutt, Operating Sysems : A modern perspective, Addison-Wesley.
3. A.S. Godbole, Operating Systems, Tata McGraw-Hill, New Delhi.
4. H.M. Deitel, An Introduction to Operating Systems, 2e, Pearson Education Asia.
5. Charles Crowley, Operating Systems, Tata McGraw-Hill Edition.
6. William Stallings, Operating Systems, 2e, Prentice Hall of India.

--- 0 ----

M.Sc. Course
in
Applied Mathematics with Oceanology
and
Computer Programming

PART-II

Paper-VI Group–B

Module No. - 67

LAB. 1 : PROGRAMMING IN C++- I

Structure :

67.1 Introduction

67.1 Objectives

67.3 What is C++?

67.4 Keyboad and Screen I/O

67.5 Predefined Manipulators

67.6 Structure and Classes

- 67.6.1 Creating objects
- 67.6.2 Defining member functions

67.7 Array of class objects

67.8 Constructor and Destructor

67.9 Objects as Function Arguments

67.10 Arrays as Class Member Data

67.11 Module Summary

67.12 Self Assessment Questions

67.13 Suggested Further Readings

67.1 Introduction

C++ is an object oriented programming (OOP) language. C++ was developed by Bjarne Stroustrup at AT & T Bell Laboratories. Since the class was a major addition to the original C language, Stroustrup called the new language 'C with classes'. However, later in 1983, the name was changed to C++. The name C++ comes from the C increment operator ++, that is, C++ is an incremented version of C.

Here we assumed that the students are familiar with the language C. The total C++ language (practical) is divided into three modules. A very small introduction of visual C++ is incorporated at the last module.

67.2 Objectives

Gone through this module the students will learn the following topic:

* Input/Output of C++
* Predefined manipulators
* C++ structure and classes
* Use of class with array
* Constructors and destructors.

67.3 What is C++?

C++ is a superset of C. Most of the features of C are available in C++. Almost all C programs are also C++ programs. However, there are a few minor differences that will prevent a C program to run under C++ compiler.

Thus C is the base language of C++ and it was chosen as base language for the following reasons.

(i) It is versatile and relatively low-level.

(ii) It is adequate for most system programming task.

(iii) It runs everywhere and on every thing, and

(iv) It fits in the UNIX programming environment.

Differences between C and C++

The following features are added in C++.

(i) OOP constructs

(ii) Enumeration names

(iii) Function prototyping

(iv) Initialization fo default parameter values

(v) Inline functions

(vi) Block declarations

(vii) Function and operator overloading

(viii) // comments

(ix) New libraries

C++ adds a complete set of OOP constructs to C. These are

(i) Classes

(ii) Derived classes

(iii) Constructors and destructors

(iv) Messages

(v) Friend functions

(vi) Virtual functions

(vii) Public, protected and private data.

Enumeration names

This feature is extension of ANSI C enum that allows avalues to be assigned to a group enumeration. For example,

enum month_of_year {jan=0, feb=100, mar+50};

C does not allow the values to be assigned to names. In C++

enum month_of_year {jan, feb, mar};

Will assigned the values as ja=0, feb=1, mar=2.

const

const is available both in C and C++. In C, const defines a numeric literal that is substituted whenever the constant is used in the source code. const effectively replaces #define in pre-ANSI C. In the other hand, C++ uses const as a declaration modifier to mean that a value cannot be changed. Thus when a constant is defined a variable is

initialized to a value that cannot be changed. Using const in a functon prototype prevent the argument from being changed by the called function. The const in C++ is more powerful than the ANSI C const.

Function prototyping

The function prototyping feature of C++ is the extension of C's feature. Function prototyping in C specify only the tyeps of the arguments. But, in C++ it also specified default values for the arguments and also specify a variable number of arguments.

Inline functions

Inline functions can improve execution efficiently of C++ programs. Practically, many very small functions are used in a program. These functions may be as short as one or two lines. When these functions are used frequently, considerable execution overhead is generated by the calls. C++ substitutes the body of the function for the function cal of functions declared as inline. Program size may significantly increase if large functions are declared inline and called in many locations.

Block declarations

In C, variables are declared at the beginning of the function. But, in C++ we can declared variables, objects at any position in the function. Also, variables may be declared within a block in C++. For example,

```
for(int i=0, i< 10; i++)
```

declares *i* is an int for the duration of the block associatd with the for loop.

Function overloading

Function and operator overloading is the important part of C++. A simple way to describe function overloading is the capability of redefining existing operators or functions. << is a good example of an overloaded operators. << is normally used to shift bits left. It is overloaded for the cout function t mean 'write to'. The compiler looks at the types of arguments to determine which meaning to use.

Comments

Another C++ coding convenience is the use of // to add a comment to the end of a line. / *...* /may still be used to delimit comments.

New libraries

The C++ language includes a host of new function libraries. For the most part, the normal C libraries, such as, jsdtib?, are still available. This helps ensure compatibility with older C programs, as well as making it easier for experienced C programmers to get started programming in C++. The complete list of C++ libraries.

alloc.h	assert.h	**bcd.h**	bios.h
complex.h	conio.h	ctype.h	dir.h
dos.h	errno.h	fnctl.h	float.h
fstream.h	**generic.h**	graphics.h	io.h
iomanip.h	**iostream.h**	limits.h	locale.h
math.h	mem.h	process.h	setjmp.h
share.h	signal.h	stdarg.h	stddef.h
stdio.h	**stdiostr.h**	stdlib.h	**stream.h**
string.h	**strstrea.h**	stat.h	timeb.h
time.h	values.h		

Note : The bold face header files are only defined in C++, these are not available in ANSI C.

The words/symbols in C++ are classified as keywords, identifiers, constants, strings and operators.

The keywords are explicity reserved identifiers and cannnot be used as names for the variables or other user defined program elements. The complete C++ keywrods are shown below. The bold face keywords are not available in ANSI C.

asm	auto	break	case
catch	char	**class**	const
continue	default	**delete**	do
double	else	enum	extern
float	for	**friend**	goto
if	**inline**	int	long
new	**operator**	**private**	**protected**
public	register	return	short
signed	size of	static	struct
switch	**template**	**this**	**throw**
try	typedef	union	unsigned
virtual	void	volatile	while

Naming rules for identifiers

The following rules should be followed while naming the variables.

(i) Only alphabets, digits and underscore are allowable symbols.

(ii) The name cannot start whith a digit.

(iii) Uppercase and lowercase letters are different, i.e., case sensitive.

(iv) A declared keyword canot be used as a variable name.

A major difference between C and C++ is the limit on the length of a name. In ANSI c, 32 characters are recognize as a variable name, while there is no limits on length of a variable in C++.

The following table gives the standard data types with their sizes and ranges of values used in C++.

Type	Bytes	Range
char	1	– 128 to 127
unsigned char	1	0 to 255
int	2	– 32768 to 32767
unsigned int	2	0 to 65536
short int	2	– 32768 to 32767
unsigned short int	2	0 to 65536
long int	4	– 2147483648 to 2147483647
unsigned long int	4	0 to 4294967295
float	4	3.4E-38 to 3.4E+38
double	8	1.7E-308 to 1.7E+308
long double	10	3.4E-4932 to 3.4E+4932

67.4 Keyboard and Screen I/O

The I/O operations in C++ are very simple compared to C. cout and cin are used as the standard output and input streams.

cout : The cout is used to display an object onto the standard device, normally the video screen. The insertion or 'get from' operator (the double less than sign<<) is used along with the cout stream.

The general syntax of the cout stream is,

cout << varibalel < variable2 << variable2 << ... << variable *n*;

For example,

cout<<"The value of x is"<<x;

cin : The cin is used to read a number, a character or a string of charctrs from a standard input device, normally the keyboard. The extract or 'put to' operator (the combination of double greater than sign >>) is used along with the cin operator.

The general syntax of the cin stream is,

cin >> varibalel >> variable2 >> ... >> variable *n*;

For example,

cin>>x>>y;

67.5 Predefined Manipulators

The following standard manipulators are normally used in the stream classes.

endl, hex, dec, oct, setbase, setw, setfill, setprecision, ends, ws, flush, setiosflags, resetionsflags.

All the above streams are available in the header file iomanip.h.

endl : the endl is an output manipulator to generate a carriage return or line feed character. The endl may be used several tiems in a C++ statement. It is equivalent to "\n".

setbase ( ) : The setbase ( ) manipulator is used to convert the base of one numeric value into another base. The common base converters in C++ are dec, hex, oct.

For example, let $x = 10$.

	Output
cout<<dec<<x<<endl;	10
cout<<hex<<x<,endl;	a
cout<<oct<<xx<<endl;	12

setw ( ) : The setw ( ) stands for the set width. The setw ( ) manupulator is used to speficy the total length of a variable for which the value of a variable is to be printed.

The general format is

setw (int w)

setfill ( ) : The setfill ( ) manupulator function is used to specify a different character to fil the unused fields width of the value. The general syntax is

setfill (char s)

For example, if $a = 100$ and $b = 400$ then the output of the statement

cout<<setfill ('*');

```
cout<<setw(6)<<a<<setw(5)<<b<<endl;
    ***100**400
```

setpression ( ) : This is used to control the number of digits of an output stream display of a floating point value. The general form is

setprecistion (int p)

which sets the precision for floating point insertions to *p*. The default precision is 6.

For example, if *a*= 0.345678 then the output of the staement

```
cout<<setprecision(2)<<a<<endl;
```

will generate the output.

36

ends : The ends manipulator used to attach a null terminating character ('/0') at the end of string. The ends manipulator takes no argument whenever it is invoked. This causes a null character to the output.

For example, if *a* = 250 then the output of the statement

```
cout<<"The value of a is"<<a<<ends;
cout<<"\""<<endl;
```

will generate the output

The value of a is 250 ₺"

ws (white space) : It is used to ignore the leading white space that precedes the first field.

flush ( ) : The flush members function (of cout) is used to cause the stream associated with the output to be completely emptied. This argument function takes no input parameters whenever it is invoked. For output on the screen, this is not necessary as all output is flushed automatically. In the case of a disk file being copied to another, it has to flush the output uffer prior to rewinding the output file for continued use.

setiosflags ( ) and resetiosflags ( ) : The setiosflags manipulator function is used to control different I/O settings. The setiosflags manipulator performs the same function as the self function.

The general syntax is

setiosflags (long f);

The resetiosflags manipulator performs the same function as that of the resetf function. The general form is

resetiosflags (long f);

For example, let *a* = 10.

```
cout<<setiosflags(ios::showbase);
cout<<setiosflags(ios::dec);cout<<a<<endl;
cout<<setiosflags(ios::hex);cout<<aa<endl;
cout<<setiosflags(ios::oct); cout<<a<<endl;
```

The output of these statements are

10

oxa

012

More I/O stream flags

Flag name	Meaning
skipws	skip white space during input
right	left justification of output
internal	pad after sign or base indicator
dec	decimal base
oct	octal base
hex	hexadecimal base
showbase	showbase for octal and hexadecimal numbers
showpoint	show the decimal point for all floating number
uppercase	show uppercase hex numbers
showpos	show '+' for positive numbers
scientific	use E for floating notation
fixed	use floating notation

Flag on : The seft ( ) function is invoked for setting the bit format flag of the I/O stream. The genral format is

cout.setf(flags to be set);

For example,

cout.setf(ios::showbase);

The bitwise OR(|) operator is used in the argument list of the setf ( ) function in order to change the bit format flag more than one.e.g.,

cout.setf(ios::showbase|ios::showpoint|ios:uppercase);

Flag off : The unsetf ( ) function is used to change the bits directly off. This function takes exactly one argument to turn off the bit pattern.

The syntax is

cout.unsetf(ios::uppercase);

67.6 Structure and Classes

In C and C++, a structure one or more data items (called members) which are grouped together as a single unit. On the other hand, a class is similar to a structure data type but it consists of not only data elements but also functions which are operated on the data elements. Secondly, in a structure, all elements are public by default, i.e., all the data member can be accessed from anywhere, while in a class it is private. In a class, the data and functions can be defined in one of the sections such as private, public and protected. Function defined within a class have a special relationship to the member data and member functions called the methods.

The general form of the class construct is:

```
class user_defined_name
    {
        private :
            variable declarations;
            function declaratons;
        protected :
            variable declarations;
            function declarations;
        public:
            variable declaratons;
            function declarations;
    };
```

The keywords private, protected and public are used to specify the three levels of assess protection for hiding data and function members internal to the class.

private : In private section, a member data can be accessed by the member function and friends of this class. The member functions and friends of this calss can always read or write private data members. The private data member is not accessible to the outside of the class.

protected : The members which are declared in the protected section, can only be accessed by the member functions and friends of this class. Also, these functions can be accessed by the member functions and friends derived from this class. It is not accessible to the outside of the class.

public : The members which are declared in the public section, can be accessed by any function in the outside class. The public implementation operations are also called as member functions or methods or interfaces to out of the class. Any function can send messages to an object of this class through these interface functins.

The public data members can always read and write outside this class. A member function can be inline, which means the member function can be defined within the body of the class. The keyword inline is used for short

functons and efficient storage.

A simple example of a class.

```
class student
    {
        private: //optional
            int rollno; //data member
            int age; //data member
            char sex; //data member
            float height' //data member
        public
            void getinfo ( ); /memeber function
            void disinfo ( ); /memeber function
    };
```

Note that the functions are declared but not defined.

67.6.1 Creating objects

If the class is declared, then one can create a numbers of variables of that type by using the class name like any other built-up-in-type variable.

The following example creates the objects x, a, b, c of type 'student'.

```
student x; //memory for x is created
student a, b, c; //memory for a, b, c are created
```

In C++, the class variables are known as **objects.** Therefore, x, a, b, c are called object of type 'student' rather than variables.

The necessary memory space is allowed to an object at this stage. Note that class specification, like a structure, provides only a **template** and does not create any memory space for the objects.

Objects can also be created in the following way:

```
class student
    {
        .....
        .....
    }   x, a, b, c;
```

67.6.2 Defining member functions

Member functions can be defined in two ways :

* outside the class definition

* inside the class definition.

Defining a member function of a class outside its scope

A member functions of a class is defined using the :: scoping operator (called scope resolution operator).

The general syntex of the member function of a class outside its scope is

return_type class_name :: member_function 9arguments)

Note that the type of member function arguments must exactly match with the types declared in the class definition of the class_name.

The important point to note is the use of the scope resolution operator (::) along with the class name in the header of the function definition. Only the scope operator identifies the function as a member of a particular class. Without this scope operator, the function definition would create an ordinary function subject to the usual function rules of access and scope.

Accessing a member of a class

A data or function member of a class construct is accessed using the (period) operation.

The general syntax for accessing a member of class is

```
class_object.data_member;
class_object.function_member;
```

Defining a member function inside the class definition

Another method of defining a member function is to replace the function declaration by the actual function definition inside the class.

For example,

```
class arith
    {
        private
            int x, y;
        public :
            int sum ( ) //function definiton
            {
              return (x+y);
            }
            int diff ( ) //function definition
            {
             return (x-y);
```

```
        }
    };
```

When a function is defined inside a class, it is treated as an inline function. Therefore, all the restrictions and limitations that apply to an inline functions are also applicable here. Normally, only small functions are defined inside the class definition.

67.7 Array of class objects

The array is a very important data type, it si homogeneous and stored in contiguous memory locatons. For practical applications such as designing a large size of database, arrays are very essential. The declaraton of an array of class objects is very similar to the declaration of the array of structures in C++.

An example of array of class objects is shown in the following program. The following program reads the information (sex, height, age) of a set of students and finds the number of male and female students, average age and average height.

Assinment CPP 67.1 (Example of array of class object)

Write a class 'student' with data member roll number, age, sex, height, weight and member functions 'getinfo' (to read information from key baord) and 'display' (to display the information of an object on the screen). Use this class to read any number of records and display all records with a proper format.

```
#include<iostream.h>
#include<conio.h>
const max=100;
class student
    {
    public:
        int rollno, age;
        char sex;
        float hegiht;
        void getinfor( )
            {
            cout<<"Enter oll no.";
            cin>>rollno;
            cout<<"Enter age";
            cin>.age;
            cout<<"Enter sex";
            cin>>sex;
```

```
                cout<<"Enter height ";
                cin>>height;
        }
    void display ( )
        {
            cout<<"The Rollno. is "<<rollno<<endl;
            cout<<"The Age is       "<<age<<endl;
            cout<<"The Sex is       "<<sex<<endl;
            cout<<"The Height is    "<<height<<endl:
        }
    };
void main ( )
    {
    clrscr ( );
    student s[max];
    int i, n;
    cout<<"Enter number of students"<<endl;
    cin>>n;
    for(i=0;i<n;i++)
        {
        cout<<endl<<"Enter the record of the student "<<i+1<<endl;
        s[i].getingo ( );
        }
    for (i=0;i<n;i++)
        {
            cout<<endl<<"Record of the student "<<i+1<<endl;
            s[i].display ( );
        }
    float heighttot=0;
    for(i=0;i<n;i++)
        heighttot+=s[i].height;

    fout<<"The average height of all the students is "<<heighttot/n'
    }
```

67.8 Constructor and Destructor

It is convenient if an object can initialize itself when it is first created, without the need to make a separae call to a member function. Automatic initialization is carried out using a special member function called a **constructor.** A constructor is a member function that is executed automatically whenever an object is created.

But,t he constructor function has some unusual aspect. First, they have exactly the same name, as the class of which they are members. This is how the compiler knows they are constructors. Since the constructor is called automatically by the system, it does not return any value.

We have mentioned that the special function constructor is called automatically when an object created for first time. We may guess that another function is called automatically when an object is destroyed. Such a functon is called a **destructor.** A destructor has the same name as the constructor but preceded by a tilde.

Like constructors, destructors do not have a return value. They hae no arguments. The most common use of destructor is to de-allocate memory that was allocated for the object by the constructor.

67.9 Objects as Function Arguments

The following program demonstrates some new aspects of classes : constructor overloading, defining member functions outside the class, objects as function arguments and return value is an object. This example, adds two distances mesured in FPS system.

```
#include <iostream.h>
class distance
    {
    private :
        int feet;
        float inches;
    public :
        distance ( ) //first constructor without arguments
            {
            feet =0;
            inches=0.0;
            }
        distance (int ft, float in)  //second constructor with two arguments
            {
            feet=ft;
            inches=in;
```

```
        }
    void getdist ( ) //function to read a length
        {
        cout<<"nEnter feet:" ; cin>>feet;
        cout<<"Enter inches: "; cin>>inches;
        }
        vodi showdist ( ) //function to display distance
        {
            cout<<"\nfeet<<"\' "<<inches<<"\" ";
        }
        distance addist (distance); //declaration of the function
    };
//definition of the function outside the class
    distance distance :: addist (distance d2)
    {
    distance temp;
    temp.incehs=inches+d2.inches;
    if)temp.inches>=12.0)
    {
        temp.inches-=12.0;
        temp.feet=1;
    }
        tem.feet+=feet+d2.feet;
        return temp;
    }
void main ( )
    {
    distance dist1, dist3; // created objects and initialized
                    //by suing first constructor
distance dist2(12, 7.35); //creating and initialized
                    //by using second constructor
dist1.getdist ( ); /reads data for dist1 using member function
dist3=dist1.addist(dist2);
```

```
        cout<<endl<<Dist1=";dist1.showdist ();
        cout<<endl<<Dist2=";dist2.showdist ();
        cout<<endl<<Dist3=";dist3.showdist ();
    }
```

67.10 Arrays as Class Member Data

Arrays canbe used as data items in classes. Here, we explain the use of arrays as class member by considering a very important data structure –stack.

Stacks are one of the main architecture of the 80x86 microprocessor used in MS-DOS computers. Functons pass their arguments and store their return address on the stack. This kind of stack is implemented partly in hardware and is most conveniently accessed in assembly language. However, stacks can also be created completely in software. Software stakes offer a storage device in certain programming situations, such as parsing algebraic expressions.

Assinment CPP 67.2 (Example of array element within class)

Write a class for stack (containing the operations push, pop and top element) and demostrate it with a proper input.

```
#include<iostream.h>
#include<conio.h>
const max=100;
class stack
    {
    private:
        int s[max], top, var;
    public:
        stack () //constructor
        {
        top=0; //initially the number of elements of the stack is 0
        }
    void push (int var)
        {
        s[++op]=var;  //observe the use of ++ operator
        }
    int pop ()
        {
```

```
        return(s[top- -]); //see use of - - operator
        }
int topelement ( )
        {
        return(s[top]);
        }
};
void main ( )
        {
        clrscr ( );
        stack s;
        s.push (100);
        s.push (200);
        s.push (300);
        cout<<"First element is "<<s.pop( )<<"\n";
        cout<<"top element is "<<s.topelement( )<<endl;
        s.push (400);
        s.push (500);
        cout<<"Second element is "<<s.pop ( );
}
```

67.11 Module Summary

The C++ programming language is divided into three moduels. In this module we studied the keyboard and screen I/O statements and some predefined manipulators. Then we introduced the main striking feature of C++, the class. Theuse of member functions inside and outside the classes are explained with examples. The use of objects as an array and functions arguments are discussed. The two special functions constructor and destructors are explainted with suitable examples. Several programs are designed using the features of C++ those are studied in this module. The module ends with an exercise and references.

67.12 Self Assessment Questions

1. Write a program to input five numbers and print them out in the reverse order of input.
2. Write a program that adds two four-element arrays together. The first array should be initialized with values 1, 2, 3, and 4; the second with values 5, 6, 7 and 8. Both arrays shouldbe passed to the function addarrays ( ), which adds corresponding variables of each array together and leaves the results in the

scond array. The second array should then be printed out by the main ( ) function.

3. Write a function that merges two ordered integer arrays into a third ordered array. The two arrays should be able to contain any number of non-negative values. The last value in each array should be the sentinel value –1. This value should also be placed at the end of the merged array. Test your program by writing the main ( ) function.
4. Write a program to calculate interest on a loan. It should allow input of principal, rate in percentage form, and time in days. Use the variables principle, rate, time and interest. The following formula is to be used :

$$interest = principal \times \frac{rate}{100} \times \frac{time}{365}$$

5. Write a program to figure out the circumference, cross-sectional area, and volume of a sphere, given a radius. The radius should be input and the rest printed out.
6. Define a class string that could work as a user-defined type. Include constructors that will enable us to create an uninitialized string

string s1; /string with length 0

and also to initialize an object with a string constant at the time of creation like

string s2("Welcome");

Include a function that adds two strings to make a third string. Note that the statement

s1=s2;

will be perfectly reasonable expression to copy one string to another.

Write a complete program to test the class to see it does the following tasks:

(a) Creates uninitialized string objects.
(b) Creates objects with string constants.
(c) Concatenates two strings properly.
(d) Displays a desired string object.

7. Define a class to represent a bank account. Include the following members :

Data members are

(i) Name of the depositor
(ii) Account number
(iii) Type of account
(iv) Balance amount in the account

Member functions are

(i) To assign initial values

(ii) To deposit an amount

(iii) to withdraw an amount after checking the balance

(iv) To display name and balance.

Write the main function to test the program.

8. Write a class to present a vector (containing at most five floats). Include member functions to perform the following tasks:

(i) To create the vector

(ii) To modify the value of a given element

(iii) To multiply by a scalar value

(iv) To display the vector in the form (4, –5, 6, 6, 8).

Write a program to test the class.

9. Write a class for stack (including the operations push, pop, top element) and demonstrated it with a proper input.

67.13 Suggested Further Readings

1. E. Balagurusamy, Object-Oriented Programming with C++, Tata McGraw Ill.
2. D. Ravichandrana, Programming with C++, Tata McGraw Hill.
3. Robert Lafore, Object-Oriented Programming in Turbo C++, Galgotia.
4. Bjarne Stroustrup, C++ Programming Language, 3ed.
5. Paul S, Wang, Standard C++ with OOP, Vikas Publishing House.
6. Steven C. Lawbr, Computer Science with C++, Vikas Publishing House.

--- 0 ---

M.Sc. Course
in
Applied Mathematics with Oceanology
and
Computer Programming

PART-II

Paper-VI

Group–B

Module No. - 68

LAB. 1 : PROGRAMMING IN C++ - II

Structure :

68.1 Introduction

68.2 Objectives

68.3 Function and Operator Overloading

- 68.3.1 Function overloading
- 68.3.2 Operator overloading
- 68.3.3 Overloading unary operator

68.4 Inheritance

- 68.4.1 Defining derived classes
- 68.4.2 Multilevel inheritance
- 68.4.3 Multiple inheritance
- 68.4.4 Hierarchical Inheritance

68.5 Module Summary

68.6 Self Assessment Questions

68.7 Suggested Further Readings

68.1 Introduction

This module is the continuation of the previous one. Here we introduce two very exciting features of C++ viz., function and operator overloading and inheritance. Using these features we can write programs in tremendously very simply way.

68.2 Objectives

Gone through this module the students will learn the following topic :

* Function overloading
* Operator overloading
* Different types of inheritance.

68.3 Functon and Operator Overloading

68.3.1 Function overloading

Overloading of functions and operators are the most exciting features of OOP. The C++ language allows the user to create new abstract data types, which is one of the major advantaes of the OOP.

Function overloading is a logical method of calling several functions with different arguments and data types that perform basically identical things by the same name. The main advantages of using function overloading are:

(i) Eliminating the use of different function names for the same operation.

(ii) Helps to understand and debug easily.

(iii) Easy maintainability of the source code.

(iv) Better understanding of the relation between the program and the outside of the class.

Function overloading is an easy concept in C++ and generally it is used within the class concept for member functions and constructors. The compiler classifies the overloaded function by its name and the number and type of arguments in the function declaration. The function declaration and definition is essential for each function with the same function name but with different arguments and data types.

The following program shows the use of function overloading without class.

```
#include <iostream.h>
int sum(int, int);
int sum(int,int,int);
```

```
float sum(float,float);
float sum(float,int);
void main ( )
    {
        int a,b, c;
        float x, y;
        cout<<"Tner three integers "<<endl;
        cin>>a>>b>>c;
        cout<<"Enter two floating nos. "<<endl;
        cin>>x>>y;
        cout<<"Sum of a and b "<<sum (a, b)<<endl;
        cout<<"Sum of a, b and c " <<sum (a, b, c)<endl;
        cout<<"Sum of x and y "<<sum (x,y)<<endl;
        cout<<"Sum of a and x "<<sum (x,a)<<endl;
    }
int sum (int a, int b)
    {
        return (a+b);
    }
int sum (int a, int b, int c)
    {
        return (a+b+c);
    }
float sum(float x, float y)
    {
        return (x+y);
    }
float sum (float a, int x)
    {
        return (a+x);
    }
```

The following is an example of function overloading with class.

Assinment CPP 68.1 (Example of function overloading).

Write classes for circle and rectangle and also write necessary member functions to perform the following operations.

(i) Area of a circle,
(ii) Circumference of a circle,
(iii) Area of a rectangle,
(iv) Perimeter of a rectangle.
Demonstrate the program.

```
#include<iostream.h>
#include<conio.h>
class circle // class for circle
    {
    int radius;
    public:
        void area ( )
        {
            cout<<"The area of circle is "<<(3.14159*radius*radius)<<end1;
        }
        void peri ( )
        {
            cout<<"Circumference of circle is "<<(2*3.1419*radius)<<end1;
        }
    void getinfo ( )
        {
        cout<<"Enter the radius of the circle "<<end1;
        cin>>radius;
        }
    }
class rectangle // class for rectangle
    {
    float a, b;
    public:
        void area ( )
        {
            cout<<"The area of rectangle is "<<(a*b)<<end1;
        }
    void peri ( )
```

```
        {
            cout<<"The perimeter of rectangle is "<<(2*(a+b))<<endl;
        }
    void getinfo ( )
        }
            cout<<"Enter the length & breath "<<endl;
            cin>>a>>b;
        }
    };
void main ( )
    {
        clrscr ( );
        circle c; //c is declared as an object of type circle
        rectangle x; // x is declared as an object of type rectangle
        c.getinfor ( ); // reading of the object c
        x.getinfo ( ); // reading of the object x
        c.area ( ) ; // computaton of area of c (circle)
        c.peri ( ); // computation of perimeter of c (circle)
        x.area ( ); // computaton of area of x (rectangle)
        x.peri ( ); // computation of perimeter of x (rectangle)
    }
```

68.3.2 Operator overloading

Operator overloading is another example of C++ polymorphism. In fact, some form of operator overloading is available in all high level languages. For example, in Pascal the + operator can be used to carry out three different operations such as adding integers, two real numbers and concatenating two strings. Operator overloading is one of the most challenging and exciting fetures of C++. The main advantages of using overloading operators in a program are that it is much easier to read and debug. Operators which already exist in the language, can only be overloaded. overloading cannot alter either the basic template of an operator, nor its place in the order of precedence.

It seems that the main idea of using operator overloading in a C++ program is to make it more natural to read and write. Even debugging such codes are much easier. As C++ is mostly used to develop a large software package in an easy way, the care must be taken while implementing the operator overloading in a program, otherwise, it leads to confusion while debugging and testing the codes.

Operator overloading is accomplished by means of a special kind of function. Operator overloading can be carried out by means of either member functions or friend functions.

