Laboratory Manual
2023-24

Sub-Computer Networks-II

SY B. Tech. (CSE)

Prepared by:

Prof. Renushe A.H

G. K. GUJAR CHARITABLE TRUST'S
Dr. Ashok Gujar Technical Institute's
Dr. Daulatrao Aher College of Engineering, Karad

Prepared by :Mrs. A. H. Renushe	Approved by: Head of Department

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**
Lab Manual
Subject : Computer Networks--II
TITLE : Index (List of Experiment)

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Computer Networks-II	
Expt. No: 01	**TITLE**: Program using UDP to connect to well known services (echo, time of the day service).

Aim: To implement Programs for UDP to connect to well known services (echo, time of the day service).

Theory:

The DAYTIME Service

The TCP/IP standards define an application protocol that allows a user to obtain the date and time of day in a format fit for human consumption. The service is officially named the DAYTIME service.

To access the DAYTIME service, the user invokes a client application. The client contacts a server to obtain the information, and then prints it. Although the standard does not specify the exact syntax, it suggests several possible formats. For example, DAYTIME could supply a date in the form:

Weekday, month day, year time-timezone like

Thursday, February 22, 1996 17:37:43-PST

The standard specifies that DAYTIME is available for both TCP and UDP. In both cases, it operates at protocol port 13.

The TCP version of DAYTIME uses the presence of a TCP connection to trigger output: as soon as a new connection arrives, the server forms a text string that contains the current date and time, sends the string, and then closes the connection. Thus, the client need not send any request at all. In fact, the standard specifies that the server must discard any data sent by the client.

The UDP version of DAYTIME requires the client to send a request. A request consists of an arbitrary UDP datagram. Whenever a server receives a datagram, it formats the current date and time, places the resulting string in an outgoing datagram, and sends it back to the client. Once it has sent a reply, the server discards the datagram that triggered the response.

The ECHO Service :

TCP/IP standards specify an ECHO service for both UDP and TCP protocols. At first glance, ECHO services seem almost useless because an ECHO server merely returns all the data it receives from a client. Despite their simplicity, ECHO services are important tools that network managers use to test reachability, debug protocol software, and identify routing problems.

The TCP ECHO service specifies that a server must accept incoming connection requests, read data from the connection, and write the data back over the connection until the client terminates the transfer. Meanwhile, the client sends input and then reads it back.

Daytime client Algorithm:

STEP 1: Start the program.
STEP 2: Declare the variables and structure.
STEP 3: Socket is created and connect function is executed.
STEP 4: If the connection is successful then server sends the message.
STEP 5: The date and time is printed at the client side.
STEP 6: Stop the program.

Daytime Server Algorithm:

STEP 1: Start the program.
STEP 2: Declare the variables and structure for the socket..

STEP 3: The socket is binded at the specified port.

STEP 4: Using the object the port and address are declared.
STEP 5: The listen and accept functions are executed.
STEP 6: If the binding is successful it waits for client request.
STEP 7: Execute the client program.

ECHO Client Algorithm:

STEP 1: Start
STEP 2: Declare the variables for the socket
STEP 3: Specify the family, protocol, IP address and port number
STEP 4: Create a socket using socket () function
STEP 5: Call the connect () function
STEP 6: Read the input message
STEP 7: Send the input message to the server
STEP 8: Display the server’s echo
STEP 9: Close the socket
STEP 10: Stop

ECHO Server Algorithm:

```
STEP 1: Start
STEP 2: Declare the variables for the socket
STEP 3: Specify the family, protocol, IP address and port number
STEP 4: Create a socket using socket () function
STEP 5: Bind the IP address and Port number
STEP 6: Listen and accept the client’s request for the connection
STEP 7: Read and Display the client’s message
STEP 8: Stop
```

Conclusion: Thus we have implemented Program for UDP to connect to well known services (echo, time of the day service).

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Computer Networks-II	
Expt. No: 02	**Title**: Implementation of concurrent TCP multiservice client/server Design

Aim: To implement concurrent TCP multiservice client/server Design.

Theory:

A connection oriented, Multiservice server Design

A connection –oriented, multiservice server can also follow an iterative algorithm.

In principle, such a server performs the same tasks as a set of iterative, connection-oriented servers. To be more precise, the single process in multiservice server replaces the master server processes in a set of connection –oriented servers. At the top level, the multiservice server uses asynchronous I/O to handle its duties

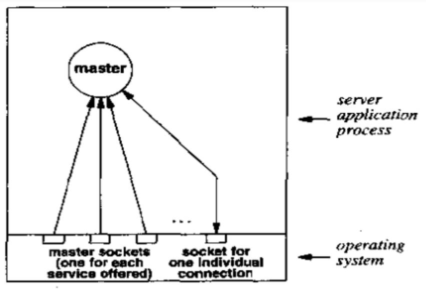

master
server application process
operating system
master sockets (one for each service offered)
socket for one individual connection

Figure 14.2 The process structure of an iterative, connection-oriented, multiservice server. At any time, the server has one socket open for each service and at most one additional socket open to handle a particular connection.

When it begins execution, the multiservice servers creates one socket for each service it offers, bind each socket to the well known port for the service ,and uses select to wait for an incoming connection request on any of them. When one of the sockets become ready, the server calls accept to obtain the new connection that has arrived .Accept creates a new socket for the incoming connection . the server uses yhe new socket to interact with a client, and then closes it. Thus, besides one master socket for each service ,the server has at most on additional socket open at any time.

A concurrent, Connection-oriented, multiservice server:

The procedure called by a multiservice server when a connection request arrives can accept and handle the new connection directly (making the server iterative), or it can create a slave process to handle it (making the server concurrent).

In an iterative implementation, once the procedure finishes communicating with the client, it closes the new connection. In the concurrent case, the master server process closes the connection as soon as it has created a slave process; the connection remains open in the slave process.

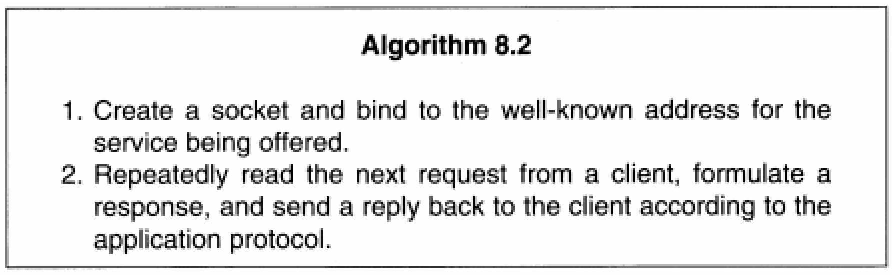
Conclusion: Thus we have studied and implemented the concurrent TCP multiservice client/server Design.

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Computer Networks-II	
Expt. No: 03	**Title**: Implementing Iterative UDP client/server

Aim: to implement Iterative UDP client/server

Theory:

Iterative servers work best for services that have a low request processing time. Because connection-oriented transport protocols like TCP have higher overhead than connectionless transport protocols like UDP, most iterative servers use connectionless transport. Creation of a socket for an iterative, connectionless server proceeds in the same way as for a connection-oriented server. The server's socket remains unconnected and can accept incoming datagram's from any client.



Algorithm 8.2

1. Create a socket and bind to the well-known address for the service being offered.
2. Repeatedly read the next request from a client, formulate a response, and send a reply back to the client according to the application protocol.

Algorithm 8.2 An iterative, connectionless server. A single process handles requests (datagrams) from clients one at a time.

Forming A Reply Address In A Connectionless Server:
The sendto socket call to specify both a datagram to be sent and an address to which it should go. Sendto has the form:
retcode = sendto(s , message, len, flags, toaddr, toaddrlen);

where s is an unconnected socket, message is the address of a buffer that contains the data to be sent, len specifies the number of bytes in the buffer, flags specifies debugging or control options, toaddr is a pointer to a sockaddr_in structure that contains the endpoint address to which the message should be sent, and toaddrlen is an integer that specifies the length of the address structure.

The call, recvfrom, takes two arguments that specify two buffers. The system places the arriving datagram in one buffer and the sender's address in the second buffer. A call to recvfrom has the form:

retcode = recvfrom(s, buf, len, flags, from, fromlen);

where argument s specifies a socket to use, buf specifies a buffer into which the system will place the next datagram, len specifies the space available in the buffer, from specifies a second buffer into which the system will place the source address, and fromlen specifies the address of an integer. Initially, fromlen specifies the length of the from buffer. When the call returns, fromlen will contain the length of the source address the system placed in the buffer. To generate a reply, the server uses the address that recvfrom stored in the from buffer when the request arrived.

Conclusions: Thus we have studied and implemented the Iterative UDP client/server.

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Computer Networks-II	
Expt. No: 04	**Title**: Studyof DNS Tools with all its options. nslookup, dig, host,whois

Aim: To study DNS Tools with all its options. nslookup, dig, host,whois

Theory:

DNS working:

Computer and other network devices communicate by IP address. It would be hard to remember the IP address of every website or resource you access, words are easier to remember. DNS will take the easy to remember name and map it to the IP address so devices can communicate.

Computer uses DNS to resolve names steps below:

1. User types in google.com into their browser. This will send a query to the DNS server to go fetch the IP address for google.com

2. The DNS server that the client uses may not know the IP address. This can be your local Active Directory DNS server or your ISP DNS server. If it doesn't know the IP address of the domain it will forward it on to the next DNS server.

3. The next DNS server says it knows the IP address and sends the request back to the computer.

4. The computer is then able to communicate to google.com.

The nslookup Command:

nslookup is a network administration command-line tool available in many computer operating systems for querying the Domain Name System (DNS) to obtain domain name or IP address mapping, or other DNS records. The name "nslookup" means "name server lookup."

The command does not use the operating system's local Domain Name System resolver library to perform its queries, and thus may behave differently from dig, which it does. Additionally, vendor-provided versions may include output of other sources of name information, such as host files and Network Information Service. Some behaviors of nslookup may be modified by the contents of resolv.conf.

Syntax:

```
nslookup [ OPTION ] [name | -] [server]
```

```
$ nslookup deltacollege.edu
```

The dig Command:

The dig command stands for Domain Information Groper. It is a network administration command-line tool for querying the Domain Name System (DNS). The dig command is useful for network troubleshooting and for educational purposes. It can operate based on command line option and flag arguments, or in batch mode by reading requests from an operating system file. When a specific name server is not specified in the command invocation, it uses the operating system's default resolver, usually configured in the file resolv.conf. Without any arguments it queries the DNS root zone.

```
Syntax: dig [ OPTIONS ]
```

The host Command:The host command in Linux system is used for DNS (Domain Name System) lookup. It is used to find the IP address of a particular domain name, or if you want to find out the domain name of a particular IP address the host command becomes handy. You can also find more specific details of a domain by specifying the corresponding option along with the domain name.

Syntax:

```
host [ OPTIONS ] hostname

$ host -v deltacollege.edu
```

WHOIS command:
is a TCP-based query and response protocol that is commonly used to provide information services to Internet users. It returns information about the registered Domain Names, an IP address block, Name Servers and a much wider range of information services.
In Linux, the **whois** command line utility is a **WHOIS** client for communicating with the WHOIS server (or database host) which listen to requests on the well-known port number **43**, which stores and delivers database content in a human-readable format.

```
$ whois 216.58.206.46
```

Conclusion: Thus we have studied different DNS tools.

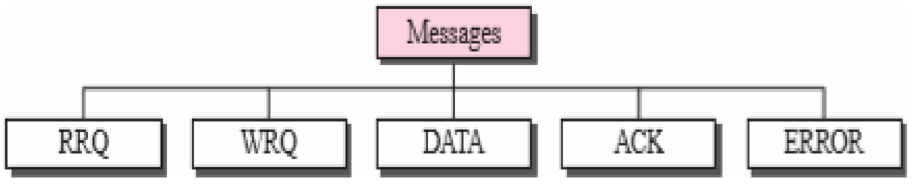
G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Computer Networks-II	
Expt. No: 05	**Title**: Study of trivial file transfer protocol (TFTP).

Aim: Study of trivial file transfer protocol (TFTP).

Theory:

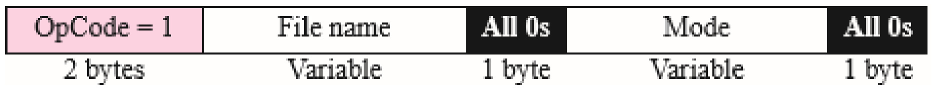
There are occasions when we need to simply copy a file without the need for all of the features of the FTP protocol. For example, when a diskless workstation or a router is booted, we need to download the bootstrap and configuration files. Here we do not need all of the sophistication provided in FTP. We just need a protocol that quickly copies the files. Trivial File Transfer Protocol (TFTP) is designed for these types of file transfer. It is so simple that the software package can fit into the read-only memory of a diskless workstation. It can be used at bootstrap time. The reason that it fits on ROM is that it requires only basic IP and UDP. However, there is no security for TFTP. TFTP can read or write a file for the client. Reading means copying a file from the server site to the client site. Writing means copying a file from the client site to the server site.

There are five types of TFTP messages, RRQ, WRQ, DATA, ACK, and ERROR.



RRQ :The read request (RRQ) message is used by the client to establish a connection for reading data from the server.

RRQ format:

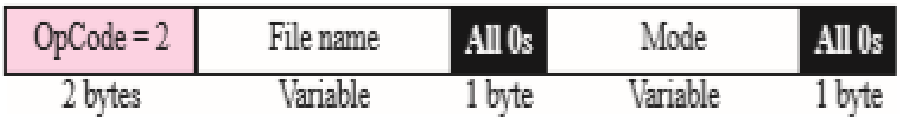


The RRQ message fields are as follows:

- OpCode. The first field is a 2-byte operation code. The value is 1 for the RRQ message.

- File name. The next field is a variable-size string (encoded in ASCII) that defines the name of the file. Since the file name varies in length, termination is signaled by a 1-byte field of 0s.
- Mode. The next field is another variable-size string defining the transfer mode. The mode field is terminated by another 1-byte field of 0s. The mode can be one of two strings: “netascii” (for an ASCII file) or “octet” (for a binary file). The file name and mode fields can be in upper- or lowercase, or a combination of both.

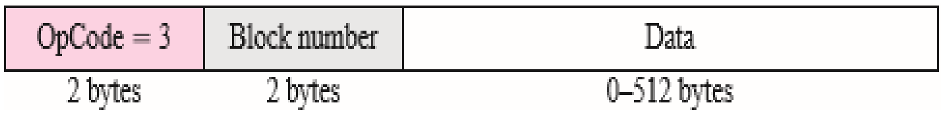
  WRQ: The write request (WRQ) message is used by the client to establish a connection for writing data to the server. The format is the same as RRQ except that the OpCode is 2.
  WRQ format:



DATA:

The data (DATA) message is used by the client or the server to send blocks of data.

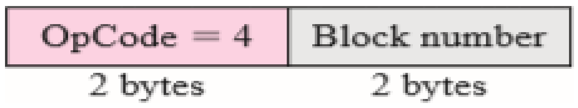
DATA format:



- OpCode: The first field is a 2-byte operation code. The value is 3 for the DATA message.
- Block number: This is a 2-byte field containing the block number. The sender of the data (client or server) uses this field for sequencing. All blocks are numbered sequentially starting with 1. The block number is necessary for acknowledgment as we will see shortly.
- Data: This block must be exactly 512 bytes in all DATA messages except the last block, which must be between 0 and 511 bytes. A non-512 byte block is used as a signal that the sender has sent all the data. In other words, it is used as an end-of-file indicator. If the data in the file happens to be an exact multiple of 512 bytes, the sender must send one extra block of zero bytes to show the end of transmission. Data can be transferred in either NVT ASCII (netascii) or binary octet (octet).

  ACK:
  The acknowledge (ACK) message is used by the client or server to acknowledge the receipt of a data block.

ACK format:



The ACK message fields are as follows:

- OpCode. The first field is a 2-byte operation code. The value is 4 for the ACK message.
- Block number. The next field is a 2-byte field containing the number of the block received.

Conclusion: thus we have studied the TFTP protocol.

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Computer Networks-II	
Expt. No: 06	**Title**: Configuration of basic service for FTP on Linux Platform

Aim: To Configure basic service FTP on Linux Platform

Theory:

File Transfer Protocol (FTP) is an application layer protocol that moves files between local and remote file systems. It runs on the top of TCP, like HTTP. To transfer a file, 2 TCP connections are used by FTP in parallel: control connection and data connection. Port number for data connection is 20 and for control connection 21.

FTP stands for File Transfer Protocol. It is similar to HTTP (HyperText Transfer Protocol), in that it specifies a language for transferring data over a network.

Step 1: Update System Packages

Start by updating your repositories – enter the following in a terminal window:

```
$ sudo apt update
```

Step 2: Install vsftpd Server on Ubuntu

A common open-source FTP utility used in Ubuntu is vsftpd

1.To install **vsftpd**, enter the command

```
$sudo apt install vsftpd
```

2. To launch the service and enable it at startup, run the commands:

```
sudo systemctl start vsftpd
```

```
sudo systemctl enable vsftpd
```

Step 3: Backup Configuration Files

Before making any changes, make sure to back up your configuration files

```
sudo cp /etc/vsftpd.conf  /etc/vsftpd.conf_default
```

Step 4: Create FTP User

Create a new FTP user with the following commands:

```
$sudo useradd -m testuser
$sudo passwd testuser
```

Step 5: Configure Firewall to Allow FTP Traffic

```
sudo ufw allow 20/tcp
sudo ufw allow 21/tcp
```

Step 6: Connect to Ubuntu FTP Server

Connect to the FTP server with the following command:
Replace ubuntu-ftp with the name of your system (taken from the command line).

Log in using the **testuser** account and password you just set. You should now be successfully logged in to your FTP server.

Configuring and Securing Ubuntu vsftpd Server:

Change Default Directory:

By default, the FTP server uses the /srv/ftp directory as the default directory. You can change this by creating a new directory and changing the FTP user home directory.

To change the FTP home directory, enter the following:

```
sudo mkdir /srv/ftp/new_location
sudo usermod -d /srv/ftp/new_location ftp
```

Restart the vsftpd service to apply the changes:

```
sudo systemctl restart vsftpd.service
```

Authenticate FTP Users:

If you want to let authenticated users upload files, edit the vsftpd.conf file by entering the following:

```
sudo nano /etc/vsftpd.conf
```

Find the entry labeled write_enable=NO, and change the value to “YES.”
Save the file, exit, then restart the FTP service with the following:

```
sudo systemctl restart vsftpd.service
```

Conclusion: Thus we have configured FTP server on Ubuntu.

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Computer Networks-II	
Expt. No: 07	Title: Configuration of basic service HTTP (Apache2) on Linux Platform

Aim: To configure HTTP (Apache2) on Linux Platform

Theory:

The Hypertext Transfer Protocol (HTTP) is a protocol used mainly to access data on the World Wide Web. HTTP functions like a combination of FTP (Chapter 21) and SMTP (Chapter 23). It is similar to FTP because it transfers files and uses the services of TCP. However, it is much simpler than FTP because it uses only one TCP connection. There is no separate control connection; only data are transferred between the client and the server. HTTP is like SMTP because the data transferred between the client and the server look like SMTP messages. In addition, the format of the messages is controlled by MIME-like headers. Unlike SMTP, the HTTP messages are not destined to be read by humans; they are read and interpreted by the HTTP server and HTTP client (browser). SMTP messages are stored and forwarded, but HTTP messages are delivered immediately. The commands from the client to the server are embedded in a request message. The contents of the requested file or other information are embedded in a response message. HTTP uses the services of TCP on well-known port 80.

Installation and configuration of Apache Web Server on Ubuntu 20.04

The Apache HTTP server is the most widely-used web server in the world. It provides many powerful features including dynamically loadable modules, robust media support, and extensive integration with other popular software.

Step 1 — Installing Apache

-Updating local packages
$ sudo apt update
-Take a super user control by typing in a terminal
$ sudo-s
-Install the apache2 package
$ sudo apt install apache2
-Start Apache2 server
$service apache2 start
Go to browser in URL type http:loacalhost
-Stop Apache2 server

```
$ Service Apache2 stop
```

Alternative command to start apache service is:

```
Apachectl –k start
```

-Go to terminal and type

```
$ cp /var /www/html/index.html  /var/www/html/second.html
$ gedit/var/www/html/index.html
```

Text editor will open

-Copy below html code in index.html file and save

```
<html>
<head>
<title> my first website</title>
</head>
</body>
<h1> my first demo website hosted on Apache2 web server</h1>
</body>
</html>
```

- Go to browser and type
  http://Localhost
  your own web site will displayed in browser

Conclusion: Thus we have configured and installed HTTP (Apache2) on Linux Platform.

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Computer Networks-II	
Expt. No: 08	**Title**: Configuration of basic service Telnet on Linux Platform

Aim: To install and configure basic service Telnet on Linux Platform

Theory:

TELNET is an abbreviation for TErminaL NETwork. It is the standard TCP/IP protocol for virtual terminal service as proposed by ISO. TELNET enables the establishment of a connection to a remote system in such a way that the local terminal appears to be a terminal at the remote system.

Local Login: When a user logs into a local time-sharing system, it is called local login. As a user types at a terminal or at a workstation running a terminal emulator, the keystrokes are accepted by the terminal driver. The terminal driver passes the characters to the operating system. The operating system, in turn, interprets the combination of characters and invokes the desired application program.

Remote Login: When a user wants to access an application program or utility located on a remote machine, he or she performs remote login. Here the TELNET client and server programs come into use. The user sends the keystrokes to the terminal driver where the local operating system accepts the characters but does not interpret them. The characters are sent to the TELNET client, which transforms the characters to a universal character set called Network Virtual Terminal (NVT) characters and delivers them to the local TCP/IP stack.

The commands or text, in NVT form, travel through the Internet and arrive at the TCP/IP stack at the remote machine. Here the characters are delivered to the operating system and passed to the TELNET server, which changes the characters to the corresponding characters understandable by the remote computer. However, the characters cannot be passed directly to the operating system because the remote operating system is not designed to receive characters from a TELNET server: It is designed to receive characters from a terminal driver. The solution is to add a piece of software called a pseudoterminal driver, which pretends that the characters are coming from a terminal. The operating system then passes the characters to the appropriate application program.

Telnet installation and Configuration

```
$ sudo apt install telnetd xinetd
```

After running the above installation command, Terminal might prompt you with the **y/n** option. To continue the procedure, hit **y** and then hit **Enter.** It will then install the Telnet server and xinetd on your Ubuntu system.

After installation, the xinetd service starts automatically. You can view the status of the service as follows:

```
$ sudo systemctl status xinetd.service
```

If the service does not start automatically, you can manually start it by running the command below:

```
$ sudo systemctl start xinetd.service
```

Now create the **/etc/xinetd.d/telnet** file using the command below:

```
$ sudo nano /etc/xinetd.d/telnet
```

Add the below lines the file:

```
service telnet

{

disable = no

flags = REUSE

socket_type = stream

wait = no
user = root

server = /usr/sbin/in.telnetd

log_on_failure += USERID

}
```

Then save and close the file and restart xinetd.service as follows:

```
$ sudo systemctl restart xinetd.service
```

Telnet server uses port 23 for listening to the incoming connections. Therefore, you will need to open this port in your firewall. Run the command below to do so:

```
$ sudo ufw allow 23
```

To allow incoming connections to port 23 from only a specific IP or subnet, run the below command instead:

```
$ sudo ufw allow from <ip or subnet> to any port 23
```

-Installing Telnet Client on Ubuntu:

```
$ sudo apt install telnet
```

-Connect to Telnet Server from Remote System

```
$ telnet <server-ip>
```

To uninstall Telnet server, run this command:

```
$ sudo apt remove telnetd xinetd
```

To uninstall Telnet client, run this command:

```
$ sudo apt remove telnet
```

Conclusion: Thus we have configured and installed Telnet server on Linux Platform.

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Computer Networks-II	
Expt. No: 09	**Title**: Study of Simple Mail Transfer Protocol.

Aim: To study of Simple Mail Transfer Protocol

Theory:

SMTP (MESSAGE TRANSFER AGENT):

The actual mail transfer is done through message transfer agents (MTAs). To send mail, a system must have the client MTA, and to receive mail, a system must have a server MTA. The formal protocol that defines the MTA client and server in the Internet is called Simple Mail Transfer Protocol (SMTP).

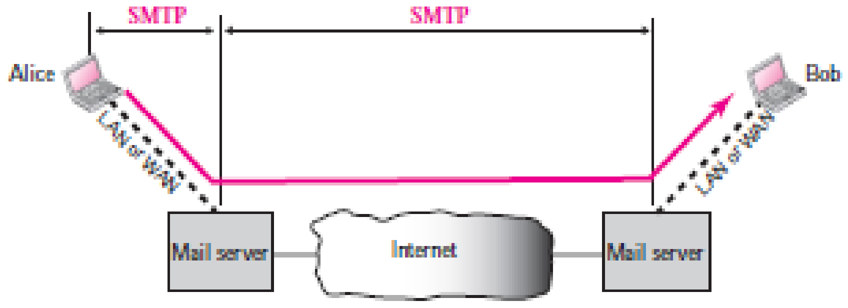


Fig: SMTP range

SMTP is used two times, between the sender and the sender's mail server and between the two mail servers. As we will see shortly, another protocol is needed between the mail server and the receiver. SMTP simply defines how commands and responses must be sent back and forth.

Commands and Responses

SMTP uses commands and responses to transfer messages between an MTA client and an MTA server.

MTA Client
Commands
Responses
MTA Server

Fig: Commands and responses

Commands:
Commands are sent from the client to the server. The format of a command is shown below: It consists of a keyword followed by zero or more arguments.

- HELO: This command is used by the client to identify itself. The argument is the domain name of the client host. The format is
  HELO: challenger.atc.fhda.edu

- MAIL FROM. This command is used by the client to identify the sender of the message. The argument is the e-mail address of the sender (local part plus the domain name). The format is
  MAIL FROM: forouzan@challenger.atc.fhda.edu

- RCPT TO. This command is used by the client to identify the intended recipient of the message. The argument is the e-mail address of the recipient. If there are multiple recipients, the command is repeated. The format is
  RCPT TO: betsy@mcgraw-hill.com

- DATA. This command is used to send the actual message. All lines that follow the DATA command are treated as the mail message. The message is terminated by a line containing just one period.
  The format is
  DATA
  This is the message to be sent to the McGraw-Hill Company.
- QUIT. This command terminates the message. The format is
  QUIT

- RSET. This command aborts the current mail transaction. The stored information about the sender and recipient is deleted. The connection will be reset.
  RSET

- VRFY. This command is used to verify the address of the recipient, which is sent as the argument. The sender can ask the receiver to confirm that a name identifies a valid recipient. Its format is
  VRFY: betsy@mcgraw-hill.com

- NOOP. This command is used by the client to check the status of the recipient. It requires an answer from the recipient. Its format is
  NOOP

- TURN. This command lets the sender and the recipient switch positions, whereby the sender becomes the recipient and vice versa. However, most SMTP implementations today do not support this feature. The format is
  TURN

- EXPN. This command asks the receiving host to expand the mailing list sent as the arguments and to return the mailbox addresses of the recipients that comprise the list. The format is
  EXPN: x y z

- HELP. This command asks the recipient to send information about the command sent as the argument. The format is
  HELP: mail

- SEND FROM. This command specifies that the mail is to be delivered to the terminal of the recipient, and not the mailbox. If the recipient is not logged in, the mail is bounced back. The argument is the address of the sender. The format is
  SEND FROM: forouzan@fhda.atc.edu

- SMOL FROM. This command specifies that the mail is to be delivered to the terminal or the mailbox of the recipient. This means that if the recipient is logged in, the mail is delivered only to the terminal. If the recipient is not logged in, the mail is delivered to the mailbox. The argument is the address of the sender. The format is
  SMOL FROM: forouzan@fhda.atc.edu

- SMAL FROM. This command specifies that the mail is to be delivered to the terminal and the mailbox of the recipient. This means that if the recipient is logged in, the mail is delivered to the terminal and the mailbox. If the recipient is not logged in, the mail is delivered only to the mailbox. The argument is the address of the sender. The format is

  SMAL FROM: forouzan@fhda.atc.edu

Conclusion: Thus we have studied commands of Simple mail Transfer Protocol.

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Computer Networks-II	
Expt. No: 10	**Title**: Capturing & Analyzing operation of various application layer protocols using network protocol analyzer. (Wireshark and tcpdump).

Aim: To Capture & Analyze operation of various application layer protocols using network protocol analyzer. (Wireshark and tcpdump).

Theory: Wireshark is an open-source network protocol analyzer that helps us to see what is happening inside a network when we try to communicate with other networks. Currently, Wireshark is the most famous application to analyze networks. Wireshark is a software tool used to monitor the network traffic through a network interface.

Purposes:
• Network administrators use it to troubleshoot network problems
• Network security engineers use it to examine security problems
• QA engineers use it to verify network applications
• Developers use it to debug protocol implementations
• People use it to learn network protocol internals

Configuring and Installing Wireshark on Ubuntu 20.04

Step 1: Update APT

```
$ sudo apt update
```

```
$ sudo apt upgrade
```

Step 2: Download and Install Wireshark

```
$ sudo apt install wireshark
```

Step 3: Enable Root Privileges

When Wireshark installs on your system, you will be prompted by the following window. As Wireshark requires superuser/root privileges to operate, this option asks to enable or disable permissions for all every user on the system. Press the "Yes" button to allow other users, or press the "No" button to restrict other users from using Wireshark.

Step 4: Launch Wireshark

In the terminal window, type the following command to start the Wireshark application.

$ wireshark

After executing this command in terminal following window will open

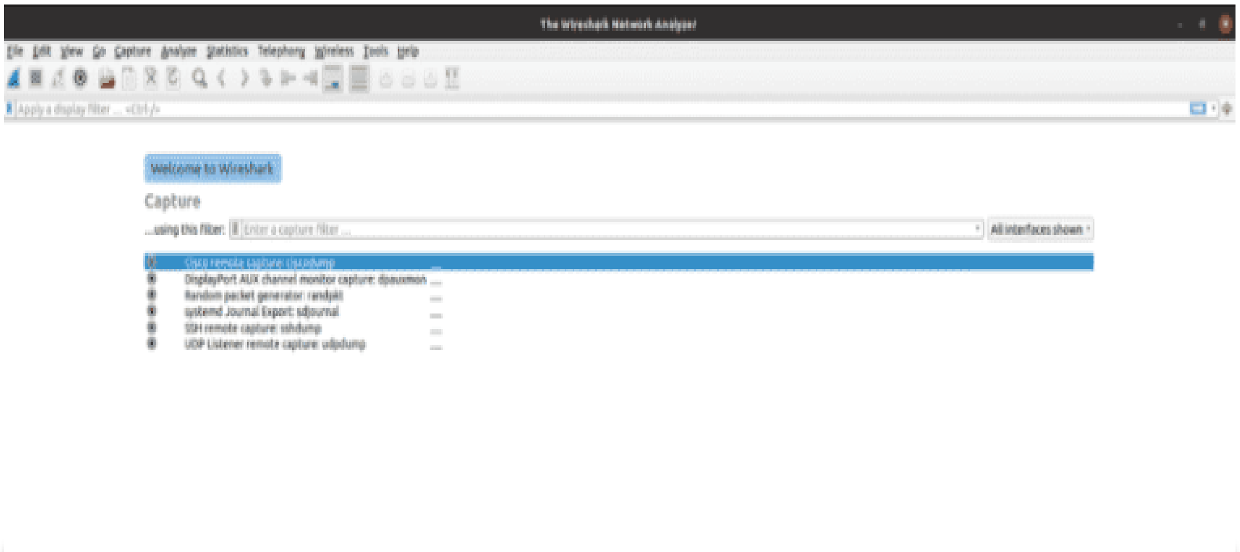

The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Welcome to Wireshark
Capture
...using this filter:
Enter a capture filter ...
All interfaces shown
Cisco remote capture: ciscodump
DisplayPort AUX channel monitor capture: dpauxmon
Random packet generator: randpkt
systemd Journal Export: sdjournal
SSH remote capture: sshdump
UDP Listener remote capture: udpdump

Figure: Wireshark Welcome screen.

Installing Wireshark:

First update the APT package repository cache with following command:

$ sudo apt update

$ sudo apt install wireshark

Press y on keyboard

$ sudo usermod –aG wireshark $(whoami)

$ sudo reboot

Starting wireshark:

$ wireshark

Then wireshark window will start

Capturing packets using wireshark:

When you start wireshark you will see a list of interfaces that you can capture packets to and from network.

Configuring and Installing TCPdump on Ubuntu 20.04

TCPdump is a free and open–source packet analyzer tool and command-line utility.It is used capturing the packets and inspect the network traffic going to and from our system. It is basic used for troubleshooting network issue and security testing. We can capture Non-TCP traffic such asUDP,ARP or ICMP.

Step1: Update the system

Apt-get update

Step2:Install TCPdump on system

Apt-get install tcpdump

Check the TCPdump version

Tcpdump --version

Step3: TCPdump syntax and Examples

Tcpdump [options][expression]

To capture all packets:

$tcpdump

To capture packets from any interface:

tcpdump -i any

Conclusion: Thus we have studied Capturing & Analyzing operation of various application layer protocols using network protocol analyzer (wireshark and TCPdump)

G. K. Gujar Memorial Charitable Trust‟s,
Dr. Ashok Gujar Technical Institute‟s
Dr. Daulatrao Aher College of Engineering, Karad.

Department of Computer Science & Engineering

Object Oriented Programming

Lab Manual

Academic year (2023-2024) IV SEMESTER

Procedure for performing experiments

a) Explanation by the faculty using OHP/PPT covering the following aspects:	20 min.

1) Aim of the experiment
2) Software/Inputs required
3) Algorithm
4) Test Data

b) Writing of source program by the students	20 min.
c) Compiling and execution of the program	60 min.
d) Assessment	20 min

Format of First Page

Name	:
Student Roll No #	:
Experiment #	:
Experiment Title	:
Date of Experiment	:
Date of Submission	:

Student Responsibilities

1. In the very beginning of the laboratory work, the student has to do programming individually. For this reason, regular attendance is strictly required.
2. Every laboratory session is divided into two parts. In the first part, the instructor will be lecturing on the test objective, procedure and data collection. In the second part, the students, organized in groups (individual student), are required to do the programming. In order to perform the experiment within the assigned period, and to gain the maximum benefit from the experiment, the students must familiarize themselves with the purpose, objective, and procedure of the experiment before coming to the laboratory. Relevant lecture notes and laboratory manual should be studied carefully and thoroughly.
3. At the end of the experiment, every student should submit the written algorithm & programs & testes result for approval by the instructor.
4. It should be understood that laboratory facilities and equipment‟s (Hardware & Software) are provided to enhance the learning process.
5. The equipment‟s must be properly cared after every laboratory session. Also, students should always take precautions to avoid any possible hazards. Students must follow laboratory regulations provided at the end of this section.