The general syntax of operator overloading is

return_type operator operator_symbol (parameters)

The keyword operator must be preceded by the return type of a function which gives information to the compiler that overloading of operator is to be carried out. Only those operators that are predefined in the C++ compiler are allowed to be overloaded.

It may be remember that the operator functions must be either meber functions (non-static) or friend functions. A basic difference between them is that a friend function will have only one argument for unary operators and two for binary operators. This is because the object used to invoke the member function is passed implicitly and therefore is available for the member function. This is not the case with friend functions. Arguments may be passed either by value or by reference.

68.3.3 Overloading unary operator

Overloading assignment operator

The assignment operator is a unary operator because it is connected only to the entity on the right, which is the operands. Whenever an assignment operator is overloaded in a base class, it cannot be inherited in a derived class. Hence the assingment operator which is to be overloaded must be either a method (or a member function) or a class object with arguments.

```
//overloading an assignment operator.
#include<iostream.h>
class sample
    {
    private :
        int real;
        int imag;
    public :
        sample (int, int);
        void display ( );
        void operator=(sample obj);
    };
    void sample :: operator = (sample obj)
    {
        real=obj.real;
        imag=obj.imag;
    }
```

```
sample :: sample(int a, int b)
{
        real=a
        imag=b;
}
void sample :: display ( )
{
        cout<<"Real part is "<<real<<endl;
        cout<<"Imaginary part is "<<imag<<endl;
}
void main ( ) {
        sample obj1 (10, 20);
        sample obj2 (5, 100);
        obj1.display ( );
        obj2.display ( );
        obj1=obj2;
        obj1.display ( );
}
```

Overloading of increment/ decrement operators

It is well known that C++ supports very unusual notation or operator that is used for incrementing and decrementing by 1. These operators can be used as either prefix or postfix. In general, overloading of these operators cannot be distinguished between prefix or postfix operation. However, whenever a psotfix operation is overloaded, it takes a single argument along a member function of a class object.

Assinment CPP 68.2 (Example of overloading of increment operator (++))

Write a program in C++ using operator overloading to find first *n* Fibonacci numbers.

```
#include<iostream.h>
#include<conio.h>

class fibonacci // class for Fibonacci
{
unsigned long into f0, f1, fib;
public;
      fibonacci ( ); // constructor
```

```
        void operator++( );
        void display ( );
};
fibonacci:: fibonacci ( )
        {
        f0=0,
        f1=1,
        fib=f0+f1;
}
void fibonacci:: operator++ ( )
{
        f=f1,
        f1=fib,
        fib=f0+f1;
}
void fibonacci:: display ( )
{
        cout<<fib<<endl;
}
void main ( )
{
        clrscr ( );
        fibonacci x;
        int n;
        cout<<"How many Fibonacci numbers do you want?"<<endl;
        cin>>n;
        cout<<"The Fibonacci numbers are"<<"\n";
        for(int i=0;i<n;i++)
                {
                x.display ( );
                ++x;
        }
}
```

Overloading binary operators

Assinment CPP 68.3 (Example of binary and input/output operators)
Design a class for complex number to include addition, subtraction, multiplication division and input and output operators. Demonstrate the class.

```
#include<iostream.h>
#include<conio.h>
class complex
    {
    public:
        float real, imag;
    public:
    complex ( ) //constructor
    {
    real=0; imag=0;
    }
    //overloading of addition operator
    friend complex operator+(complex a, complex b)
    {
        complex c;
        c.real=a.real+b.real;
        c.imag=a.imag+b.imag;
        return c;
    }
    //overloading of subtraction operator
    complex operator - (comlex a)
    {
        complex c;
        c.real=real-a.real;
        c.imag=imag-a.imag;
        return c;
    }
    //overloading of multiplication operator
    friend complex operator*(complex a, complex b)
    {
        complex c;
```

```
            c.real=a.real*b.real-a.imag*b.imag;
            c.imag=a.real*b.imag+a.imag*b.real;
            return c;
        }
//overloading of division operator
friend complex operator/(complex a, complex b)
        {
            complex c; float t;
            t=b.real*b.real+b.imag*b.imag;
            c.real=(a.real*b.real+a.imag*b.imag)/t;
            c.imag=(a.imag*b.real-a.real*b.imag)/t;
            return c;
        }
//overloading of output operator
friend ostream & operator<<(ostream &s, complex &a)
        {
            cout<<"("<<a.real<<", "<<a.imag<<")"<<endl;
            return s;
        }
//overloading of input operator
friend istream & operator>>(istream &s, complex &a)
        {
            cin>>a.real>>a.imag;
            return s;
        }
};
void main (void)
        {
        clrscr ( );
        complex a, b, c;
        cin>>a>>b;
        c=a+b; // addition of a and b and storing it in c
        cout<<c<<endl; //printing of c
        cout<<a-b<<endl; //substraction and printing of a and b
        cout<<a*b<<endl; //product of a and b and direct printing
        cout<<a/b<<endl; //division of a by b and printing
```

```
    }
```

Assinment CPP 68.4 Design a class for matrix to include the operations addition, multiplication, the functions inverse, determinant and functions for input and output. Save this class to the file "matrix.h" and demonstrate the class.

```
//The file "matrix.h" that contains the class matrix.
const max=10;
class matrix {
    private :
        int row, col;
        float a[max] [max];
    public :
        int read ( );
        int print ( );
        matrix operator+(matrix b);
        matrix operator* (matrix b);
        matrix operator~( ) //for inverse
        float det ( );
};
    int matrix :: read ( )
    {
        cout<<"Enter number of rows and columns"<<endl;
        cin>>row>>col;
        if(row*col==0)
        {
        cout<<"Invalid matrix !"<<endl;
        return 0;
    }
    cout<<"Enter the elements of matrix (rowwise)"<<endl;
    for(int i=0; i<row;i++)
    for (int j=0; j<col; j++) cin>>a[i][j];
    return 1;
}

int matrix :: print ( )
```

```
{
    if (row*col= = 0)
    {
        cout<<"Invalid matrix !"<<end1;
        return 0;
    }
    for (int i=0; i<row;i++)
    {
        for (int j=0, j<col; j++) cout <<a[i][j]<<" ";
        cout<<endl1;
    }
    return 1;
}

matrix matrix :: operator+(matrix b)
{
    matrix c;
    c.row=0; c.col=0;
    if((row!=b.row)||(col!=b.col))
    {
        cout<<"Matrix addition is not possible"<<end1;
        return c;
    }
    c.row=row;
    c.col=b.col;
    for(int i=0;i<row;i++)
    for (int j=0; j<col; j++)
        c.a[i] [j]=a[i] [j] +b.a[i] [j];
    return c;
}

matrix matrix :: operator*(matrix b)
    {
    matrix c;
    if (col!=b.row)
    {
```

```
            cout<<"Matrix multiplication is not allowed"<<endl;
            c.row=0; c.col=0;
        }
        c.row=row;
        c.col=b.col;
        for(int i=0; i<row; i++)
        for (int j=0; j<b.col;j++)
        {
            c.a[i] [j]=0.0;
            for(int k=0; k<col; k++) c.a[i] [j]+=a[i] [k]*b.a[k] [j];
        }
        return c;
}

float matrix : det ( )
{
        int i, j, k;
        float prod, sign=1, temp;
        if (row!=col)
        {
            cout<<"The matrix is not square ! Deteminant value
                is not possible"<<endl1;
            exit(0);
        }
        for (k=0;k<row;k++) {
        {
        temp=fabs (a[k] [k]); j=k; // initial maximum
        for (int i=k+1; i<ro; i++)
            if(temp<fabs(a[i] [k]))
            {
                temp=fabs(a[i] [k]); j=i;
            }
        }
        //j is the pivot position
        if (j! =k) // interchange k and j rows
        {
```

```
            sign= – sign;
            for (i=0; i<row;i++)
            {
            temp=a[j] [i]; a [j] [i] =a[k][i]; a[k] [i]=temp;
            }
      }
      for (j=k+1; j<row; j++) // makes a[k] [k] to a [row] [k] zero
      {
            temp=a[j] [k]/a [k] [k];
            for(i=0; i<row; i++) a[j] [i]–=temp*a[k] [i];
      }
} // end of k loop
prod=sign;
for(i=0;i<row; i++) prod*=a[i] [i]; // product of diagonal elements
return prod;
}// of function det

matrix matrix:: operator~( )
      {
      const zero=0.00001;
      int m, k, i, j;
      float temp;
      matrix b;
            if(row!=col)
      {
            cout<<"The matrix is not square ! Inversion is not
                  possible"<<end1;
            exit (0);
      }
      /* augment the matrix A */
      for (i=0;i<row; i++) for (j=0; j<row; j++) a[i] [row+j]=0;
      for (i=0; i<row;i++) a [i] [row+i]=1;
      m=2*row;
      for (k=0; k<row; k++)
            {
            //finds pivot element and its position
```

```
            temp=fabs(a[k] [k]); j=k; // initial maximum
            for (i=k+1; i<row; i++)
            if (temp<fabs(a[i] [k]))
                  {
                  temp=fabs (a[i] [k]); j=i;
                  }
            if (fabs(a[j] [k]<=zero) // if a [j] [k] =0
            {
            cout<<"The matrix is singular and is not invertible"<<end1;
            exit(0);
            }
      if(j!=k) / * interchange k and j rows */
            {
            for (i=0; i<m. i++)
            {
            temp=a[j] [i]; a [j] [i]=a [k] [i]; a [k] [i] =temp;
            }
      }
      if(a[k] [k]!=1)
            {
            temp=a[k] [k];
            for(i=0;i<m; i++) a [k] [i]/=temp;
            }
      for (j=k+1; j<row; j++) // makes a [k+1] [k] to a [row] [k] zero
      {
      temp=a[j] [k];
      for (i=0; i<m; i++) a [j] [i]- -=temp*a[k] [i];
      }
} // end of k loop
// make left half of A to a unit matrix
for (k=1; m>row; k++)
      {
      for (j=k-1; j>=0; j- -)
      {
      tempa[j] [k];
      for (i=0; i<m; i++) a[j] [i] -=temp*a[k] [i];
```

```
            }
        }
        b.row=row; b.col=col;
        for (i=0;i<row; i++) {
            for (j=row; j<m; j++)
            b.a [i] [j-row]=a [i] [j];
        }
return b;
}//end of inverse
//end of the file "matrix;h"
```

The following program demonstrates the class matrix written in the file "matrix.h".

```
//File name matrix.cpp
#include<iostream.h>
#include<conio.h>
#include<ath.h>
#include<stdlib.h>
#include<matrix.h> // user defined header file
void main ( ) {
    clrscr ( );
    cout<<"---------- INPUT ---------"<<endl1;
    matrix a, b, c;
    a. read ( );
    b. read ( );
    cout<<"--------Addition ---------"<<end1;
    c=a+b;
    c.print ( );
    cout<<"----------Multiplication ----------"<<endl1;
    (a*b).print ( );
    cout<<"---------- Determinant---------"<<endl1;
    cout<<a.det ( )<<end1;
    cout<<"----------Inverse ----------"<<end1;
    (~a).print ( );
    }
```

Assinment CPP 68.5 Write a program to find the solution of a system of linear equations by matrix inverse method. Use the header file "matrix.h".

```
//File name matrix.cpp
#include<iostream.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
#include<matrix.h> //user defined header file
//Assumed that the system of equations be ax=b.
//Its solution is x=(~a)b, where (~a) is the inverse of the matrix a.
void main ( )
    {
        clrscr ( );
        matrix a, b, x;
        a. read ( );
        b. read ( );
        x = (~a)*b;
        cout<<"The solution vector is"<<endl;
        x.print ( );
    }
```

Overloading of comparison operators

Comparison in logical operators require two objects to be compared and hence the result will be out of these:

less than
less than or equal to
greater than
greater than or equal to
equal to
not equal to

The return value of the operator is an integer. Operator overloading accepts an object on its right as a parameter and the object on the left is passed by the this pointer.

```
//Overloading of << operator
#include<iosream.h>
class sample
    {
    private :
```

```
        int x, y;
    public:
        sample ( );
        sample (int, int);
        void display ( );
        int operator <(sample a);
    };
sample :: sample ( )
    {
    x=y=0;
    }
sample :: sample (int a, int b)
    {
    x=a; y=b;
    }
void sample :: display ( )
    {
    cout<<"First value is "<<x<<" "<<"Second value is "<<y<<endl;
    }
int sample :: operator<(sample a)
    {
    return (x<a.x);
    }
void main ( )
    {
    sample a(10, 0), b(30, 20);
    cout<<(a<b)<<endl;
    cout<<(b<a)<,endl;
    }
```

Rules for overloading operators

(i) Only existing operators can be overloaded. New operators cannot created.

(ii) The overloaded operator must have at least one operand that is of user_defined type.

(iii) We cannot change the basic meaning of an operator. That is, we cannot redefined the plus (+) operatro to subtract one value from the other.

(iv) Overloaded operators follow the syntax rules of the original operators. That cannot be overriden.

(v) There are some operatros that cannot be overloaded. (e.g., size of, ., ::, ? etc.).

(vi) We cannot use friend functions to voerload certain operatros. However, member functions can be used to overload them.

(vii) Unary operators, overloaded by means of a member function, take no explicit arguments and return no explicit arguments and return values. But, those overloaded by means of a friend function take one reference argument (the object of the relevant class).

(viii) Binary operators overloaded through a member function take one explicit argument and those which are overloaded through a friend function take two explicit arguments.

(ix) When using binary operators overloaded through a member function, the left hand operator must be an object of the relevant class.

(x) Binary arithmetic operators such as +, –, * and / must explicitly return a value. They must not attempt to change their own arguments.

Following operators can be overloaded
Type 1: Binary operators

Operator	Meaning
[ ]	Array element reference
( )	Function call
new	New operator
delete	Delete operator
*	Multiplication
/	Division
%	Modulus
+	Addition
–	Subtraction
<<	Left shift

>>	Right shift
<, <=, >, >=, = =, !=	Comparison operators
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
&&	Logical AND
||	Logical OR
=	Assignment
*=, /=, %=, +=, –=	Arithmetic with assign
<<=	Shift left and assign
>>=	Shift right and assign
&=	Bitwise right and assign
|=	Bitwise AND and assign
^=	Bitwise one's complement and assign

Type 2 : Unary operators

Operator	Meaning
–>	Indirect member operator
!	Logical nagation
*	Pointer reference (indirection)
&	Address operator
~	Ones complement
–> *	Indirect pointer to member
+	Addition
–	Subtraction
++	Increment
– –	Decrement
–	Unary minus

Type 3 : Operators common to unary and binary forms

The following operators are used both for unary and binary forms :

Operator	Meaning
+	Addition
–	Subtraction and unary minus
*	Multiplication and pointer reference
&	Bitwise AND and address of

Type 4: Operators that cannot be overloaded

Operator	Meaning
.	Direct member or class member operator
.*	Direct pointer to member
::	Scope resolution operator
?:	Conditional operator
size of	Size in bytes operator
#, ##	Preprocessing symbols

68.4 Inheritance

The C++ class can be reused in several ways. Once a class has been written and tested, it can be adapted by other programmers to suit their requirements. This is basically done by creating new classes, reusing the properties of the existing ones. The mechanism of deriving a new class from an old one is called **inheritance** or **derivation.** The old class is referrd to as the **base class** and the new one is called the **derived class.**

The derived class inherits some or all from the base class. A class can also inherit properties from class. A class can also inherit properties from more than one class or from more more than one level. A derived class with only one base class is called **single inheritance** and one with several base classes is called **multiple inheritance.** On the other hand, the members of one class may be inherited by more than one class. This process is known as **hierarchical inheritance.** The mechanism of deriving a class from another deriving a class from another derived class is known as **multilevel inheritance.**

68.4.1 Defining derived classes

A derived class is defined by specifying its relationship with the base class in addition to its own details. The general form of defining a derived class is :

```
class derived_class-name:visibility_mode base_class_name
{

};
```

The colon indicates that the derived_class_name is derived from the base-class_name. The visibility mode is optional and if present, may be either private or public. The default visibility mode is private. Visibility mode specifies whether the features of he base class are privately derived or publicly derived.

For example,

class ABC : private XYZ // private derivation

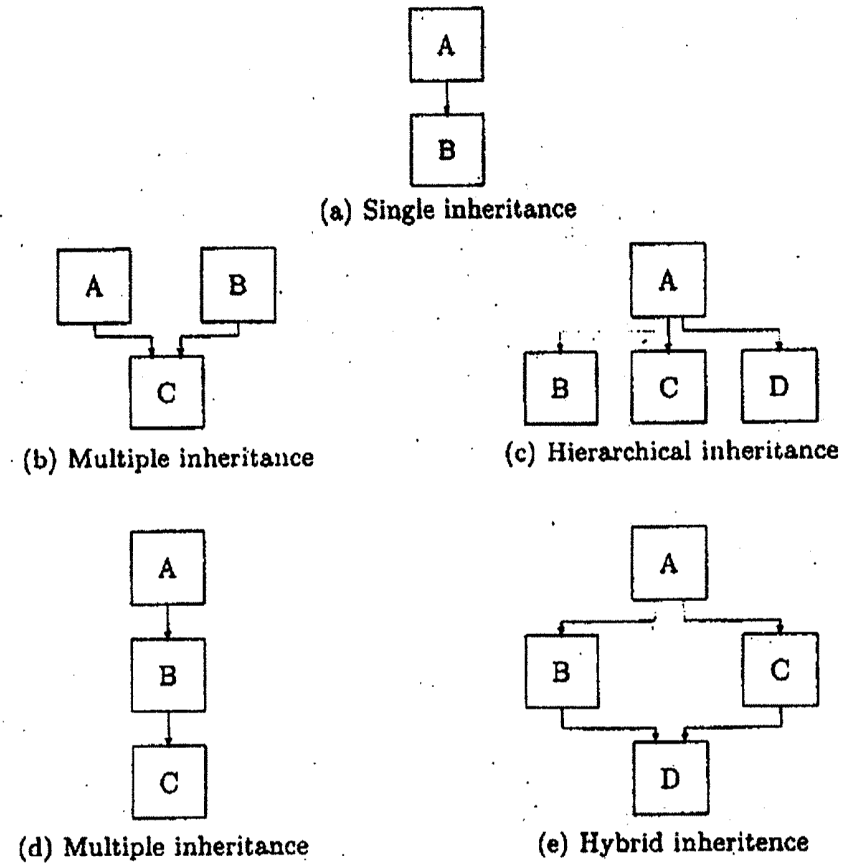


Figure 68.1: Different inheritances.

```
    {
        members of ABC
    };
class ABC : public XYZ //public derivation
    {
        members of ABC
    };
class ABC : XYZ // private derivation by default
    {
        members of ABC
    };
```

When a base class is **privately inherited** by a derived class, 'public members' of the base class become 'private members' of the derived class and therefore the public members of the base calss can only be accessed by

the members functions of the derived class. They are inaccessible to the objects of the derived class. A public member of a class can be acessed by its own objects using the dot operator. The result is tha no member of the base class is accessible to the objects of the drived class.

On the other hand, when the base class is publicly inheried, 'public members' of the base class become 'public members' of the derived class and therefore they are accessible to the objects of the derived class. In both the cases, the private members are not inherited and therefore, the private members of a base class will never become the members of its derived class.

In inheritance, some of the base class data elements and member functions are 'inherited' into the derived class. We can add our own data and member functions and thus extend the functionality of the base class. Inheritance when used to modify and extend the capabilities of the existing classes, becomes a very powerful tool for incremental program development.

Example on single inheritance.

```
//Single inheritance, public derivation.
#include<iostream.h>
class B
    {
        int a; //private, not inheritable
    public:
        int b; //public ; ready for inheritance
        void getab ( );
        int geta ( );
        void showa ( );
    };
class D : public B //public derivation
    {
        int c;
    public:
        void mul ( );
        void display ( );
    };
//definitions of functions
void B:: getab ( )
{
    a= 5;
    b = 10;
}
```

```
int B :: geta ( )
        {
        return a;
        }
void B :: showa ( )
        {
        cout <<"a= "<<a<<end1;
        }
void D::mu1 ( )
        {
        c=b*geta ( );
        }
void D :: display ( )
        {
        cout<<"a= "<<geta ( )<<end1;
        cout<<"b= "<<b<<end1;
        cout<<"c= "<<c<<end1;
        }
void main ( )
        {
        D d;
        d.getab ( );
        d.mul ( );
        d. showas ( );
        d.b=20;
        d.mul ( );
        d.display ( );
        }
```

A sample of input / output :

a=5 a=5 b=10 c=50

a=5 b=20 c=100

```
//Single inheritance, private derivation.
#include<iostream.h>
```

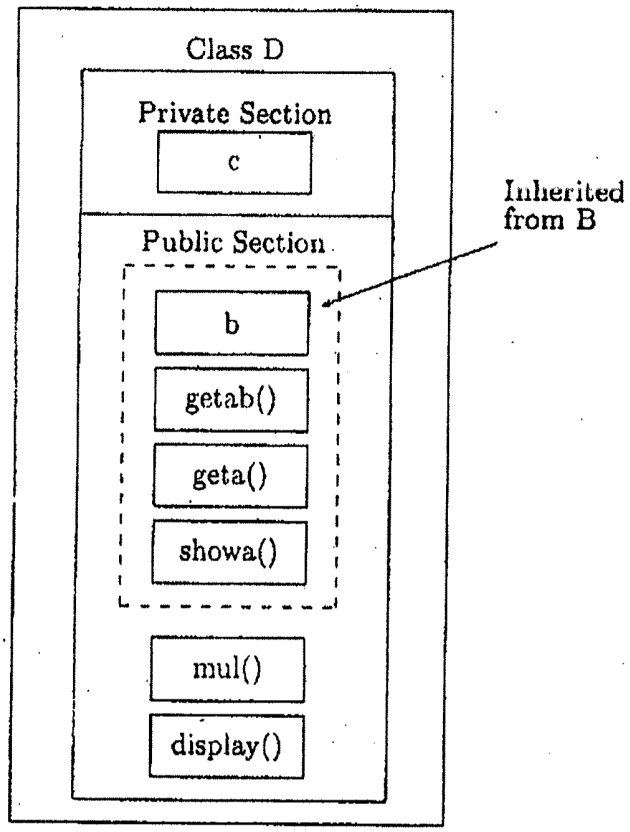


Figure 68.2: Status of the class D for public derivation.

```
class B
    {
        int a;
    public:
        int b;
        void getab ( );
        int geta ( );
        void showa ( );
};
class D :private B //private derivation
{
    int c;
    public:
    void mul ( );
    void display ( );
```

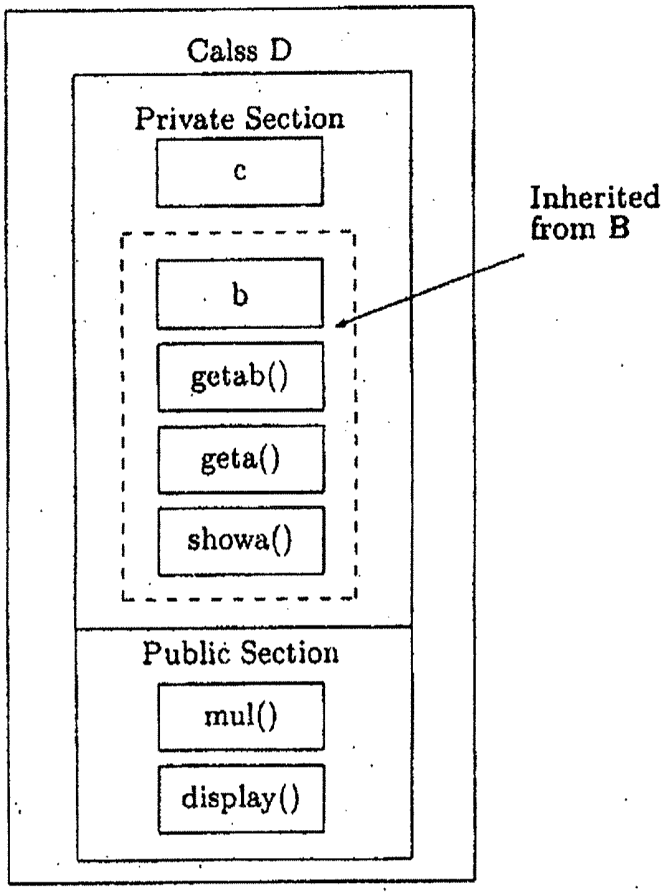


Figure 68.3: Status of the class D for private derivation.

Making a private member inheritable

We have seen that a private member of a base class cannot be inherited and therefore it is not available for the derived class directly.

C++ provides a third visibility modifier, **protected,** which serve a limited purpose in inheritance. A member declaed as protected is accessible by the member functions within its class and any class immediately derived from it. If cannot be accessed by the functions outside these two classes. A class can now use all the three visibility modes as illustrated below:

When a protected member is inherited in public mode, it become protected in the derived class too, and therefore is accessible by the member functions of the derived class. It is also ready for further inheritance. A protected members, inherited in the private mode derivation, becomes private in the derived class. Although, it is available to the member function of the derived class, it is not available for further inheritance (since private members cannot be inherited).

Example **of private derivation** for protected data :

Base class visibility	Derived class visibility	
	Public derivation	**Private derivation**
Private	Not inherited	Not inherited
Protected	Protected	Private
Public	Public	Private

```
class B
    {
    int a;
    protected :
        int b;
        void getab( );
    public :
        int c;
        void getc ( );
};
class D : private B  //private derivation
{
    int d;
    public :
        void mul ( );
        void display ( );
};
```

Example of **public derivation** for protected data :

```
class B
{
    private :
        int a;
    protected :
        int b;
        void getab ( );
    public :
    int c:
```

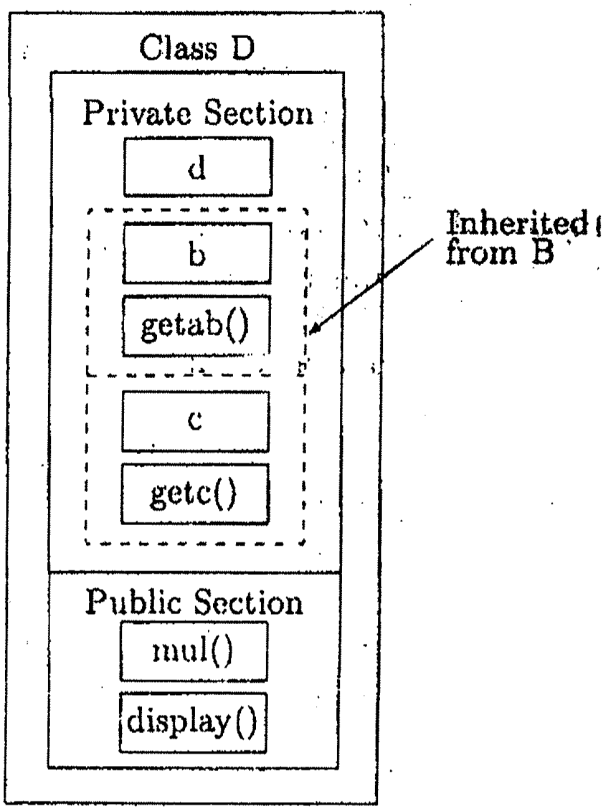


Figure 68.4: Status of the protected data for private derivation.

```
    void get c ( );
    };
class D : public B //public derivation
    {
    private :
        int d;
    protected :
        int c;
        void getc ( );
    public :
        int x;
        void mul ( ) ;
        void display ( );
    };
```

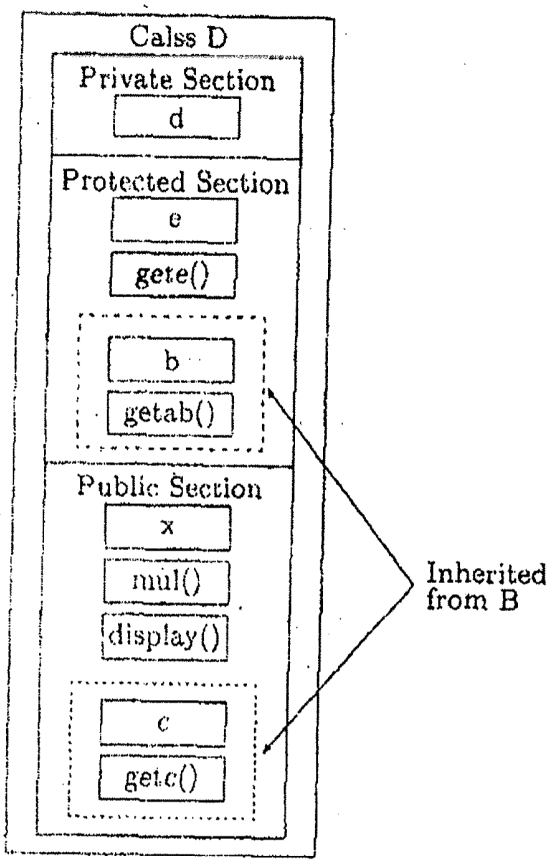


Figure 68.5: Status of the protected data for public derivation.

68.4.2 Multilevel inheritance

In C++, a class can be derived from another derived class.