	G. K. Gujar Memorial Charitable Trust"s, Dr. Ashok Gujar Technical Institute"s Dr. Daulatrao Aher College of Engineering, Karad.
	Department of Computer Science & Engineering

EXPERIMENT LIST

Department: Computer Science and Engineering **Academic Year:** 2023-24

Class: Second Year (SY BTech CSE) **Semester:** IV

Subject: Object Oriented Programming **Semester Duration:** 6 months

Expt. No.	TITLE OF THE EXPERIMENT
1	Study and comparison of Procedure Oriented Programming & Object Oriented Programming features with an example.
2	Introduction of Object Oriented Programming and Implementation of sample C++ program.
3	Implementation of Constructor & Destructor.
4	Implementation of Function Overloading and Constructor Overloading
5	Implementation of Operator Overloading.
6	Implementation of Multilevel Inheritance.
7	Implementation of Multiple Inheritance.
8	Implementation of Hierarchical Inheritance.
9	Implementation of Friend Function and Friend Class.
	Review - I
10	Implementation of Virtual Function and Virtual Class.
11	Implementation of Student database using concept of File Handling.
12	Implementation of Conversion of Polish Expressions.
13	Implementation of Searching Algorithms using Function Template and Virtual Function.

INRODUCTION TO C++ PROGRAMMING

Title: Introduction to Object Oriented Programming language.

Objective:

Learn the fundamentals of C++ programs

Prerequisites: Procedure Oriented Language i.e Fundamentals of C Language

Theory:

While writing the C++ program, there is need to follow the rules which are specified as follows:

1) Use of modern way of writing C++ programs
2) Rules for declaring the local variables.
3) Following the rule No Default to int
4) Use of new C++ headers
5) Use of the namespace std and working with an old and new compilers

The old-style c++ program has the following structure:

```
#include<iostream.h>
int main( )
{
//statements
return (0);
}
```

The new-style c++ program has the following structure:

```
#include<iostream>
Using namespace std;
int main( )
{
//statements
return (0);
```

}
The key differences between old-style and modern code has two features: 1) new-style headers and the namespace statement. A simple C++ program to display the addition of two numbers is

```
#include  <iostream>
using namespace std;
int main()
{
int n1, n2,ans;
cout<<”enter any two numbers”;
cin>>n1>>n2;
ans = n1 + n2;
        cout<<”the addition = “<<ans; return(0); }
```

Program Features: The C++ program is a collection of functions as like that of C program. The above example contains the main function only. The execution is initiated with the main (), every C++ program must have function main ().

1) **Comments:** C++ programming language contains two types of comments: 1) single line comment (example // program title – addition of two numbers) and 2) multiline comments, written as

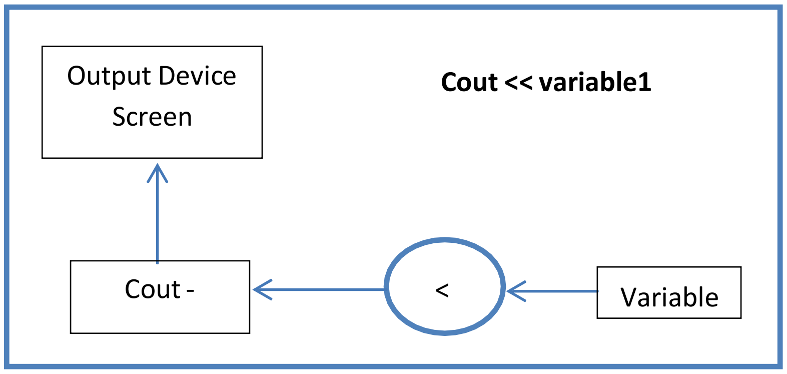
```
/*
                    --------------
                    --------------
*/
```

2)The iostream file- #include <iostream>, this directives causes the preprocessors to add the contents of the iostream file to the program. It contains declarations for the identifier cout, cin and the operator output operator “<<”, input operator “>>”. This file should be included at beginning of all programs that uses the input/output statements.

3) Namespaces – namesspaces are the declarative region, used to localize the names of the identifiers to avoid the name collisions. Elements declared in one namespace are different from the elements which are declared in other namespace. To include the name in C++ program, there is need to use statement “using namespace <namespace_name>”. in program std namespace is used to include the names of the C++ library functions to define them as global for the C++ programs.

4) Input / Output statements:

The output statement is used as cout with operator “<<”. The identifier cout is a predefined object that represents the standard output stream in C++. The standard output stream is referred to the screen/o/p device. The operator “<<” is called the ***insertion or put to operator or output operator.***It inserts or sends the contents of the variable on its right to the left side of it to the object. The operator “<<” is also called as left-shift operator, used to shift the data from the variable to device.



Theinput statement is used as cin>>n1, this causes the program to wait for the user to give the input which will be stored into variable n1. The identifier cin is predefined object in C++ that corresponds to the standard input stream. The input stream is referred to the keyboard. The operator “>>” is called as extraction/get from /input operator that extracts data from keyboard and assigns itto the variable present at the right side of it. The both input “>>” and output “<<” operators can be overloaded.

5) Declaring local variables – in C89 standard, the variables must be declared at the beginning of the program whereas the C99 standard allows declaration of variables anywhere in a program when even there is need to use them in application logic. The encapsulation of code and data make it sense to

declare variables close to the position from where they are used instead of declaring them at the beginning of the program. Therefore, it is possible to avoid the accidental use of variables and this approach is used in program which contains larger functions but it is harder to maintain the programs.

6) No Default to int –C89 and the original specifications of C++ language states that, when no explicit data type is specified in the declaration, then type “int” is assumed. This rule is called as “default – to – int” this default-to-int rule is dropped from C99 standard, and there is need to specify the type. This rule is applicable mostly to the function declaration, therefore in C99 standard while declaring the function there need to specify the return datatype even though function returns value of “int” type. As a practical matter, nearly all C and C++ compilers provides support to this rule.

Functions in C++

1) The Main Function
2) Function Prototyping
3) Approach to call function: call by value, call by reference
4) Default arguments
5) const arguments
6) Function overloading
7) Use of other library functions
8) Nesting of member functions

Problem Statements: Implement the following C++ programs.

1) Write a program to display the average of three numbers.
2) Write a C++ program to generate the mathematical calculator.
3) Write a C++ program to display the count of odd, even and divisible by 5 numbers with in the specified range from 1 to 100.
4) Write a C++ program to find out even and odd number.

Theory Questions

1) What is meant by Object Oriented Programming?
2) Explain the difference between C & C++ & Old style Vs modern Style C++
3) What is the use of Namespaces concept in C++.

EXPERIMENT NO: 1

Title: - Study and comparison of procedure oriented programming & Object oriented programming features with an example.

Aim: -To study comparison of procedure oriented programming & Object oriented programming features.

Theory:

Conventional programming using languages such as COBOL FORTRAN and C is commonly known as procedure oriented programming. In procedure oriented approach the problem is viewed as sequence of things to be done such as reading calculating and printing. A number of functions are written to accomplish these tasks. The primary focus is on functions.

Procedure oriented programming basically consist of writing a list of instructions for the computer to follows, and organizing these instructions in to group known as functions. We normally use flowcharts to organize these actions and represent the flow of control from one action to another.

While we concentrate on the development of functions very little attention is given to the data that are being used by various functions. In a multi function program many important data item are placed as global so that they may be accessed by all the functions. Each function may have its own local data.

Global data are more vulnerable to an inadvertent change by functions in a large programming it is too difficult to identify what data is used by which functions. In case we need to revise an external data structure, we should also revise all function that accesses the data. This provides an opportunity for bugs to creep in.

Another serious drawback with the procedural approach is that it does not model real world problems very well. His is because functions are action oriented and do not really corresponding to the element of the problem.

The striking features of Procedure-Oriented Programming are:

- Emphasis is on doing things(Algorithm)
- Large programs are divided in to smaller programs known as functions.
- Most of the functions share global data.
- Data move openly around the system from function to function.
- Function transforms data from one form to another
- Employs top-down approach in program design

Object Oriented Programming treats data as a critical element in the program development and does not allow it too freely around the system. It ties data more closely to the functions that operate on it and protects it from accidental modification from outside functions. Object Oriented Programming allows decomposition of a problem into a number of entities called „Objects‟ and builds data and function around these objects.

The striking features of Object-Oriented Programming are:

- Emphasis is on data rather than procedure
- Programs are divided into objects.
- Data structures are designed such that they characterize the objects.
- Functions that operate on the data of an object are tied together in the data structure.
- Data is hidden and cannot be accessed by external functions.
- Objects may communicate with each other through functions.
- New data and functions can be easily added whenever necessary.
- Follows bottom up approach in program design.

Basic Concepts in Object Oriented Programming:

- Objects
- Classes
- Data Abstraction and Encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding
- Message passing

Benefits of Object Oriented Programming:

- Through inheritance, we can eliminate redundant code and extend the use of existing classes.
- The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.

- It is possible to have multiple instances of an object to co-exist without any interference.
- It is possible to map objects in the problem domain to those objects in the program.
- It is easy to partition the work in a project based on objects.
- The data centered design approach enables us to capture more details of a model in implementable form.
- Object-oriented systems can be easily upgraded from small to large systems.
- Message passing techniques for communication between objects makes the interface descriptions with external systems much simpler.
- Software complexity can be easily managed.

C++ is an object oriented language. Three most important facilities that C++ adds on to C are classes, function overloading, and operator overloading. These features enable us to create abstract data type, inherit properties from existing data types & support polymorphism, thus making C++ a truly object oriented language. C++ is a versatile language for handling very large programs. It is suitable for virtually any programming task including development of editors, compilers, databases, communication systems and any complex real life application systems.

Output Operator:

Output statement introduces two new C++ features, cout and <<. The identifier **cout** is a predefined object that represents the Standard Output Stream(i.e. Screen) in C++. The operator **<<** is called the **insertion** or **put to** operator. It inserts the contents of the variable on its right to the objects on its left.

e.g. cout<<" Two numbers are"<<a<<b;

Input Operator:

The input statement causes the program to wait for the user to type in a number. The number keyed in is placed in the variable. The identifier cin is predefined object in C++ corresponds to Standard Input Stream(i.e Keyboard). The operator >> is known as extraction or get from operator. It extracts the value from keyboard and assigns it to variable at right.

e.g cin>>number;

The iostream File:

#include<iostream>

This directive causes the preprocessor to add contents of the iostream file to the program. It contains declarations for the identifier cout and operator <<.

PROGRAM -

//program to implement arithmetic operation using c language.

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

```
#include<conio.h>
void main()
{       int a,b,c,opt;
        clrscr();
        printf("\n Enter Two Number for arithmetic operation\t\n");
        scanf("%d%d",&a,&b);
        while(1)
        {       clrscr();
                printf("\n\t Number1 : %d",a);
                printf("\n\t Number2: %d",b);
                printf("\n\t 1.Addition \n\t 2.Substraction \n\t 3.Multiplication ");
                printf("\n\t 4.Division \n\t 5.exit \nOption: ");
                scanf("%d",&opt);
                if(opt>=5)
                        break;
                switch(opt)
                {               case 1:
                                c=a+b;
                                printf("\n\t Addition is : %d",c);
                                break;
                        case 2:
                                c=a-b;
                                printf("\n\subtraction is : %d",c);
                                break;
                        case 3:
                                c=a*b;
```

```
                    printf("\n\t Multiplication is : %d",c);

                    break;

                case 4:

                    c=a/b;

                    printf("\n\t Division is : %d",c);

                    break;

        }getch();

    }}
```

OUTPUT

```
    Enter Two Number for arithmetic operation

    20

    10

    Number1 :20

    Number2 :10

    1.Addition

    2.Substraction

    3.Multiplication

    4.Division

    5.exit

    option:1

        Addition is: 30

    1.Addition

    2.Substraction

    3.Multiplication

    4.Division

    5.exit
```

```
option:2
        subtraction is : 10

1. Addition
2.Substraction
3.Multiplication
4.Division
5.exit
option:3
        Multiplication is :200
1.Addition
2. Substraction
3.Multiplication
4.Division
5.exit
option:4
        Division is :2
1.Addition
2.Substraction
3.Multiplication
4.Division
5.exit
option:5
```

//program to implement arithmetic operation using c++ language.

```
#include<iostream.h>
#include<conio.h>
```

```
void main()
{       int a,b,c,opt;
        clrscr();
        cout<<"\n Enter Two Number for arithmetic operation\t"<<endl;
        cin>>a>>b;
        cout<<endl;
        while(1)
        {
                clrscr();
                cout<<endl;
                cout<<" Number 1:"<<a<<endl;
                cout<<" Number 2:"<<b<<endl;
                cout<<"\t 1.Addition \n\t 2.Substraction \n\t 3.Multiplication ";
                cout<<"\n\t 4.Division \n\t 5.exit \nOption: ";
                cin>>opt;
                cout<<endl;
                if(opt>=5)
                break;
                switch(opt)
                {               case 1:
                                c=a+b;
                                cout<<"Addition is : "<<c<<endl;
                                break;
                        case 2:
                                c=a-b;
                                cout<<"Subtraction is : "<<c<<endl;
```

```
                    break;

                case 3:

                    c=a*b;

                    cout<<"Multiplication is : "<<c<<endl;

                    break;

                case 4:

                    c=a/b;

                    cout<<"division is : "<<c<<endl;

                    break;

            }       getch();

        }}
```

OUTPUT:-

```
Enter Two Number for arithmetic operation

8

4

Number1 :8

Number2 :4

1.Addition

2.Substraction

3.Multiplication

4.Division

5.exit

option:1

Addition is: 12

1.Addition

2.Substraction
```

```
3.Multiplication

4.Division

5.exit

option:2

        Subtraction is : 4

1.Addition

2.Substraction

3.Multiplication

4.Division

5.exit

option:3

        Multiplication is :32

1.Addition

2.Substraction

3.Multiplication

4.Division

5.exit

option:5
```

Conclusion: Thus, we have studied comparison of procedure oriented programming & Object oriented programming features with an example.

Experiment No 2

Title: - Introduction of object oriented programming and structure of C++ program

Aim: -To study Basic concepts of OOPS and Structure of C++ program

Theory:

In this experiment we will learn about Objects, Classes, Inheritance, Data Abstraction, Data Encapsulation, Polymorphism, Overloading and Reusability. Before starting to learn C++ it is essential that one must have a basic knowledge of the concepts of Object oriented programming. Some of the important object oriented features are namely:

- Objects
- Classes
- Inheritance
- Data Abstraction
- Data Encapsulation
- Polymorphism
- Overloading
- Reusability

In order to understand the basic concepts in C++, the programmer must have a command of the basic terminology in object-oriented programming. Below is a brief outline of the concepts of Object-oriented programming languages:

Objects:

Object is the basic unit of object-oriented programming. Objects are identified by its unique name. An object represents a particular instance of a class. There can be more than one instance of an object. Each instance of an object can hold its own relevant data. An Object is a collection of data members and associated member functions also known as methods.

Classes:

Classes are data types based on which objects are created. Objects with similar properties and methods are grouped together to form a Class. Thus a Class represents a set of individual objects. Characteristics of an object are represented in a class as Properties. The actions that can be performed by objects become functions of the class and are referred to as Methods.

For example consider we have a Class of Cars under which Santro Xing, Alto and WaganR represents individual Objects. In this context each Car Object will have its own, Model, Year of Manufacture, Color, Top Speed, Engine Power etc., which form Properties of the Car class and the associated actions i.e., object functions like Start, Move, and Stop form the Methods of Car Class. No memory is allocated when a class is created. Memory is allocated only when an object is created, i.e., when an instance of a class is created.

Inheritance:

Inheritance is the process of forming a new class from an existing class or base class. The base class is also known as parent class or super class, the new class that is formed is called derived class. Derived class is also known as a child class or sub class. Inheritance helps in reducing the overall code size of the program, which is an important concept in object-oriented programming.

Data Abstraction:

Data Abstraction increases the power of programming language by creating user defined data types. Data Abstraction also represents the needed information in the program without presenting the details.

Data Encapsulation:

Data Encapsulation combines data and functions into a single unit called *class*. When using Data Encapsulation, data is not accessed directly; it is only accessible through the functions present inside the class. Data Encapsulation enables the important concept of data hiding possible.

Polymorphism:

Polymorphism allows routines to use variables of different types at different times. An operator or function can be given different meanings or functions. Polymorphism refers to a single function or multi-functioning operator performing in different ways.

Overloading:

Overloading is one type of Polymorphism. It allows an object to have different meanings, depending on its context. When an existing operator or function begins to operate on new data type, or class, it is understood to be overloaded.

Reusability:

This term refers to the ability for multiple programmers to use the same written and debugged existing class of data. This is a time saving device and adds code efficiency to the language. Additionally, the programmer can incorporate new features to the existing class, further developing the application and allowing users to achieve increased performance. This time saving feature optimizes code, helps in gaining secured applications and facilitates easier maintenance on the application.

A sample program to understand the basic structure of C++

Include Header Files
Class Declaration
Member Function Definition
Main Function Program

//Program to read employee details and to output the data

```
 #include <iostream>              //Preprocessor directive

using namespace std;

class employee                    // Class Declaration

{

   private:

     char empname[50];

     int empno;

    public:

     void getvalue()

     {

        cout<<"INPUT Employee Name:";

        cin>>empname;

        cout<<"INPUT Employee Number:";

        cin>>empno;

     }

      void displayvalue()

     {

        cout<<"Employee Name:"<<empname<<endl;

        cout<<"Employee Number:"<<empno<<endl;

     }
};

main()

{     employee e1;                    // Creation of Object

    e1.getvalue();

    e1.displayvalue();

}
```

Overview of the Basic Structure of C++ Programming

- The // in first line is used for representing comment in the program.
- The second line of the program has a # symbol which represents the preprocessor directive followed by header file to be included placed between < >.
- The next structure present in the program is the class definition. This starts with the keyword class followed by class name employee. Within the class are data and functions. The data defined in the class are generally private and functions are public. These explanations we will be detailed in later sections. The class declaration ends with a semicolon.
- main() function is present in all C++ programs.
- An object e1 is created in employee class. Using this e1 the functions present in the employee class are accessed and there by data are accessed.
- The input namely ename and eno is using the input statement namely cin and the values are outputted using the output statement namely cout.

PROGRAM –

Class, Object, Control Structures:

```
#include<iostream.h>

#include<conio.h>

class operation

{       private:

                int addi,subs,multi;

                float a,b,divi;

        public:

                void getdata();

                void add( );

                void sub( );

                void mult( );
                void div( );
}x;
void operation::getdata( )
        {       cout<<"Enter the value for a,b";
                cin>>a>>b;
        }
void operation::add( )
        {       getdata( );
                addi=a+b;
                cout<<"Adddition of two numbers"<<addi;
        }
void operation::sub( )
        {       getdata( );
                subs=a-b;
                cout<<"Substraction of two numbers"<<subs;
        }
void operation::div( )
        {       getdata( );
                divi=a/b;
                cout<<"Division of two numbers"<<divi;
        }
void operation::mult( )
        {       getdata( );
                multi=a*b;
                cout<<"Multiplication of two numbers"<<multi;
        }
```

```
void main( )
        {       int choice;
                char ch;
                clrscr( );
                do
                 {
                        cout<<"1.Addition\n2.Substraction\n3.Division\n4.Multiplication";
                        cout<<"Enter u r choice";
                        cin>>choice;
                        switch(choice)
                                {
                                        case 1:
                                                x.add( );
                                                break;
                                        case 2:
                                                x.sub( );
                                                break;
                                        case 3:
                                                x.div( );
                                                break;
                                        case 4:
                                                x. mult( );
                                                break;
                                }
                                cout<<"\nDo you want to continue";
                                cin>>ch;
```

```
                    }while(ch=='y');
                    getch( );
          }
```

OUTPUT:

```
1. Addition
2. Subtraction
3. Division
4. Multiplication
Enter u r choice 1
Enter values for a &b 10 20
Addition of two numbers 30
Do you want to continue Y
1. Addition
2. Subtraction
3. Division
4. Multiplication
Enter u r choice 2
Enter values for a & b 20 5
Subtraction of two numbers 15
Do you want to continue N
```

Conclusion: Thus, we have studied basic concept of object oriented programming concepts and structure of C++ program.

Experiment No: 3

Title: - Implementation of Constructor & Destructor.

Aim: - To study the concept of Constructor & Destructor and implement program for Constructor & Destructor.

Theory:-

Constructor

A constructor is special member function whose task is to initialize object. The objects of its class are special because its name same as the class name. The constructor is involved whenever as object of it associated class it is called constructor because it construct the value of data members of class.

When class contain constructor like the one define above it is guaranteed that an object created by the class will be initialized automatically.

e.g. integer nt i

Not only used the objects int i of type integer but also initialize its data members and to zero. There is no need to write any statement to invoke this function for object separately.

The constructor functions have some special characters.

1. They should be declared in the public section.

2. They are invoked automatically when the subject are created.

3. They do not have return types not even void and therefore they cannot return a value.

4. They cannot be inherited through a defined class can call base class constructor.

5. Like other C++ function they can have default arguments.

6. Constructors cannot be virtual.

7. We cannot refer to their address

8. An object with a constructor cannot be used as a member of union.

9. They make implicit calls to the operators new and delete when memory allocation is required.

Destructors: Destructor is a function that automatically executed when an object is destroyed. A destructor function gets executed whenever an instance of the class to which it belongs goes out of existence. The primary usage of the destructor function is t release space on the heap. A destructor function may be involved explicitly.

Syntax rules for writing a destructor function. The rules for writing a destructor function are:

1. A destructor function name is the same as that of the class.
2. It is declared with no return types. Since it cannot ever return a value.
3 .It cannot be declared static constructor variable.
4. It takes no argument and therefore cannot be over loaded.
5. It should have public access in the class declaration.
6. The general syntax of the destructor function in C++ is,

```
Class user _ name
{     private:
                \\ data variable
                \\ methods
                protected:
                \\data
public:
        user_name ( ); \\ constructor
        ~user_name ( ); \\ destructor
                        \\ methods
};
```

Program

Constructor & Destructor:

```
#include<iostream.h>
#include<conio.h>
class account
{
        private:

        float bal,intr,amt,acc;

       float rate,dep,mon;

       char nm[10];

        public:

                account();

                ~account();

                void getbal();

                void deposit();

                void compound();

                void  menu();

                void withdraw();

                void display();

};

account::account()

  {     cout<<"\nEnter the account number:";

        cin>>acc;

        cout<<"\nEnter the name of the customer:";

        cin>>nm;

        rate=0.02;

        intr=0;

        amt=0;

        dep=0;

        cout<<"\nEnter u r balance";

        cin>>bal;

  }
```

```
 account::~account()
{       cout<<"\nThanks";
        cout<<"\nMeet Again";
        getch();
}
  void account::deposit(void)
        {           cout<<"\nEnter the deposit:";
                    cin>>dep;
                    bal=bal+dep;
                    cout<<"\nThe total balance="<<bal;
        }
  void account::getbal(void)
        {     cout<<"\nBalance is:"<<bal;
        }
  void account::compound(void)
        {           cout<<"\nHow many Months:";
                    cin>>mon;
                    intr=bal*rate*mon;
                    bal=bal+intr;
                    cout<<"\nThe total balance"<<bal;
        }
  void account::withdraw()
        {           cout<<"\nEnter the amount for withdraw=";
                    cin>>amt;
                    if(amt<bal)
```

```
                {       bal=bal-amt;

                        cout<<"\nTotal balnce is="<<bal;

                }  else

                 {      cout<<"\n\nSORRY You Can NOt \withdraw";

                 }

        }
void account::menu()

        {       cout<<"1.Deposit\n2.Withdraw\n3.Compound\n4.Balance\n5.Display";

        }
void account::display()

        {       cout<<"Acc.no.\tName\tDiposit\tWithdraw\tInterest\tBalance\n";

                cout<<acc<<"\t"<<nm<<"\t"<<dep<<"\t"<<amt<<"\t"<<intr<<"\t"<<bal;

        }
void main()

  {     clrscr();

        account ac;

        int ch;

        char choice;

        do

         {      ac.menu();

                cout<<"\nEnter u r choice";

                cin>>ch;

                switch(ch)

                  {     case 1:

                                ac.deposit();

                                break;
```

```
                    case 2:
                            ac.withdraw();
                            break;
                    case 3:
                            ac.compound();
                            break;
                    case 4:
                            ac.getbal();
                            break;
                    case 5:
                            ac.display();
                            break;
                    default:
                            cout<<"Sorry u r entered wrong choice";
              }
            cout<<"\nDo U Want To Continue";
            cin>>choice;


     } while(choice=='y' || choice=='Y');
getch();
   }
```

OUTPUT:

```
Enter the account number: 09
Enter the name of the customer: Ram
Enter u r balance 10000
1. Deposit
```

```
2. Withdraw

3. Compound

4. Balance

5. Display

Enter u r choice 1

Enter the deposit: 10000

The total balance =20000

Do u want to continue Y

1. Deposit

2. Withdraw

3. Compound

4. Balance

5. Display

Enter u r choice 2

Enter amount for withdraw= 5000

The total balance =15000

Do u want to continue N

Thanks

Meet Again
```

Conclusion: Thus, we have studied concept of Constructor & Destructor and implement a program.

Experiment No: 4

Title: - Implementation of Function overloading and Constructor overloading,
Aim: - To study & implement the concept of Function overloading and Constructor overloading,
Theory:-

Overloading Functions:

Function overloading is the ability to have multiple definitions for the same function name within the same scope. The usual reason for picking a function name is to indicate the function‟s chief purpose. Readable programs generally have a diverse and literate choice of identifiers. Sometimes different functions are used for the same purpose. For example, consider a function that averages a sequence of *double* values versus one that averages a sequence of *int* values. Both are conveniently named *average (),* as in the following code. An overloaded function can be written in which there are distinct argument lists. The list of arguments must differ in either type or number or both. The compiler chooses the function with matching types and arguments. The *signature matching algorithm* gives the rules for performing this. By *signature,* we mean the list of types that are used in the function declaration. The rules for this match are very detailed. We discuss them in more detail in Section 5.9, *Overloading and Signature Matching*, on page 208. Conceptually, the algorithm picks the best available match.

Therefore, if the arguments are exactly matched to the signature, that is the match selected. In the preceding code, the arguments exactly matched the two versions of the overloaded function *average()*. It is important not to abuse this feature. Multiple functions with the same name can

often be confusing. It is not readily apparent to the programmer which version is being called. The use of function overloading is good design when each version of the overloaded function conceptually performs the same computation.

Polymorphism Using Function Overloading:

Polymorphism is a means of giving different meanings to the same function name or

operator, dependent on context. The appropriate meaning is selected on the basis of the type of data being processed. We have encountered one form of polymorphism when writing expressions of mixed type. Depending on the type of the operands, the division operator on native types might be either an integer division or a floating-point division. Object orientation takes advantage of polymorphism by linking behavior to the object‟s type. Operators, such as + and <<, have distinct meanings overloaded by operand type. For example, the expression cout << x is by convention expected to display an appropriate representation of x, depending on the type of object x. Overloading of functions gives the same function name different meanings. The name has several interpretations that depend on function selection. This is called *ad hoc polymorphism*. Operators are overloaded and selected based on the signature-matching algorithm. Overloading operators gives them new meanings. For example, the meaning of the expression a + b differs depending on the types of the variables a and b. Overloading the operator + for user-defined types allows them to be used in addition expressions in much the same way native types are used. The expression a + b could mean string concatenation, complex-number addition, or integer addition, depending on whether the variables were the user-defined ADT my string, the standard library class complex, or the native type int. Mixed-type expressions are also made possible by defining conversion functions. One principle of OOP is that user-defined types must enjoy the same privileges as native types. Where the C++ standard library adds the complex number type, the programmer expects the convenience of using it without regard to a native/nonnative distinction. Operator overloading and user-defined conversions let us use complex numbers in much the same way as we can use int or double. Later, we will discuss two other powerful forms of polymorphism, namely, parametric polymorphism using templates and pure polymorphism using virtual functions.

Overloaded Constructors:

It‟s convenient to be able to give variables of type Distance a value when they are first created. That is, we would like to use definitions like

```
Distance width(5, 6.25);
```

Which defines an object, width simultaneously initializes it to a value of 5 for feet and 6.25 for inches. To do this we write a constructor like this:

```
Distance (int ft, float in) : feet(ft), inches(in)
  { }
```

This sets the member data feet and inches to whatever values are passed as arguments to the constructor. So far so good. However, we also want to define variables of type Distance without initializing them,

```
Distance dist1, dist2;
```

In that program there was no constructor, but our definitions worked just fine. How could they work without a constructor? Because an implicit no–argument constructor is built into the program automatically by the compiler and it‟s this constructor that created the objects, even though we didn‟t define it in the class. This no–argument constructor is called the *default constructor*. If it weren‟t created automatically by the constructor, you wouldn‟t be able to create objects of a class for which no constructor was defined.

Often we want to initialize data members in the default (no–argument) constructor as well. If we let the default constructor do it, we don‟t really know what values the data members may be given. If we care what values they may be given, we need to explicitly define the constructor. We show how this looks:

```
Distance (): feet (0), inches (0.0)  //default constructor
  {  }      //no function body, doesn‟t do anything
```

The data members are initialized to constant values, in this case the integer value 0 and the float value 0.0, for feet and inches respectively. Now we can use objects initialized with the no–argument constructor and be confident they represent no distance (0 feet plus 0.0 inches) rather than some arbitrary value.

Since there are now two explicit constructors with the same name, Distance (), we say the constructor is *overloaded*. Which of the two constructors is executed when an object is created depends on how many arguments are used in the definition:

```
Distance length;           // calls first constructor
Distance width (11, 6.0);  // calls second constructor
```

Programs:-

a) Function Overloading:-

```
#include<iostream>

using namespace std;

class geometry

{       private:

        float l,b,h,dia,s,v,x,in,rn,ht;

        public:

        void area(float l,float b,float h)

        {       s=2*l*b*b*h*h*l;

                cout<<"Area of cuboid is:"<<s<<"sq.unit";

        }

        void area(float x)

        {

                s=6*x*x;

                cout<<"\nsurface area of cube:"<<s<<"sq.unit";

        }

        void area(float rn, float ht)

        {       float pi=3.142;

                s=2*pi*rn*(ht+rn);

                cout<<"\nSurface area of cylinder is:"<<s<<"sq.unit";
        }
        void vol(float l,float b,float h)
        {       v=l*b*h;
                cout<<"\n\nvolume of cubid is:"<<v<<"cb.unit";
        }
        void vol(float x)
        {       v=x*x*x;
                cout<<"\n\nvolume area of cube:"<<v<<"sq.unit";
        }
        void vol(float rn, float ht)
        {       float pi=3.142;
                v=pi*rn*ht*rn;
                cout<<"\n\nvolume of cylinder is:"<<v<<"sq.unit";
        }
};
int main()
{       geometry  g;
        int  ch,i=10;
        while(1)
        {       cout<<"\n\n\nFIND VOLUME&SURFACE AREA OF SHAPES.";
                cout<<"\n1.CUBOID\n2.CUBE\n3.CYLINDER\n4.EXIT.";
                cout<<"\nCHOICE::";
                cin>>ch;
                if(ch==4)
                break;
                switch(ch)
        {     case 1:
                          int l,b,h;
                          cout<<"\nenter l,b & h of cuboid";
                          cin>>l>>b>>h;
                          g.area(l,b,h);
                          g.vol(l,b,h);
                          break;
                  case 2:
                          int s;
                          cout<<"\nenter side of cube";
                          cin>>s;
                          g.area(s);
                          g.vol(s);
                          cout<<"\nDiagonal of cube:"<<1.73*i<<"unit";
                          break;
                  case 3:
                          int rn,ht;
                          cout<<"\nenter rn&ht of cylinder";
                          cin>>rn>>ht;
                          g.area(rn,ht);
                          g.vol(rn,ht);
                          break;
              default:
                          cout<<"\nWrong choice";
        }
}       return 0;         }
```

OUTPUT:

```
FIND VOLUME&SURFACE AREA GEOMETRIC SHAPES.

1 .CUBOID

2. CUBE

3. CYLINDER

4. EXIT.

CHOICE::2

Enter side of cube4

Surface area of cube: 96sq.unit

Volume area of cube: 64sq.unit

Diagonal of cube: 17.3unit.

FIND VOLUME&SURFACE AREA GEOMETRIC SHAPES.

1. CUBOID

2. CUBE

3. CYLINDER

4. EXIT.

CHOICE::1

Enter l,b & h of cuboid4

5

6

Area of cuboid is: 28800sq.unit

Volume of cubid is: 120cb.unit

FIND VOLUME&SURFACE AREA GEOMETRIC SHAPES.

1. CUBOID

2. CUBE

3. CYLINDER
```

```
4. EXIT.

CHOICE::3

Enter rn & ht of cylinder3

5

Surface area of cylinder is: 150.815994sq.unit

Volume of cylinder is: 141.389999sq.unit

FIND VOLUME&SURFACE AREA GEOMETRIC SHAPES.

1. CUBOID

2. CUBE

3. CYLINDER

4. EXIT.

CHOICE::4
```

b) Constructor Overloading:

```
#include<iostream>

Using namespace std;

int ch;

class constr

{           public:

                        int a,b,c;

                        constr(int,int);

                        constr(float,float);

                        constr(int,int,int);

                        void putdata();

                        void putdata1();

};
```

```
constr::constr(int x,int y)
{        if(ch==1)
         {        int c;
                  c=x+y;
                  cout<<"\n\tADDITION="<<c;
         }
                   if(ch==2)
         {        int c;
                  c=x-y;
                  cout<<"\n\tSUBSTRACTION="<<c;
         }
}
constr::constr(float x,float y)
{        if(ch==1)
         {        float c;
                  c=x+y;
                  cout<<"\n\tADDITION="<<c;
         }         if(ch==2)
         {        float c;
                  c=x-y;
                  cout<<"\n\tSUBSTRACTION="<<c;
         }
}        constr::constr(int a,int b,int c)
         {        int d;
                  d=a*b*c;
                  cout<<"\n\tMULTIPLICATION="<<d;       }
```

```
int main()
{    static    int    x,y,z;
          static float p,q;
          while(1)
          { cout<<"\n\t1.ADDITION\n\t2.SUBSTRACTION\n\t3.MULTIPLICATION\n\t4.EXIT";
          cout<<"\n\tENTER YOUR CHOICE\t";
          cin>>ch;
          switch(ch)
          {                 case 1:
                             cout<<"\n\tENTER TWO INTEGER NUMBERS\t";
                             cin>>x>>y;
                             constr c1(x,y);
                             cout<<"\n\tENTER TWO FLOAT NUMBERS\t";
                             cin>>p>>q;
                             constr c2(p,q);
                             break;
                  case 2:
                             cout<<"\n\tENTER TWO INTEGER NUMBERS\t";
                             cin>>x>>y;
                             constr c3(x,y);
                             cout<<"\n\tENTER TWO FLOAT NUMBERS\t";
                             cin>>p>>q;
                             constr c4(p,q);
                             break;
                  case 3:
                             cout<<"\n\tENTER THREE INTEGER NUMBERS\t";
```

```
                        cin>>x>>y>>z;

                        constr c5(x,y,z);

                        break;

                case 4:

                        return(0);

                        break;

                default:

                                cout<<"\n\tWRONG CHOICE";

        }       return 0;

}
```

OUTPUT:

```
    1. ADDITION

    2. SUBSTRACTION

    3. MULTIPLICATION

    4. EXIT

    ENTER YOUR CHOICE

    1

    ENTER TWO INTEGER NUMBERS

    1

    1

    ADDITION=2

    ENTER TWO FLOAT NUMBERS

    1.2

    2.2

    ADDITION=3.4

    1. ADDITION
```

```
 2. SUBSTRACTION

 3. MULTIPLICATION

 4. EXIT

 ENTER YOUR CHOICE

3

 ENTER THREE INTEGER NUMBERS

  2

  5

  1

 MULTIPLICATION=10

 1. ADDITION

 2. SUBSTRACTION

 3. MULTIPLICATION

 4. EXIT

 ENTER YOUR CHOICE        4
```

Conclusion: - Thus we implemented the concept of Function overloading, Constructor overloading, Operator overloading.