The class A serves as a base class for the derived class B which in turn serves as a base class for the derived class C. The class B is known as **intermediate base class** since it provided a link for the inheritance between A and C. The chain ABC is known as **inheritance path.**

A derived class with multilevel inheritance is declared as follows:

```
clas sA //base class
    {
    .....
    };
class B : public A //B derived from A
    {
    .....
```

```
{;
class C : public B //C derived from B
{
.....
};
```

This process can be extended to any number of levels.

68.4.3 Multiple inheritance

This is known as **multiple inheritance.** Multiple inheritance allows us to combine the features of several existing classes as a starting point for defining new classes. It is like a child inheriting the physical features of one and the intelligence of another.

The general form of a derived class with multiple base classes is as follows:

```
class D : visibility B1, visibility B2, ...
{
.....
.....
}
```

where, visibility may be either **public** or **private**. The base classes are separated by commas. For exmple,

```
class P : public M, pubc N
{
public:
void display ();
};
```

where the classes *M* and *N* are

```
class M
{
protected :
    int m;
public :
    void getm (int);
};
```

```
class M
    {
    protected :
        int n;
    public :
        void getn (int);
    };
```

The derived class P, as declared above contain all the members of M and N in addition to its own members as shown below:

```
class P
    {
    protected :
        int m; //from M
        int n; //from N
    public :
        void getm (int); //from M
        void getn (int); //from N
        void display ( ); //own member
    };
```

68.4.4 Hierarchical inheritance

Additional members are added through inheritance to extend the capabilities of a class. Another interesting application of inheritance is to use it as a support to the hierarchical design of a program. Many programming problems can be cast into a hierarchy where certain features of one level are shared by many others below that level.

Assinment CPP 68.6 A program consists of two base classes and one derived class. The base class basic-info contains the data members: name, roll number and sex. An another base class aceademic_fit contains the data members: course, semester and rank. The derived class financial_assit contains the data amount besides the data mebers of the base classes. The derived class has been declared as an array of class objects. The member functions are used to get information of the derived class from the keyboard and display the contents of the array of class objects on the screen.

```
//Multiple inheritance using array of objects
#include<iostream.h>
```

```
#include<iomanip.h>
const int max=100;
class basic_info
    {
        private :
            char name [20];
            long int rollno;
            char sex;
        public :
            void getdata ( );
            void display ( );
        }; //end of class basic_info
    class academic_fit
    {
        private :
            char course [20];
            char semester [10];
            int rank;
        public :
            void getdata ( );
            void display ( );
    }; //end of academic_fit
class financial_assit : private basci_info, private academic_fit
    {
        private :
            float amount;
        public :
            void getdata ( );
            void displya ( );
    }, //end of class financial_assit
void basic_info :: getdata ( )
    {
        cout<<"Enter a name"<endl
        cin>>name;
        cout<<"Roll No. "<<endl;
        cin>>rollno;
```

```
            cout<<"Sex "<<endl;
            cin>>sex;
        }
void basic_info :: display ( )
        {
            cout<<"Name "<<" "<<rollno<<" "<<sex;
        }
void academic_fit :: getdata ( )
        {
            cout<<"Course name (B.Tech/MCA/BCA/MSc. etc) ?"<<endl;
            cin>>course;
            cout<<"Semester (first/second etc)?"<<endl;
            cin>>semester;
            cout<<"Rank of the student "<<endl;
            cin>>rank;
        }
void acedemic_fit :: display ( )
        {
            cout<<course<<" "<<semester<<" "<<rank;
        }
void financial_assit :: getdata ( )
        {
            basic_info () :: getdata ( );
            academic_fit :: getdata ( );
            cout<<"Amount in rupees ? "<,endl;
            cin>>amount;
        }
void financial_assit :: display ( )
        {
            basic_info :: display ( );
            academic_fit : display ( );
            cout<<setprecision(2);
            cout<<amount<<" ";
        }
void main ( )
        {
```

```
    financial_assit f[max];
    int n;
    cout<<"How many students ?"<<endl;
    cin>>n;
    cout<<"Enter the following information for";
    cout<<"financial assistance "<<endl;
    for (int i=0; i<=n–1; i++) {
        cout<<"Record no.  "<<i+1<<endl;
        f[i].getdata ( );
        cout<<endl;
        }
    cout<<endl;
    cout<<"Academic Performance for Financial Assistance"<endl;
    cout<<"-----------------------------------------"<<endl;
    cout<<"Name Roll No. Sex Course Semester Rank Amount "<<endl;
    cou<<"-----------------------------------------"<<endl;
    for (i=0; i<n–1, i++) {
        f[i].display ( );
        cout<<endl;
    }
    cout<<"-----------------------------------------"<<endl;
}
```

68.5 Module Summary

In this module we studied two important features of C++. These are function and operator overloading and inheritance. The overloading of functions are explained with examples. The overloading of several operators are also explained by different examples. The different kinds of inheritance are studied with examples and diagrammatic representations. The module ends with an exercise and references.

68.6 Self Assessment Questions

1. Write a program in C++ using operator overloading to find first *n* fibonacci numbers.
2. Write a program in C++ to perform the following using function overloading:

Area of a circle, circumference of a circle, area of a rectangle, perimeter of rectangle. Write classes for circle and rectangle.

3. Develop a program in C++ using operator overloading to add and multiply two given complex numbers.
4. Develop a program in C++ using function overloading for adding two given integer matrices and to floating point matrices separately.
5. Develop a program in C++ using operator overloading for multiplying two matrices.
6. Design a class for matrix. Overload +, , * operators.
7. Design a class for matrix. Find the inverse of a square matrix using operator overloading.
8. Design a class for matrix. Find the determinant value of the matrix using partial pivoting.
9. Design a class for matrix containing matrix multiplication and inversion operators. Use it to solve a sysem of linear equations.
10. Design a class for complex number and overload the following operators : +, –, *, /, >>, <<. Demonstrate your class with an example.
11. Write a class 'student' with data member roll no, age, sex, height, weight and member functions 'getinfo' (will read information from key board) and 'display' (will display information of an object on the screen). Use this class to read *N* such records and display all records with a proper format.
12. Assume that a bank maintains two kinds of accounts for customers, one called as savings account and the other as current account. The savings account provides compound interest and withdrawl facilities but no cheque book facility. The current account provides cheque book facility but no interest. Current account holders should also maintain a minimum balance and if the balance falls below this level, a service charge is imposed.

    Create a class account that stores customer name, account number and type of account. From this derive the classes curacct and savacct to make them more specific their requirements. Include necessary member functions in order to achieve the following tasks :

    (i) Accept deposit from a customer and update the balance

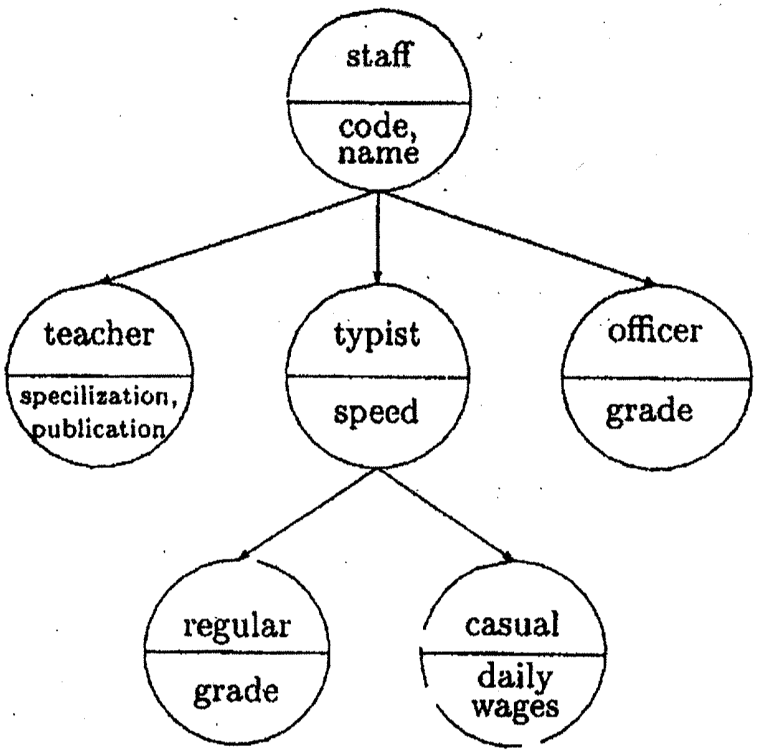
    (ii) Display the balance

    (iii) Compute the deposit interest

    (iv) Permit withdrawal and update and balance

    (v) Check for the minimum balance, impose penalty necessary and updata the balance.

13. An educational institution wishes to maintain a database of its employees. The daabase is divided into a number of classes whose hierarchical relationships are shown in the following figure. The figure also shows the minimum information required for each class. Specify all the classes and define functions to create the database and retrieve individual information as and when required.



68.7 Suggested Further Readings

1. E. Balagurusamy, Object-Oriented Programming with C++, Tata McGraw Hill.
2. D. Ravichandran, Programming with C++, Tata McGraw Hill.
3. Robert Lafore, Object-Oriented Programming in Turbo C++, Galgotia.
4. Bjarne Stroustrup, C++ Programming Language, 3ed.
5. Paul S. Wang, Standard C++ with OOP, Vikas Publishing House.
6. Steven C. Lawbr, Computer Science with C++, Vikas Publishing House.

--- 0 ---

M.Sc. Course
in
Applied Mathematics with Oceanology
and
Computer Programming

PART-II

Paper-VI

Group–B

Module No. - 69

LAB. 3 : PROGRAMMING IN C++ - III

Structure :

69.1 Introduction

69.2 Objectives

69.3 Data File Operations

- 69.3.1 Opening a file
- 69.3.2 Closing a file
- 69.3.3 Stream state member functions
- 69.3.4 Printer Output
- 69.3.5 read ( ) and write ( ) member functions
- 69.3.6 Binary file operations
- 69.3.7 File pointers
- 69.3.8 Random access file operations

69.4 Templates and Exception Handling

- 69.4.1 Class template
- 69.4.2 Exception handling

69.5 Visual C++ Features

- 69.5.1 Creating system resources
- 69.5.2 Designing dialog boxes

69.5.3 Editing a resource HotSpots
69.6 Module Summary
69.7 Self Assessment Questions
69.8 Suggested Further Readings

69.1 Introduction

This is the third and the last module of the C++ programming. In the previous two modules we assumed that the data are supplied through key board and the output will displayed on the monitor. But, these techniques have some limitations. If the data size or the output size is large then we have to store input or output in a file. From the output file we can take printout of the output whenever required. Thus the handling of data files is a very important task in any programming languages. The C++ language has special features to handling data files. Here we studied all these features.

The C++ language has a special feature for generic data processing, called *templates*. Using this feature we can write a function in a very general form, i.e., for any type of data and using this function we can find the value of this function for a particular type of data.

The errors occur during the execution of a program can be identified by a C++ feature, called *exception handling* and the corresponding action can also be taken using this feature.

69.2 Objectives

Gone through this module the students will learn the following topic:

- Data file declaration
- File creation, opening, reading, writing and closing
- Bindary file handling
- Random access file operations
- Class temple
- Exception handling
- Introduction to visual C++

69.3 Data File Operations

File is a collection of data or a set of characters or may be a text or a program. Basically, there are two types of files in the C++: sequential files and random access files. The sequential files are very easy to create than random access files. In sequential files the data or text will be stored or read back sequentially. In random access files, data can be accessed and processed randomly.

The header file, fstream.h supports the highly sophisticated input/output stream processing techniques and to implement I/O for the advanced language features such as classes, derived classes, function overloading, virtual function and multiple inheritance.

69.3.1 Opening a file

ifstream

The header file ifstream.h is a derived class from the base class of istream and to used to rad a stream of objects from a file.

```
# include <fstream.h>
void main ( )
    {
        ifstream infile;
        infile.open ("ami.dat"); // opening a file
        .....
        .....
        .....
    }
    or
    void main ( )
      }
        ifstram infile ("ami.dat");
        .....
        .....
        .....
     }
```

ofstream

The header file ofstream.h is derived from the base class of ostream and is used to write a stream of objects in a file.

```
*include <fstream.h>
void main ( )
    {
        ofstream outfile;
```

```
        outfile.open ("ami.out");//opening a file as output
        .....
        .....
        .....
    }
    or
    void main ( )
     {
        ofstream outfile ("ami.out");
        .....
        .....
        .....
     }
```

fstream

The header file fstream.h is a derived class from the base class of iostream and is used for both reading and writing a stream of objects on a file. The statement #include<fstream.h> automatically includes the header file iostream.h.

```
#include <fstream.h>

void main ( )
    {
        fstream infile;
        infile.open ("ami.dat",ios : : in//ios: out);
        .....
        .....
        .....
    }
```

When a file is opened for both reading and writing, the I/O streams keep track of two file pointers–one for input operation and other for output operation.

Note 69.3.1 The default mode for ifstream is ios::in, that of for ofstream is ios::out. However, there is no default mode for fstream.

Following is a list of member functions used as file attributes for the various kinds of file opening operation.

Name of member functions	Meaning
ios::in	opening a file for reading
ios::out	opening a file for writing
ios::app	append at the end of a file
ios::ate	seek to the end of a file upon opening instead of beginning
ios::trunc	delete a file if it exists and recreate it
ios::nocreate	open a file if a file does not exist
ios:: replace	open a file if a file does exist
ios::binary	open a file for binary mode; default is text.

69.3.2 Closing a file

The member function close ( ) is used to close a file which has been opened for file processing such as to read to write, etc.

69.3.3 Sream stae member functions

The stream stae member functions give the information status like end of a file has been reached or file open failure and son on. The following are the stream state member functions:

eof ( ); fail ( ); bad ( ); good ( )

eof ( )

The eof ( ) stream state member function is used to check whether a file pointer has reached the end ofa file character or not. If it is successful, eof ( ) member function returns a zero, otherwise returns 1. The general syntax is:

```
#include<fstream.h>
void main ( )
    {
        ifstream infile;
        infile.open ("ABC.DAT");
        while (!infile.eof ( ) )
    {
        .....
        .....
    }
}
```

fail ( )

The fail ( ) stream stae member function is used to check whether a file has been opened for input or output successfully, or any invalid operations are attempted or there is an unrecovarable error. If it fails, it returns a non-zero value.

```
#include<fstream.h>
void main ( )
    {
        ifstream infile;
        infile.open (ABC.DAT");
        while(!infile.fail( ))
        {
        cout<<"File open to fail !"<<endl;
        continue;
        .....
        .....
        }
    }
```

bad ( )

The bad ( ) stream state member function is used to check whether any invalid file operations has been attempted or there is an unrecoverable error. The bad ( ) member function returns a non-zero if it is true, otherwise returns a zero.

```
#include<fstrem.h>
#include<stdlib.h>
void main ( )
    {
        ifstream infile;
        infile.open ("ABC.DAT");
        if (infile.bad ( ))
        {
        cerr<<"File does not opened"<<endl;
        exit(1);
        }
    .....
    .....
```

```
    }
```

good ( )

The good ( ) stream state member function is used to check whether the previous file operation has been successful or not. The function good ( ) returns a non-zero if all stream stae bits are zero.

```
#include<fstrem.h>
#include<stdlib.h>
void main ( )
    {
        ifstream infile;
        infile.open ("ABC.DAT");
        if (infile.good ( ))
        {
         .....
         .....
        }
    }
```

Reading/Writing a character from a file

The get ( ) and put ( ) functions are used to read and write an alphanumeric character from a file/to a file.

Assinment CPP 69.1 Write a program in C++ which will read the content of a file and display it on the screen.

```
//Reading and displying of a file on the screen.
#include<fstream.h>
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
void main ( )
    {
        clrscr ( );
        ifstream infile;
        char fname1 [12];
        char ch;
        cout<<"Enter a file name";
```

```
cin>>fname1;
infile.open(fname1);
if(infile.fail())
    {
        cout<<"No such file exists !"<<endl;
        exit(1);
    }
while(!infile.eof())
    {
        ch=(char)infile.get();
        cout<<ch;
    }
    infile.close();
    }
```

Assinment CPP 69.2 Write a program in C++ which will read the content of a file and display it on the screen. Use command line arguments.

```
//Reading and displying of a file on the screen.
//Input file is taken from comman line.
#include<fstream.h>
#include<iostream.h>
#include<stdlib.h>
#include<<conio.h>
void main (int argc, char *argv [ ])
    {
        clrscr ( );
        ifstream infile;
        char ch;
        infile.open (argv[1];
        if (infile.fail ( ))
        {
         cout<<"No such file exists !" <<endl;
         exit (1);
        }
    while (!infile.eof( ))
        {
```

```
        infile.close ( );
        ofile.close ( );
    }
```

3.4 Printer Output

nding data to the printer is similar to sending data to a disk file. DOS predefines a number of special filenames for dware devices. They are listed below.

CON	Console (keyboard and screen) output
AUX or COM1	First serial port
COM2	Second serial port
PRN or LPT1	First parallel port
LPT2	Second parallel port
NUL	Dummy (non-existent) devices

ssinment CPP 69.4 Write a program in C++ which will print a file to printer file. Use command line arguments.

```
inting of a file using command line argument.
nclude<fstream.h>//for file functions
nclude<process.h> //for exit
oid main(int argc, char* argv [ ] )
    {
    if (argc ! = 2)
    {
        cout<<"/n Incorrect number of inputs !"<<endl;
        exit(1);
    }
    char ch;
    ifstream infile;
    infile.open (argv[1]; //open file for input
    if(!infile)
    {
        cout<<"\n Can't open ! "<<argv[1];
        exit (1);
    }
    ofstream outfile;
```

```
        ch =(char)infile.get ( );
        cout<<ch;
        }
        infile.close ( );
}
```

Assiment CPP 69.3 Write a program in C++ which will copy a file to another file. use command line arguments.

```
//Copy a file to another file using command line argument.
#include<fstream.h>
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
void main (int argc, char *argv[ ] )
        {
                clrscr ( );
                ifstream ifile;
                ofstream ofile;
                char ch;
                ifile.open (argv[1];
                if (ifile.fail ( ) )
        {
                cout<<"No such file exists !"<<endl;
                exit (1);
        }
        ofile.open (argv [2]);
        if (ofile.fail ( ) )
        {
                cout<<"Unable to create a file !"<<endl;
                exit (1);
        }
        while (!ifile.eof ( ) )
        {
                ch=(char) ifile.get ();
                ofile.put (ch); // or we can write 'ofile<<ch;'
        }
```

```
        outfile.open ("PRN"); //open file for printer
        while (infile.get(ch) !=0)
            outfile.out (ch); //write the char. to printer
    }
```

Assinment CPP 69.5 Write a program in C++ to convert all lower case alphabets to upper case of a file.

```
//Converting a lower case to upper case letters
#include<fstream.h?
#include<string.h>
#include<stdlib.h>
#include<ctype.h>
void main (void)
    {
    ofstream ofile;
    ifstream ifile;
    char fname1 [12], fname2 [12];
    char ch, uch;
    cout<<"Enter a file name to be converted \n";
    cin>>fname1;
    cout<<"New file name\n";
    cin>>fname2;
    ifile.open (fname1);
    if(ifile.fail ( ))
    {
        cout<<"No such file exists !"<<endl;
        exit (1);
    }
    ofile.open (fname2);
    if (ofile.fail ( ))
    {
        cout<<"Unable to create a file !"<<endl;
        exit (1);
    }
    while (!ifile.eof ( ))
    {
```

```
ch=(char) ifile.get ( ); //read a char. from the input file
uch=toupper (ch); //convet to upper
ofile.put (uch); // write the converted char. to output file
}
ifile.close ( );
ofile.close ( );
}
```

Assinment CPP 69.6 Write a program in C++ to read the sides of some tringles from a file (the file name should be supplied) and write the sides and areas of all the triangles to a file whose name is to be supplied by user.

```
#include<fstream.h>
#include<stdlib.h>
#include<math.h>
void main ( )
    {
        ifstream ifile;
        ofstram ofile;
        char fname1 [12], fname2 [12];
        float x, y, z;
        cout<<"Enter the input file";
        cin>>fname1;
        ifile.open (fname1, ios::in);
        cout<<"Enter the output file";
        cin>>fname2;
        ofile.open (fname2, ios::out);
        if (ifile.fail ( ))
        {
            cerr<<"File does not exist !"<<endl;
            exit (1);
        }
        if (ofile.fail ( ))
        {
            cerr<<"File creation error !"<<endl;
            exit (1);
        }
```

```
while(!ifile.eof())
    {
        ifile>>x>>y>>z; //reading of sides from input file
        float s = (x+y+z)/2;
        flat area =sqrt (s*(s-x)*(s-y)*(s-z));
        //computation of area of the triangle
        //writing sides and area to the output file
        ofile<<x<<"\t"<<y<<"\t"<<zz<<"\t"<<area<<" "<<endl;
    }
ifile.close();
ofile.close();
}
```

A sample of input / output :

Input file 'ss.dat' shown below.

```
2   3   4
3   4   5
5   6   7
1   2   3
```

Output file 'ss.out.

```
2   3   4   2.904737
3   4   5   5
5   6   7   14.696939
1   2   3   0
```

Assinment CPP 69.7 Write a program in C++ to merge two files into a one file heading.

```
#include<fstream.h>
#include<string.h>
#include<stdlib.h>
#include<ctype.h>
void main (void)
{
    ofstream ofile;
    ifstream ifile;
    char fname1 [12], fname2 [12], fname3 [12];
```

```
char ch;
cout<<"Enter two files which are to be maerged \n";
cin>>fname1>>fname2;
cout<<"Enter new file name\n";
cin>>fname3;
ifile.open (fname1);
if (ifile.fail ( ))
        {
        cout<<"Does not exists the file "<<fname1<<end1;
        exit (1);
        }
        ofile.open (fname3);
        if (ofile.fail ( ) )
        {
                cout<<"Unable to create a new file !" <<endl;
                exit (1);
        }
        while (!ifile.eof ( ) )
        {
                ch=(char) ifile.get ( ); //read a char, from first input file
                ofile.put (ch); //write the char.to output file
        }
ifile.close ( );
ifile.open (fname2); //opening of second iniput file
if (ifile.fail ( ) )
        {
                cout<<"Does not exists the file "<<fname2<<endl;
                exit (1);
        }
while (!ifile.eof ( ) )
        {
        ch = (char) ifile.get ( ); //read a char. from second input file
        cfile.put (ch); //write the char. to output file
        }
        ifile.close ( );
        ofile.close ();
        }
```

69.3.5 Read ( ) and write ( ) member functions

The member functions read ( ) and write ( ) are used to read and write a stream of objects from a specified file respectively.

Reading an object from a file

The read ( ) member functions is used to get data for the stream of object from a specified file.

The general syntax of the read ( ) member function is

```
infile.read ( (char *) & x, sizeof (x) );
```

Writing an object to file

The write ( ) member function is used to save the stream of objects on a specified file.

The general syntax of write ( ) member function is

```
infile.writer ((char *) & x, sizeof (x) );
```

The contents of the object x are then written to disk. using the write ( ) function. This function, like the other output functions we have seen, is a member of ostream. It takes two arguments: the address of the object to be written, and the length of the object in bytes. We use the size of operator to find the length of the object x. The address of the object must be cast to type pointer to 'char'.

69.3.6 Binary file operatons

In C++, in general the file stream operations are performed in text mode but it also supports binary file operations. A binary file is a sequential access file in which data are stored and read back one after another in the binary format instead of ASCII characters. For example, a binary file contains integer, floating point number, array of structures, etc. Binary file processing is well suited for the design and development of a complex data base or to read and write a binary information.

The text file created by C++ can be edited by an ordinary editor or by a word processor. The text file can easily be transferred from one computer system to another. On the other hand a binary file is more accurate for numbers because it stores the exact internal representation of a value. There are no conversion errors or round off erros. Sving data in binary format can be faster as there is no conversion taking place while storing data to a file. The binary format data file normally takes less space. However, binary format data file cannot be easily transferred from one computer system to another due to variations in the internal representation of the data from one computer to another.

Assinment CPP 69.8 Write a program to read an array of class object of student information such as name, age, sex, height and weight from the keyboard and to store them on a specified file using read ( ) and write ( ) member

functions. Again, the same file is opened for reading and displaying the contents of the file on the screen.

```
//array of class objects and binary file operation.
#include<fstream.h>
#include<iostream.h>
#include<iomanip.h>
const max=100;
class studentinfo
    {
    protected :
        char name [20];
        int age;
        char sex;
        float height;
        float weight;
    public :
        void display ( );
        void getdata ( );
    };
void studentinfo : : getdata ( )
    {
        cout<<"Name  :"; cin>>name;
        cout<<"Age  :"; cin>>age;
        cout<<"Sex  :"; cin>>sex;
        cout<<"Height:"; cin>>height;
        cout<<"Weight:"; cin>>weight;
    }
void studentinfo : : display ( )
    {
        cout<<name<<setw (5)<<age<<setw (10)<<sex;
        cout<<setw (5)<<height<<setw (5)<<weight<<endl;
    }
void main (void)
    {
        studentinfo s [ax];
        ftream infile;
```

```
    char fname [11];
    int i, n;
    cont<<"Enter a file name to be stored" <<endl;
    cin>>fname;
    infile.open (fname, ios :: in \\ios :: out);
    cout<<"How many objects are to be stored">>endl;
    cin>>n;
    //reading from the keyboard
    cout<<"Enter the following information"<<end1;
    for (i=0, i<n;i++)
        {
        cout<<endl<<"Student-"<<i+1<<endl;
        x[i].getdata ( );
        }
    //storing onto the file
    infile.open (fname, ios :: out);
    cout<<"Storing onto the file ..." <<endl;
    for (i=0, i<n; i++)
        infile.write ((char*) & x[i], sizeof (x[i]));
    infile.close ( );
    //reading from the file
    infile.open(fname,ios::in);
    cout<<"Reading from the file...."<<endl;
        {
        infile.read((char*) &x[i], sizeof(x[i]));
        x[i].display ( );
        }
    infile.close ( );
} //end of main
```

69.3.7 Fiel pointers

Each file object has associated with it two integer values called the **get pointer** and the **put pointer.** These are also called the **current get position** and the **current put position** or simply the **current position.** These values specify the byte number in the file where writing or reading will take place.

The seekg ( ) and tellg ( ) function allow to set and examine the get pointer and the seekp ( ) and tellp ( )

functions perform these same actions on the put pointer.

Specifying the offset

he seekg ( ) function can be used in two ways. We have seen the first, where the single argument represents the position. We can also use it with age arguments, where the first arguments represents an offsef from a particular location from which the offst is measured. There are three possibalities for the second arguments: beg is the begining of the file, cur is the current pointer position and end is the end of the file.

The statement

seekp (–10, ios:: end);

indicate the position of the 10th byte from the end of he file.

The following program finds a record of a person from a binary file.

```
//seeks particular record in a file
#include<fstream.h>
calss person
    {
        prtected:
            char name [40]; //person's name
            int age;        // person's age
        public :
            void showdata ( )   //display person's record
            {
            cout<<endl<<"Name:  "<<name;
            cout<<endl<<"Age :  "<<age;
        }
    }
void main ( )
    {
        person p;
        ifstream infile;//declartion of input file
        infile.open ("Person.dat"); //open file for input
        infileseekg (0,ios::end);  //goto 0 bytes from end
        int endposition=infle.tellg ( ); //find where we are
        int n=endposition/sizeof (person); //number of records
        cout<<endl<<"There are "<<n<<" records in the file"<<endl;
        cout<<endl<<"Enter record number";
```

```
        cin>>n;
        int position=(n-1)*szeof(person);
        infile.seekg (position); //bytes from begin read one person
        infile.read((char*) &p,size of (p));
        p.showdta ( );// display the record
    }
```

69.3.8 Random access file operations

The random access file must be opened with the following mode of access:

Mode of access	Meaning
ios::in	in order to read a file
ios::out	in order to write a file
ios::ate	in order to append a file
ios::binary	binary format

The use of these modes is illustrated in the following program segement.

```
#include<fstream.h>
void main ( )
    {
        fstream file;
        file.open(fnamel ios::in//ios::out//ios::ate//ios::binary);
        .....
        .....
    }
```

The file enumerated values are listed in the following table.

Enumerated value	File position
ios::beg	from the beginning of the file
ios::cur	from the current file ponter position
ios::end	from the end of the file

* seekg ( ) The seekg ( ) member function is used to position file operations for random input operations.

```
#include<fstream.h>
void main ( )
    {
```

```
fstream infile;
.....
.....
infile.seekg (40); //goto byte number 40
infile.seekg (40, ios::beg);  //same as the above
infile.seekg (0, ios::end); //goto the end of the file
infile.seekg (0); //goto start of the file
infile.seekg (-10, ios::cur); //the file pointer is moved back
                              //from current position by 10 bytes
}
```

*seekp (The seekp ( ) member function is used to position file operations for random output operations.

* tellg ( ) This member function is used to check the current position of the input stream.

* tellp ( ) The tellp ( ) member function is used to check the current position of the output stream.

69.4 Templates and Exception Handling

Template is a method for writing a single function or class for a family of similar functions or classes in a generic manner. When a single function is written for a family of similar functions, it is called as a 'function template'. In this function at least one formal argument is generic.