Experiment No: 5

Title: - Implementation of Operator overloading.

Aim: - To study & implement the concept of Operator overloading.

Theory:- Overloading Operators

The keyword operator is used to define a type-conversion member function, as well as

to overload the built-in C++ operators. Just as a function name such as print() can be given a variety of meanings depending on its arguments, so can an operator such as + be given additional meanings. *Overloading operators* allows infix expressions of both ADTs and built-in types to be written. In many instances, this important notational convenience leads to shorter, more readable programs.

Unary and binary operators can be overloaded as non static member functions. Implicitly, they are acting on a class value. Most unary operators can be overloaded as ordinary functions, taking a single argument of class or reference-to-class type. Most binary operators can be overloaded as ordinary functions, taking one or both arguments of class or reference-to-class type. The operators =, ( ), [ ], and -> must be overloaded with a non static member function.

Although meanings can be added to operators, their associativity and precedence remain the same. For example, the multiplication operator remains of higher precedence than the addition operator. Almost all operators can be overloaded. The exceptions are the member operator. the member object selector .*, the ternary conditional expression operator ? : , the sizeof operator, and the scope resolution operator :: .

Available operators include all of the arithmetic, logical, comparison, equality, assignment, and bit operators. Furthermore, the increment and decrement operators, ++ and --, a have distinct prefix and postfix meanings. The subscript or index operator [] and the function call () can also be overloaded. The structure pointer operator -> and the member pointer selector operator ->* can be overloaded. It is also possible to overload new and delete. The assignment, function call, subscripting, and class pointer operators can be overloaded only by non static member functions.

PROGRAM:- //Programe to overload „+" and „>" operator

```
#include<iostream>
using namespace std;
class number
{ int a;
  public:
            number()
           {
             a=0;
            }
        number(int x)
```

```
            {    a=x;
            }
number operator +(number &n1);
int operator >(number & n);
void display();
};
number number::operator +(number &n1)
     { number n;
       n.a=a+n1.a;
       return n;
     }
int number::operator >(number &n)
   {    return(a>n.a);
  }
void number::display()
  {  cout<<a<<"\n";
  }
void main()
  { int a,b,c;
     cout<<"\n\tPROGRAM FOR OPERATOR OVERLOADING\n";
     cout<<"\n\tENTER THREE FOR A,B,C\t";
     cin>>a;
     cout<<"\n\t\t\t\t";
     cin>>b;
     cout<<"\n\t\t\t\t";
     cin>>c;
  number n(a),n1(b),n2(c);
  number  n3,n4;
  n3=n+n1;
  cout<<"\n\tX=A+B=\t";
  n3.display();
  n4=n+n2;
  cout<<"\n\tY=A+C=\t";
  n4.display();
  if(n3>n4)
  cout<<"\n\tX IS GREATER ";
  else
  cout<<"\n\tY IS GREATER";
}
```

OUTPUT

```
ENTER THREE FOR A,B,C  10
20
30
X=A+B=  30
Y=A+C=  40
Y IS GREATER
```

Conclusion: - Thus we have implemented the concept of Operator overloading.

Experiment No 6

Title: - Implementation Multilevel inheritance

Aim: - To study & implement the concept of Multilevel inheritance.

Theory:-

What is Inheritance:-

Inheritance is the process by which new classes called *derived* classes are created from existing classes called *base* classes. The derived classes have all the features of the base class and the programmer can choose to add new features specific to the newly created derived class.

For example: a programmer can create a *base* class named *fruit* and define *derived* classes as mango, orange, banana, etc. Each of these derived classes, (mango, orange, banana, etc.) has all the features of the *base* class (fruit) with additional attributes or features specific to these newly created derived classes. Mango would have its own defined features, orange would have its own defined features, banana would have its own defined features, etc. This concept of *Inheritance* leads to the concept of *polymorphism*.

Features or Advantages of Inheritance:

Reusability:

Inheritance helps the code to be reused in many situations. The base class is defined and once it is compiled, it need not be reworked. Using the concept of inheritance, the programmer can create as many derived classes from the base class as needed while adding specific features to each derived class as needed.

Saves Time and Effort:

The above concept of reusability achieved by inheritance saves the programmer time and effort. Since the main code written can be reused in various situations as needed.

Overriding--With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class.

Data hiding -- Base class can decide to keep some data private so that it cannot be altered by the derived class

Inheritance can also make application code more flexible to change because classes that inherit from a common superclass can be used interchangeably. If the return type of a method is superclass

Types/Forms of Inheritance

There are 05 forms of Inheritance as shown in figure

01. **Single Inheritance:** It is the inheritance hierarchy wherein one derived class inherits from one base class.
02. **Multiple Inheritance:** It is the inheritance hierarchy wherein one derived class inherits from multiple base class(es)
03. **Hierarchical Inheritance:** It is the inheritance hierarchy wherein multiple subclasses inherit from one base class.
04. **Multilevel Inheritance:** It is the inheritance hierarchy wherein subclass acts as a base class for other classes.
05. **Hybrid Inheritance:** The inheritance hierarchy that reflects any legal combination of other four types of inheritance.

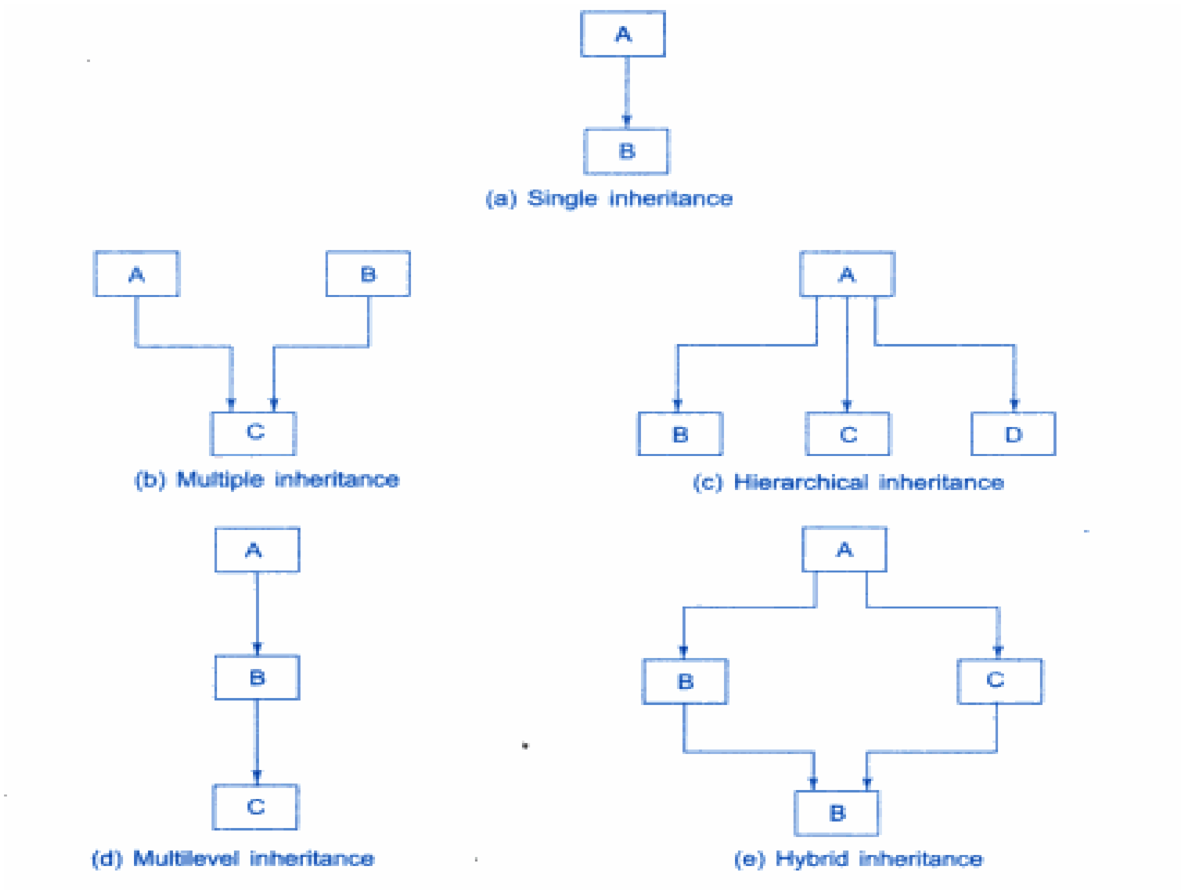


Fig. 8.1 ⇔ Forms of inheritance

Simple / Single Inheritance:-

The transitive nature of inheritance is reflected by this form of inheritance. When a class is derived from a class which is a derived class itself – then this is referred to as multilevel inheritance. For example

ClassA, ClassB and ClassC.
ClassB is derived from ClassA and ClassC is derived ClassB This is multi level inheritance..

```
ClassA
   ^
   |
ClassB
   ^
   |
ClassC
```

PROGRAM:-

Simple Inheritance :-

```
#include <iostream.h>

class base {

  int i, j;

public:

  void set(int a, int b) { i=a; j=b; }

  void show() { cout << i << " " << j << "\n"; }

};

class derived : public base {

  int k;

public:

  derived(int x) { k=x; }

  void showk() { cout << k << "\n"; }

};

int main()

{
  derived ob(3);

  ob.set(1, 2); // access member of base

  ob.show(); // access member of base

  ob.showk(); // uses member of derived class

  return 0;

}
```

OUTPUT

```
1

2

3
```

a) **Multilevel Inheritance :-**

```
#include<iostream>

using namespace std;

int ch;

class ONE

{       protected:

                        int cc1,cc2,milage1,milage2;

        public:

                        char type1[20],type2[20];

                        void getdata()

                        {       cout<<"\n\tENTER CC\t";

                                cin>>cc1;

                                cout<<"\n\tENTER MILAGE\t";

                                cin>>milage1;

                        }

                      void getdata1()

                        {       cout<<"\n\tENTER CC\t";
```

```
                    cin>>cc2;
                    cout<<"\n\tENTER MILAGE\t";
                    cin>>milage2;
                }
};
class TWO:public ONE
{       protected:
                int seat1,seat2,model1,model2;
        public:
                void getdata2()
                {       ONE::getdata();
                        cout<<"\n\tENTER NUMBER OF SEATS\t";
                        cin>>seat1;
                        cout<<"\n\tENTER MODEL\t";
                        cin>>model1;
                }
                void getdata3()
                {       ONE::getdata1();
                        cout<<"\n\tENTER NUMBER OF SEATS\t";
                        cin>>seat2;
                        cout<<"\n\tENTER MODEL\t";
                        cin>>model2;
                }
};
class THREE:public TWO
{
public:         void getdata4()
                {       TWO::getdata2();
                }
                void getdata5()
                {        TWO::getdata3();
                }
                void display();
                void compare();
}obj;
void THREE::display()
{       int j;
cout<<"\n";
for(j=0;j<=50;j++)
cout<<"_";
cout<<"\n\tINFORMATION OF FIRST VEHICLE";.
cout<<"\n";
for(j=0;j<=50;j++)
cout<<"_";
cout<<"\n\tTYPE=\t"<<obj.type1;
cout<<"\n\tMILAGE=\t"<<obj.milage1;
cout<<"\n\tCC=\t"<<obj.cc1;
cout<<"\n\tNo OF SEATS="<<obj.seat1;
cout<<"\n\tMODEL=\t"<<obj.model1;
cout<<"\n";
for(j=0;j<=50;j++)
cout<<"_";
cout<<"\n\tINFORMATION OF SECOND VEHICLE";

cout<<"\n";

for(j=0;j<=50;j++)

cout<<"_";

cout<<"\n\tTYPE=\t"<<obj.type2;

cout<<"\n\tMILAGE=\t"<<obj.milage2;

cout<<"\n\tCC=\t"<<obj.cc2;

cout<<"\n\tNo OF SEATS="<<obj.seat2;

cout<<"\n\tMODEL=\t"<<obj.model2;

cout<<"\n";

for(j=0;j<=50;j++)

cout<<"_";

}void THREE::compare()

{       int j;

if(ch==1)

{       if((obj.milage1>obj.milage2))

        {       cout<<"\n\tFIRST VEHICLE IS BETTER THAN SECOND
                VEHICLE";

        }

        else

        {       cout<<"\n\tSECOND VEHICLE IS BETTER THAN FIRST
                VEHICLE";

        }

        cout<<"\n";

        for(j=0;j<=50;j++)

        cout<<"_";

}
if(ch==2)
{       if((obj.cc1>obj.cc2))
        {       cout<<"\n\tFIRST VEHICLE IS BETTER THAN SECOND
                VEHICLE";
        }
        else
        {       cout<<"\n\tSECOND VEHICLE IS BETTER THAN FIRST
                VEHICLE";
        }
        cout<<"\n";
        for(j=0;j<=50;j++)
        cout<<"_";
}
}
int main()
{       cout<<"\n\tPROGRAM FOR MULTILEVEL INHERITANCE\n";
while(1)
{       cout<<"\n\t1.TWO WHEELER\n\t2.FOUR WHEELER\n\t3.EXIT";
        cout<<"\n\tENTER YOUR CHOICE\n\t";
        cin>>ch;
        switch(ch)
        {       case 1:
                        cout<<"\n\tFOR FIRST VEHICLE\n";
                        obj.THREE::getdata4();
                        cout<<"\n\n\tFOR SECOND VEHICLE\n";
                        obj.THREE::getdata5();
                        strcpy(obj.type1,"TWO WHEELER");
                        strcpy(obj.type2,"TWO WHEELER");
                        obj.display();
                        obj.compare();
                        break;
                    case 2:
                        cout<<"\n\tFOR FIRST VEHICLE\n";
                        obj.THREE::getdata4();
                        cout<<"\n\n\tFOR SECOND VEHICLE\n";
                        obj.THREE::getdata5();
                        strcpy(obj.type1,"FOUR WHEELER");
                        strcpy(obj.type2,"FOUR WHEELER");
                        obj.display();
                        obj.compare();
                        break;
                    case 3:
                        return (0);
                        break;
                default:
                        cout<<"\n\tWRONG CHOICE";
                }
        }
}
```

OUTPUT

```
PROGRAM FOR MULTILEVEL INHERITANCE
1. TWO WHEELER
2. FOUR WHEELER
3. EXIT
```

```
 ENTER YOUR CHOICE

1

FOR FIRST VEHICLE

ENTER CC        135

ENTER MILAGE    60

ENTER NUMBER OF SEATS   2

ENTER MODEL     2000

FOR SECOND VEHICLE

ENTER CC        100

ENTER MILAGE    50

ENTER NUMBER OF SEATS   2

ENTER MODEL     2003

_______________________________________________________

INFORMATION OF FIRST VEHICLE

________________________________________________________

TYPE=  TWO WHEELER

MILAGE= 60

CC=     135

No OF SEATS=2

MODEL=  2000

_______________________________________________________

INFORMATION OF SECOND VEHICLE

________________________________________________________

TYPE=  TWO WHEELER

MILAGE= 50

CC=     100
```

```
No OF SEATS=2

MODEL= 2003

______________________________________________________

FIRST VEHICLE IS BETTER THAN SECOND VEHICLE

______________________________________________________

1. TWO WHEELER

2. FOUR WHEELER

3. EXIT

  ENTER YOUR CHOICE

2

FOR FIRST VEHICLE

ENTER CC        600

ENTER MILAGE    25

ENTER NUMBER OF SEATS   4

ENTER MODEL     1999

FOR SECOND VEHICLE

ENTER CC        800

ENTER MILAGE    30

ENTER NUMBER OF SEATS   4

ENTER MODEL     2001

______________________________________________________

INFORMATION OF FIRST VEHICLE

______________________________________________________

TYPE=  FOUR WHEELER

MILAGE= 25

CC=     600
```

```
No OF SEATS=4

MODEL=  1999

_______________________________________________

INFORMATION OF SECOND VEHICLE

_______________________________________________

TYPE=  FOUR WHEELER

MILAGE= 30

CC=     800

No OF SEATS=4

MODEL 2001

_______________________________________________

SECOND VEHICLE IS BETTER THAN FIRST VEHICLE

_______________________________________________

1. TWO WHEELER

2. FOUR WHEELER

3. EXIT

ENTER YOUR CHOICE

3
```

Conclusion: - Thus we implemented the concept of Inheritance in terms of Multilevel Inheritance.