C++ provides certain features with the capability to define a single function for a group of similar functions. When a single function is written for a family of similar functions, they are called as function templates. The main advantage of using function template is avoiding unnecessary repetition of the source code. The object code becomes more compact and efficient than the conventional way of declaring and defining the functions. Secondly, full data type checking is carried out. The function template does not specify the actual data types of the arguments that a function accepts but it uses a generic or parameterized data type.

The general syntax for declaring a function template in C++ is,

```
template <class T>
T function_name (T arguments)
    {
    .....
    .....
    return (T);
    }
```

or,

```
template <class T>
T function_name (T arguments)
        {
        .....
        .....
        return (T);
        {
```

Assinment CPP 69.9 Write a program to define a function template for summing an array of inegers and an array of floating point numbers.

```
#include<iostram.h>
//definition of generic function 'sum'
template <class T>
T sum (T *array, int n)
        {
                T temp=0;
                for (int i=0;i<n;i++)
                        temp+=array [i];
                return(temp);
        }
        }
int sum(int *a, int n); //declaraion of function sum for integer
data float sum(float *a, int n); //declaration of function sum for
                                 //float data
void main ( )
        {
        int n=3, sum1;
        float sum2;
        static int a[3]={10, 15, 30};
        static float b[3]={10.5,1.8,3.4};
        cout<<"Sum of integers "<<sum(a,n);
        cout<<"Sum of floating point numbers"<<sum(b,n);
```

Assinment CPP 69.10 Write a program to define the function template for swapping two items of the various data types such as integer and floating point numbers.

```
#include<iostream.h>
//definition of the generic function swap
template <class T>
T swap (T &x, T &y)
        {
        T temp;
        temp=x;
        x=y;
        y=temp;
        return(0);
        {
int swap(int &a, int &b);
float swap(flaot &a,float &b);
void main ( )
        {
        int ix, iy;
        float fx, fy;
        cout<<"Enter two integers"<<endl;
        cin>>ix>>iy;
        cout<<"Enter two floating numbers"<<endl;
        cin>>fx>>fy;
        swap(ix,iy);
        cout<<"After swapping integers "<<"ix="<<ix<<"iy="<<iy<<endl;
        swap(fx,fy);
        cout<<"After swapping floating numbers "<<"fx="<<fx<<"fy="<<fy<<endl;
}
```

69.4.1 Class template

A class template is a class definition that describes a family of related classes. C++ offers the user the ability to create a class that contain one or more types that are generic or parameterized. The manner of declaring the class template is the same as that of a function template. The keywod template must be inserted as a first word for defining a class temple.

The general syntax of the class template is

```
template <class T>
        class user+defined_name
        {
                private:
                .....
                .....
                public :
                .....
                .....
        };
```

Assinment CPP 69.11 Write a program to find sum of two numbers using class template.

```
#include<class T>
        class sample
        {
        private :
                T value, value1, value2;
        public :
                void getdata ( );
                void sum ( );
        };
template <class T>
        void sample<T> ::getdata ( )
        {
                cin>>value1>>value2;
        }
template <class T>
        void sample<T>::sum( )
        {
                T value;
                value=value1+value2;
                cout<<"Sum="<<value<<endl;
        }
void main ( )
        {
```

```
        sample <int> a;
        sample <float> b;
        cout<<"Enter any two integers"<<endl;
        a. getdata ( );
        a. sum ( );
        cout<<"Enter any two floating point numbers"<<endl;
        b.getdata ( );
        b. sum ( );
}
```

The following example illustrates the use of construction and destruction in class template.

```
//construction with single argument and destructor of class
//template
#include<iostream.h>
template<class T>
        class sample
        {
        private
                T value;
        public :
                sample (T n); //construction
                ~sample ( ); //destruction
                void display ( );
        };
template <class T>
sample<T>::~sample (T n) : value (n) { {

template <class T>
        sample<T>::~sample ( ) { }

void sample<T>::display ( )
        {
                cout<<"Content of the value="<,value<<end1;
        }
        void main ( )
        {
```

```
    sample <int> obj1(10);
    cout<<"Integer : "<<end1;
    obj1.display ( );
    sampleMfloat>obj2(-25.05);
    cout<<"Floating point number:"<<end1;
    obj2.display ( );
    sampleMMdouble>obj3(167623L);
    cout<<"Double precision number :"<<end1;
    obj3.display ( );
}
```

69.4.2 Exception handling

An exception is an error or an unexpected event. The exception handler is a set of codes executes when an exception occurs. Exception handling is one of the most recently added features and perhaps it may not be support by much earlier version of the C++ compilers.

Exception handling in C++ provides a better method by which the caller of a function can be informed that some error condition has occurred. The following keywords are used for handling error functions in C++.

try
catch
throw

Whenever a caller of a function finds an error, it is difficult to check or trace these critical errors. In C++, these types of error conditions are handled easily using the above keywords. The program must be developed with exception handling in such way that it determines the possible errors the program might encounter and then include codes to handle them.

Exception handling provides another way to transfer control and information from a point in the execution of a program to an exception handler. A handler will be invoked only by a throw expression in a code executed in the handler's try block or the function called from the handler's try block.

```
class sample
    {
    private :
        char *str;
    public :
        enum{minsize=1, maxsize=1000};
        sample ( );
```

```
            sample(int);
            void display ( ) ;
        };
sample::sample (int size)
{
        if((size<minsize)||(size>maxsize))
            throw(size);
        str=new char[size];
        if(str==0)
            throw("Out of memory\n");
        }
        void funct(int n)
        {
            try
        {
            sample obj(n); //bject is created
        }
            catch(int k)
        {
            cerr<<"Out of range ... "<<endl;
            funct(sample::maxsize);
        }
}//end of funct definition
```

69.5 Visual C++ Features

The Microsoft Visual C++ compiler includes several advanced tools that are useful for proram development. This section we examines the creation of bitmaps, cursors, and icons from within the Visual C++ IDE (Integrated Development Environment). The Visual C++ compiler helps to locate syntax errors during the build (compile and link) operation. Syntax errors are often the easiest to fix because of the detailed help provided by the online help facilities.

69.5.1 Creaing system resources

Customizing a Wndows application with own icons, cursors, bitmaps, and dialog boxes is easy with Microsoft's Visual C++ IDE. This is not a just a compiler, it is also an easy-to-use, powerful resource editor.

Designing bitmaps

Here we discuss how to use the Visual C++ IDE to draw a bitmap. All other graphic figures, such as icons and cursors can be created in a similar manner.

The Visual C++ IDE allows to design device-independent colour bitmap images. These bitmaps are functionally device independent with respect to resolution. The image file format allows to create a bitmap that always looks the same, regardless of the resolution of the display on which it appears.

The first step to create an application resource such as a bitmap is to click on the Insert | Resource .. menu item.

The resulting dialog box, shown here, shown the drop-down list that display the kinds of resources available. Beceause we want to creat a bitmap, this resource is highlighted. Simply press ENTER to confirm selection.

Creating a bitmap with the Visual C++ resource editor is about as easy as creating a picture with Windows Paintbrush. The Visual C++ resource editor first presents with a blank bitmap grid and the drawing tools toolbar.

We can use the toolbar to select the brush size, the brush colour, and various drawing modes, such as fills and predefined shapes.

To set a resource's properties, begin by pressing ALT-ENTER. This step displays the particular re-source's Properties dialog box. The following illustration shows the Bitmap Properties dialog box with the details of the bitmap's width, height, colors, filename and save compressed properties.

69.5.2 Designing dialog boxes

The initial steps required for creating a dialog box resource are identical to those described previously for creating a bitmap resource. First, choose the kind of resource we wish to create as shown earlier. However, select the Dialog option.

The objects toolbox allows to place a variety of controls in the dialog box design. These include (starting at the top and proceeding from left to right, not including the 'arrow' selection control) bitmap, label, edit box, frame, button, checkbox, radio button, combo box, list box, orizontal scroll bar, vertical scroll bar, and a user-defined control.

69.35.3 Editing a resource HotSpots

A cursor resource differs slightly from a bitmap or an icon resource in that it can contain a hotspot. A cursor's hotspot represents the part of the image that registers the cursor's screen coordinates.

Creating a cursor involves the same initial steps used for creating bitmaps and dialog boxes. This process is started by first selecting the Inset|Resource ... menu item command and then choosing the cursor resource.

A cursor's hotspot is set by first clicking on the hotspot button to the right of the Hotspot:label in the design toolbar. Once we have clicked on the button, simply move the mouse pointer into the cursor's bitmap design and click on the appropriate cell.

69.6 Module Summary

In this module we introduced the data file handling feature of C++. The concept of generic data handling is also discussed through templates. A brief discussion on exception handling and visual C++ are incorporated here.

69.7 Self Assessent Questions

1. Write a program in C++ which will read the content of a file and display it on the screen. Use command line arguments.
2. Write a program in C++ which make a copy of a file. Use command line arguments.
3. Write a program in C++ which will print a file on the printer. use command line arguments.
4. Suppose a file TRIAL.DAT contains the sides of 50 traingles. Write a program in C++ which read these sides and calculate area of each triangle and write these sides and area of another file TRIA.OUT.
5. Write a program in C++ to merge two files into a one file heading.
6. Write a program in C++ to read the class object of student_info such as name, age, sex, height and weight from the keyboard and store them on a specified file using read ( ) and write ( ) member functions. Again, the same file is opened for reading and displaying the contents of the file on the screen.
7. A file 'emp.dat' contain employee code, name, basic pay of some employee. Write a program in C++ to read records from 'emp.dat' and calculate D.A. (60% of basic pay), H.R.A. (15% of basic pay but not more than Rs. 2000/-), M.A. (Rs. 100/- for all employee) and gross pay (sum of basic pay, D.A., H.R.A., and M.A.) and write all inforamtion to the file 'emp.out' and then display all information form the file 'emp.out' on the screen with proper format.
8. Write a function template for finding the eminimum value contained in any array. Use it to find the minimum from a set of integers and from a set of floats.
9. Write a class template to represent a generic vector. Include member functions to perform the follwoing tasks :

(i) To create the vector

(ii) to modify the value of a given element

(iii) To multiply by a scaler value

(iv) To display the vector of the form (3,4,5,6,7,....).

10. Write a program containing a possible exception. use a try blcok to throw it and a catch block to handle it properly.

11. Write a program with the following :

(i) A function to read two double type numbers from keyboard

(ii) A function to calculate the division of these two numbers

(iii) A try block to detect and throw an exception when a wrong type of data is keyed in

(iv) A try block to detect and throw an exception if the condition 'divide-by-zero' ocurs

(v) Appropriate catch block to handle the exceptions thrown.

69.8 Suggested Further Readings

1. E. Balagurusamy, Object-Oriented Progamming with C++, Tata McGraw Hill.
2. D. Ravichandran, Programming with C++, Tata McGraw Hill.
3. Rober Lafore, Object-Oreinted Programming in Turbo C++, Galgotia.
4. Bjarne Stroustrup, C++ Programming Language, 3ed.
5. Paul S. Wang, Standard C++ with OOP, Vikas Publishing House.
6. Steven C. Lawbr, Computer Science with C++, Vikas Publishing House.

--- 0 ---

M.Sc. Course in
Applied Mathematics with Oceanology
and
Computer Programming

Part-II

Paper-VI **Group-B**

Module No. 70
Lab. 4: Database Management System (DBMS)-I

Structure

70.1. Introduction
70.2. Objectives
70.3. What is database management ?
70.4. How to start database software ?
70.5. Creation of a database file
70.6. Listing contents of database file
70.7. Searching for information in the Database file with LOCATE
70.8. Searching information within a field
70.9. Editing data in a record
70.10. Editing multiple records with BROWSE
70.11. Editing selected fields with BROWSE
70.12. Moving between records
70.13. Replace field contents with REPLACE
70.14. Deleting records with DELATE
70.15. Recalling deleted records
70.16. Removing records permanently with PACK
70.17. Condition clause
70.18. Modifying structure of DBF file
70.19. Entering data in Memo field
70.20. Listing the memo field
70.21. Sorting with SORT
    70.21.1. Sorted on multiple fields

70.22. Indexing database file with INDEX
    70.22.1. Advantages of indexing over sorting
    70.22.2. Automatic updating of index files
    70.22.3. Searching data in indexed file
70.23. File utilities
70.24. Memory variable and functions
    70.24.1. Creating memory variable using = command
70.25. Printing of output
70.26. Displaying and manipulating memory variables
    70.26.1. Time and Date functions
    70.26.2. More functions regarding date
70.27. Converting numeric data to characters with STR()
70.28. Command files
    70.28.1. Creating a command file
70.29. Font setting in Dot Matrix Printer
70.30. Module Summary
70.31. Self Assessment Questions
70.32. Further Suggested Readings

70.1. Introduction

A database management system (DBMS) is a collection of programs that enables us to store, modify and extract information from a database. There are different types of DBMS ranging from small systems that run on personal computer to huge systems that run on mainframes. The following are examples of database applications:

- Computerized library systems
- Railway reservation systems
- Flight reservation systems
- Inventory system
- Admission (to an institution) system

etc.

A database management system is a piece of software that provides services for accessing a database, while maintaining all the required data.

There are several packages are available to implement the DBMS. Some of them are dBase III (plus), Foxbase, Foxpro, etc.

Here we discuss mainly about the package Foxbase.

70.2. Objectives

Through this module, the students will learn the following fundamental topics on data bases.

- Database Management System
- Create a database file
- List/Display of records
- Searching record
- Modify/Edit records
- Replace of fields
- Deletion of records
- Sorting
- Indexing
- File utilities
- Memory variables
- Printing
- Time and Date functions
- Command file

70.3. What is database management ?

Managing a database primarily involves the following tasks:

- CREATE a new database file
- ADD a new data to the database
- DELETE data from the database
- EDIT (MODIFY) data on the database
- SEARCH the database for types of information
- SORT the database into some meaningful order
- PRINT data from database.

70.4. How to start database software ?

(To start Foxpro)

Type `foxpro` where the software foxpro is available then press `enter`. Then a screen will display containing a small window called command window. All commands are typed within this command window.

(To start Foxbase)

Type `foxplus` where the software foxbase is available then press `enter`. Then a screen will display containing the word Foxbase in giant letters with a dot (.) prompt. Type all the commands after the dot prompt.

To exit from database software type `QUIT` and press `ENTER`.

70.5. Creation of a database file

Type `CREAT <filename>` to crate a new database file. The number of characters in file name should be not more than 8. The extension of the database file is .dbf. The extension is not necessary while creating a dbf (the database file generally referred as **dbf** file) file. After typing the command `CREAT <filename>` a window will display containing several headings. This command actually creates the **structure** of the database file. The structure is the name, type and width of the fields of the dbf file. The first line is look like the following form.

```
Name                    Type      Width      Dec
```

Field name should satisfy the following conditions:

1. Should be start with an alphabet
2. Length not more than 10 characters
3. Combinations of alphanumeric characters and underscore (_)

The fields can be classified as seven different types:

1. Character
2. Numeric
3. Date
4. Logical
5. Memo
6. Float (not available in foxbase)
7. General (not available in foxbase)

`Width` is the number of digits to be used in the field and `Dec` is the number of digits after decimal point.

Enter one by one filed name, data type, width and dec. After completion of the entry of field names press OK button (in foxpro) /`Ctrl and End` (in foxbase).

After completion of the above step the following message will display on the screen.

```
Input data record now?
```

Now we can enter the data or can enter any time. If you like to enter the data now press `Yes` otherwise press `No`. If the answer is Yes then a new screen will come on the screen containing the fields names together with a fixed space for entering data. Now we can enter the data one by one. To save the entire date press `ctrl and end` keys together.

The structure of a typical database file containing the filed names, data type, width and dec is shown below.

Name	Type	Width	Dec
NAME	Character	25	
ADDRESS	Character	35	

SEX	Character	1	
DJOIN	Date	8	
BASIC	Numeric	5	0
DA	Numeric	7	2
HRA	Numeric	7	2
PF	Numeric	5	0
PUJAADV	Numeric	5	0
GROSS	Numeric	7	2
TDEDUC	Numeric	7	2
NETPAY	Numeric	7	2
REMARK	Memo		

Suppose the above structure is stored into the database file EMP.DBF.

10.6. Listing contents of database file

Two commands `LIST` and `DISPLAY ALL` are used to display the content of a database file. The `LIST` command will display all the records without break, but, `DISPLAY ALL` will display all the records page wise.

Listing selected fields

. LIST NAME, DJOIN, BASIC, SEX

will display only the fields NAME, DJOIN, BASIC and SEX of all employee.

. DISPLAY ALL NAME, DJOIN, BASIC, SEX

the action is same but the records will display page wise.

. DISPLAY ALL NAME, DJOIN, BASIC FOR BASIC>5000

will display the name, date of joining and basic of all employees whose BASIC is greater than 5000.

LIST OFF

will display all fields of all records without displaying the record number.

LIST OFF NAME, DJOIN, BASIC

the action is same but will display the fields NAME, DJOIN and BASIC.

. LIST NAME, BASIC+DA+HRA

will display the NAME and sum of BASIC, DA and HRA of all employees.

. LIST NEXT 4

will display all fields of next four records starting from the current record.

. LIST TO PRINT

will PRINT all fields of all records to the default printer.

. LIST NAME, NETPAY TO PRINT

will PRINT NAME and NETPAY of all records to the default printer.

. LIST NEXT 4 TO PRINT

will PRINT next 4 records to the default printer.

When the above commands execute then at the top of each page (screen/paper) the fields names will display /print. But if we don't need to display/print the field names then we can OFF it by using the following command.

. SET HEADING OFF

Again, if we need to display/print the heading (i.e., the filed names) then we can use the following command.

. SET HEADING ON

70.7. Searching for information in the Database file with LOCATE

. USE EMP (open the database file EMP)

. LOCATE FOR NAME="Arun"

it will display the record number of the record whose name is exactly "Arun".

. DISPLAY

will display the content of that record.

. CONTINUE

will search for next record.

If there is any record with NAME equal to "Arun" then will display the record number otherwise will display the following message.

```
End of LOCATE scope
```

70.8. Searching information within a field

. LOCATE FOR "Kumar" $NAME

The $ symbol preceding a field name tells to look for a specified information anywhere within that filed.

The command .CONTINUE will continue the search for next record.

. LIST FOR "Kumar" $NAME

will display all fields of all records which contain "Kumar" within the NAME field.

70.9. Editing data in a record

. USE EMP

. EDIT

will edit the current record.

. EDIT 4

. EDIT RECORD 4

both the commands will edit record number 4.

. EDIT FIELDS NAME, BASIC

is used to edit only NAME and BASIC fields for current record.

. EDIT FOR BASIC>2000

will display the first record which meets this condition for editing. To get next record with BASIC>2000 press PgDn key.

70.10. Editing multiple records with BROWSE

. BROWSE

May use the following key.

Down arrow ↓	goto next record
At end	will ask:
	Add new record (Y/N) ?
PgUp	will show the previous page
PgDn	will show the next page
(Ctrl)^(right arrow) →	goto next field (right)
(Ctrl)^(left arrow) ←	goto next previous (left)
(Ctrl)^ Home	Will display 5 options
	Bottom : goto bottom of file
	Top : goto top of file
	Lock : will lock fields left side of the screen
	Records no. & freeze : will freeze a particular field
(Ctrl)^ End	End of Browse

70.11. Editing selected fields with BROWSE

. BROWSE FIELDS NAME, BASIC

will edit only the fields NAME and BASIC.

. BROWSE WIDTH 8

will display all fields in 8 characters.

70.12. Moving between records

. GOTO 4

. GO 4

. 4

The control goes to the record number 4, i.e., the current record becomes 4 (note that these four commands have the same effect).

. GO BOTTOM (goto the end of the file)
. GO TOP (goto the top of the file)

70.13. Replace field contents with REPLACE

. GOTO 4
. REPLACE BASIC WITH 3000
1 record replaced
This command will changed the BASIC of the record 4 by 3000.

. REPLACE DA WITH BASIC* .40 ALL
will changed the DA of all records by 40% of the corresponding BASIC, i.e., if the BASIC of a record is 2000 then the DA of this record changed to 2000*.40=800.

. REPLACE HRA WITH 2000 FOR HRA>2000
. REPLACE HRA WITH 2000 FOR HRA>2000 ALL
. REPLACE HRA WITH 2000 FOR HRA>2000 WHILE EOF()
These three commands (have same effect) will change the HRA of all records if the HRA is greater than 2000.

70.14. Deleting records with DELATE

(will mark the records for deletion)

The following two commands
. GOTO 3
. DELETE
and
. DELETE RECORD 3
have the same effect. These commands marked the record number 3 for deletion.
. DELETE NEXT 4
will mark next 4 (from the current record) records for deletion.
. DELETE ALL FOR BASIC<2000
This command will mark all the records whose BASIC is less than 2000.

70.15. Recalling deleted records

. RECALL RECORD 3
and
. GOTO 3
. RECALL
have the same effect. These commands unmarked the record number 3 (if it is marked or unmarked previously).

. RECALL NEXT 5

will unmark next 5 records.

. RECALL ALL

will unmark all records.

70.16. Removing records permanently with PACK

. PACK

will delete all marked records permanently from the database file.

70.17. Condition clause

Operator	Description	Example
=	Equal to	LIST FOR BASIC=2200
<	Less than	LIST FOR BASIC<2500
>	Greater than	LIST FOR BASIC>3000
<> or #	Not equal to	LIST FOR BASIC<>2500
<=	Less than or equal to	LIST FOR BASIC<=2500
>=	Greater than or equal to	LIST FOR BASIC>=3500
$	Substring comparison	LIST FOR "Maiti" $NAME

70.18. Modifying structure of DBF file

. USE EMP

. DISPLAY STRUCTURE or DISP STRU

This command will display the name, type and width of all fields of the database file EMP.

NOTE : ***In database software (foxbase/foxpro) full name of the commands need not be typed. Only the first four characters are to be typed.***

. MODIFY STRUCTURE or MODI STRU

A new screen will come containing the fields of the database file EMP. Now we can modify the name, type and width of the fields. Also we can enter a new field or we can delete an existing field.

- Press `ctrl(^)N` to create a blank field in the database structure.
- Now we can type the desired field name, type and width.
- Press at the end `ctrl(^)END.`

To delete an existing field press `ctrl(^)U` on the respective field.

70.19. Entering data in Memo field

To enter data in the memo field, press the `ctrl(^)Home` keys (while the cursor is positioned in the memo field).

Then enter data. The keys `PgUp, PgDn, Tab, Del, Ins, ctrl(^)T` (to delete word), `ctrl(^)Y` (to delete a line) may be used.

At the end press `ctrl(^)End` or `ctrl(^)W` to save and exit.

70.20. Listing the memo field

. LIST

will display the contents of the memo filed.

. LIST NAME, REMARK

will display NAME and content of REMARK.

. SET MEMOWIDTH TO 30

will display memo field for 30 characters per line.

70.21. Sorting with SORT

Let us sort the EMP database file on the NAME filed and store the sorted data in a new file, say, EMPS. The following command do this task.

. USE EMP
. SORT ON NAME TO EMPS

The sort command will sort the database file (in use) on the NAME field and store sorted data in the EMPS database file. This command sort the file on the NAME field in ascending order. If the EMP file contains 50 records then after sorting the following message will display on the screen.

```
100% Sorted.        50 Records Sorted
```

To sort data in descending order, "/D" may be added after the filed name. For example, to sort the EMP database file in descending order on the NAME field and store the sorted data in the EMPSS database file the following command to be used.

. USE EMP
. SORT ON NAME/D TO EMPSS

It may be noted that the records of the file EMP remain same in their original order, but, the order of the records in the sorted file EMPSS is changed accordingly.

70.21.1. Sorted on multiple fields

The following command will sort the file EMP with respect to the fields NAME and ADDRESS and store to the file EMP1.

. USE EMP
. SORT ON NAME,ADRESS TO EMP1

The file EMP actually will sort according to the field NAME in ascending order. But, if the NAME of two or more records are same then these records (the records with same NAME fields) will sort according to their ADRESS field in ascending order.

70.22. Indexing database file with INDEX

Indexing is another way to arrange records in the desired order. Whereas the SORT command create another database file to store sorted records, index does not create another database file, it creates an INDEX file. For example, if we create an index file for the EMP database file on the NAME field, it will arrange all the names in alphabetical order and write them in the index file. It will also write the record number against each name, where that name can be found in the EMP database file.

The following command
. USE EMP
. INDEX ON NAME TO NAMEIDX
will index the EMP file and the following message will display on the screen.

```
100% indexed 50 Records indexed
```

The foxbase/foxpro will assigns ".idx" as the file extension of index file.

70.22.1. Advantages of indexing over sorting

Indexing has three major advantages over sorting.

(i) Index files are small
(ii) Index files are automatically updated when the information is changed, added or deleted in the database file.
(iii) Index files help in quickly searching for information through FIND/SEEK commands.

70.22.2. Automatic updating of index files

In a sorted database file if we add a record or modify the data in any record in the database file, we have to resort it so that the sorted files are correct. On the other hand, if indexed files are being used, there is no need to reindex the files. The index file automatically will reindex. If the EMP file contains 50 records and if we add one more record then the following command may be used.

. USE EMP (will open the EMP file)
. SET INDEX TO NAMEIDX (will make the index file of this database file active)
. APPEND BLANK (will append one empty record to the end of the file EMP)
. EDIT (will present a blank record for filling in data which was added by APPEND BLANK command)

70.22.3. Searching data in indexed file

. USE EMP INDEX NAMEIDX

will open the database file EMP with index file NAMEIDX.

Now, to locate the name "Arun Samanta" within this file type the following command.

. FIND Arun Samanta

Within a moment it display a dot (in foxbase) which indicates that it has found the record which contains "Arun Samanta" in the NAME field, and the record pointer point to that particular record. If it does not find a record, it will display a "No find." Message. To display all fields of this record type DISPLAY.

70.23. File utilities

. DIR

will display all the DBF files containing in the current directory/folder.

. COPY FILE <SourceFile> TO <DestinationFile>

will copy named "SourceFile" to the file "DestinationFile".

. RENAME <OldFile> TO <NewFile>

will change "OldFile" name to the new name "NewFile".

. ERASE <FileName>

. DELETE FILE <FileName>

will delete the file named "FileName".

Without quit the database software we can execute any DOS command from foxbase/foxpro command. For example,

. RUN ERASE *.PRG

. ! ERASE *.PRG

The above two commands (both have same purpose) will delete all the files whose extension PRG.

70.24. Memory variable and functions

With foxbase/foxpro, we can store the value of the intermediate results of calculations, contents of a field etc. to the RAM. In any database software some space of the RAM keeps reserved for storing such data. This particular memory area may be used as a "scratch pad area" and data items stored in this area are called *Memory Variables.*

Creating memory variable using STORE command

. STORE 10 TO X, PAY

10

This command creates a numeric type memory variable and stores 10 to the variables X and PAY.

. STORE "Aniket Pal" TO NAME
Aniket Pal
This command creates a character type memory variable and stores "Aniket Pal" to the variable NAME.

. STORE DATE() TO TODAY
08/12/04
This command creates a date type memory variable and stores the current date (which is set to the computer) to the variable TODAY.

It may be noticed that when STORE command is being processed the values stored in the memory variables are display on the screen. To avoid this unwanted display the following command may be used.
. SET TALK OFF

70.24.1. Creating memory variable using = command

The above task can be done by using "=" sign.
. X=10
10
. PAY=10
10
. NAME= "Aniket Pal"
Aniket Pal
. TODAY=DATE()
08/12/04
. WORD=SPACE(25)
This command will creates a character type variable WORD containing 25 blank spaces.

70.25. Printing of output

Printing of output of a database file is very simple. To activate the printer use the following command.
. SET PRINT ON
After setting this command all the information display on the screen will also printed to the printer. To stop print use the following command.
. SET PRINT OFF

70.26. Displaying and manipulating memory variables

?/?? are two very useful commands to display any kind of information: field, memory variable, any constant, function etc.

. USE EMP

. ? NAME

This command will display the NAME of the first record.

. USE EMP
. GOTO 4
. ? NAME, BASIC

This command will display the NAME, BASIC of the 4th record.

. ? "Today's date is: ", DATE()

If the system date (the date which is set to the computer) is 13 August 2004 then the above command will display the following message:

```
Today's date is: 08/13/04
```

. A=10
. B=20
. ? A (will display the value of A)
. ? B (will display the value of B)
. ? A+B (will display the sum of A and B)

The following will display for the above commands.

```
10
20
30
```

Unlike the ? command, which displays the result in the next line, the ?? command displays the result in the same line.

70.26.1. Time and Date functions

The function TIME() displays the current system time to the screen.

.TIME()

```
22:39:40
```

(where 22:39:40 is the current time).

. DATE()

```
08/13/04
```

This function displays the current system date. Note that the date is displays in American format (mm/dd/yy). To display the date in British format (dd/mm/yy) the command SET DATE BRITISH to be used before execution of the function DATE(). To displays the date again in American format use the command SET DATE AMERICAN. Assume that the system date is 13 August 2004.

. SET DATE BRITISH

```
. DATE()
13/08/04
```

It also be noted that the year is displays into 2 digits. To display the year into 4 digits use the command SET CENTURY ON. Note the following output.

```
. SET DATE BRITISH
. SET CENTURY ON
. DATE()
13/08/2004
```

To display the year into 2 digits use the command SET CENTURY OFF.

70.26.2. More functions regarding date

The **DTOC** (Date TO Character) function converts a date-type memory variable into a character string, e.g.,

```
.TODAY = DTOC(DATE())
13/08/04
```

This function converts the system date in a string, i.e., TODAY is a string type variable contains "13/08/04".

The **CTOD** (Character TO Date) function convert a date stored as a character string into a date-type variable.

```
. TEMP = "13/08/04"
. CURRENT=CTOD(TEMP)
```

The first line stores the string "13/08/04" to the variable TEMP and the second command converts this string to a date-type value, i.e., CURRENT is a date-type variable.

Assume that the memory variable TODAY store the system date 13/08/04. Observe the following functions with this date.

```
. ? DAY(TODAY)
13
```

This function displays the day of the date TODAY.

```
. ? MONTH(TODAY)
8
```

This function displays the month of the date TODAY.

```
. ? YEAR(TODAY)
2004
```

This function displays the year of the date TODAY.

. ? CMONTH(TODAY)
August
This function displays the name of month as character expression of the date TODAY.

. ? DOW(TODAY) (Day Of Week)
6
This function displays the day of week as number of the date TODAY.

. ? CDOW(TODAY) (Character Day Of Week)
Friday
This function displays the day of week as character of the date TODAY.

We can add subtract days from date type variable.

. ? TODAY+20
20/09/04
Since TODAY is 13/08/04, the expression TODAY+20 gives the date after 20 days from 13/08/04.

. ? TODAY-20
24/07/04
Since TODAY is 13/08/04, the expression TODAY-20 gives the date 20 days ago from 13/08/04.

70.27. Converting numeric data to characters with STR()

. N=234.4561
The value of N becomes 234.4561. The function STR(N,6,2) converts the number N as character whose total length is 6 and number of digits after decimal point is 2, i.e.,
. STR(N,6,2)
234.46

The function STR(N,3) converts the number N as character whose total length is 3 and number of digits after decimal point is 0, i.e.,
. STR(N,3)
234

The function SRT(N,5) converts the number N as character whose total length is 5 but, the leading two characters will be blank spaces.
. STR(N,5)
  234

The function SRT(N) converts the number N as character but, before convert to character it should be rounded to an integer and the length of this string will be 10 (default length).

```
. STR(N)
        234
```

The function LEN gives the length of a character variable, e.g., if X="India" then the function LEN(X) gives the output 5.

```
. X="India"
. LEN(X)
5
```

70.28. Command files

All the command discussed previously are executed one by one. But, we can store all these command into a file, called command file. The extension of the command file is assigned to "prg" Once the commands are saved in a command file, we can execute all commands stored in that file by just using the command file name. The following command is used to run a command file.

```
. DO <FileName>
```

70.28.1. Creating a command file

The command MODIFY is used to create a command file. To create the command file TEST PRG use the following command.

```
. MODIFY COMMAND TEST     or     MODI COMM TEST
```

After typing this command a menu will come on the screen. Then type the following commands and save it by pressing `Ctrl End` keys.

```
USE EMP
LIST
GO TOP
DISPLAY NAME, BASIC, DA, HRA ALL
DISPLAY NAME FOR BASIC>2000
CLOSE DATABASES
```

To run this command file TEST type the following command.

```
. DO TEST
```

The action of each command is similar to the previous case. The last command will close the database file EMP, which was opened at the first line.

70.29. Font setting in Dot Matrix Printer

The font reside in printer can be on and off using the appropriate escape sequence. The following table gives a list of on and off option for different font.

Font Name	On	Off
Vertical enlarge	w1	w0
Horizontal enlarge	W1	W0
Underline	-1	-0
Bold	G	H
Italics	4	5
Compress	CHR(15)	CHR(18)

The following program segment illustrates the use of this table.

```
SET TALK OFF
SET PRINT ON
? "Normal Vidyasagar University"

?CHR(27)+"w1"
?"Vertical enlarge Vidyasagar University"
?CHR(27)+"w0"

?CHR(27)+"-1"
?"Underline Vidyasagar University"
?CHR(27)+"-0"

?CHR(27)+"G"
?"Bold Vidyasagar University"
?CHR(27)+"H"

?CHR(27)+"4"
?"Italics Vidyasagar University"
?CHR(27)+"5"

?CHR(27)+CHR(15)
?"Compressed Vidyasagar University"
?CHR(27)+CHR(18)

?"Normal font"
??CHR(27)+"-1"
??"Underline font"
??CHR(27)+"-0"
??CHR(27)+"4"
```

??"Italics font"
?CHR(27)+"4"

The output is shown below:
Normal Vidyasagar University
Vertical enlarge Vidyasagar University
Underline Vidyasagar University
Bold Vidyasagar University
Italics Vidyasagar University
Compressed Vidyasagar University
Normal font Underline font *Italics font*

70.30. Module Summary

In this module all basic commands of foxbase/foxpro are introduced. At least one example is supplied corresponding to each command/ clause. The definition of database management system and starting procedure are given at the beginning.

70.31. Self Assessment Questions

1. What are the field types supported by foxbase/foxpro?
2. Which field will be most appropriate to store the following information in a database file?
   Name of a book, Author(s) name, Price of the book, Accession number of the book, Date of purchase of the book, Summary of the book.
3. Create a database file VEHICLE with the following structure:

Filed Name	Type	Width
VEH_NAME	Character	30
MODEL	Character	15
DATE_MADE	Date	8

   Add 5 records to it.
4. In the EMP database file, find out and display the record of an employee whose name is supplied externally.
5. Write a short note on LOCATE.
6. What are the uses of the command LIST and DISPLAY?
7. What is the command to edit only the NAME and BASIC fields in the EMP database file?
8. What command is used to view multiple records from a database file on the screen and edit them?
9. Modify the structure of the EMP database file to add a new field EMPCODE and increase the width of the NAME field to 30.

10. Copy the EMP file to a new database file, say, EMP100. Rename EMP100 as EMP200. Erase the database file EMP200.
11. Sort the EMP database file on the NAME field. Store the sorted data in the EMP1 database file and display all the records.
12. Sort the EMP database file in descending order of the BASIC and NAME fields in EMP2 database file.
13. Index the EMP database file on the NAME field and store the index file as EMPIND. Use this index file to find "Arun Samanta" from EMP database file.
14. What are the merit and demerit of sort and index?
15. Display the system time and date to the screen.
16. Change the date format to British. Write the command to display the year of a date into 4 digits.
17. The following commands are stored into the file SAMPLE.

```
. SET DATE BRITI
. SET CENT ON
. ? DATE()
. ? CMONTH(DATE())
. ? CDOW(DATE())
. A=100
. B=200
. ?A+B
```

What is the output of the file SAMPLE, assume that the system date is 15 August 2004.

18. What is the command to run a command file.
19. What is the command to set printer on and off.

70.32. Further Suggested Readings

1. dBase III Plus Made Simple with dBase IV and FoxBase+, R.K.Taxali, Tata McGraw-Hill.
2. Special Edition Using Visual FoxPro 5, Antonovich, et. al., PHI.
3. Special Edition Using Visual FoxPro 6, Bazian., PHI.

M.Sc. Course in
Applied Mathematics with Oceanology
and
Computer Programming

Part-II

Paper-VI Group-B

Module No. 71

Lab. 5: Database Management System (DBMS)-2

Structure

71.1. Introduction
71.2. Objectives
71.3. Key Words
71.4. Assignments
  71.4.1. Maintaining a database for a class of students
  71.4.2. Package for publication of result of M.Sc. Part I examination
  71.4.3. Package for maintaining a database for books on a library
  71.4.4. Package for maintaining a database for telephone directory
  71.4.5. Package for maintaining a database for capitals and countries
  71.4.6. Package for pay packet of an institute
  71.4.7. Package for maintaining students records
  71.4.8. Package for writing a common letter to the members of an organization
  71.4.9. Program to controlling a Dot Matrix printer
71.30. Module Summary
71.31. Self Assessment Questions
71.32. Further Suggested Readings

71.1. Introduction

This is the continuation of the module 70. In the module 70 we have introduced the several commands of foxpro/foxbase. Here we design some complete programs called the commands files using the commands of previous module with some other commands like do-while structure, etc. The techniques to design the pop-up menu with appropriate messages are incorporated here,

71.2. Objectives

In this module, the students will learn the technique to design menu based programs, use of colour,

boarder, use of multiple data base files within a single program, integration of several different programs, addition, modification, deletion of records, etc., use of different fonts in printer, etc

71.3. Key Words

Do-While, pop-up menu, colour, boarder, append a file, delete a file, modification of a record delete a record, entry of new records, sorting, searching, merging of database files.

71.4. Assignments

71.4.1. Maintaining a database for a class of students

***Assignment* F1.** Create a database file for some students with the following fields

- Name
- Roll No.
- Date of birth

Write a program in FoxBase+ to draw the pop-up menu containing the following items

- Adding of new record(s).
- Display the record of each student in a given format.
- Display the record of a particular student whose roll number is given in the given format
- Sorting on name.
- Listing of name of all records of the sorted file.
- Exit from the menu.

Also implement all the menu items.

Structure of the database file F1.dbf is shown below

Structure for database : C:\PROGFOX\F1.DBF
Number of data records: 7
Date of last update : 09/06/06

Field	Field Name	Type	Width	Dec
1	NAME	Character	20	
2	ROLL	Numeric	4	
3	DBIRTH	Date	8	
** Total **			3	

```
**************************FILE NAME F1.PRG*****************************
**************************MENU PROGRAM*********************************
SET TALK OFF
CLEAR
CH=1
I=10
```

```
J=10
DO WHILE .T.
    CLEAR
    SET COLOR TO G
    @I,J    PROMPT " 1. ENTRY OF NEW RECORD"
    @I+1,J PROMPT " 2. DISPLAY THE RECORDS OF ALL STUDENTS"
    @I+2,J PROMPT " 3. SEARCHING OF A STUDENT WHOSE ROLL IS GIVEN"
    @I+3,J PROMPT " 4. SORTING ON NAME"
    @I+4,J PROMPT " 5. LISTING OF NAME OF RECORDS OF SORTED FILE"
    @I+5,J PROMPT " 6. EXIT"
    SET COLOR TO R
    @I-1,J-1 TO I+6,J+52 DOUBLE
     SET COLOR TO
     MENU TO CH
     DO CASE
        CASE CH=1
           DO F11
        CASE CH=2
           DO F13
        CASE CH=3
           DO F12
        CASE CH=4
           DO F14
        CASE CH=5
           DO F15
        CASE CH=6
           CLEAR
           SET COLOR TO
           CLOSE ALL
           EXIT
     ENDCASE
  ENDDO

************************FILE NAME F11.PRG*****************************
**********************ADDING OF NEW RECORD*****************************
```

```
SET TALK OFF
CLEAR
NAME1=SPACE(20)
ROLL1=0
DBIRTH1=DATE()
DO WHILE.T.
        I=10
        J=10
        @I,J SAY"ENTER NAME OF THE STUDENT"GET NAME1
        @I+1,J SAY"ENTER ROLL OF THE STUDENT"GET ROLL1
        @I+2,J SAY"ENTER DBIRTH OF THE STUDENT"GET DBIRTH1
        @I-1,J-1 TO I+4,J+49 DOUBLE
        READ
        OPT="Y"
        @I+11,J SAY"IS THE DATA CORRECT(Y/N)?" GET OPT PICT"!"
        READ
        IF(OPT="Y")THEN
                EXIT
        ENDIF
ENDDO
USE F1
APPEND BLANK
CLEAR
REPL NAME WITH NAME1
REPL ROLL WITH ROLL1
REPL DBIRTH WITH DBIRTH1
CLOSE DATABASES

**************************FILE NAME F13.PRG**************************
***********DISPLAY THE RECORD OF A PARTICULAR STUDENT***********
**********************WHOSE ROLL NO.IS SUPPLIED**********************
SET TALK OFF
CLEAR
I=10
J=10
```

```
ROLL1=0
@I,J SAY "ENTER ROLL OF A STUDENT" GET ROLL1 PICT "9999"
@I-1,J-1 TO I+1,J+35 DOUBLE
READ
USE F1
S=0
DO WHILE .NOT.EOF()
        CLEAR
        IF(ROLL1=ROLL) THEN
                @I+2,J SAY "NAME:"+NAME
                @I+3,J SAY "ROLL NO:"+STR(ROLL,4)
                @I+3,J+25 SAY "DATE OF BIRTH:"+DTOC(DBIRTH)
                READ
                S=1
                GO BOTTOM
                EXIT
        ELSE
             SKIP
        ENDIF
ENDDO
IF (S=0) THEN
        @I+5,J SAY "NO SUCH NAME EXIT!"
        WAIT""
ENDIF
SET COLOR TO
CLOSE DATABASES

**************************FILE NAME F12.PRG**************************
******DISPLAY THE RECORDS OF ALL STUDENTS WHOSE ****************
SET TALK OFF
CLEAR
USE F1
DO WHILE .NOT. EOF()
        I=10
        J=15
```

```
        CLEAR
        @I,J SAY"NAME:"+NAME
        @I+1,J SAY"ROLL NO:"+STR(ROLL,4)
        @I+1,J+17 SAY"DATE OF BIRTH:"+DTOC(DBIRTH)
        WAIT " "
       SKIP
ENDDO
READ
SET COLOR TO
CLOSE DATABASES

************************FILE NAME F14.PRG*****************************
************************SORTING ON NAME ******************************
SET TALK OFF
CLEAR
USE F1
DO WHILE .NOT. EOF()
      SORT ON NAME TO F11
ENDDO
CLOSE DATABASES

*************************FILE NAME F15.PRG*****************************
********* ROLL AND NAME OF ALL RECORD OF THE SORTED FILE*********
SET TALK OFF
CLEAR
USE F11
DISPLAY ALL ROLL,NAME
READ
CLOSE DATABASES
```

71.4.2. Package for publication of result of M.Sc. Part I examination

Assignment **F2.** Create a database file for the M.Sc. Part-I students of Applied Mathematics, V.U. having the following fields

- Roll no.
- Name
- Address
- Marks of Paper-I Gr. A

- Marks of Paper-I Gr. B]
- Marks of Paper-I Gr. C
- Marks of Paper-II Gr. A
- Marks of Paper-II Gr. B
- Marks of Paper-III Gr. A
- Marks of Paper-III Gr. B
- Marks of Paper-III Gr. C
- Marks of Paper-IV Gr. A
- Marks of Paper-IV Gr. B
- Marks of Paper-V Gr. A
- Marks of Paper-V Gr. B
- Total marks
- Percentage of marks
- Class obtained

Write a program in FoxBase+ to draw the pop-up menu containing the following items

- Adding record(s) of new students.
- Edit/modify all the records of a particular student whose roll no. is given.
- Entry the marks of
- Specified group of a specified paper.
    - All group of all paper.
- Display the records of all students.
- Preparation of result by calculating total marks, percentage of marks and class.
- Tabulation.
- Printing of mark sheet.
- Quit from the menu.

Write program to implement all of the above tasks.

Structure of the database file F2.dbf is shown below

Field	Field Name	Type	Width	Dec
1	ROLLNO	Numeric	2	
2	NAME	Character	18	
3	ADDRESS	Character	10	
4	MP1GA	Numeric	2	
5	MP1GB	Numeric	2	
6	MP1GC	Numeric	2	

7	MP2GA	Numeric	2	
8	MP2GB	Numeric	2	
9	MP3GA	Numeric	2	
10	MP3GB	Numeric	2	
11	MP3GC	Numeric	2	
12	MP4GA	Numeric	2	
13	MP4GB	Numeric	2	
14	MP5GA	Numeric	2	
15	MP5GB	Numeric	2	
16	TMARK	Numeric	3	
17	PMARK	Numeric	4	2
18	CLASS	Character	2	
** Total **		64		

```
**********ASSIGNMENT NO:2**********
**********FILE NAME:F2.PRG**********
**********MAIN MENU**********
SET TALK OFF
CH=1
DO WHILE.T.
        I=10
        J=10
        CLEAR
        SET COLOR TO R+*
        @1,70 SAY DATE()
        SET COLOR TO GR+
        @ I,J  PROM  " 1. ENTRY OF NEW RECORD OF A STUDENT"
        @I+1,J PROM" 2. MODIFY/EDIT"
        @I+2,J PROM" 3. MARKS ENTRY (OF A GROUP OF A PAPER)"
        @I+3,J PROM" 4. MARKS ENTRY OF ALL GROUP OF ALL PAPERS"
        @I+4,J PROM" 5. DISPLAY OF RECORDS OF ALL STUDENTS"
        @I+5,J PROM" 6. CALCULATION OF TOTAL MARKS, PERCENTAGE OF
                        MARKS AND CLASS "
        @I+6,J PROM" 7. TABULATION"
```

```
@I+7,J PROM" 8. PRINTING OF MARK SHEET"
@I+8,J PROM" 9. EXIT"
@I+9,J PROM" 10.QUIT"
SET COLOR TO G
@I-1,J-1 TO I+10,J+69 DOUBLE
SET COLOR TO RG+
MENU TO CH
DO CASE
        CASE CH=1
                DO F21
        CASE CH=2
                DO F22
        CASE CH=3
                DO F23
        CASE CH=4
                DO F24
        CASE CH=5
                DO F25
        CASE CH=6
                DO F26
        CASE CH=7
                DO F27
        CASE CH=8
                DO F28
        CASE CH=9
                CLEAR
                CLOSE ALL
                SET COLOR TO
                EXIT
        CASE CH=10
                CLEAR
                CLOSE ALL
                SET COLOR TO
                QUIT
```

```
        ENDCASE
ENDD

**************************FILE NAME F21.PRG**************************
************************ENTRY OF NEW RECORDS************************
SET TALK OFF
CLEAR
I=10
J=10
ROLL1=0
NAME1=SPACE(20)
ADDRESS1=SPACE(30)
USE F2
APPEND BLANK
DO WHILE .T.
        @I,J SAY "ENTRY OF THE ROLL" GET ROLL1
        @I+1,J SAY "ENTRY OF THE NAME" GET NAME1
        @I+2,J SAY "ENTRY OF THE ADDRESS" GET ADDRESS1
        @I-1,J-1 TO I+3,J+53 DOUBLE
        READ
        SIG ="Y"
        @I+7,J SAY "IS THE DATA CORRECT (Y/N)" GET SIG
        READ
        IF(SIG="Y") THEN
                EXIT
        ENDIF
ENDDO
REPL ROLLNO WITH ROLL1
REPL NAME WITH NAME1
REPL ADDRESS WITH ADDRESS1
CLOSE DATABASE

**************************FILE NAME F23.PRG**************************
*****ENTRY THE MARKS OF SPECIFIED GROUP OF A SPECIFIED PAPER****
SET TALK OFF
CLEAR
```

```
I=5
J=5
PA=1
GR=SPACE(1)
@I,J SAY "ENTER PAPER NUMBER (1 .. 5) "GET PA PICT "9" RANGE 1,5
@I+1,J SAY "ENTER GROUP                    " GET GR
@I-1,J-1 TO I+2,J+34 DOUBLE
READ
PAPER="MP"+STR(PA,1)+"G"+GR
USE F2
DO WHILE .NOT. EOF()
        MARK=0
        @I+3,J  SAY "NAME:" +NAME
        @I+3,J+32 SAY "ROLL NO."+STR(ROLLNO,4)
        @I+4,J SAY "ENTER THE MARKS:" GET MARK
        READ
        REPLACE &PAPER WITH MARK
        SKIP
ENDDO
CLOSE DATABASE

**************************FILE NAME F24.PRG************************
**ENTRY OF MARKS OF ALL GROUPS OF ALL PAPERS OF ALL STUDENTS**
SET TALK OFF
CLEAR
I=7
J=7
USE F2
BROWS FIELDS NAME,ROLLNO,MP1GA,MP1GB,MP1GC,MP2GA,MP2GB,
MP3GA,MP3GB,MP3GC,MP4GA,MP4GB,MP5GA,MP5GB
CLOSE DATABASE

***********************FILE NAME F25.PRG*****************************
***********DISPLAY THE RECORD OF ALL STUDENTS*********************
SET TALK OFF
CLEAR
```

```
I=1
J=1
USE F2
DISPLAY  ALL ROLLNO,NAME,MP1GA,MP1GB,MP1GC,MP2GA,MP2GB,
MP3GA,MP3GB,MP3GC,MP4GA,TMARK,CLASS
READ
CLOSE DATABASES

**********FILE NAME F26.PRG**********
******CALCULATION OF TOTAL, PERCENTAGE OF MARKS AND CLASS****
SET TALK OFF
CLEAR
USE F2
DO WHILE .NOT. EOF()
        REPL TMARK WITH MP1GA+MP1GB+MP1GC+MP2GA+MP2GB
        +MP3GA+MP3GB+MP3GC+M4GA+MP4GB +MP5GA+MP5GB
        REPL PMARK WITH TMARK/5
        CLASS1="X"
        IF(PMARK>=45) THEN
                CLASS1="II"
        ENDIF
        IF(PMARK>=60) THEN
                CLASS1="I"
        ENDIF
        REPL CLASS WITH CLASS1
        SKIP
ENDDO
CLOSE DATABASES

**********FILE NAME F27.PRG**********
**********TABULATION PROGRAM**********
SET TALK OFF
SET STAT OFF
PAGE=1
PAGELEN=5
KOUNT=0
```

```
SLNO=0
USE F2
SET PRINT ON
DO HEADF21
DO WHILE.NOT.EOF()
        SLNO=SLNO+1
        KOUNT=KOUNT+1
        ?STR(SLNO,3)+SPACE(4)+NAME+SPACE(2)+STR(ROLLNO,2)+SPACE(5)
        ??ADDRESS+SPACE(4)+STR(MP1GA,2)+SPACE(4)
        ??STR(MP1GB,2)+SPACE(5)+STR(MP1GC,2)+SPACE(5)+STR(MP2GA,2)
        ??SPAC(5)+STR(MP2GB,2)+SPAC(5)+STR(MP3GA,2)+SPAC(5)
        ??+STR(MP3GB,2)+SPAC(5)
        ??STR(MP3GC,2)+SPACE(5)+STR(MP4GA,2)+SPACE(5)
        ??+STR(MP4GB,2)+PACE(5)
        ??STR(MP5GA,2)+SPACE(5)+STR(MP5GB,2)+SPACE(6)
        ??+STR(TMARK,3)+SPACE(3)
        ??STR(PMARK,5)+SPACE(5)+CLASS
        SKIP
        IF(KOUNT>=PAGELEN)THEN
                KOUNT=0
                PAGE=PAGE+1
                EJECT
                DO HEADF21
        ENDIF
ENDDO
WAIT""
SET PRINT OFF
CLOSE ALL

**********FILE NAME HEADF21.PRG**********
**********HEADING OF TABULATION SHEET**********
?SPACE(65)+"PAGE NO."+STR(PAGE,3)
?CHR(27)+"w1"+SPACE(27)+"VIDYASAGAR UNIVERSITY"
?CHR(27)+"w0"
?CHR(27)+"-1"+SPACE(27)+"Midnapore"+SPACE(1)+","+SPACE(1)+"West Bengal"
```

```
?CHR(27)+"-0"
?SPACE(30)+"The Tabulation Sheet"
?"SL.NO"+SPAC(2)+"Mr./Miss"+SPAC(11)+"Rollno"+SPAC(3)+"Address"+SPAC(6)
??" P1GrA"+SPACE(1)+" P1GrB"+SPACE(1)+" P1GrC"+SPACE(1)+" P2GrA"
??SPACE(1)+" P2GrB"+SPACE(1)+" P3GrA"+SPACE(1)
??" P3GrB"+SPACE(1)+" P3GrC"+SPACE(1)+" P4GrA"+SPACE(1)+" P4GrB"
??SPACE(1)+" P5GrA"+SPACE(1)+" P5GrB"+SPACE(2)+"TMARK"+SPACE(2)
??"PMARK"+SPACE(2)+"CLASS"
?REPL("~",152)

**************************FILE NAME F28.PRG************************
************************PRINTING OF MARKSHEET**********************
SET TALK OFF
SET STAT OFF
PAGE=1
PAGELEN=1
KOUNT=0
SLNO=0
USE F2
SET PRINT ON
DO WHILE.NOT.EOF()
        DO HEADF22
        SLNO=SLNO+1
        KOUNT=KOUNT+1
        ?STR(SLNO,3)+SPACE(4)+NAME+SPACE(2)+STR(ROLLNO,2)+SPACE(5)
        ??ADDRESS+SPACE(4)+STR(MP1GA,2)+SPACE(4)
        ??STR(MP1GB,2)+SPACE(5)+STR(MP1GC,2)+SPACE(5)+STR(MP2GA,2)
        ??SPAC(5)+STR(MP2GB,2)+SPAC(5)+STR(MP3GA,2)
        ??+SPAC(5)+STR(MP3GB,2)+SPAC(5)
        ??STR(MP3GC,2)+SPACE(5)+STR(MP4GA,2)+SPACE(5)+STR(MP4GB,2)
        ??SPACE(5)+STR(MP5GA,2)+SPACE(5)+STR(MP5GB,2)+SPACE(6)
        ??STR(TMARK,3)+SPACE(3)
        ??STR(PMARK,5)+SPACE(5)+CLASS
        ?REPL("-",152)
        SKIP
```

```
        IF(KOUNT>=PAGELEN)THEN
                KOUNT=0
                PAGE=PAGE+1
                EJECT
        ENDIF
        WAIT""
ENDDO
SET PRINT OFF
CLOSE ALL

************************FILE NAME HEADF22.PRG**********************
******************HEADING OF MARKSHEET PRINTING*****************
?SPACE(65)+"PAGE NO."+STR(PAGE,3)
?CHR(27)+"w1"+SPACE(10)+"VIDYASAGAR UNIVERSITY"
?CHR(27)+"w0"
?CHR(27)+"-1"+SPACE(8)+"Midnapore"+SPACE(1)+","+SPACE(1)+"West Bengal"
?CHR(27)+"-0"
?SPACE(2)+"THE Following is the statement of the marks obtained by"
?"Sl. No"+SPAC(2)+" Mr./Miss"+SPAC(11)+"Rollno"+SPAC(3)+"Address"+SPAC(6)
??" P1GrA"+SPACE(1)+" P1GrB"+SPACE(1)+" P1GrC"+SPACE(1)+" P2GrA"
??SPACE(1)+" P2GrB"+SPACE(1)+" P3GrA"+SPACE(1)
??" P3GrB"+SPACE(1)+" P3GrC"+SPACE(1)+" P4GrA"+SPACE(1)+" P4GrB"
??SPACE(1)+" P5GrA"+SPACE(1)+" P5GrB"+SPACE(2)+"TMARK"+SPACE(2)
??"PMARK"+SPACE(2)+"CLASS"
?REPL("~",152)
```

71.4.3. Package for maintaining a database for books on a library

Assignment **F3.** Create two database files of which are contains the records of books fields

- Title
- Author
- Acc no.
- Call no.
- Subject
- Branch
- Issuing card no.
- Date of issue

and the other file contains the record of member with the following fields

- Name
- Roll No (if student)
- Department
- Year (if student)
- Status (student/teacher/non teaching, officer etc.)
- Acc no.1
- Acc no.2
- Acc no.3
- Card no.

(assumed that a member can issue at most three books at a time).
Write a program to draw the pop-up menu containing the items.

- Entry of new book
- Entry of new member
- Listing of books issued by a member
- Searching of a book according to
    - Acc no.
    - Subject
    - Branch
    - Title
    - Author
- Exit from menu.

Now write FoxBase programs to implement all the tasks.