Experiment No 7

Title: - Implementation of Multiple Inheritance

Aim: - To study & implement the concept of Multiple Inheritances.

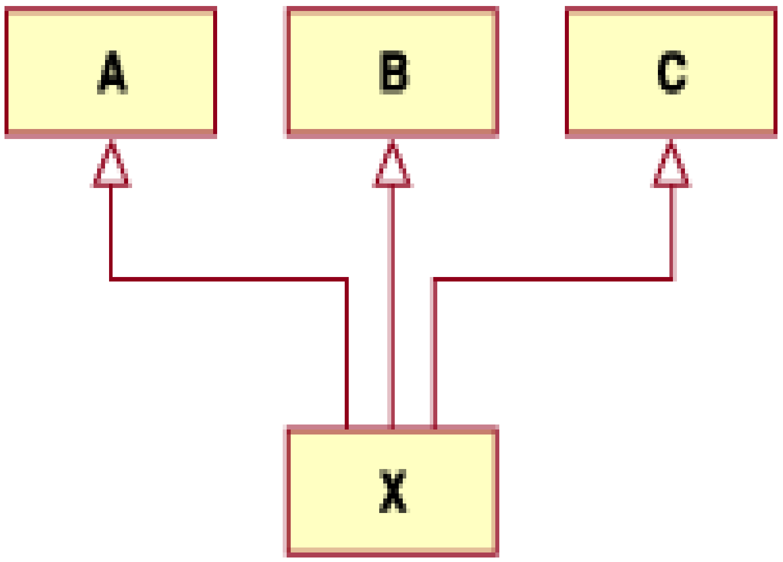
Theory:-

Multiple inheritance:

You can derive a class from any number of base classes. Deriving a class from more than one direct base class is called *multiple inheritance*. In the following example, classes A, B, and C are direct base classes for the derived class X:

```
class A { /* ... */ };
class B { /* ... */ };
class C { /* ... */ };
class X : public A, private B, public C { /* ... */ };
```

The following *inheritance graph* describes the inheritance relationships of the above example. An arrow points to the direct base class of the class at the tail of the arrow:

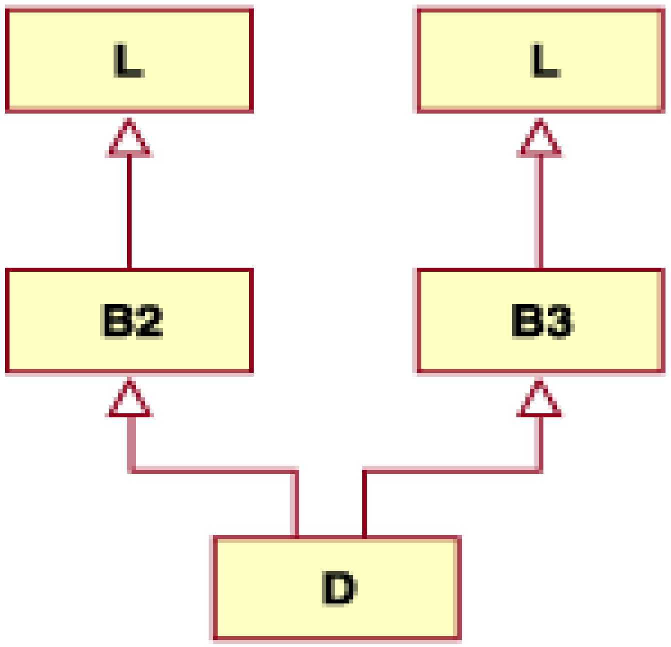


The order of derivation is relevant only to determine the order of default initialization by constructors and cleanup by destructors.

A direct base class cannot appear in the base list of a derived class more than once:

```
class B1 { /* ... */ };                    // direct base class
class D : public B1, private B1 { /* ... */ }; // error
```

However, a derived class can inherit an indirect base class more than once, as shown in the following example:



```
class L { /* ... */ };                  // indirect base class
class B2 : public L { /* ... */ };
class B3 : public L { /* ... */ };
class D : public B2, public B3 { /* ... */ }; // valid
```

In the above example, class D inherits the indirect base class L once through class B2 and once through class B3. However, this may lead to ambiguities because two subobjects of class L exist, and both are accessible through class D. You can avoid this ambiguity by referring to class L using a qualified class name. For example:

B2::L

or

B3:: L.

You can also avoid this ambiguity by using the base specified virtual to declare a base class.

PROGRAMME:-
// Program to implement Multiple Inheritance :-

```
#include<iostream>
```

```
using namespace std;

class employee1

{       protected:

                char nm[20];

                char dpt[20],des[10];

                int code;

        public:

                 void getdata()

                 {       cout<<"\n\tENTER THE EMPLOYEE NAME:\t";

                         cin>>nm;

                         cout<<"\n\tENTER CODE:\t";

                         cin>>code;

                         cout<<"\n\t ENTER DEPARTMENT:\t";

                         cin>>dpt;

                         cout<<"\n\tENTER DESSIGNATION:\t";

                         cin>>des;

                 }};

class employee2

{       protected:

                 int acno,date,month,year;

        public:

                void getdata1()

                {       cout<<"\n\t ENTER THE ACCOUNT NO:\t";

                        cin>>acno;

                        cout<<"\n\t ENTER DATE OF JOINING:\t";

                        cin>>date>>month>>year;     }};
```

```
class employee3:public employee1,public employee2
{       protected:
                        float days,sal,salary,den,hra,travel,gross,
                              prfund,gpf,lic,dedn,net,total1,total2;
                public:
                        void getdata2()
                        {       employee1::getdata();
                                employee2::getdata1();
                                cout<<"\n\tENTER PAYABLE DAYS:\t";
                                cin>>days;
                                cout<<"\n\tENTER SALARY PER DAY:\t";
                                cin>>sal;
                        }
                         void formula()
                        {       salary=days*sal;
                                den=salary*0.05;
                                hra=salary*0.12;
                                travel= salary*0.09;
                                gross=salary+den+hra+travel;
                                prfund=salary*0.04;
                                gpf=salary*0.01;
                                lic=salary*0.12;
                                dedn=prfund+gpf+lic;
                                total1=gross;
                                total2=dedn;
                                net=total1-total2;      }
                void display();

};

void employee3::display()

{       int i,j;

        cout<<"\n";

        for(i=0;i<65;i++)

        cout<<"_";

        cout<<"\n\t\t\tWIPRO\n";

        for(i=0;i<65;i++)

        cout<<"_";

        cout<<"\n|\tNAME="<<nm<<"\t\t|\tACCOUNT NO="<<acno<<"\t\t|";

        cout<<"\n|\tCODE="<<code<<"\t\t|\tDESIGNATION="<<des<<"\t|";

        cout<<"\n|\tDEPT="<<dpt<<"\t\t|\tDOJ="<<date<<"/"<<month<<"/"<<

                        year<<"\t\t|";

        cout<<"\n|\t\t\t\t|\tDAYS PAYMENT="<<days<<"\t\t|\n";

        for(i=0;i<65;i++)

        cout<<"_";

        cout<<"\n|PAYMENT\t|\tRs\t|\tDEDUCTION |    Rs\t|\n";

        for(i=0;i<65;i++)

        cout<<"_";

        cout<<"\n|BASIC SALARY=\t\t"<<salary<<"\t|\t

                        PR.FUND=\t"<<prfund<<"\t|";

        cout<<"\n|\tDENTAL=\t\t"<<den<<"\t|\tGPF=\t\t"<<gpf<<"\t|";

        cout<<"\n|\tHRA=\t\t"<<hra<<"\t|\tLIC=\t\t"<<lic<<"\t|";

        cout<<"\n|\tTRAVELLING=\t"<<travel<<"\t|\n";

        for(i=0;i<65;i++)
        cout<<"_";
        cout<<"\n|\tTOTAL1="<<total1<<"\t\t|\tTOTAL2="<<total2<<
                        "\t\t|\n";
        for(i=0;i<65;i++)
        cout<<"_";
        cout<<"\n\tNET PAYMENT="<<net;
        cout<<"\n";
        for(i=0;i<65;i++)
        cout<<"_";
 }
void main()
{       employee3 e;
        e.getdata2();
        e.formula();
        e.display();
}
```

OUTPUT

```
ENTER THE EMPLOYEE NAME:      Rajendara
ENTER CODE:    123
ENTER DEPARTMENT:    Construction
ENTER DESSIGNATION:    Engineer
ENTER THE ACCOUNT NO: 234
ENTER DATE OF JOINING: 25  04 2000
ENTER PAYABLE DAYS:    30
ENTER SALARY PER DAY:  900
```

```
____________________________________________________________

                        WIPRO

____________________________________________________________

|    NAME=Rajendara        |    ACCOUNT NO=234           |

|    CODE=123              |    DESIGNATION=Engineer     |

|    DEPT=Construction     |    DOJ=25/4/2000            |

|                          |    DAYS PAYMENT=30          |

____________________________________________________________

|PAYMENT           |    Rs    |    DEDUCTION |  Rs    |

____________________________________________________________

|BASIC SALARY=       27000  |    PR.FUND=     1080   |

|DENTAL=              1350  |    GPF=          270   |

|HRA=                 3240  |    LIC=         3240   |

|TRAVELLING=          2430  |

____________________________________________________________

|    TOTAL1=34020             |    TOTAL2=4590          |

____________________________________________________________

 NET PAYMENT=2943

____________________________________________________________
```

Conclusion: **-** Thus we implemented the concept of Inheritance in terms of Multiple Inheritance.

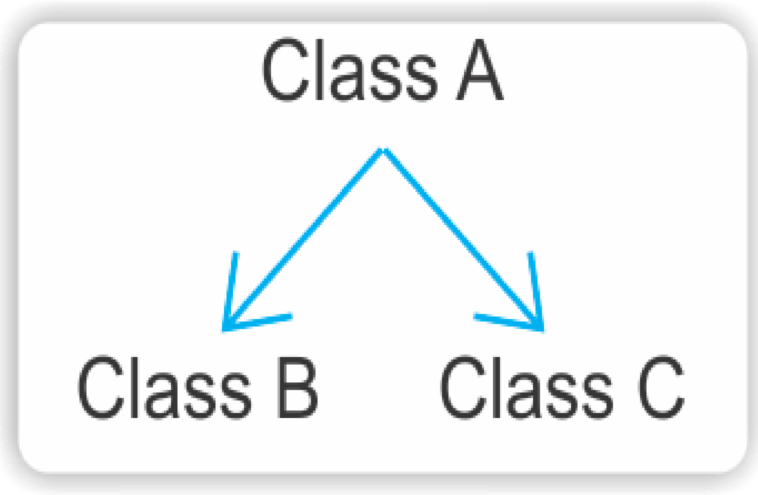
Experiment No 8

Title: - Implementation Hierarchical Inheritance

Aim: - To study & implement the concept of Hierarchical inheritance.

Theory:-

Hierarchical inheritance
It is the inheritance hierarchy wherein multiple subclasses inherit from one base class.
In these inheritance , **derived class** inherits property of **Base Class** but in these inheritance only one Base Class and multiple derived class are involved.



SYNTAX:-

```
class base_class_name
{
private:
…………….
public:
…………….
};
          class derived_class_name1 : public base_class_name
          private:
          ……………
          public:
          ……………
          };
class derived_class_name2 : public base_class_name
private:
…………..
public:
…………..
};
```

In the Below program Class A is Base Class and Class B ,Class C , derived class.

PROGRAME:

```
#include<iostream.h>
#include<conio.h>

class A  //Base Class
{
public:
      int a,b;
        void getnumber()
    {
    cout<<"\n\nEnter Number :::\t";
    cin>>a;
    }
};
class B : public A  //Derived Class 1
{
    public:
    void square()
    {
     getnumber(); //Call Base class property
    cout<<"\n\n\tSquare of the number :::\t"<<(a*a);
    cout<<"\n\n\t------------------------------------------------------------------";
    }
};
 class C :public A //Derived Class 2
{    public:
       void cube()
    {    getnumber(); //Call Base class property
    cout<<"\n\n\tCube of the number :::\t"<<(a*a*a);
    cout<<"\n\n\t------------------------------------------------------------------";
    }
};
 int main()
{clrscr();
 B b1;         //b1 is object of Derived class 1
b1.square();  //call member function of class B
C c1;         //c1 is object of Derived class 2
c1.cube();    //call member function of class C
getch();
}
```

OUTPUT

```
Enter number : : : 2

Square of Number  : : : 4

-----------------------------------------------------------

Enter number : : : 3

Cube of Number  : : : 9

------------------------------------------------------------
```

Conclusion: - Thus we implemented the concept of Inheritance in terms of Hierarchical Inheritance.

Experiment No 9

Title: - Implementation of friend class and friend function.

Aim: - To study & implement the concept of Friend class and function.

Theory:-

Friend Functions

The concepts of encapsulation and data hiding dictate that nonmember functions should not be able to access an object‟s private or protected data. The policy is, if you‟re not a member, you can‟t get in. However, there are situations where such rigid discrimination leads to considerable inconvenience.

Friend Classes

The member functions of a class can all be made friends at the same time when you make the entire class a friend.

// Program to implement concept of friend class.

```
#include <iostream>
using namespace std;

class alpha
  {
  private:
    int data1;

   public:
    alpha() : data1(99) { }  //constructor

    friend class beta;         //beta is a friend class
  };

class beta
  {                            //all member functions can

    public:                    //access private alpha data
    void func1(alpha a) { cout << "\ndata1=" << a.data1; }
    void func2(alpha a)  { cout << "\ndata1=" << a.data1; }

  };
```

```
int main()
  {
  alpha a;
  beta b;

  b.func1(a);
  b.func2(a);
  cout << endl;
  return 0;
}
```

In class alpha the entire class beta is proclaimed a friend. Now all the member functions of beta can access the private data of alpha (in this program the single data item data1).

Note that in the friend declaration we specify that beta is a class using the class keyword:
friend class beta;

We could also have declared beta to be a class before the alpha class specifier, as in previous examples.

class beta;

and then, within alpha, referred to beta without the class keyword:

friend beta;

b) Friend class and function:-

1) Friend Class:

```
#include<iostream>
Using namespace std;
class data
{
    int rno,year;
    char name[30],class1[10],branch[20];
  public:
 void getdata()
 {
  cout<<"\n\tENTER ROLL NO\t";
  cin>>rno;
  cout<<"\n\tENTER NAME\t";
 cin>>name;
 cout<<"\n\tENTER YEAR\t";
 cin>>year;
 cout<<"\n\tENTER CLASS\t";
 cin>>class1;
 cout<<"\n\tENTER BRANCH\t";
 cin>>branch;
 }
  friend class exam;
}obj;
class exam
 {
   int mark[5],tot,avg;
  char grade[20];
  public:
 void getmark();
 void total();
 void grade1();
 void display();
 }obj1;

          void exam::getmark()
            {
              int i;
              cout<<"\n\tENTER MARKS OF SUBJECTS AS FOLLOWS";
              cout<<"\n\t1.DMS\t2.AM\t3.DSMP\t4.DS\t5.CN-1\n";
             for(i=0;i<5;i++)
            cin>>mark[i];
            }

void exam::total()
 {
  int i;
  tot=0;

  for(i=0;i<5;i++)
  {
     tot=tot+mark[i];
   }
   avg=tot/5;
}

          void exam::grade1()
          {
            if(avg>=66)
            strcpy(grade,"DISTINCTION");
            else if(avg>=60)
            strcpy(grade,"FIRST CLASS");
            else if(avg>=50)
            strcpy(grade,"SECOND CLASS");
            else if(avg>=40)
            strcpy(grade,"PASS CLASS");
            else
            strcpy(grade,"FAIL");
           }
```

```
void exam::display()
{
  int  j;
  cout<<"\n\n";
  for(j=0;j<=50;j++)
  cout<<"*";
  cout<<"\n\tSHIVAJI UNIVERSITY,KOLHAPUR";
  cout<<"\n\t\tRESULT";
  cout<<"\n\tNAME="<<obj.name;
 cout<<"\n\tROLL NO="<<obj.rno<<"\tYEAR="<<obj.year;
 cout<<"\n\tCLASS="<<obj.class1<<"\tBRANCH="<<obj.branch;
 cout<<"\n\n";
 for(j=0;j<=48;j++)
 cout<<"_";
 cout<<"\n|\tSR NO\t|\tSUBJECT\t|\tMARKS\t|";
 cout<<"\n";
 for(j=0;j<=48;j++)
 cout<<"_";
 cout<<"\n|\t1.\t|\tDMS\t|\t"<<mark[0]<<"\t|";
 cout<<"\n";
 for(j=0;j<=48;j++)
 cout<<"_";
 cout<<"\n|\t2.\t|\tAM\t|\t"<<mark[1]<<"\t|";
 cout<<"\n";
 for(j=0;j<=48;j++)
 cout<<"_";
 cout<<"\n|\t3.\t|\tDSMP\t|\t"<<mark[2]<<"\t|";
 cout<<"\n";
 for(j=0;j<=48;j++)
 cout<<"_";
 cout<<"\n|\t4.\t|\tDS\t|\t"<<mark[3]<<"\t|";
 cout<<"\n";
 for(j=0;j<=48;j++)
 cout<<"_";
 cout<<"\n|\t5.\t|\tCN-1\t|\t"<<mark[4]<<"\t|";
 cout<<"\n";
 for(j=0;j<=48;j++)
 cout<<"_";
 cout<<"\n\t\t|\tTOTAL\t|\t"<<tot<<"\t|";
 cout<<"\n";
 for(j=0;j<=48;j++)
 cout<<"_";
 cout<<"\n\tAVERAGE="<<avg;
 cout<<"\n\tGRADE="<<grade;
 cout<<"\n\n";
 for(j=0;j<=50;j++)
 cout<<"*";
}
void main()
{
obj.getdata();
obj1.getmark();
obj1.total();
obj1.grade1();
obj1.display();
}
```