Structure of the database file F31.DBF is shown below

Field	Field Name	Type	Width	Dec
1	TITLE	Character	25	
2	AUTHOR	Character	20	
3	ACCNO	Numerical	5	
4	CALLNO	Numerical	5	
5	SUB	Character	14	
6	BRAN	Character	10	
7	ISSCDNO	Numerical	5	
8	DISS	Date	8	

Structure of the database file F32.DBF is shown below

Field	Field Name	Type	Width	Dec
1	NAME	Character	19	
2	ROLLNO	Numerical	2	
3	DEP	Character	15	
4	YEAR	Numerical	4	
5	STATUS	Character	10	
6	ACCNO1	Numerical	5	
7	ACCNO2	Numerical	5	
8	ACCNO3	Numerical	5	
9	CRDNO	Numerical	5	

```
**********FILE NAME F3.PRG**********
**********PROGRAM OF MAIN MENU**********
SET TALK OFF
CH=1
DO WHILE .T.
   I=10
   J=10
   CLEAR
   SET COLOR TO G+
   @I,J   PROM " 1. ENTRY OF NEW BOOK"
   @I+1,J PROM " 2. ENTRY OF NEW MEMBER"
   @I+2,J PROM " 3. LISTING OF BOOKS ISSUED BY A MEMBER"
   @I+3,J PROM " 4. SEARCHING OF A BOOK ACCORDING TO ACC NO."
   @I+4,J PROM " 5. SEARCHING OF A BOOK ACCORDING TO SUBJECT"
   @I+5,J PROM " 6. SEARCHING OF A BOOK ACCORDING TO BRANCH"
   @I+6,J PROM " 7. SEARCHING OF A BOOK ACCORDING TO TITLE"
   @I+7,J PROM " 8. SEARCHING OF A BOOK ACCORDING TO AUTHOR"
   @I+8,J PROM " 9. EXIT"
   SET COLOR TO GB
   @I-1,J-1 TO I+9,J+48 DOUBLE
   SET COLOR TO G+
   MENU TO CH
 DO CASE
```

```
        CASE CH=1
              DO F31
       CASE CH=2
             DO F32
       CASE CH=3
             DO F33
       CASE CH=4
             DO F34
       CASE CH=5
             DO F35
       CASE CH=6
             DO F36
       CASE CH=7
             DO F37
       CASE CH=8
             DO F38
      . CASE CH=9
             SET COLOR TO
             CLEAR
             EXIT
    ENDCASE
 ENDDO
 SET COLOR TO
************************FILE NAME F31.PRG*****************************
************************ENTRY OF NEW BOOK*****************************
SET TALK OFF
CLEAR
I=10
J=10
TITLE1=SPACE(25)
AUTHOR1=SPACE(20)
ACNO1=0
CALNO1=0
SUBJECT1=SPACE(14)
```

```
BRANCH1=SPACE(10)
USE F31
APPEND BLANK
DO WHILE .T.
    @I,J  SAY "ENTER TITLE OF THE BOOK " GET TITLE1
    @I+1,J SAY "ENTER AUTHOR NAME     " GET AUTHOR1
    @I+2,J SAY "ENTER ACC NO.         " GET ACNO1 PICT "99999"
    @I+3,J SAY "ENTER CALL NO.        " GET CALNO1 PICT "99999"
    @I+4,J SAY "ENTRY SUBJECT NAME    " GET SUBJECT1
    @I+5,J SAY "ENTER BRANCH (SC./ARTS.)" GET BRANCH1
    @I-1,J-1 TO I+6,J+52 DOUBLE
    READ
    SIG ="Y"
    @I+7,J SAY "IS ENTERED DATA CORRECT (Y/N) ?" GET SIG
    READ
    IF(SIG="Y") THEN
        EXIT
    ENDIF
ENDDO
REPL TITLE WITH TITLE1
REPL AUTHOR WITH AUTHOR1
REPL ACCNO WITH ACNO1
REPL CALLNO WITH CALNO1
REPL SUB WITH SUBJECT1
REPL BRAN WITH BRANCH1
CLOSE DATABASE

**************************FILE NAME F32.PRG**************************
************************ENTRY OF NEW MEMBER************************
SET TALK OFF
CLEAR
I=10
J=10
NAME1=SPACE(19)
ROLLNO1=0
```

```
DEP1=SPACE(15)
YEAR1=0
STATUS1=SPACE(10)
CRDNO1=0
USE F32
APPEND BLANK
DO WHILE .T.
    SET COLOR TO G
    @I,J  SAY "ENTER NAME           " GET NAME1
    @I+1,J SAY "ENTER ROLLNO         " GET ROLLNO1 PICT "99"
    @I+2,J SAY "ENTER NAME OF DEPARTMENT" GET DEP1
    @I+3,J SAY "ENTER YEAR OF ADMISSION " GET YEAR1 PICT "9999"
    @I+4,J SAY "ENTER STATUS(STU./TEAC.)" GET STATUS1
    @I+5,J SAY "ENTER OF CARD NO.      " GET CRDNO1 "99999"
    SET COLOR TO R+
    @I-1,J-1 TO I+6,J+50 DOUBLE
    READ
    SIG ="Y"
    @I+9,J SAY "IS ENTERED DATA CORRECT (Y/N) ?" GET SIG
    READ
    IF(SIG="Y") THEN
        EXIT
    ENDIF
ENDDO
REPL NAME WITH NAME1
REPL ROLLNO WITH ROLLNO1
REPL DEP WITH DEP1
REPL YEAR WITH YEAR1
REPL STATUS WITH STATUS1
REPL CRDNO WITH CRDNO1
CLOSE DATABASE

*****************************FILE NAME F33.PRG************************
**********LISTING OF BOOKS ISSUED BY A PARTICVULAR MEMBER*******
SET TALK OFF
```

```
CLEAR
I=10
J=10
SELECT A
    USE F31
SELECT B
    USE F32
CRDNO1=0
SET COLOR TO RG
@I,J SAY "ENTER OF CARD NO." GET CRDNO1 PICT "99999"
SET COLOR TO BG
@I-1,J-1 TO I+1,J+28 DOUBLE
READ
S=0
DO WHILE .NOT. EOF()
    SELECT B
    IF(CRDNO1=CRDNO) THEN
        DO WHILE .NOT. EOF()
            SELECT A
            IF(CRDNO1=ISSCDNO) THEN
                S=1
                CLEAR
                SET COLOR TO RG+
                @I+2,J SAY "TITLE:" +TITLE
                @I+3,J SAY "AUTHOR:" +AUTHOR
                @I+4,J SAY "ACC.NO.:" +STR(ACCNO,5)
            ENDIF
            SKIP
        ENDDO
    ENDIF
    SELECT B
    SKIP
ENDDO
IF(S=0) THEN
```

```
    SET COLOR TO R+*
    @I+5,J SAY "NO BOOKS ISSUED BY THIS CARD HOLDER !"
ENDIF
READ
SET COLOR TO
CLOSE DATABASES

*****************************FILE NAME F34.PRG************************
***********SEARCHING WITH RESPECT TO ACCESSION NUMBER*********
SET TALK OFF
CLEAR
I=10
J=10
ACCNO1=0
SET COLOR TO RG
@I,J SAY "ENTER OF ACC. NO." GET ACCNO1 PICT "99999"
SET COLOR TO BG
@I-1,J-1 TO I+1,J+28 DOUBLE
READ
S=0
USE F31
DO WHILE .NOT. EOF()
    IF(ACCNO1=ACCNO) THEN
        S=1
        @I+2,J SAY "TITLE:" +TITLE
        @I+3,J SAY "AUTHOR:" +AUTHOR
        @I+4,J SAY "ACC.NO.:" +STR(ACCNO,5)
        IF(ISSCDNO=0) THEN
            @I+5,J SAY "THIS BOOK IS AVAILABLE"
        ELSE
            @I+5,J SAY "THIS BOOK IS ISSUED"
        ENDIF
        READ
    ENDIF
    SKIP
```

```
ENDDO
IF(S=0) THEN
    SET COLOR TO R+*
    @I+5,J+5 SAY"INVALID ACCESSION NUMBER !"
ENDIF
WAIT""
CLOSE DATABASES
SET COLOR TO

**********FILE NAME F35.PRG**********
**********SEARCHING WITH RESPECTIVE TO SUBJECT**********
SET TALK OFF
CLEAR
I=2
J=2
SUBJECT1=SPACE(14)
@I,J SAY "ENTER OF SUBJECT" GET SUBJECT1
@I-1,J-1 TO I+1,J+35 DOUBLE
READ
S=0
USE F31
DO WHILE .NOT. EOF()
    IF(UPPER(TRIM(SUBJECT1))=UPPER(TRIM(SUB))) THEN
        S=1
        CLEAR
        SET COLOR TO G+
        @I+2,J SAY "TITLE:" +TITLE
        @I+3,J SAY "AUTHOR:" +AUTHOR
        @I+4,J SAY "ACC.NO.:" +STR(ACCNO,5)
        IF(ISSCDNO=0)THEN
            SET COLOR TO GR
            @I+5,J SAY "THIS BOOK IS AVAILABLE"
        ELSE
            SET COLOR TO R+*
            @I+5,J SAY "THIS BOOK IS ISSUED"
```

```
        ENDIF
        READ
    ENDIF
    SKIP
ENDDO
IF(S=0)THEN
    SET COLOR TO R+*
    @I+5,J+5 SAY "NO BOOKS AVAILABEL WITH THIS SUBJECT NAME !"
ENDIF
WAIT""
CLOSE DATABASES
SET COLOR TO

**************************FILE NAME F36.PRG**************************
****************SEARCHING WITH RESPECT TO BRANCH*****************
SET TALK OFF
CLEAR
I=2
J=2
BRAN1=SPACE(10)
@I,J SAY "ENTER OF BRANCH" GET BRAN1
@I-1,J-1 TO I+1,J+28 DOUBLE
READ
S=0
USE F31
DO WHILE .NOT. EOF()
    IF(UPPER(TRIM(BRAN1))=UPPER(TRIM(BRAN))) THEN
        S=1
        CLEAR
        SET COLOR TO BR
        @I+2,J SAY "TITLE:" +TITLE
        @I+3,J SAY "AUTHOR:" +AUTHOR
        @I+4,J SAY "ACC.NO.:" +STR(ACCNO,5)
        IF(ISSCDNO=0)THEN
            SET COLOR TO R+
```

```
            @I+5,J SAY "THIS BOOK IS AVAILABLE"
        ELSE
            SET COLOR TO R+*
            @I+5,J SAY "THIS BOOK IS ISSUED"
        ENDIF
        READ
    ENDIF
    SKIP
ENDDO
IF(S=0) THEN
    SET COLOR TO R+*
    @I+5,J+5 SAY "NO BOOKS AVAILABLE FOR THIS BRANCH !"
ENDIF
WAIT""
CLOSE DATABASES
SET COLOR TO

*****************************FILE NAME 37.PRG***********************
*****************SEARCHING WITH RESPECTTO TITLE*****************
SET TALK OFF
CLEAR
I=2
J=2
TITLE1=SPACE(25)
@I,J SAY "ENTER OF TITLE" GET TITLE1
@I-1,J-1 TO I+1,J+42 DOUBLE
READ
S=0
USE F31
DO WHILE .NOT. EOF()
    IF(UPPER(TRIM(TITLE1))=UPPER(TRIM(TITLE)))THEN
        S=1
        SET COLOR TO BG
        @I+2,J SAY "TITLE:" +TITLE
        @I+3,J SAY "AUTHOR:" +AUTHOR
```

```
        @I+4,J SAY "ACC. NO.:" +STR(ACCNO,5)
        IF(ISSCDNO=0)THEN
            SET COLOR TO R+
            @I+5,J SAY "THIS BOOK IS AVAILABLE"
        ELSE
            SET COLER TO R+*
            @I+5,J SAY "THIS BOOK IS ISSUED"
        ENDIF
        READ
    ENDIF
    SKIP
ENDDO
IF(S=0) THEN
    SET COLOR TO R+*
    @I+5,J+5 SAY "NO BOOKS AVAILABLE WITH THIS TITLE !"
ENDIF
WAIT""
CLOSE DATABASES
SET COLOR TO

*****************************FILE NAME F38.PRG************************
***************SEARCHING WITH RESPECT TO AUTHOR***************
SET TALK OFF
CLEAR
I=10
J=10
AUTHOR1=SPACE(20)
@I,J SAY "ENTER OF AUTHOR" GET AUTHOR1
@I-1,J-1 TO I+1,J+36 DOUBLE
READ
S=0
USE F31
DO WHILE .NOT. EOF()
    IF(UPPER(TRIM(AUTHOR1))=UPPER(TRIM(AUTHOR)))THEN
      S=1
```

```
        SET COLOR TO RG+
        @I+2,J SAY "TITLE:" +TITLE
        @I+3,J SAY "AUTHOR:" +AUTHOR
        @I+4,J SAY "ACC. NO.:" +STR(ACCNO,5)
        IF(ISSCDNO=0) THEN
           SET COLOR TO R+
           @I+5,J SAY "THIS BOOK IS AVAILABLE"
        ELSE
           SET COLOR TO R+*
           @I+5,J SAY "THIS BOOK IS ISSUED"
        ENDIF
     ENDIF
     SKIP
ENDDO
IF(S=0) THEN
   SET COLOR TO R+*
   @I+5,J+5 SAY "NO BOOKS AVAILABLE WITH THIS AUTHOR !"
ENDIF
WAIT""
CLOSE DATABASES
SET COLOR TO
```

71.4.4. Package for maintaining a database for telephone directory

Assignment **F4.** Create a database file that will maintain a list of

- Names
- Addresses
- Telephone numbers

Write a program in FoxBase+ to draw a pop-up menu that contains the following items

- Add new record(s)
- Delete a record
- Modify a record
- Retrieve and display an entire record for a given name
- Generate a complete list of all records.
- Exit from the menu.

Also write programs to implement all of the above tasks.

Structure of the database file F4.DBF is shown below

Field	Field Name	Type	Width	Dec
1	NAME	Character	20	
2	ADDRESS	Character	14	
3	TELNO	Numerical	12	

```
**************************FILE NAME F4.PRG**************************
************************MAIN MENU PROGRAM************************
SET TALK OFF
SET STAT OFF
CH=1
I=10
J=10
DO WHILE .T.
    CLEAR
    SET COLOR TO GB+
    @I,J   PROM " 1. ADD A NEW RECORD"
                    MESS "ADD NAME, ADDRESS AND TEL. NO."
    @I+1,J PROM " 2. DELETE A RECORD" MESS "DELETE A RECORD WHEN
                    CONSUMER NAME IS GIVEN"
    @I+2,J PROM " 3. MODIFY A RECORD"
                    MESS "CORRECTION OF EXISTING RECORDS"
    @I+3,J PROM " 4. RETRIEVE AND DISPLAY AN ENTIRE RECORD
                    FOR A CONSUMER"  MESS "NAME, ADDRESS AND TEL.
                    NO. TO BE DISPLYED"
    @I+4,J PROM " 5. GENERATE A COMPLETE LIST OF ALL RECORDS"
                    MESS "LIST OF NAME AND TEL. NO. OF ALL CONSUMER"
    @I+5,J PROM " 6. EXIT FROM THE MENU"
    SET COLOR TO R+
    @I-1,J-1 TO I+6,J+56 DOUBLE
    SET COLOR TO GB+
    SET MESS TO 23
    MENU TO CH
    DO CASE
        CASE CH=1
```

```
            DO F41
        CASE CH=2
            DO F42
        CASE CH=3
            DO F43
        CASE CH=4
            DO F44
        CASE CH=5
            DO F45
        CASE CH=6
             SET COLOR TO
            CLEAR
            EXIT
            CLOSEALL
    ENDCASE
ENDDO

*****************************FILE NAME F41.PRG**********************
************************PROGRAM FOR NEW ENTRY**********************
SET TALK OFF
CLEAR
I=10
J=10
NAME1=SPACE(20)
ADD1=SPACE(14)
TELNO1=0
USE F4
APPEND BLANK
DO WHILE .T.
    I=10
    J=10
    SET COLOR TO GB+
    @I,J SAY "ENTER OF THE NAME" GET NAME1
    @I+1,J SAY "ENTER OF THE ADDRESSES" GET ADD1
    @I+2,J SAY "ENTER OF THE TELEPHONE NO." GET TELNO1
```

```
                PICT "999999999999"
    SET COLOR TO R+
    @I-1,J-1 TO I+3,J+45 DOUBLE
    READ
    SIG ="Y"
    @I+4,J SAY "IS THE DATA CORRECT (Y/N)" GET SIG
    READ
    IF(SIG="Y") THEN
       EXIT
    ENDIF
ENDDO
REPL NAME WITH NAME1
REPL ADDRESS WITH ADD1
REPL TELNO WITH TELNO1
SET COLOR TO
CLOSE DATABASE

*****************************FILE NAME 42*****************************
**************DELETE A RECORD WITH RESPECT TO NAME**************
SET TALK OFF
CLEAR
I=10
J=10
NAME1=SPACE(20)
SET COLOR TO GB+
@I,J SAY "ENTER CONSUMER NAME " GET NAME1
SET COLOR TO R+
@I-1,J-1 TO I+1,J+42 DOUBLE
READ
USE F4
S=0
DO WHILE .NOT. EOF()
    IF(UPPER(TRIM(NAME1))=UPPER(TRIM(NAME))) THEN
       DELETE
       PACK
```

```
        SET COLOR TO R+
        @I+3,J+2 SAY "1 RECORD DELETED !"
        S=1
     ELSE
        SKIP
     ENDIF
ENDDO
IF(S=0)THEN
     SET COLOR TO R+*
     @I+3,J+2 SAY "NO RECORDS AVAILABLE WITH THIS NAME !"
ENDIF
WAIT""
SET COLOR TO
CLOSE DATABASE

*****************************FILE NAME F43.PRG************************
*******************************MODIFY/EDIT****************************
SET TALK OFF
CLEAR
I=10
J=10
NAME1=SPACE(20)
SET COLOR TO GB+
@I,J+5 SAY "ENTRY OF THE NAME" GET NAME1
SET COLOR TO R+
@I-1,J+4 TO I+1,J+43 DOUBLE
READ
USE F4
S=0
DO WHILE .NOT. EOF()
     IF(UPPER(TRIM(NAME1))=(UPPER(TRIM(NAME)))) THEN
        S=1
        SET COLOR TO GB+
        @I+3,J+5 SAY "NAME:              " GET NAME
        @I+4,J+5 SAY "ADDRESS:           " GET ADDRESS
```

```
        @I+5,J+5 SAY "TELEPHONE NO.:" GET TELNO PICT "999999999999"
        READ
        GO BOTTOM
        EXIT
      ELSE
        SKIP
      ENDIF
ENDDO
IF(S=0) THEN
   SET COLOR TO R+*
   @I+5,J+6 SAY "NO RECORDS WITH THIS NAME EXIT !"
   WAIT""
ENDIF
SET COLOR TO
CLOSE DATABASE

**************************FILE NAME F44.PRG*****************************
*******RETRIEVE AND DISPLAY OF A RECORD OF A CONSUMER**********
SET TALK OFF
CLEAR
I=10
J=10
NAME1=SPACE(20)
SET COLOR TO GB+
@I,J SAY "ENTRY OF THE NAME" GET NAME1
SET COLOR TO R+
@I-1,J-1 TO I+1,J+38 DOUBLE
READ
S=0
USE F4
DO WHILE.NOT.EOF()
   IF(TRIM(UPPER(NAME1))=TRIM(UPPER(NAME)))THEN
      S=1
      SET COLOR TO GB+
      @I+2,J SAY "NAME       : "+NAME
```

```
        @I+3,J SAY "ADDRESS     : "+ADDRESS
        @I+4,J SAY "TELEPHONE NO.: "+STR(TELNO,12)
        EXIT
        WAIT""
     ELSE
        SKIP
     ENDIF
ENDDO
IF(S=0)THEN
   SET COLOR TO R+*
   @I+5,J SAY "NO RECORDS AVAILABEL WITH THIS NAME !"
ENDIF
WAIT""
SET COLOR TO
CLOSE DATABASE

**************************FILE NAME F45.PRG*************************
****GENERATE A COMPLETE LIST OF NAMES, ADDRESS AND TEL. NO.****
SET TALK OFF
CLEAR
USE F4
SET COLOR TO GB+
DISPLAY ALL
WAIT
SET COLOR TO
CLOSE DATABASES
```

71.4.5. Package for maintaining a database for capitals and countries

***Assignment* F5.** Create a database file containing the list of countries and their corresponding capitals. Write a program to obtain a popup menu containing the items.

- Search the capital of a country
- Search the country whose capital is given
- Add a new record
- Delete a record
- Generate a complete list of all countries and capitals.
- Exit from menu.

Write programs to access the database file to execute the above tasks.

Structure of the database file F5.DBF is shown below

Field	Field Name	Type	Width	Dec
1	COUNTRY	Character	11	
2	CAPITAL	Character	12	

```
**********FILE NAME F5.PRG**********
**********MAIN MENU PROGRAM**********
SET TALK OFF
SET STAT OFF
CH=1
DO WHILE .T.
        I=10
        J=10
        CLEAR
        SET COLOR TO GB+
        @I,J   PROM " 1. SEARCH FOR CAPITAL"
                        MESS "SEARCH ON COUNTRY TO DISPLAY CAPITAL"
        @I+1,J PROM " 2. SEARCH FOR COUNTRY" MESS "SEARCH ON
                        CAPITAL TO DISPLAY COUNTRY NAME"
        @I+2,J PROM " 3. ADD A NEW RECORD" MESS " "
        @I+3,J PROM " 4. DELETE A RECORD" MESS "DELETE A RECORD
                        WHOSE COUNTRY NAME IS GIVEN"
        @I+4,J PROM " 5. COMPLETE LIST OF ALL COUNTRY & CAPITAL"
                        MESS " "
        @I+5,J PROM " 6. EXIT FROM MENU"  MESS " "
        SET COLOR TO R+
        @I-1,J-1 TO I+6,J+44 DOUBLE
        SET COLOR TO GB+
        SET MESS TO 23
        MENU TO CH
        DO CASE
                CASE CH=1
                        DO F51
                CASE CH=2
```

```
                DO F52
            CASE CH=3
                DO F53
            CASE CH=4
                DO F54
            CASE CH=5
                DO F55
            CASE CH=6
                SET COLOR TO
                CLEAR
                EXIT
                CLOSEALL
        ENDCASE
ENDDO
SET STAT ON
SET COLOR TO

**************************FILE NAME F51.PRG************************
****************SEARCH ON COUNTRY TO GET CAPITAL****************
SET TALK OFF
CLEAR
I=10
J=10
COUNTRY1=SPACE(11)
SET COLOR TO RG+
@I,J SAY "ENTER NAME OF THE COUNTRY" GET COUNTRY1
SET COLOR TO R+
@I-1,J-1 TO I+1,J+38 DOUBLE
READ
USE F5
S=0
DO WHILE .NOT. EOF()
        IF(UPPER(LTRIM(COUNTRY1))=UPPER(LTRIM(COUNTRY))) THEN
            @I+2,J SAY "COUNTRY NAME: " +COUNTRY
            @I+3,J SAY "CAPITAL NAME: " +CAPITAL
```

```
            S=1
            READ
            EXIT
        ELSE
            SKIP
        ENDIF
ENDDO
IF(S=0)THEN
        SET COLOR TO R+*
        @I+2,J SAY "THE COUNTRY "+TRIM(COUNTRY1)+" IS NOT AVAILABLE IN THE
              DATABASE !"
        WAIT""
ENDIF
SET COLOR TO
CLOSE DATABASE

************************FILE NAME F52.PRG*****************************
***********SEARCH ON CAPITAL TO FIND COUNTRY NAME**************
SET TALK OFF
CLEAR
I=10
J=10
CAPITAL1=SPACE(12)
SET COLOR TO BG+
@I,J SAY "ENTER NAME OF THE CAPITAL" GET CAPITAL1
SET COLOR TO R+
@I-1,J-1 TO I+1,J+38 DOUBLE
READ
USE F5
S=0
DO WHILE .NOT. EOF()
        IF(UPPER(RTRIM(LTRIM(CAPITAL1)))
                        =UPPER(RTRIM(LTRIM(CAPITAL)))) THEN
            @I+3,J SAY "COUNTRY NAME:"+COUNTRY
            @I+4,J SAY "CAPITAL NAME:"+CAPITAL
```

```
            S=1
            READ
            EXIT
        ELSE
            SKIP
        ENDIF
ENDDO
IF(S=0)THEN
        SET COLOR TO R+*
        @I+2,J SAY "THE CAPITAL "+TRIM(CAPITAL1)
                    +" IS NOT AVAILABLE IN THE DATABASE !"
        WAIT""
ENDIF
SET COLOR TO
CLOSE DATABASE

*****************************FILE NAME F53.PRG*************************
**********************NEW DATA ENTRY PROGRAM**********************
SET TALK OFF
CLEAR
I=10
J=10
COUNTRY1=SPACE(11)
CAPITAL1=SPACE(12)
USE F5
APPEND BLANK
DO WHILE .T.
    I=10
    J=10
    SET COLOR TO BG+
    @I,J SAY "ENTER NAME OF THE COUNTRY" GET COUNTRY1
    @I+1,J SAY "ENTER NAME OF THE CAPITAL" GET CAPITAL1
    SET COLOR TO R
    @I-1,J-1 TO I+2,J+38 DOUBLE
    READ
```

```
    SIG ="Y"
    @I+3,J SAY "IS THE DATA CORRECT (Y/N) ?" GET SIG PICT "!"
    READ
    IF(SIG="Y") THEN
       EXIT
    ENDIF
ENDDO
REPL COUNTRY WITH COUNTRY1
REPL CAPITAL WITH CAPITAL1
SET COLOR TO
CLOSE DATABASE

**************************FILE NAME F54.PRG*****************************
**********DELETE A RECORD WHEN COUNTRY NAME IS GIVEN*********
SET TALK OFF
CLEAR
I=10
J=10
COUNTRY1=SPACE(11)
SET COLOR TO BG+
@I,J SAY "ENTRY NAME OF THE COUNTRY" GET COUNTRY1
SET COLOR TO R+
@I-1,J-1 TO I+1,J+38 DOUBLE
READ
USE F5
S=0
DO WHILE .NOT. EOF()
      IF(UPPER(TRIM(COUNTRY1))=UPPER(TRIM(COUNTRY))) THEN
            DELETE
            PACK
            SET COLOR TO R+
            @I+3,J SAY "ONE RECORD DELETED !"
            S=1
      ELSE
            SKIP
```

```
        ENDIF
ENDDO
IF(S=0)THEN
        SET COLOR TO R+*
        @I+3,J+2 SAY "RECORD WITH THIS COUNTRY NAME DOES NOT EXIST!"
ENDIF
WAIT""
SET COLOR TO
CLOSE DATABASE

*****************************FILE NAME F55.PRG*****************************
**GENERATION OF A COMPLETE LIST OF ALL COUNTRIES AND CAPITALS**
SET TALK OFF
CLEAR
USE F5
DISPLAY ALL
READ
CLOSE DATABASES
```

71.4.6. Package for pay packet of an institute

***Assignment* F6.** Create a database file containing the employee code, name and basic pay of employees of an organization. .

Write a program to draw a popup menu containing the items

- Add a new record.
- Calculation of allowances & gross income.
- Display the records of a particular employee
- Search employee code of a specified name.
- Generate a complete list of employees and corresponding records.
- Exit form menu.

Also write programs to use the pop-up menu to accomplish the all above tasks.
Add employee code, name and basic pay as new record.
Calculate D. A, H. R. A, M. A and gross income of each employee
(D. A is 64% of basic pay, H. R. A is 15% of basic pay but not more than Rs.2000, M. A is Rs.100 for all employee and Gross income = Basic pay + D. A + H. R. A + M. A)

Structure of the database file F6.DBF is shown below

Field	Field Name	Type	Width	Dec
1	NAME	Character	20	
2	ECODE	Character	5	
3	BPAY	Numerical	5	
4	DA	Numerical	7	2
5	HRA	Numerical	5	
6	MA	Numerical	4	
7	GROSS	Numerical	8	2

```
**************************FILE NAME F6.PRG**************************
************************MAIN MENU PROGRAM**************************
SET TALK OFF
SET STAT OFF
CH=1
DO WHILE .T.
    I=10
    J=10
    SET COLOR TO BG+
    CLEAR
    @I,J   PROM " 1. ADD A NEW RECORD"
    @I+1,J PROM " 2. CALCULATION OF ALLOWANCES & GROSS INCOME"
    @I+2,J PROM " 3. DISPLAY THE RECORD OF A PARTIULAR EMPLOYEE"
    @I+3,J PROM " 4. SEARCH ON EMPLOYEE NAME "
    @I+4,J PROM " 5. A COMPLETE LIST OF EMPLOYEES "
    @I+5,J PROM " 6. EXIT"
    SET COLOR TO B+
    @I-1,J-1 TO I+6,J+54 DOUBLE
    SET COLOR TO BG+
    MENU TO CH
    DO  CASE
        CASE CH=1
            DO F61
        CASE CH=2
            DO F62
```

```
        CASE CH=3
             DO F63
        CASE CH=4
             DO F64
        CASE CH=5
             DO F65
        CASE CH=6
             SET COLOR TO
             CLEAR
             EXIT
    ENDCASE
ENDDO
SET STAT ON
SET COLOR TO

************************FILE NAME F61.PRG*****************************
*********************ENTRY OF NEW RECORD*****************************
SET TALK OFF
CLEAR
I=10
J=10
NAME1=SPACE(20)
ECODE1=SPACE(5)
BPAY1=0
USE F6
APPEND BLANK
DO WHILE .T.
    SET COLOR TO GB+
    @I,J  SAY "ENTER NAME       " GET NAME1
    @I+1,J SAY "ENTER EMPLOYEE CODE" GET ECODE1
    @I+2,J SAY "ENTER BASIC PAY   " GET BPAY1 PICT "99999"
    SET COLOR TO BR
    @I-1,J-1 TO I+3,J+43 DOUBLE
    READ
    SIG ="Y"
```

```
    SET COLOR TO RG+
    @I+7,J SAY "IS THE DATA CORRECT (Y/N) ?" GET SIG
    READ
    IF(SIG="Y") THEN
        EXIT
    ENDIF
ENDDO
REPL NAME WITH NAME1
REPL ECODE WITH ECODE1
REPL BPAY WITH BPAY1
CLOSE DATABASE
SET COLOR TO
**************************FILE NAME F62.PRG**************************
***********CALCULATION OF TOTAL PAY (GROSS) AND NET PAY**********
SET TALK OFF
USE F6
DO WHILE .NOT. EOF()
    REPL DA WITH BPAY*0.64
    REPL HRA WITH BPAY*0.15
    IF(HRA>2000) THEN
        REPL HRA WITH 2000
    ENDIF
    REPL MA WITH 100
    REPL GROSS WITH BPAY+DA+HRA+MA
    SKIP
ENDDO
CLOSE DATABASES

************************FILE NAME F63.PRG**************************
**********DISPLAY THE RECORD OF A PARTICULAR EMPLOYEE**********
SET TALK OFF
CLEAR
I=10
J=10
ECODE1=SPACE(5)
```

```
SET COLOR TO BR+
@I,J+5 SAY "ENTRY OF THE EPMLOYEE CODE" GET ECODE1
SET COLOR TO R+
@I-1,J+4 TO I+1,J+37 DOUBLE
READ
USE F6
S=0
DO WHILE .NOT. EOF()
    IF(UPPER(TRIM(ECODE1))=UPPER(TRIM(ECODE))) THEN
        @I+2,J+5 SAY "EMPLOYE NAME: " + NAME
        @I+3,J+5 SAY "EMPLOYE CODE: " + ECODE
        @I+4,J+5 SAY "BASIC PAY   : " +STR(BPAY,5)
        @I+5,J+5 SAY "DA          : " +STR(DA,7,2)
        @I+6,J+5 SAY "HRA         : " +STR(HRA,7,2)
        @I+7,J+5 SAY "MA          : " +STR(MA,4)
        @I+8,J+5 SAY "GROSS PAY   : " +STR(GROSS,8,2)
        READ
        S=1
        GO BOTTOM
        EXIT
    ELSE
        SKIP
    ENDIF
ENDDO
IF(S=0) THEN
    SET COLOR TO R+*
    @ I+5,J+6 SAY "EMPLOYEE WITH CODE "+TRIM(ECODE)+" DOES NOT EXIST !"
    WAIT""
ENDIF
SET COLOR TO
CLOSE DATABASE

**************************FILE NAME F64.PRG*****************************
**********SEARCHING WITH RESPECT TO EMPLOYEE NAME**************
SET TALK OFF
```

```
CLEAR
I=10
J=10
NAME1=SPACE(20)
SET COLOR TO GB+
@I,J+5 SAY "ENTRY OF THE EPMLOYEE NAME" GET NAME1
SET COLOR TO GB+
@I-1,J+4 TO I+1,J+52 DOUBLE
READ
USE F6
S=0
DO WHILE .NOT. EOF()
    IF(UPPER(TRIM(NAME1))=UPPER(TRIM(NAME))) THEN
        SET COLOR TO RG+
        @I+2,J+5 SAY "EMPLOYE NAME: " + NAME
        @I+3,J+5 SAY "EMPLOYE CODE: " + ECODE
        READ
        S=1
        GO BOTTOM
        EXIT
    ELSE
        SKIP
    ENDIF
ENDDO
IF(S=0) THEN
    SET COLOR TO R+*
    @ I+5,J+6 SAY "RECORD WITH THIS NAME DOES NOT EXIST"
    WAIT""
ENDIF
SET COLOR TO
CLOSE DATABASE

**************************FILE NAME F65.PRG**************************
*********GENERATION OF A COMPLETE LIST OF EMPLOYEES AND*********
**********************CORRESPONDING RECORDS************************
```

```
SET TALK OFF
SET COLOR TO RB+
CLEAR
USE F6
DISPLAY ALL
READ
SET COLOR TO
CLOSE ALL
```

71.4.7. Package for maintaining students records

***Assignment* F7.** Write an interactive file oriented program that will maintain a list of roll numbers, name and marks of students. Include a menu that will allow the user to select any of the following activities

- Add a new record
- Delete a record
- Search a record
- Generate a complete list of roll number, names and marks.
- End of execution

Structure of the database file F7.DBF is shown below

Field	Field Name	Type	Width	Dec
1	ROLLNO	Numerical	3	
2	NAME	Character	20	
3	MARK	Numerical	3	

```
*****************************FILE NAME F7.PRG*****************************
************************MAIN MENU PROGRAM************************
SET TALK OFF
SET STAT OFF
CH=1
DO WHILE .T.
    I=10
    J=10
    CLEAR
    SET COLOR TO B+
    @I,J  PROM " 1. ADD A NEW RECORD"
    @I+1,J PROM " 2. DELETE A RECORD"
    @I+2,J PROM " 3. SEARCH A RECORD"
```

```
    @I+3,J PROM " 4. GENERATE A COMPLETE LIST OF RECORDS"
    @I+4,J PROM " 5. EXIT"
    SET COLOR TO G+
    @I-1,J-1 TO I+5,J+41 DOUBLE
    SET COLOR TO B+
    MENU TO CH
    DO CASE
       CASE CH=1
          DO F71
       CASE CH=2
          DO F72
       CASE CH=3
          DO F73
       CASE CH=4
          DO F74
       CASE CH=5
          SET COLOR TO
          CLEAR
          EXIT
          CLOSEALL
   ENDCASE
ENDDO
SET COLOR TO

*************************FILE NAME F71.PRG*****************************
************************ENTRY OF NEW RECORD*****************************
SET TALK OFF
CLEAR
I=10
J=10
ROLLNO1=0
NAME1=SPACE(20)
MARK1=0
USE F7
APPEND BLANK
```

```
DO WHILE .T.
    SET COLOR TO BG+
    @I,J   SAY "ENTER ROLL NO.   : " GET ROLLNO1 PICT "999"
    @I+1,J SAY "ENTER NAME       : " GET NAME1
    @I+2,J SAY "ENTER TOTAL MARK : " GET MARK1 PICT "999"
    SET COLOR TO R+
    @I-1,J-1 TO I+3,J+42 DOUBLE
    READ
    SIG ="Y"
    SET COLOR TO G+
    @I+4,J SAY "IS THE DATA CORRECT (Y/N) ?" GET SIG
    READ
    IF(SIG="Y") THEN
        EXIT
    ENDIF
ENDDO
REPL ROLLNO WITH ROLLNO1
REPL NAME WITH NAME1
REPL MARK WITH MARK1
CLOSE DATABASE
SET COLOR TO

*****************************FILE NAME F72.PRG*****************************
******************DELETE WITH RESPECT TO NAME**********************
SET TALK OFF
CLEAR
I=10
J=10
NAME1=SPACE(20)
SET COLOR TO B+
@I,J SAY "ENTER NAME" GET NAME1
SET COLOR TO R+
@I-1,J-1 TO I+1,J+35 DOUBLE
READ
USE F7
```

```
S=0
DO WHILE .NOT. EOF()
    IF(UPPER(TRIM(NAME1))=UPPER(TRIM(NAME))) THEN
        DELETE
        PACK
        SET COLOR TO R+*
        @I+4,J SAY "ONE RECORD DELETED !"
        S=1
    ELSE
        SKIP
    ENDIF
ENDDO
IF(S=0)THEN
    SET COLOR TO R+*
    @I+3,J+2 SAY "NO RECORDS AVAILABLE WITH THIS NAME !"
ENDIF
WAIT""
SET COLOR TO
CLOSE DATABASE

**************************FILE NAME F73.PRG**************************
*****************SEARCHING WITH RESPECT TO NAME*******************
SET TALK OFF
CLEAR
I=10
J=10
NAME1=SPACE(20)
SET COLOR TO G+
@I,J SAY "ENTER NAME" GET NAME1
SET COLOR TO GB+
@I-1,J-1 TO I+1,J+35 DOUBLE
READ
USE F7
S=0
DO WHILE .NOT. EOF()
```

```
    IF(UPPER(LTRIM(NAME1))=UPPER(LTRIM(NAME))) THEN
        SET COLOR TO RB+
        @I+2,J SAY "NAME    : " +NAME
        @I+3,J SAY "ROLL NO.: " +STR(ROLLNO,3)
        @I+4,J SAY "MARKS   : "+STR(MARK,3)
        S=1
        READ
        EXIT
    ELSE
        SKIP
    ENDIF
ENDDO
IF(S=0)THEN
    SET COLOR TO R+*
    @I+2,J SAY "NO RECORDS AVAILABLE WITH THIS NAME !"
    WAIT""
ENDIF
SET COLOR TO
CLOSE DATABASE

**************************FILE NAME F74.PRG**************************
*******GENERATION OF A COMPLETE LIST OF ALL RECORDS*************
SET TALK OFF
CLEAR
SET COLOR TO RG+
USE F7
DISPLAY ALL
READ
CLOSE DATABASES
```

71.4.8. Package for writing a common letter to the members of an organization

***Assignment* F8.** Create a database file which contains the following fields:

- Name
- Sex
- Address
- C/O

- Vill/Street
- P.O.
- Dist.
- State
- PIN

Write program to enable the pop-up menu to perform the following task (Put prefix Mr. before male and Mrs./Miss. before female candidate). Prepare letters using TEXT ... ENDTEXT statement.

- Add a new record.
- Prepare a letter to a member whose name is given.
- Prepare letters of all member of same District.
- Prepare letters to all members of same PIN code area.
- Prepare letters to all members of same state.

Structure of the database file F9.DBF is shown below.

Field	Field Name	Type	Width	Dec
1	NAME	Character l	20	
2	SEX	Character	1	
3	ADDRESS	Character	25	
4	CAREOF	Character	20	
5	VILL	Character	15	
6	PO	Character	15	
7	DIST	Character	20	
8	STATE	Character	13	
9	PIN	Numeric	6	

```
**************************FILE NAME F9.PRG*****************************
**********************MAIN MENU PROGRAM*****************************
SET TALK OFF
SET STAT OFF
CH=1
DO WHILE .T.
    I=10
    J=10
    CLEAR
    SET COLOR TO RG+
    @I-1,J PROM "  ADD NEW RECORD "
```

```
    @I,J  PROM "  SEND LETTER TO A MEMBER"
    @I+1,J PROM "  SEND LETTERS TO ALL MEMBERS OF SAME DIST"
    @I+2,J PROM "  SEND LETTERS TO ALL MEMBERS OF SAME STATE"
    @I+3,J PROM "  SEND LETTERS TO ALL MEMBERS OF SAME PIN "
    @I+4,J PROM "  EXIT"
    SET COLOR TO B+
    @I-2,J-1 TO I+5,J+46 DOUBLE
    SET COLOR TO RG+
    MENU TO CH
    DO CASE
        CASE CH=1
            DO F90
        CASE CH=2
            DO F91
        CASE CH=3
            DO F92
        CASE CH=4
            DO F93
        CASE CH=5
            DO F94
        CASE CH=6
            CLEAR
            EXIT
            CLOSEALL
   ENDCASE
ENDDO
SET STAT ON
SET COLOR TO

**************************FILE NAME F90.PRG*****************************
************************ENTRY OF NEW RECORD*****************************
SET TALK OFF
CLEAR
I=5
J=5
```

```
NAME1=SPACE(20)
SEX1="M"
ADDRESS1=SPACE(25)
CAREOF1=SPACE(20)
VILL1=SPACE(15)
PO1=SPACE(15)
DIST1=SPACE(20)
STATE1=SPACE(13)
PIN1=0
USE F9
APPEND BLANK
DO WHILE .T.
    SET COLOR TO GB+
    @I,J    SAY "ENTER NAME"      " GET NAME1
    @I+1,J SAY "ENTER SEX (F/M)  " GET SEX1
    @I+2,J SAY "ENTER ADDRESS  " GET ADDRESS1
    @I+3,J SAY "C/O                  " GET CAREOF1
    @I+4,J SAY "VILL                 " GET VILL1
    @I+5,J SAY "P.O.                 " GET PO1
    @I+6,J SAY "DISTRICT             " GET DIST1
    @I+7,J SAY "STATE                " GET STATE1
    @I+8,J SAY "PIN CODE             " GET PIN1 PICT "999999"
    SET COLOR TO BR
    @I-1,J-1 TO I+9,J+40 DOUBLE
    READ
    SIG ="Y"
    SET COLOR TO RG+
    @I+12,J SAY "IS THE DATA CORRECT (Y/N) ?" GET SIG
    READ
    IF(SIG="Y") THEN
        EXIT
    ENDIF
ENDDO
REPL NAME WITH NAME1
```

```
REPL SEX WITH SEX1
REPL ADDRESS WITH ADDRESS1
REPL CAREOF WITH CAREOF1
REPL VILL WITH VILL1
REPL PO WITH PO1
REPL DIST WITH DIST1
REPL STATE WITH STATE1
REPL PIN WITH PIN1
CLOSE DATABASE
SET COLOR TO
*****************************FILE NAME F92.PRG*************************
********SENDING LETTER TO A MEMBER WHOSE NAME IS GIVEN**********
SET TALK OFF
I=10
J=10
CLEAR
NAME1=SPACE(20)
SET COLOR TO G+
@I,J SAY "ENTER NAME " GET NAME1
READ
S=0
USE F9
DO WHILE .NOT. EOF()
    IF(UPPER(TRIM(NAME1))=UPPER(TRIM(NAME))) THEN
        CLEAR
        S=1
        SEX1="Miss."
        IF(SEX="M") THEN
            SEX1="Mr."
        ENDIF
        ?"Dear"+SPACE(1)+SEX1+NAME
        TEXT
        You are well now .....
                                Your faithfully
                                  Pancha
```

```
        ENDTEXT
        ?"To"
        ?SEX1+" "+ NAME
        ?"C/O  : " +CAREOF
        ?"Vill : " +VILL
        ?"P.O. : " +PO
        ?"Dist : " +DIST
        ?"Pin  : "+STR(PIN,7)
        ?"State: " +STATE
        WAIT""
    ENDIF
    SKIP
ENDDO
IF(S=0) THEN
    SET COLOR TO R+*
    @I+5,J+5 SAY "NO RECORDS EXIST WITH THIS NAME !"
ENDIF
WAIT""
SET COLOR TO
CLOSE DATABASES

*****************************FILE NAME F92.PRG************************
********SENDING LETTER TO THE MEMBERS WITH SAME DISTRICT********
SET TALK OFF
I=10
J=10
CLEAR
DIST1=SPACE(20)
SET COLOR TO G+
@I,J SAY "ENTER DISTRICT NAME " GET DIST1
READ
S=0
USE F9
DO WHILE .NOT. EOF()
    IF(UPPER(TRIM(DIST1))=UPPER(TRIM(DIST))) THEN
```

```
        CLEAR
        S=1
        SEX1="Miss."
        IF(SEX="M") THEN
            SEX1="Mr."
        ENDIF
        ?"Dear"+SPACE(1)+SEX1+NAME
        TEXT
        You are well now .....
                                    Your faithfully
                                      Pancha
        ENDTEXT
        ?"To"
        ?SEX1+" "+ NAME
        ?"C/O  : " +CAREOF
        ?"Vill : " +VILL
        ?"P.O. : " +PO
        ?"Dist : " +DIST
        ?"Pin  : "+STR(PIN,7)
        ?"State: " +STATE
        WAIT""
    ENDIF
    SKIP
ENDDO
IF(S=0) THEN
    SET COLOR TO R+*
    @I+5,J+5 SAY "NO RECORDS EXIST WITH THIS DISTRICT NAME !"
ENDIF
WAIT""
SET COLOR TO
CLOSE DATABASES

**************************FILE NAME F93.PRG**************************
********SENDING LETTER TO ALL MEMBERS WITH SAME STATE*********
SET TALK OFF
```

```
I=10
J=10
CLEAR
STATE1=SPACE(13)
SET COLOR TO G+
@I,J SAY "ENTER STATE:" GET STATE1
READ
S=0
USE F9
DO WHILE .NOT. EOF()
    IF(UPPER(TRIM(STATE1))=UPPER(TRIM(STATE))) THEN
        S=1
        SEX1="Miss."
        IF(SEX="M") THEN
            SEX1="Mr."
        ENDIF
        CLEAR
        ?"Dear"+SPACE(1)+SEX1+NAME
        TEXT
        You are well now .....
                                    Your faithfully
                                      Pancha
        ENDTEXT
        ?"To"
        ?SEX1+" "+NAME
        ?"C/O  : " +CAREOF
        ?"Vill : " +VILL
        ?"P.O. : " +PO
        ?"Dist : " +DIST
        ?"Pin  : "+STR(PIN,7)
        ?"State: " +STATE
        WAIT""
        CLEAR
    ENDIF
```

```
    SKIP
ENDDO
IF(S=0)THEN
    @I+5,J+5 SAY"NO SUCH STATE EXIT!"
ENDIF
WAIT""
CLOSE DATABASES
**************************FILE NAME F94.PRG*************************
***************SENDING LETTER WITH SAME PIC CODE******************
SET TALK OFF
SET STAT ON
I=10
J=10
CLEAR
PIN1=0
SET COLOR TO BG+
@I,J SAY "ENTER PINCODE:" GET PIN1 PICT "999999"
SET COLOR TO RB+
@I-1,J-1 TO I+1,J+28 DOUBLE
READ
S=0
USE F9
DO WHILE .NOT. EOF()
    IF(PIN1=PIN) THEN
        CLEAR
        S=1
        SEX1="Miss."
        IF(SEX="M") THEN
            SEX1="Mr."
        ENDIF
        ?"Dear"+SPACE(1)+SEX1+NAME
        TEXT
        You are well now .....
                                Your faithfully
                                  Pancha
```

```
        ENDTEXT
        ?"Name : " +NAME
        ?"C/O  : " +CAREOF
        ?"Vill : " +VILL
        ?"P.O. : " +PO
        ?"Dist : " +DIST
        ?"Pin  : "+STR(PIN,7)
        ?"State: " +STATE
        WAIT""
    ENDIF
    SKIP
ENDDO
IF(S=0)THEN
    SET COLOR TO R+*
    @I+5,J+5 SAY "NO RECORDS WITH THIS PIN CODE DO NOT EXIT !"
ENDIF
WAIT""
SET COLOR TO
CLOSE DATABASES
```

71.4.9. Program to controlling a Dot Matrix printer

***Assignment* F9.** Write a program in FoxBase+ to print the following matter

Fourth Reunion Celebration

Dear Sir/Madam

You are cordially invited in the occasion of **fourth Reunion** to be held on **January 17, 2007.**

Thanking you,
Sincerely yours,

Secretary,
Fourth Reunion Celebration Committee.

(Use the specified fonts shown above and attach an output copy printed on dot-matrix printer)

```
********** FILE NAME F10.PRG **********
******** PROGRAM TO WRITE A LETTER WITH SPECIFIC FONT STYLE ******
SET TALK OFF
SET STAT OFF
?CHR(27)+"W1"+SPACE(20)+"Fourth Reunion Celebration"
?CHR(27)+"W0"
?SPACE(1)
?CHR(27)+"4"+"Dear Sir/Madam"
??CHR(27)+"5"
?SPACE(8)+"You are cordially invited in the occasion of "
??CHR(27)+"G"+"Fourth Reunion"
??CHR(27)+"H"+"to be held"
??"on"
??CHR(27)+"G"+"January 17, 2007."
??CHR(27)+"H"
?SPACE(8)+"Thanking you,"
?SPACE(8)+"Sincerely yours,"
?
?
?
?SPACE(34)+"Secretary,"
?SPACE(34)+"Fourth Reunion Celebration."
SET TALK ON
SET STAT ON
```

71.30. Module Summary

In this module, we have designed some interactive database packages. In these packages, most of the features of foxpro are covered. The pop-up menu is designed for all programs with different colours and boarders.

71.31. Self Assessment Questions

1. Suppose a database file contains the fields NAME, ROLLNO, AGE and ADDRESS. Write an interactive program to enter five records.
2. Suppose a database file contains the fields NAME, ROLLNO, AGE and ADDRESS. Write an interactive program to modify a particular record when the value of the NAME filed is given.
3. Suppose a database file contains the fields NAME, ROLLNO, AGE and ADDRESS. Write an interactive program to delete some records when the ROOLNO of such records are supplied.

4. Create a database file for some students with the following fields
    - Name
    - Roll No.
    - Date of birth

   Write a program in FoxBase+ to draw the pop-up menu containing the following items
    - Display the records of each student in a given format.
    - Display the records of a particular student whose roll no. is given in the given format.
    - Adding of new record(s) .
    - Sorting on roll no. or name.
    - Listing of roll no. and name of all records of the sorted file.
    - Exiting from the menu.

   Also implement all the menu items.

5. Create a database file for the M.Sc. Part-I students of Applied Mathematics, V.U. having the following fields
    - Roll no.
    - Name
    - Address
    - Marks of Paper-I Gr. A
    - Marks of Paper-I Gr. B
    - Marks of Paper-I Gr. C
    - Marks of Paper-II Gr. A
    - Marks of Paper-II Gr. B
    - Marks of Paper-III Gr. A
    - Marks of Paper-III Gr. B
    - Marks of Paper-III Gr. C
    - Marks of Paper-IV Gr. A
    - Marks of Paper-IV Gr. B
    - Marks of Paper-V Gr. A
    - Marks of Paper-V Gr. B
    - Total marks
    - Percentage of marks
    - Class obtained

   Write a program in FoxBase+ to draw the pop-up menu containing the following items
    - Adding record(s) of new students.

- Edit/modify all the records of a particular student whose roll no. is given.
- Entry the marks of
- Specified group of a specified paper.
    - All group of all paper.
- Display the records of all students.
- Preparation of result by calculating total marks, percentage of marks and class.
- Tabulation.
- Printing of mark sheet.
- Quit from the menu.

Write program to implement all of the above tasks.

6. Create two database files of which are contains the records of books fields
    - Title
    - Author
    - Acc no.
    - Call no.
    - Subject
    - Branch
    - Issuing card no.
    - Date of issue

and the other file contains the record of member with the following fields

- Name
- Roll No (if student)
- Department
- Year (if student)
- Status (student/teacher/non teaching, officer etc.)
- Acc no.1
- Acc no.2
- Acc no.3
- Card no.

(assume that a member can issue at most three books at a time).

Write a program to draw the pop-up menu containing the items.

- Entry of new book
- Entry of new member
- Listing of books issued by a member

- Searching of a book according to
  - Acc no.
  - Subject
  - Branch
  - Title
  - Author
- Exit from menu.

Now write FoxBase programs to implement all the tasks.

7. Create a database file that will maintain a list of
   - Names
   - Addresses
   - Telephone numbers

   Write a program in FoxBase+ to draw a pop-up menu that contains the following items
   - Add new record(s)
   - Delete a record
   - Modify a record
   - Retrieve and display an entire record for a given name
   - Generate a complete list of all records.
   - Exit from the menu.

   Also write programs to implement all of the above tasks.

8. Create a database file containing the list of countries and their corresponding capitals.

   Write a program to obtain a popup menu containing the items.
   - Search the capital of a country
   - Search the country whose capital is given
   - Add a new record
   - Delete a record
   - Generate a complete list of all countries and capitals.
   - Exit from menu.

   Write programs to access the database file to execute the above tasks.

9. Create a database file containing the employee code, name and basic pay of employees of an organization..

   Write a program to draw a popup menu containing the items
   - Add a new record.
   - Calculation of allowances & gross income.
   - Display the records of a particular employee

- Search employee code of a specified name.
- Generate a complete list of employees and corresponding records.
- Exit form menu.

Also write programs to use the pop-up menu to accomplish the all above tasks.
Add employee code, name and basic pay as new record.
Calculate D. A, H. R. A, M. A and gross income of each employee
(D. A is 60% of basic pay, H. R. A is 15% of basic pay but not more than Rs.2000, M. A is Rs.100 for
all employee and Gross income = Basic pay + D. A + H. R. A + M. A)

10. Write an interactive file oriented program that will maintain a list of roll numbers, name and marks of students. Include a menu that will allow the user to select any of the following activities
    - Add a new record
    - Delete a record
    - Search a record
    - Generate a complete list of roll no., names and marks.
    - End of execution

11. Suppose a file PARTI contains the roll no.; name, and total marks (out of 400) of Part-I exam. Another file P ART2 contains the roll no., name of those students in the same order as in the file PARTI with marks in four papers PAPER5, PAPER6, PAPER?, PAPER8 of Part-II examination. Write a program to prepare the result which to stored in the file RESULT of this examination containing roll no., name, marks in Part-I, total marks of Part-II, total percentage of marks and class.

    The class may be determined as:
    % of marks ≥ 60 gets 1 st class
    ≤ 45 % of marks < 60 gets 2nd class
    .....% of marks <45 fail
    Display the complete list of all fields of the file RESULT.

12. Create a database file which contains the following fields:
    - Name
    - Sex
    - Address
    - C/O
    - Vill/Street
    - P.O.
    - Dist.
    - State

- PIN

Write a program to draw the pop-up menu containing the items

- Send letter to each member
- Send letter to the members of some
- Dist.
- State
- PIN

Also write program to enable the pop-up menu to perform the following task (Put prefix Mr. before male and Mrs./Miss. before female candidate)

- Prepare letters to each member using TEXT ENDTEXT statements where the body of the letter is to be given. Prepare letters of all member of same District.
- Prepare letters to all members of same PIN code area.
- Prepare letters to all members of same district.
- Prepare letters to all members of same state.

13. Suppose a file PARTI contains the roll no.; name, and total marks (out of 400) of Part-I exam. Another file PART2 contains the roll no., name of those students in the same order as in the file PARTI with marks in four papers PAPER5, PAPER6, PAPER7, PAPER8 of Part-II examination. Write a program to prepare the result which to stored in the file RESULT of this examination containing roll no., name, marks in Part-I, total marks of Part-II, percentage of marks and class.

    The class may be determined as:

    % of marks ≥ 60 gets 1st class

    a. 45 % of marks < 60 gets 2nd class

    % of marks <45 fail

    Display the complete list of all fields of the file RESULT.

71.32. Further Suggested Readings

1. dBase III Plus Made Simple with dBase IV and FoxBase+, R.K.Taxali, Tata McGraw-Hill.
2. Special Edition Using Visual FoxPro 5, Antonovich, et. al., PHI.
3. Special Edition Using Visual FoxPro 6, Bazian., PHI.

M.Sc. Course in
Applied Mathematics with Oceanology
and
Computer Programming

Part-II

Paper-VI Group-B

Module No. 72
Lab. 6: Internet

Structure

72.1. Introduction
72.2. Objectives
72.3. Keywords
72.4. Working with e-mail
    72.4.1. Sending Mail
    72.4.2. Sending Mail to People on Other Networks
    72.4.3. Reading Mail
    72.4.4. Replying to Mail
    72.4.5. Deleting Mail
72.5. Sending Mail to Multiple Recipients
72.6. Sending Files via E-Mail
72.7. Forwarding Mail
72.8. Managing an Address Book
72.9. What's New on the Web?
72.10. Web Addresses (URLs)
72.11. The Browser
72.12. Navigation
    72.12.1. Entering to a Site
    72.12.2. Opening a file available in the local hard disk
    72.12.3. Going back Home
    72.12.4. Searching
    72.12.5. Stop

72.12.6. Backward
72.12.7. Forward
72.12.8. Favorites
72.12.9. Adding to Favorites
72.13. More on Browser
72.13.1. Saving the current document
72.13.2. Source option
72.14. Searching the Web
72.14.1. How to search more effectively
72.15. Module Summary
72.16. Self Assessment Questions
72.17. Further Suggested Readings

72.1. Introduction

Todays the Internet seems to be everywhere. Web addresses appear on television advertisement, newspapers advertisement, magazines, etc. There are TV shows and magazines devoted to the Internet. And every new computer program that comes out has some Internet features. For most people, the basic appeal of the Internet is e-mail and the World Wide Web. More and more people are connecting to the net every day, and some businesses are beginning to rely on it for communication, for information, and even for transportation of certain types of products, such as software or writing. In this module you will learn about the access of the Internet.

72.2. Objectives

The main objective of this module is how we can send, read, delete, reply etc. of e-mail and attachment of file with e-mail. The other objective is how we can search on the net using any search engine.

72.3. Keywords

E-mail: read, send, reply, forward; WWW, searching.

72.4. Working with e-mail

The following items you will most often with e-mail:

- Send mail
- Read incoming mail
- Reply to mail
- Delete mail
- Exit the mail program

72.4.1. Sending Mail

All mail programs have a New Message or Compose E-mail command, often located on a message menu, and they usually have a keyboard shortcut for the command as well, such as Ctrl+N for New

Message. When you start a new message, your program will open a new window. Figure 2.2 shows a new message window.

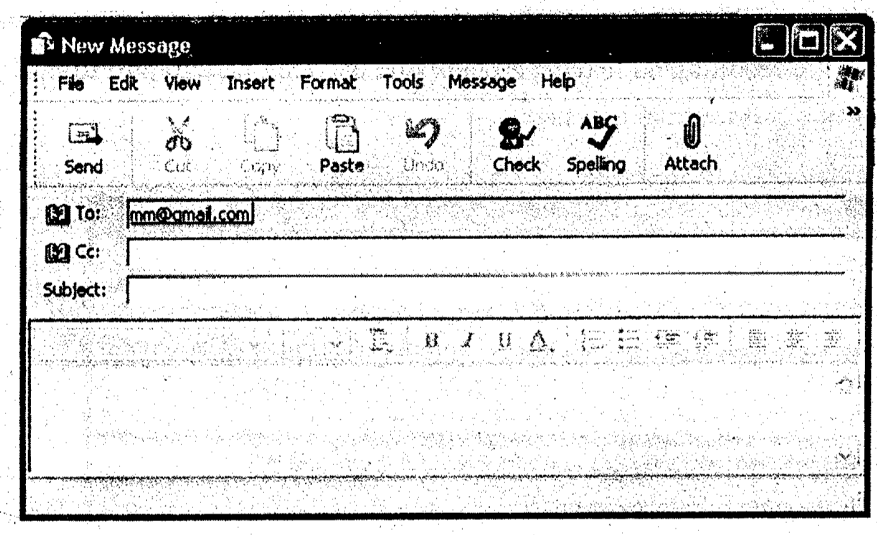


Figure 72.1. A blank New Message window.