OUTPUT:

```
ENTER ROLL NO  123
ENTER NAME      Ganesh
ENTER YEAR      2010
ENTER CLASS     SE
ENTER BRANCH    CSE
ENTER MARKS OF SUBJECTS AS FOLLOWS
1. DMS  2.AM     3. DSMP  4.DS    5.CN-1
70
89
66
58
68

****************************************************************************
                        SHIVAJI UNIVERSITY, KOLHAPUR
                                                   RESULT
  NAME=Ganesh
  ROLL NO=123     YEAR=2010
  CLASS=SE        BRANCH=CSE
__________  _________________________  _____________
|    SR NO   |     SUBJECT |     MARKS   |
__________  _________________________  _____________
|      1.    |       DMS      |      70       |
__________  _________________________  _____________
|      2.    |       AM       |      89       |
__________  _________________________  _____________
|      3    |       DSMP     |      66       |
__________  _________________________  _____________
|    4.     |        DS      |       58     |
__________  _________________________  _____________
|    5.     |       CN-1    |       68     |
__________  _________________________  _____________
           |      TOTAL  |         351    |
__________________________________________________
  AVERAGE=70
  GRADE=DISTINCTION
```

****

2) Friend Function:

//Program to implement concept of friend function

```
#include<iostream>
Using namespace std;
class one
{
   int a,b,c;
   public:
 void getdata();
 friend int largest(one obj);
}obj2;

            void one::getdata()
            {
               cout<<"\n\tENTER THREE NUMBERS\n";
               cin>>a>>b>>c;
            }

int largest(one obj)
{
    int max;
  max=obj.a;
  if(obj.b>max)
  max=obj.b;
  if(obj.c>max)
  max=obj.c;
  eturn(max);
}

            void main()
            {
               int maxi;
               cout<<"\n\tPROGRAM FOR FRIEND FUNCTION";
              obj2.getdata();
              maxi=largest(obj2);
              cout<<"\n\tLARGEST NO="<<maxi;
            }
```

OUTPUT

```
ENTER THREE NUMBERS

20
30
10
LARGEST NO=30
```

Experiment No 10

Title: - Implementation of Virtual function and Virtual class,

Aim: - To study & implement the concept of Virtual function and Virtual class.
Theory:-

Virtual Functions:

Virtual means *existing in appearance but not in reality*. When virtual functions are used, a program that appears to be calling a function of one class may in reality be calling a function of a different class. Why are virtual functions needed? Suppose you have a number of objects of different classes but you want to put them all in an array and perform a particular operation on them using the same function call. For example, suppose a graphics program includes several different shapes: a triangle, a ball, a square, and so on. Each of these classes has a member function draw() that causes the object to be drawn on the screen.

Now suppose you plan to make a picture by grouping a number of these elements together and you want to draw the picture in a convenient way. One approach is to create an array that holds pointers to all the different objects in the picture. The array might be defined like this:

```
shape* ptrarr[100]; // array of 100 pointers to shapes
```

If you insert pointers to all the shapes into this array, you can then draw an entire picture using a simple loop:

```
for(int j=0; j<N; j++)
ptrarr[j]->draw();
```

This is an amazing capability: Completely different functions are executed by the same function call. If the pointer in ptrarr points to a ball, the function that draws a ball is called; if it points to a triangle, the triangle-drawing function is called. This is called *polymorphism*, which means *different forms*. The functions have the same appearance, the draw() expression, but different actual functions are called, depending on the contents of ptrarr[j]. Polymorphism is one of the key features of Object-Oriented Programming, after classes and inheritance.

For the polymorphic approach to work, several conditions must be met. First, all the different classes of shapes, such as balls and triangles, must be derived from a single base class (called shape in MULTSHAP). Second, the draw() function must be declared to be virtual in the base class. This is all rather abstract, so let‟s start with some short programs that show parts of the situation, and put everything together later.

Virtual Base Classes

Before leaving the subject of virtual programming elements, we should mention *virtual base classes as they relate to multiple inheritance*.

Consider the situation shown in Figure 11.3, with a base class, Parent; two derived classes, Child1 and Child2; and a fourth class, Grandchild, derived from both Child1 and Child2.

In this arrangement, a problem can arise if a member function in the Grandchild class wants to access data or functions in the Parent class.

A compiler error occurs when the getdata() member function in Grandchild attempts to access basedata in Parent. Why? When the Child1 and Child2 classes are derived from Parent, each inherits a copy of Parent; this copy is called a *subobject*. Each of the two subobjects contains its own copy of Parent‟s data, including basedata. Now, when Grandchild refers to basedata, which of the two copies will it access? The situation is ambiguous, and that‟s what the compiler reports.

PROGRAMS :-

a) Virtual Functions :-
// Program to implement concept of virtual function..

```
#include <iostream.h>
class convert
{
protected:
          double val1; // initial value
          double val2;  // converted value
public:
         convert(double i) {
         val1 = i;
  }

  double getconv() { return val2; }
  double getinit() { return val1; }
  virtual void compute() = 0;
};
```

// Liters to gallons.

```
class l_to_g : public convert {
public:
  l_to_g(double i) : convert(i) { }
  void compute() {
    val2 = val1 / 3.7854;
  }
};
```

```
// Fahrenheit to Celsius
class f_to_c : public convert {
public:
        f_to_c(double i) : convert(i) { }
        void compute() {
        val2 = (val1-42) / 1.8;
  }
};

int main()
{
  convert *p;  // pointer to base class
  l_to_g lgob(4);
  f_to_c fcob(70);

  // use virtual function mechanism to convert
  p = &lgob;
  cout << p->getinit() << " liters is ";
  p->compute();
  cout << p->getconv() << " gallons\n";  // l_to_g

  p = &fcob;
  cout << p->getinit() << " in Fahrenheit is ";
  p->compute();
  cout << p->getconv() << " Celsius\n"; // f_to_c
  return 0;
}
```

OUTPUT:

```
4 liters is 1.056691 gallons
70 in Fahrenheit is 15.555556 Celsius
```

Conclusion: - Thus we implemented the concept of Virtual function, Virtual class, Friend class and function.

Experiment No 11

Title: - Implementation of File handling

Aim: - **To study & implement the concept of File handling.**

Theory:-

In C++, you open a file by linking it to a stream. Before you can open a file, you must first obtain a stream. There are three types of streams: input, output, and input/output.To create an input stream, you must declare the stream to be of class **ifstream**. To create an output stream, you must declare it as class **ofstream**. Streams that will be performing both input and output operations must be declared as class **fstream**. For example, this fragment creates one input stream, one output stream, and one stream capable of both input and output:

ifstream in; // input

ofstream out; // output

fstream io; // input and output

Once you have created a stream, one way to associate it with a file is by using **open()**. This function is a member of each of the three stream classes. The prototype for each is shown here:

void ifstream::open(const char **filename*, ios::openmode *mode* = ios::in);

void ofstream::open(const char **filename*, ios::openmode *mode* = ios::out | ios::trunc);

void fstream::open(const char **filename*, ios::openmode *mode* = ios::in | ios::out

Reading and Writing Text Files:

It is very easy to read from or write to a text file. Simply use the << and >> operators the same way you do when performing console I/O, except that instead of using **cin** and **cout**, substitute a stream that is linked to a file. For example, this program creates a short inventory file that contains each item's name and its cost.

Characters vs. Bytes:

Before beginning our examination of unformatted I/O, it is important to clarify an important concept. For many years, I/O in C and C++ was thought of as *byte oriented*. This is because a **char** is equivalent to a byte and the only types of streams available were **char** streams. However, with the advent of wide characters (of type **wchar_t**) and their attendant streams, we can no longer say that C++ I/O is byte oriented. Instead, we must say that it is *character oriented*. Of course, **char** streams are still byte oriented and we can continue to think in terms of bytes, especially when operating on nontextual data. But the equivalency between a byte and a character can no longer be taken for granted. All of the streams used in this book are **char** streams since they are by far the most common. They also make unformatted file handling easier because a **char** stream establishes a one-to-one correspondence between bytes and characters, which is a benefit when reading or writing blocks of binary data.

put( ) and get( ):

One way that you may read and write unformatted data is by using the member functions **get()** and **put()** . These functions operate on characters. That is, **get()** will read a character and **put()** will write a character. Of course , if you have opened the file for binary operations and are operating on a **char** (rather than a **wchar_t** stream), then these functions read and write bytes of data. The **get()** function has many forms, but the most commonly used version is shown here along with **put()** :

istream &get(char &*ch*);

ostream &put(char *ch*);

Detecting End-of-File:

As we have seen, objects derived from ios contain error-status flags that can be checked to determine the results of operations. When we read a file little by little, as we do here, we will eventually encounter an end-of-file (EOF) condition. The EOF is a signal sent to the program from the operating system when there is no more data to read. In ILINE we could have checked for this in the line

while(!infile.eof()) // until eof encountered

However, checking specifically for an eofbit means that we won‟t detect the other error flags, such as the failbit and badbit, which may also occur, although more rarely. To do this, we can change our loop condition:

while(infile.good()) // until any error encountered

You can also test the stream directly. Any stream object, such as infile, has a value that can be tested for the usual error conditions, including EOF. If any such condition is true, the object returns a zero value. If everything is going well, the object returns a nonzero value. This value is actually a pointer, but the "address" returned has no significance except to be tested for a zero or nonzero value. Thus we can rewrite our while loop again:

while(infile) // until any error encountered

This is certainly simple, but it may not be quite so clear to the uninitiated what it does.

Closing Files:

So far in most of the programs there has been no need to close streams explicitly because they are closed automatically when they go out of scope; this invokes their destructors and closes the associated file. Since both the output stream os and the input stream is are associated with the same file, the first stream must be closed before the second is opened. We use the close() member function for this. You may want to use an explicit close() every time you close a file, without relying on the stream"s destructor. This is potentially more reliable, and certainly makes the listing more readable.

Mode Bits for open() Function :

Mode Bit	**Result**
in	Open for reading (default for ifstream)
out	Open for writing (default for ofstream)
ate	Start reading or writing at end of file (AT End)
app	Start writing at end of file (APPend)
trunc	Truncate file to zero length if it exists (TRUNCate)
nocreate	Error when opening if file does not already exist
noreplace	Error when opening for output if file already exists, unless ate or app is set
binary	Open file in binary (not text) mode

Programs:-

1) Writing Data:

// Program to write formatted output to a file, using <<

```
#include <fstream.h>              //for file O

#include <iostream.h>

#include <string.h>

int main()

  {         char ch = „x‟;

            int j = 77;
            double d = 6.02;
            string str1 = "Hello";        //strings without
            string str2 = "World";        //  embedded spaces

            ofstream outfile("fdata.txt"); //create ofstream object
            outfile << ch                 //insert (write) data
             << j       << „ „            //needs space between numbers
            << d        << str1
            << „ „                        //needs spaces between strings
            << str2;
  out << "File written\n";
  return 0;
  }
```

OUTPUT

```
File written
//File fdata.txt created
 x77 6.02Hello World
```

2) Reading Data:

// Program to read formatted output from a file, using >>

```
#include <fstream.h>             //for file O
#include <iostream.h>
#include <string.h>
int main()
  {
  char ch;
  int j;
  double d;

  string str1;

  string str2;

  ifstream infile("fdata.txt");  //create ifstream object

                      //extract (read) data from it

  infile >> ch >> j >> d >> str1 >> str2;

  cout << ch << endl              //display the data

    << j << endl

    << d << endl

    << str1 << endl

    << str2 << endl;

  return 0;

  }
```

OUTPUT

```
X
77
 6.02
Hello World
```

Conclusion: - Thus we implemented the concept of File Handling.

Experiment No 12

Title: - Implementation of Conversion of Polish expressions.

Aim: - To study & implement the concept of Polish expressions conversion.

Theory:-

Polish notation, also known as prefix notation, is a form of notation for logic, arithmetic, and algebra. Its distinguishing feature is that it places operators to the left of their operands. If the parity of the operators is fixed, the result is a syntax lacking parentheses or other brackets that can still be parsed without ambiguity. The Polish logician Jan Łukasiewicz invented this notation around 1920 in order to simplify sentential logic.Here is a quotation from Nicod's Axiom and Generalizing Deduction. I came upon the idea of a parenthesis-free notation in 1924. I used that notation for the first time in my article Łukasiewicz .The reference cited by Jan Łukasiewicz above is apparently a lithographed report in Polish.

Alonzo Church mentions this notation in his classic book on Mathematical logic as worthy of remark in notational systems even contrasted to Whitehead and Russell's logical notational exposition and work in Principia Mathematic.

While no longer used much in logic, Polish notation has since found a place in computer science. The Arithmetic expression for adding the numbers one and two is, in prefix notation, written "+ 1 2" rather than "1 + 2". In more complex expressions, the operators still precede their operands, but the operands may themselves be nontrivial expressions including operators of their own. For instance, the expression that would be written in conventional infix notation as

(5 − 6) * 7

can be written in prefix as

*(− 5 6) 7

or simply

* − 5 6 7

Since the simple arithmetic operators are all binary (at least, in arithmetic contexts), any prefix representation thereof is unambiguous, and bracketing the prefix expression is unnecessary. In the previous example, the parentheses in the infix version were required, since moving them

5 − (6 * 7)

or simply removing them

5 − 6 * 7

would change the meaning and result of the overall expression. However, the corresponding prefix version of this second calculation would be written

− 5 * 6 7

The processing of the subtraction is deferred until both operands of the subtraction have been read in (i.e., 5 and the product of 6 and 7). As with any notation, the innermost expressions are evaluated first, but in prefix notation this "innermost-ness" can be conveyed by order rather than bracketing.

Prefix notation of simple arithmetic is largely of academic interest. Unlike the similar postfix reverse Polish notation, prefix notation has been used in few if any commercially-made calculators. However, prefix-notated arithmetic is frequently used as a first, conceptual step in the teaching of compiler construction.

Computer programming

Prefix notation has seen wide application in Lisp s-expressions, where the brackets are required due to the arithmetic operators having variable parity. The postfix reverse Polish notation is used in many stack-based programming languages like PostScript, and is the operating principle of certain calculators, notably from Hewlett-Packard.

Although obvious, it is important to note that the number of operands in an expression must equal the number of operators plus one, otherwise the statement makes no sense (assuming only binary operators are used in the expression). This can be easy to overlook when dealing with longer, more complicated expressions with several operators, so care must be taken to double check that an expression makes sense when using prefix notation.

Order of operations

Order of operations is defined within the structure of prefix notation and can be easily determined. One thing to keep in mind is that when executing an operation, the operation is applied TO the first operand BY the second operand. This is not an issue with operations that commute, but for non-commutative operations like division or subtraction, this fact is crucial to the analysis of a statement. For example, the following statement:

/ 10 5 = 2 (Prefix)

is read as "Divide 10 BY 5". Thus the solution is 2, not ½ as would be the result of an incorrect analysis.

Prefix notation is especially popular with stack-based operations due to its innate ability to easily distinguish order of operations without the need for parentheses. To evaluate order of operations under prefix notation, one does not even need to memorize an operational hierarchy, as with infix notation. Instead, one looks directly to the notation to discover which operator to evaluate first. Reading an expression from left to right, one first looks for an operator and proceeds to look for two operands. If another operator is found before two operands are found, then the old operator is placed aside until this new operator is resolved. This process iterates until an operator is resolved, which must happen eventually, as there must be one more operand than there are operators in a complete statement. Once resolved, the operator and the two operands are replaced with a new operand. Because one operator and two operands are removed and one operand is added, there is a net loss of one operator and one operand, which still leaves an expression with N operators and N+1 operands, thus allowing the iterative process to continue. This is the general theory behind using stacks in programming languages to evaluate a statement in prefix notation, although there are various algorithms that manipulate the process. Once analyzed, a statement in prefix notation becomes less intimidating to the human mind as it allows some separation from convention with added convenience.