Type the address of the person to whom you wish to send the mail. The person's address must be of the form username@address.domain, where username is the person's identifier (the name they log in with); address is the identifier of the person's network and/or machine on the network (the address might consist of several words-the host and subdomain-separated by dots); and domain is the three-letter code indicating whether the address is a business (.com), a non-profit organization (.org), a university (.edu), a branch of the government (.gov), a part of the military (.mil), and so on. (Some e-mail programs require special text before or after the Internet e-mail address.) By the way, all the rules mentioned in the previous category apply only to sending mail over the Internet. Generally, if you are sending mail to someone on your own network (or another member of your online service or subscriber of your service provider), you only have to specify the username, not any of the Internet information.

In the e-mail address mmpalvu@gmail.com (you pronounce the @ sign as "at", and the . as "dot"). This address is loged on as "mmpalvu" and its service provider is gmail and gmail is a commercial business.

72.4.2. Sending Mail to People on Other Networks

Many people have Internet addresses even though they are not, strictly speaking, on the Internet. Most other networks have gateways that send mail to and from the Internet. If you want to send mail to someone on another network, you'll need to know their identifiers on that network and how their network address appears in Internet form.

To send a copy of the e-mail message to another recipient, press Tab again and type that person's address on the Cc: line. Press Tab a few more times until the insertion point jumps into the message area.

Most mail programs can word-wrap your message, so you only have to press Enter when you want to start a new paragraph. We recommend leaving a blank line between paragraphs, to make them easier to read. Figure 72.2 shows a short e-mail message.

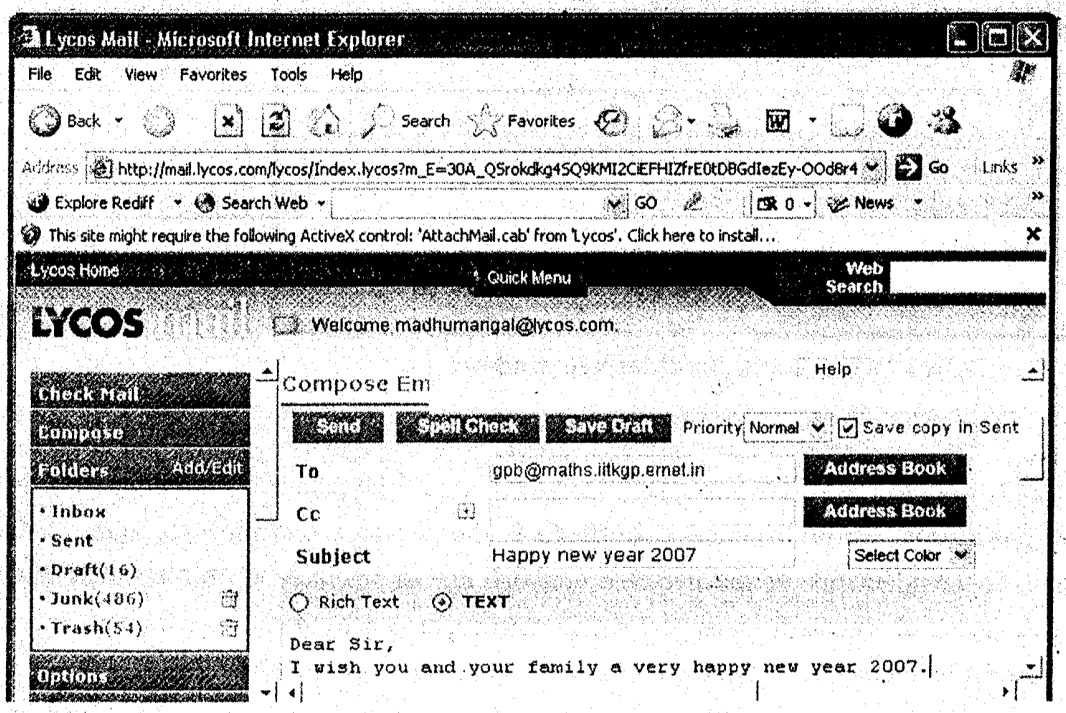


Figure 72.2 A short e-mail message.

When you are done, send the message by clicking the Send button.

72.4.3. Reading Mail

Whenever we connect to the Net, the first thing we do is check own e-mail. It's like checking your mailbox when you get home, except the contents are usually more interesting. Some mail programs combine the process of sending queued messages with checking for new mail. Most also check for new mail when you first start them. Unread (usually new) mail typically appears with some indicator that it's new, such as the Subject line appearing in bold, or a bullet or checkmark appearing next to new messages, etc.. This is supposed to help you avoid accidentally missing messages. To view the contents of a mail message, highlight it in the window and press Enter (or double-click it). The

message will appear in its own window, much like an outgoing message. Figure 2.3 shows an incoming message in Microsoft Internet Mail.

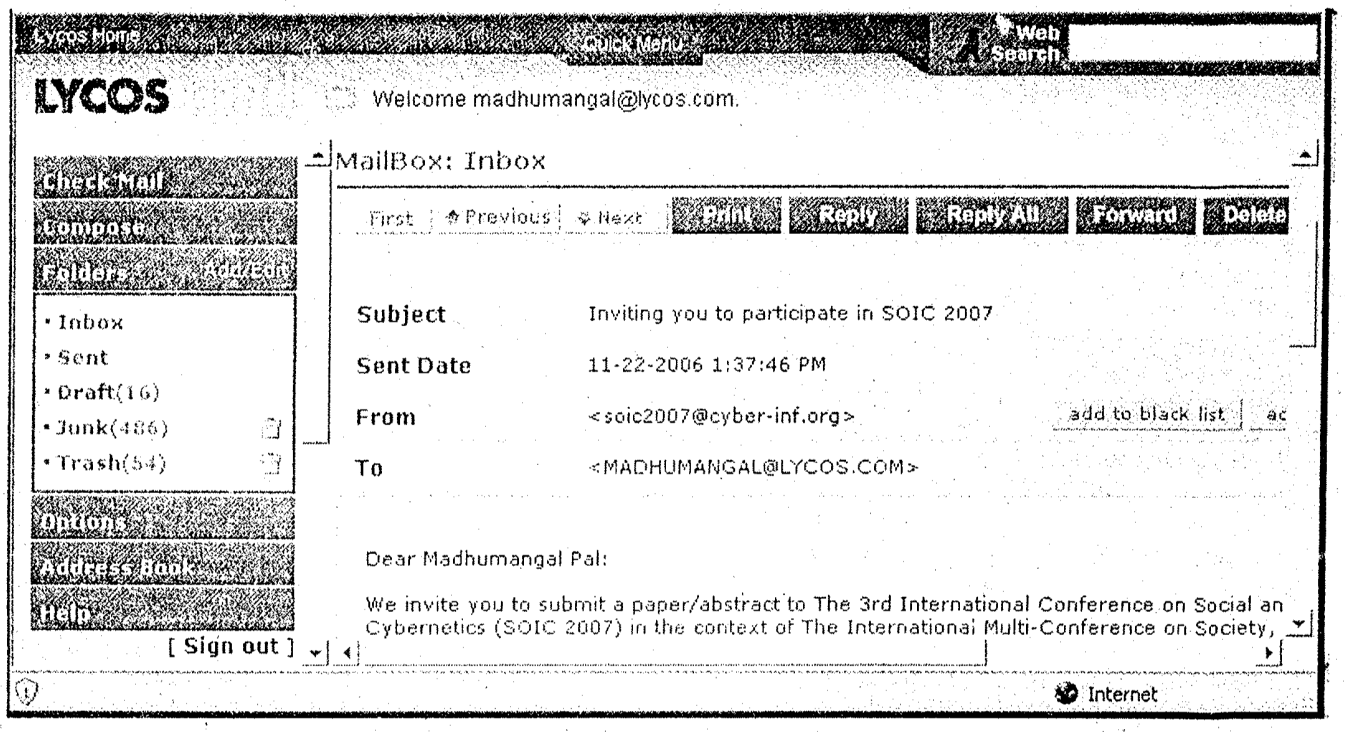


Figure 72.3. An e-mail received by madhumangal@lycos.com.

If the message continues beyond the bottom of the window, use the scroll bar to see the next screenful.

After reading the message, you can close or reply to the message.

72.4.4. Replying to Mail

Somewhere near the New Message command (probably on the same menu or button bar), you'll find the Reply command. To reply to mail, highlight the message in the Inbox, or open it and then select the Reply command.

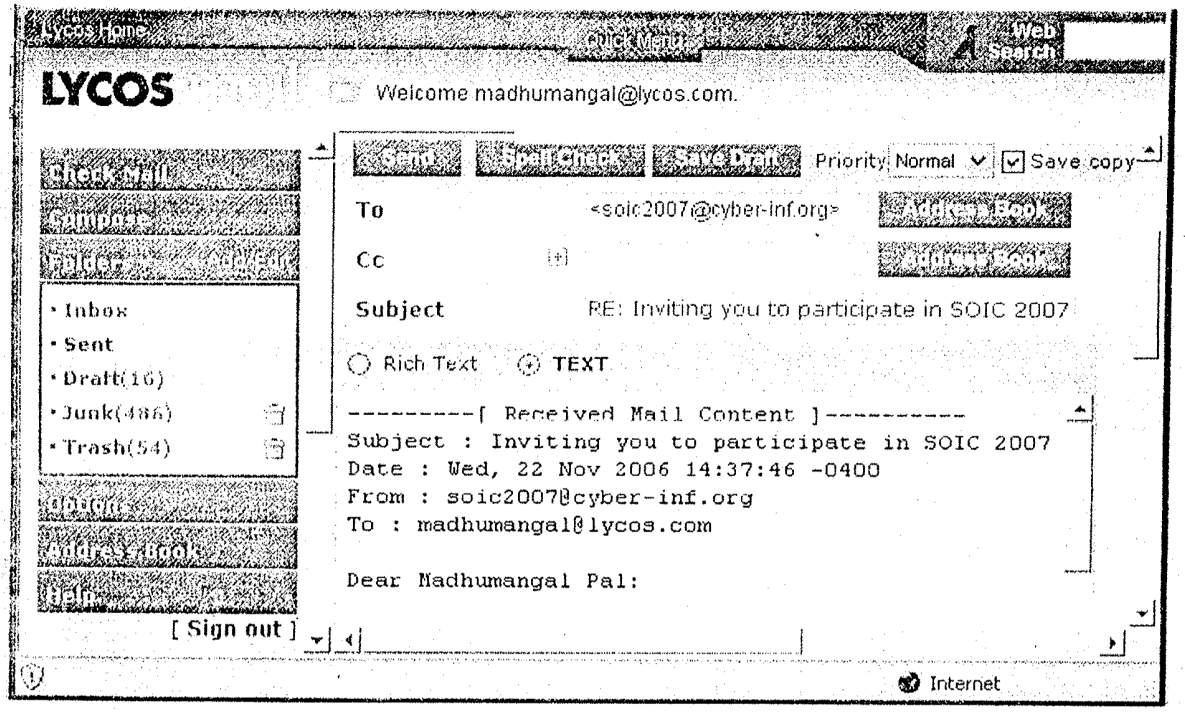


Figure 72.4. A reply with the original message included.

Your program will also start a new message automatically addressed to the sender of the message you're replying to. Some mail programs will also automatically include the contents of the original message (or will give you the choice of including the contents or not). Often, especially with e-mail programs that were designed primarily for use on the Internet, the included message will appear with each line preceded by a ">" character to indicate that it is quoted text, although different mail programs have different ways of showing quoted messages. Sometimes, you'll want to reply to everyone who was sent a copy of the original message. Most e-mail programs also offer a variation on the normal reply command that includes all original recipients in your reply.

Type a subject and press Tab. (People often fail to change the subject line of messages, even when the conversation has evolved its way onto a new topic.) Add other recipients if necessary or tab your way into the message area to type your reply, and then choose the Send (or Queue) command.

72.4.5. Deleting Mail

If you have read a piece of mail and you're positive that you have no need to save it, you should delete it so it doesn't clutter up your Inbox (and waste precious hard-disk storage space). To delete a message, you typically highlight it and press Delete (or click the Delete or Trash button, if there is

one). In most programs, this moves the message to the Deleted-mail or Trash mailbox until you empty the trash (or quit the program).

If you change your mind, try opening the Trash mailbox (or Deleted-Mail mailbox) and then looking for a command that allows you to transfer mail from one box to another. It may even be called Transfer. When you find it, transfer the mail back to your Inbox.

After you know the basics of e-mail-sending, reading, and replying to messages-there are a few more things you'll want to know about that will come in very handy as you start communicating over the Internet. Some of them are basic and available in just about every e-mail program (forwarding a message on to a new recipient, sending mail to more than one person at a time, and saving old messages somewhere besides in your inbox). Other, more advanced or sophisticated features vary from program to program.

72.5. Sending Mail to Multiple Recipients

Sometimes you'll want to send a message to more than one recipient. You can usually do this in one of several ways. Most programs allow you to list multiple recipients in the To: line, usually separated by commas (some programs require that you use a different character, such as a semicolon, to separate addresses). Most also have a Cc: line which, so far, suggested you just Tab right past to get to the message area. As with traditional paper office memos, the Cc: line, in an e-mail message is for people who you want to receive a copy of the message, but who are not the primary, recipient.

Some programs also offer a Bcc: line, which lets you list one or more people to receive blind copies of that message. This means that the primary (and Cc:) recipients will not see the names of people receiving blind copies.

72.6. Sending Files via E-Mail

Just "attach" a file to an e-mail message and it zips across the globe to your recipient, without having to be put on a disk and sent by mail, or courier. Naturally, it's not that simple. Some files are just too big to send this way (anything close to a megabyte is probably too big). But even with files of a more appropriate size, you may encounter hitches. Most of the problem is in coordinating between computer types, file types, compression formats, and encoding formats. Getting all the elements to work out can be a little like trying a combination lock. Let's start with what an attachment really is.

One of the most important functions of e-mail is its ability to let you send files called attachments along with your messages. An attachment is any data file, in any form, which your program will send along with your e-mail message. Each program is different in the way it handles file attachments, and some of the online services still don't let you send or receive files over the Internet. Also, because different programs have different ways of encoding attached files (translating the files into a form that

can be shipped over the Internet), you may have to compare details with your sender or recipient to make sure that both of your programs can "speak" the same code. Internet mail generally consists of only straight text files, although there are some protocols for sending other forms of information. For example, some mail programs use MIME (Multipurpose Internet Mail Extensions) to send other kinds of data, including colour pictures, sound files, and video clips. If you are sent e-mail with a MIME attachment, you may not be able to see the pictures, hear the music, or view the movies, but the text in the attachment should come through just fine. You'll be asked if you want to view it or save it.

Quite aside from e-mail issues, you might be trying to send a file that your recipient doesn't have the right application for reading, so that's another thing you may have to work out in advance. For example, if you use Word for the Macintosh and your recipient uses Word Perfect for DOS, then you may have to save your file in a format that your recipient's program can understand; this may involve both of you poking around your program's Open and Save As commands to see what options are available.

When you receive a file attachment, your e-mail program will usually decode it and tell you where it's been placed (unless it doesn't recognize the coding format, in which case you'll get a bunch of garbage at the end of the message and no file attachment, if this happens, you need to negotiate with it).

72.7. Forwarding Mail

If someone sends you mail and you'd like to send a copy of it to someone else, with most mail programs, you can select a Forward command. The Forward command is often near (such as on the same menu as) the Reply command, and it works in almost the same way. The difference is that your mail program won't insert the original sender's e-mail address into the To: line. Instead it will be blank, as in a new message. But the original message will be automatically included in the new message, often with some characters (like the standard ">" Internet e-mail quoting character) or other formatting to distinguish it from what you yourself write.

To forward a message, type the recipient's name on the To: line and then Tab your way down to the message area. Edit the message and/or add your own note, perhaps an explanation of why you are forwarding the message, to the beginning. Then send the message as usual.

72.8. Managing an Address Book

If there are people with whom you correspond regularly through e-mail or people whose e-mail addresses are hard to remember or difficult to type correctly, most e-mail programs enable you to create aliases (sometimes called nicknames) for these people. Aliases are shorter words that you type instead of the actual address. These lists of addresses are usually grouped together in something called

an address book. Modeled on real-world address books, these windows or modules often have room for other vital info (such as street addresses, and phone and fax numbers).

Some e-mail and groupware programs share a single address box with other applications on your computer, so your contact information is available to various programs.

When you type an alias or choose a name from an address book, your e-mail program inserts the correct address into the To: line of your message (some programs can also insert an address into the Cc: line). You can also set up an alias for a list of addresses so you can send mail to a group of people all at once.

72.9. What's New on the Web?

More and more day-to-day applications are gaining the capability to browse the Web and display documents in the web format (HTML). The competition among web browsers is fierce, with Netscape and Microsoft duking it out for the dominant share of the market. Every viable browser includes mail and news modules as well, with web-editing quickly becoming another common feature. Upcoming releases of popular browsers will blend together your desktop view of your own computer with the web interface. Meanwhile, in-house "intranets" these days incorporate most of the features of web sites and local, small-scale networks run along the lines of the Internet.

Those are the changes taking place in the world of tools. On the Web itself, commercial sites keep sprouting up like mushrooms, although there is no proven business model yet for making money on the Web. A brave new method for delivering content to your screen via the Web, called "intranet" allows you to surf the Web passively.

72.10. Web Addresses (URLs)

If you've ever noticed an advertisement that says "check out our web site"; you have seen the arcane way they describe how to find it, giving you the address, or URL, which almost always begins with the letters http:. (URL stands for Uniform Resource Locator). A web address starts with the name of a protocol (a method for connecting to information); most of the time it is HTTP. HTTP stands for Hypertext Transfer Protocol and means that the resource in question will be found on a web server. This protocol is followed by :// (the oft-heard "colon-slash-slash" or even more cumbersome, "colon-forward-slash-forward-slash"), and then an Internet address of the form site.subdomain.domain. After that, you might find a colon and a port number (this is fairly rare), a path which is a list of folders or direction names leading to the resource in question, separated by more forward slashes. After the path, you might find a file name, which often ends with the extension .html. An HTML file (which means Hypertext Markup Language file) is the primary type of document on the Web.

Fortunately, most of the time, you won't have to type in web addresses yourself because you'll be following links that have the URL encoded into them.

72.11. The Browser

The Internet browser is a software which communicates between user and the Internet. Browsers are also referred to as Web Clients, or Universal Clients, because in their client/server model, the Browser functions as the client program.

The Browser do the following tasks:

- It sends a request to a web server for information.
- It receives the information and then displays it on the user's computer.

A Browser can be graphical, text-based and intuitive. A Graphical Browser allows us to view images on the computer, `point and click' with a mouse, hypertext links, use drop-down menus and toolbar buttons to navigate and access resources on the Internet. The WWW (World Wide Web) incorporates hypertext, images, sound, video, etc. It can be fully exploited with a graphical browser.

Browsers often include `helper applications' which are actually software programs that are needed to display images, hear sounds or run animation sequences. These helper applications are automatically invoked by the Browser when a user selects a link to a resource that requires them.

Many different Browsers are available today. All perform the same basic functions (transferring hypertext) but many support special features that are unique.

Examples of common Browsers are:

- Microsoft Internet Explorer
- Netscape Communicator.

72.12. Navigation

Here we discuss the method to navigate through the Internet using Microsoft Internet Explorer.

72.12.1. Entering to a Site

If the address of an Internet site is known then we can enter to this site by simply typing the address of that site on the address bar. For example, click on the address bar and type **www.google.com** and press then press Enter key. On the Status Bar of the Internet Explorer, we will see messages such as `Finding Site', `Web Site Found, awaiting reply', etc. If the web site has been found, the opening page of the **google** web site will be displayed on the monitor. The previously typed address can be selected from the list that is displayed by the Address Bar.

If the Address box is not visible then we can select Address Bar from the View menu.

72.12.2. Opening a file available in the local hard disk

If we develop web pages then it will be stored on the local hard disk. Such files can also be opened with the help of the Browser to see how they will eventually appear on the Internet. When this is done, the dynamic icon will become static.

To open a web page from the local hard disk (available in the local machine) follow these steps:

- From the **File menu**, select **Open**.
- Click **Open File** button, select the file to open. But, this will over-write the present contents of the window.
- We can open a file in a new window, by checking the box **Open in new window** and clicking **OK**.

72.12.3. Going back Home

We can browsed through a lot of web pages of different web sites and we would like to return to the Start page. This can be done by clicking the Home button.

72.12.4. Searching

A search page is a collection of other Internet sites. The search pages are, in general, different. Some search pages provide searching capabilities, such as the ability to search by topic or by keyword. Others simply provide a well-organised list of links to selected Internet sites.

We can quickly go to the Search page by clicking the Search button on the toolbar. To go to the Search page from any page, click on the Search button.

72.12.5. Stop

Generally, a web page contains many of pictures. Some we may be only interested in the text part of the document which loads first. To stop loading other details such as images and other unwanted information, click on the Stop button. This would not affect the contents of the web page already displayed but would stop display of any further contents.

'Stop' will also be useful when a wrong address is inadvertently entered and some unwanted information starts loading.

72.12.6. Backward

Suppose we have browsed through a number of pages. To go back to the previous page click on the Back button on the last visited page.

72.12.7. Forward

To go to the next page that we have already visited, click on the Forward button. This key will be useful to go forward, immediately after going backward.

72.12.8. Favorites

Favorites is a folder which stores all the web addresses as Internet Shortcut with an extension .URL. Web site www.rediffmail.com provides free e-mail accounts to the registered users. Suppose we have an e-mail account named asarkar@rediffmail.com, where we receive all mails. To view the mails we would have to go to the web site. Suppose we would be checking for mails once in the morning and evening. Therefore, this site is a frequently visited one. To go to the web site, we would need to type

the address of the site again and again. Instead, we can add the address to favorites and by selecting this address from the list we can go to the site. Similarly, all frequently visited sites can be listed from the favorites list.

We can open the Favorites folder by selecting **Open Favorites** from the **Favorites** menu or by clicking **Favorites** button from the toolbar.

72.12.9. Adding to Favorites

When a page is on display, clicking **Add to favorites** button will help to add the current address to the favorites lists. When the Browser does this, it prompts for a name which we would refer to (or leave it to be the default). We can also choose multiple folders to store favorites of different types.

72.13. More on Browser

72.13.1. Saving the current document

To make a local copy of the current web page, choose **File** menu option and **Save As File** menu item. This will store only the textual portions of the page. To save the picture in the page we have to click the right mouse button on every single picture and choose **Save Picture** from the menu pop-up that appears next to the picture. It may be of interest to know that the Browser automatically adds pictures to the cache1, cache2, cache3, cache4 folders under **C:\Windows\Temporary Internet Files.**

72.13.2. Source option

An HTML file usually consists of printable text characters. It is only the web browser that interprets this language and produces the visual effects from it. To view the HTML source script that is producing the visual effect, we choose from the **View** menu option, the **Source** menu item. This will display the HTML script.

72.14. Searching the Web

The Internet is a vast collection of information on almost any topic in the world. How we can get information on the Internet or the World Wide Web ? There are specific sites called Search Engines to help people who are interested to search an information.

These sites maintain a database of the numerous sites and pages that are available on the WWW. When we get to such a site it displays an on-line form into which we can enter a string of text or keywords to indicate the information we are seeking. The Search Engine then looks up the database and presents a web page of links to possible sites and pages that satisfy our query. Search Engines like Google (www.google.com), AltaVista (www.altavista.com), Excite (www.excite.com), Yahoo (www.yahoo.com), Infoseek (www.infoseek.com), Lycos (www.Lycos.com) and Webcrawler (www.webcrawler.com) are popular sites which are accessed by millions of users each day. The Search Engines keep updating their database of web sites and pages through two methods. Anybody who has created a site or Web page can go to that site and add his URL or address directly into the database along with some other information. The user can then reach that site with just one click. Some of the

Search Engines not only check for keywords that occur within a file name or the description but also intelligently guess as to what the user is looking for. For example, if we type the word and ask the Search Engine to look for the word 'Kids', it not only lists all documents where the word 'Kids' appears but also looks for other words like 'Children' and 'Babies'. This kind of search is called a Concept Search.

72.14.1. How to search more effectively

Search Engines accept a combination of words in the query following certain conjunction rules to filter the search process and generate results more effectively.

1. **Multiple keyword searches:** When we give two words separated by a space, then a search engine looks for documents containing either of these two words. For example, typing Gold and Platinum will find all documents which contain the words Gold and Platinum. Some Search Engines intelligently check to see which of the documents have both the words and then place those links first in the line for our decision.
2. **Case sensitive searches:** Most of the Search Engines are case sensitive; which means that **Number** is not the same as **number**. But typing **paris** will also find a document with the word **Paris**. If we type **paRIS** we may most probably not find any matches at all. Capital letters enforce an exact case match on the entire word and therefore when looking for proper nouns spell them only in upper case. For acronyms like ISO or UN you should use capitals or upper case. When looking for words which could appear in capitals or lower case, it is better to use lowercase.
3. **Using the * (wild card) notation:** If we want to search for words **sing, singer, singing** then we can use the * notation. For example, 'sing*' will find the words with similar patterns. But when using the * notation we must be careful because it will also find words like **single, singular or singalong**, etc. We can use the * notation before or after a word and even in between a word. Care should be taken when we construct our query using the * notation. If we are looking for words matching **color** or **colour** using col*r will get matches like **collector** and **collider**.
4. **Quotes:** A complete phrase can be searched by putting quotations. For example, if we type **"Fibonacci Numbers"** we will get web pages which have the entire phrase Fibonacci Numbers and not all the documents which have Fibonacci and Numbers. When we are looking for a particular phrase within a page, it is better to use the double quote marks before and after the phrase. This is one of the best ways to restrict searches on the Web, because when looking for phrases, instead of keywords, the number of matches is reduced drastically. This ensures that we have the relevant information rather than millions of documents.
5. **Using + and - signs to find files:** The string **- circle + rectangle** will find all documents that have the word **rectangle** but do not have the word **circle**. Using the + (Plus) and - (minus)

signs can help prune the matches from the search and will eliminate what we are not looking for.

6. **Parentheses:** The string - **(circle + rectangle) + square** will find all documents which contain **square** but not either of the words **circle** or **rectangle**. Using parentheses, we can further constrain our searches to find the best matches for searches.

There are many Search Engines on the Web and each of them follow some variations in how they search, what syntax can be used to search and how queries are ranked are displayed. Many of them provide help for using the Search Engine. Most of the large sites on the Web today have their own Search Engines which will help us to find information within that site. Sites like google.com, microsoft.com, yahoo.com and others have their own Search Engines. This will soon become a common feature on all large web sites to help user find information within that web site. Search Engines like Altavista, Excite, Lycos, etc., do not find information related to just one site but cover a fair number of sites and pages on the Web. It is nearly impossible for any Search Engine to index all the web sites and pages because millions of pages get added every day.

72.15. Module Summary

In this module we introduce the concept of e-mail and its use, such as sending, reading, deleting, replying, forwarding, etc. Also, we introduce the basic idea for searching on Internet using search engine with different techniques.

72.16. Self Assessment Questions

1. Write a procedure to sending e-mail to a specified address and a group of addresses.
2. Write a process to reading e-mail.
3. How you can reply of e-mail ?
4. Write a process to create a new e-mail address for a particular person.
5. What is the process to attach a file with e-mail ?
6. What is the procedure to search the string "Interval Graph"?
7. Write a note to search a phrase with different facilities.

72.17. Further Suggested Readings

1. The ABCs of the Internet, C. Crumlish, BPB Publications.
2. 10-Minute Guide to Microsoft Internet explorer 5, Freeze, PHI.
3. 10-Minute Guide to Internet, Kent, PHI.
4. 10-Minute Guide to the Internet and the World Wide Web, 2nd ed., Grimes, PHI.
5. Using the World Wide Web, 2nd ed., Wall.
6. Internet Communications Using SIP, H.Sinnceich and A.B.Johnston, Narosa.

"Learner's Feed-back"

After going through the Modules/ Units please answer the following questionnaire.
Cut the portion and send the same to the Directorate.

To
The Director
Directorate of Distance Education,
Vidyasagar University,
Midnapore- 721 102.

1. The modules are : (give ✓ in appropriate box)
   ☐ Easily understandable; ☐ very hard; ☐ partially understandable.
2. Write the number of the Modules/Units which are very difficult to understand :
   .....
   .....
   .....
3. Write the number of Modules/ Units which according to you should be re-written :
   .....
   .....
   .....
4. Which portion/page is not understandable to you? (Mention the page no. And Portion)
   .....
   .....
   .....
   .....
5. Write a short comment about the study material as a learner.
   .....
   .....
   .....
   .....
   .....

.....
(full Signature of the Leaner)

Enrolment No. .....

Date : .....

Phone/ Mobile No. .....



Published by : Directorate of Distance Education,
Vidyasagar University, Midnapore - 721 102, W.B.
Printed by : Trade Made, Howrah-711103 & ISO 9001 : 2008 Company