Program:-

/*Program to implement conversion of Polynomial Expression*/

```
 #include<iostream>
using namespace std;
struct node
{       int coe,exp;
          struct node *next;
};
class polynomial
{         struct node *start,*n,*p;
          public:
          void get_poly();
          void show();
          void add(polynomial p1,polynomial p2);
};
void polynomial::get_poly()
{         char c='y';
          start=NULL;
          while(c=='y'||c=='Y')
```

```
        {       n=new node;
                cout<<"Enter The Coefficient :";
                cin>>n->coe;
                cout<<"Enter the Exponent :";
                cin>>n->exp;
                n->next=NULL;
                p->next=n;
                if(start==NULL)
                {       start=n;
                }
                p=n;
                cout<<"Enter y To add more Nodes";
                cin>>c;
        }}
void polynomial::show()
{       struct node *ptr;
        ptr=start;
        while(ptr!=NULL)
        {       cout<<ptr->coe<<"X^"<<ptr->exp<<"";
                ptr=ptr->next;
                if(ptr!=NULL)
                        cout<<"+";
        }       cout<<"\n";
}void polynomial::add(polynomial p1,polynomial p2)
{       struct node *p1ptr,*p2ptr;
        int coe,exp;
```

```
start=NULL;

p1ptr=p1.start;

p2ptr=p2.start;

while(p1ptr!=NULL && p2ptr!=NULL)

{  if(p1ptr->exp==p2ptr->exp)

        {       coe=p1ptr->coe+p2ptr->coe;

                exp=p1ptr->exp;

                p1ptr=p1ptr->next;

                p2ptr=p2ptr->next;

        }

        else if(p1ptr->exp>p2ptr->exp)

        {       coe=p1ptr->coe;

                exp=p1ptr->exp;

                p1ptr=p1ptr->next;

        }else if(p1ptr->exp<p2ptr->exp)

        {       coe=p2ptr->coe;

                exp=p2ptr->exp;

                p2ptr=p2ptr->next;

        }n=new node;

        if(start==NULL)

                start=n;

        n->coe=coe;

        n->exp=exp;

        n->next=NULL;

        p->next=n;

        p=n;    }
```

```
if(p1ptr==NULL)
{       while(p2ptr!=NULL)
        {       coe=p2ptr->coe;
                exp=p2ptr->exp;
                p2ptr=p2ptr->next;
                n=new node;
                if(start==NULL)
                        start=n;
                n->coe=coe;
                n->exp=exp;
                n->next=NULL;
                p->next=n;
                p=n;
        }
}else if(p2ptr==NULL)
{       while(p1ptr!=NULL)
        {       coe=p1ptr->coe;
                exp=p1ptr->exp;
                p1ptr=p1ptr->next;
                n=new node;
                if(start==NULL)
                        start=n;
                n->coe=coe;
                n->exp=exp;
                n->next=NULL;
                p->next=n;
```

```
                    p=n;
                }}}
int main()
{       int ch;
        char l;
        polynomial p1,p2,sum;
        cout<<"\n First Polynomial";
        p1.get_poly();
        cout<<"\n Second Polynomial";
        p2.get_poly();
        cout<<"\n First Polynomial is :";
        p1.show();
        cout<<"\n Second Polynomial is :";
        p2.show();
        do
        {       cout<<"\n 1.Addition";
                cout<<"\n  2.Exit";
                cout<<"\n Enter the Choice";
                cin>>ch;
                switch(ch)
                {               case 1:
                                cout<<"\n The sum of two polynomial is: ";
                                sum.add(p1,p2);
                                sum.show();
                                break;
                        case 2:
                            cout<<"\n Exit";
                            break;
                }
                cout<<"\n Do you want continue ";
                cin>>l;
        }while(l=='y'||l=='Y');
   return 0;
}
```

OUTPUT

```
 First Polynomial
 Enter The Coefficient: 2
 Enter the Exponent: 4
 Enter y to add more Nodes n
 Second Polynomial
 Enter the Coefficient: 4
 Enter the Exponent: 2
 Enter y to add more Nodes n
 First Polynomial is: 2X^4
 Second Polynomial is: 4X^2
 1. Addition
 2. Exit
 Enter the Choice1
 The sum of two polynomial is: 2X^4+4X^2
  Do you want continue n
```

Conclusion: - Thus we implemented the concept of Conversion of Polish Expression using C++.

Experiment No 13

Title: - Implementation of sorting algorithms is using function template and virtual function.

Aim:- To study & implement the concept of sorting algorithms using function template and virtual function.

Theory:-

Function Templates:

Many functions have the same code body, regardless of type; for example, initializing the contents of one array from another of the same type uses the same code body. The essential code is

```
for (i = 0; i < n; ++i)

a[i] = b[i];
```

Many programmers automate this with a simple macro:

Now that‟s what I call a generic waiter - he can balance anything!

```
#define COPY(A, B, N) \

{ int i; for (i=0; i < (N); ++i) (A)[i] = (B)[i]; }
```

Programming that works regardless of type is a form of generic programming. The use of define macros is a form of generic programming. Its advantages are several, including simplicity, familiarity, and efficiency. There is familiarity because of a long tradition in C programming of using such macros. It is very efficient. There is no function call overhead.

The disadvantages of using macros include type-safety, unanticipated evaluations and

Scoping problems. Using define macros can often work, but doing so is not type-safe. Macro substitution is a preprocessor textual substitution that is not syntactically checked until later. Another problem with define macros is that they can lead to repeated evaluation of a single parameter. Definitions of macros are tied to their position in a file and not to the C++ language rules for scope.

The code

```
#define CUBE(X) ((X)*(X)*(X))
```

behaves differently from the code

```
template<class T> T cube (T x) { return x * x * x;}
```

When cube(sqrt(7)) is invoked, the function sqrt(7) is called once, not three times as with the CUBE define macro. Templates are safer when types can be mixed in an expression and conversions are inappropriate

Future Directions:

C++ templates evolved considerably from their initial design in 1988 until the standardization of C++ in 1998 (the technical work was completed in November 1997). After that, the language definition was stable for several years, but during that time various new needs have arisen in the area of C++ templates. Some of these needs are simply a consequence of a desire for more consistency or orthogonality in the language. For example, why wouldn't default template arguments be allowed on function templates when they are allowed on class templates? Other extensions are prompted by increasingly sophisticated template programming idioms that often stretch the abilities of existing compilers. In what follows we describe some extensions that have come up more than once among C++ language and compiler designers. Often such extensions were prompted by the designers of various advanced C++ libraries (including the C++ standard library). There is no guarantee that any of these will ever be part of standard C++. On the other hand, some of these are already provided as extensions by certain C++ implementations.

Program:-

// Program to implement generic bubble sort using concept of template .

```
#include <iostream.h>

template <class X>

void bubble

    (  X *items,  // pointer to array to be sorted

     int count) // number of items in array

    {    register int a, b;

        X t;

  for(a=1; a<count; a++)

     for(b=count-1; b>=a; b--)

    if(items[b-1] > items[b]) {

    // exchange elements

          t = items[b-1];

    items[b-1] = items[b];
```

```
      items[b] = t;
        }
}
int main()
{  int iarray[7] = {7, 5, 4, 3, 9, 8, 6};
  double darray[5] = {4.3, 2.5, 1.9, 100.2, 3.0};
  int i;
  cout << "Here is unsorted integer array: ";
  for(i=0; i<7; i++)
  cout << iarray[i] << ' ';
  cout << endl;
   cout << "Here is unsorted double array: ";
  for(i=0; i<5; i++)
  cout << darray[i] << ' ';
  cout << endl;
  bubble(iarray, 7);
  bubble(darray, 5);
  cout << "Here is sorted integer array: ";
  for(i=0; i<7; i++)
  cout << iarray[i] << ' ';
  cout << endl;
  cout << "Here is sorted double array: ";
  for(i=0; i<5; i++)
  cout << darray[i] << ' ';
  cout << endl;
  return 0;}
```

OUTPUT

```
Here is unsorted integer array: 7, 5, 4, 3, 9, 8, 6

Here is unsorted double array: 4.3, 2.5, 1.9, 100.2, 3.0

Here is sorted integer array: 3,4,5,6,7,8,9

Here is sorted double array: 1.9, 2.5, 3.0, 4.3, 100.2
```

Conclusion: - Thus we implemented the concept of sorting algorithms using function template and virtual function.

Experiment No 14

Title: - Implementation of Linear Search and Binary Search using concept of template.

Aim:- To study & implement the concept of Linear Search and Binary Search using function template.

Theory:-
Search using Function Templates

linear search or sequential search is a method for finding a particular value in a list that checks each element in sequence until the desired element is found or the list is exhausted.The list need not be ordered.

Linear search is the simplest search algorithm; it is a special case of brute-force search. Its worst case cost is proportional to the number of elements in the list. Its expected cost is also proportional to the number of elements if all elements are searched equally. If the list has more than a few elements and is searched often, then more complicated search methods such as binary search or hashing may be appropriate. Those methods have faster search times but require additional resources to attain that speed.

a binary search or half-interval search algorithm finds the position of a specified input value (the search "key") within an array sorted by key value.[1][2] For binary search, the array should be arranged in ascending or descending order. In each step, the algorithm compares the search key value with the key value of the middle element of the array. If the keys match, then a matching element has been found and its index, or position, is returned. Otherwise, if the search key is less than the middle element's key, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the search key is greater, on the sub-array to the right. If the remaining array to be searched is empty, then the key cannot be found in the array and a special "not found" indication is returned.

A binary search halves the number of items to check with each iteration, so locating an item (or determining its absence) takes logarithmic time. A binary search is a dichotomic divide and conquer search algorithm.

Program:-

// Program to implement Linear Search and Binary Search using function template

```
#include<iostream>

template<class T>

void Lsearch(T *a, T item, int n)
```

```
{           int z=0;
            for (inti=0;i<n;i++)
 {       if(a[i]== item)
            {           z=1;
                        cout<<"\n Item found at position = "<<i+1<<"\n\n";
            }else
            if(z!=1)
            {           z=0;
            }
 }if(z==0)
            cout<<"\n Item not found in the list\n\n";0
}
            Template <class T>
            Void Bsearch(T *a, T item, int n)
            {           int beg=0,end=n-1;
                        int mid=beg + end/2;
                        while ((a[mid]!=item) && (n>0))
                         {          if(item>a[mid])
                                    beg=mid;
                                    else
                                    end=mid;
                                    mid=(beg + end)/2;
                                    n--;
                         }
            If ( a[mid] == item)
            cout<<"\n Item found at position = "<<mid+1<<"\n\n";
```

```
else
cout<<"\n Item not found in the list\n\n";
}void main()
{         intiarr[10] = {2,42,56,86,87,99,323,546,767,886};
          doubledarr[6]= {2.4, 5.53,44.4, 54.45, 65.7,89.54};
          intiele;
          double dele;
          cout<<"\n Elements of Integer Array \n";
          for(inti=0;i<10;i++)
           {  cout<<" "<<iarr[i]<<" ,";
           }
          cout<<"\n\n Enter an item to be search: ";
          cin>>iele;
          cout<<"\n\n Linear Search Method\n";
          Lsearch(iarr,iele,10);
          cout<<"\n\n Binary Search method\n";
          Bsearch(iarr,iele,10);
          cout<<"\n Elements of double Array \n";
          for(inti=0;i<6;i++)
           {  cout<<" "<<darr[i]<<" ,";
           }
          cout<<"\n\n Enter an item to be search: ";
          cin>>dele;
          cout<<"\n\n Linear Search Method\n";
          Lsearch(darr,dele,6);
          cout<<"\n\n Binary Search method\n";
```

```
        Bsearch(darr,dele,6);

        system("pause");

    }
```

OUTPUT

```
Elements of Integer Array
 2 , 42 , 56 , 86 , 87 , 99 , 323 , 546 , 767 , 886 ,
 Enter an item to be search: 323
 Linear Search Method
 Item found at position = 7
 Binary Search method
 Item found at position = 7

 Elements of double Array
 2.4 , 5.53 , 44.4 , 54.45 , 65.7 , 89.54 ,
 Enter an item to be search: 5.53
 Linear Search Method
 Item found at position = 2
 Binary Search method
 Item found at position = 2
Press any key to continue . . .
```

Conclusion: - Thus we implemented the concept of Linear Search and Binary Search using function template.

Experiment No 15

Title: - Implementation of stack using class template

Aim: - To study & Implementation of stack using class template
Theory:-

Class Templates:

The template concept can be extended to classes. Class templates are generally used for data storage (container) classes. However, the examples of these classes that we presented could store data of only a single basic type. The Stack class in the STAKARAY program "Arrays and Strings," for example, could store data only of type int. Here‟s a condensed version of that class.

```
class Stack
  {
  private:
    int st[MAX];         //array of ints
    int top;             //index number of top of stack
  public:
    Stack();             //constructor
    void push(int var);  //takes int as argument
    int pop();           //returns int value
  };
```

A template argument, named Type in this example, is then used (instead of a fixed data type like int) every place in the class specification where there is a reference to the type of the array st. There are three such places: the definition of st, the argument type of the push () function, and the return type of the pop() function. Class templates differ from function templates in the way they are instantiated. To create an actual function from a function template, you call it using arguments of a specific type. Classes, however, are instantiated by defining an object using the template argument.

```
Stack<float> s1;
```

This creates an object, s1, a stack that stores numbers of type float. The compiler provides space in memory for this object‟s data, using type float wherever the template argument Type appears in the class specification. It also provides space for the member functions (if these have not already been placed in memory by another object of type Stack<float>).

Class Name Depends on Context

In the TEMPSTAK example, the member functions of the class template were all defined within the class. If the member functions are defined externally (outside of the class specification), we need a new syntax. The next program shows how this works.

Program:-

// Program to implement generic stack using template.

```
#include <iostream.h>
const int SIZE = 10;
// Create a generic stack class
        template <class StackType>
        class stack
        {     StackType stck[SIZE]; // holds the stack
              int tos; // index of top-of-stack
              public:
              stack()
             {      tos = 0;
              } // initialize stack
           void push(StackType ob); // push object on stack
         StackType pop(); // pop object from stack
        };
// Push an object.
template <class StackType>
void stack<StackType>::push(StackType ob)
{  if(tos==SIZE)
 {    cout << "Stack is full.\n";
    return;
  } stck[tos] = ob;
  tos++;
}// Pop an object.
template <class StackType>
```

```
StackType stack<StackType>::pop()
{    if(tos==0)
  {    cout << "Stack is empty.\n";
    return 0; // return null on empty stack
   }  tos--;
  return stck[tos];
}
int main()
{  // Demonstrate character stacks.
  stack<char> s1, s2; // create two character stacks
  int i;
  s1.push('a');
  s2.push('x');
  s1.push('b');
  s2.push('y');
  s1.push('c');
  s2.push('z');
  for(i=0; i<3; i++) cout << "Pop s1: " << s1.pop() << "\n";
  for(i=0; i<3; i++) cout << "Pop s2: " << s2.pop() << "\n";
  // demonstrate double stacks
  stack<double> ds1, ds2; // create two double stacks
  ds1.push(1.1);
  ds2.push(2.2);
  ds1.push(3.3);
  ds2.push(4.4);
  ds1.push(5.5);
  ds2.push(6.6);

  for(i=0; i<3; i++) cout << "Pop ds1: " << ds1.pop() << "\n";

  for(i=0; i<3; i++) cout << "Pop ds2: " << ds2.pop() << "\n";

  return 0;

}
```

OUTPUT

```
Pop s1: c

Pop s1: b

Pop s1: a

Pop s2: z

Pop s2: y

Pop s2: x

Pop ds1:5.5

Pop ds1:3.3

Pop ds1:1.1

Pop ds2:6.6

Pop ds2:4.4

Pop ds2:2.2
```

Conclusion: - Thus we implemented the concept of stack using class template.

Experiment No 16

Title: - Implementation of queue using class template

Aim: - To study & Implementation of queue using class template
Theory:-

Queue using Class Template

Queue is a particular kind of abstract data type or collection in which the entities in the collection are kept in order and the principal (or only) operations on the collection are the addition of entities to the rear terminal position, known as enqueue, and removal of entities from the front terminal position, known as dequeue. This makes the queue a First-In-First-Out (FIFO) data structure. In a FIFO data structure, the first element added to the queue will be the first one to be removed. This is equivalent to the requirement that once a new element is added, all elements that were added before have to be removed before the new element can be removed. Often a peek or front operation is also entered, returning the value of the front element without dequeuing it. A queue is an example of a linear data structure, or more abstractly a sequential collection.

Templates are the foundation of generic programming, which involves writing code in a way that is independent of any particular type.

A template is a blueprint or formula for creating a generic class or a function. The library containers like iterators and algorithms are examples of generic programming and have been developed using template concept.

There is a single definition of each container, such as vector, but we can define many different kinds of vectors for example, vector <int> or vector <string>.

You can use templates to define functions as well as classes, let us see how do they work:

Function Template:

The general form of a template function definition is shown here:

```
template<class type> ret-type func-name(parameter list)

{

  // body of function

}
```

Here, type is a placeholder name for a data type used by the function. This name can be used within the function definition.

Program:-

// Program to implement queue using class template.

```
#include<iostream>
#include<stdlib.h>
#include<process.h>
template<class T>
class Queue
{      private:
             intfront,rear;
             T *queue;
             intmaxsize;


       public:
             Queue(intmaxqueuesize)
             {
                    front=0;
                    rear=-1;
                    maxsize=maxqueuesize;
                    queue=new T[maxsize];
             }
             ~Queue()
             {
                   delete[] queue;
             }
             intisempty();
             intisfull();
             void insert();
             void deletion();
             void atfront();
             void atrear();
             void display();
};
template<class T>
int Queue<T>::isempty()
{
   if(front==0&&rear==-1||front==rear)
   return 1;
   else
   return 0;
```

```
}
template<class T>
int Queue<T>::isfull()
{
   if(rear==maxsize-1)
   return 1;
   else
   return 0;
}
template<class T>
void Queue<T>::atfront()
{
    if(isempty())
    cout<<"\nSorry the queue is empty!";
    else
    cout<<"\nFront element of the queue is : "<<queue[front];
}
template<class T>
void Queue<T>::atrear()
{
    if(isempty())
    cout<<"\nSorry the queue is empty!";
    else
    cout<<"\nRear element of the queue is : "<<queue[rear];
}
template<class T>
void Queue<T>::insert()
{
    T ele;
    if(isfull())
    cout<<"\nSorry the queue is full!";
    else
    {
    cout<<"\nEnter the element to insert : ";
    cin>>ele;
    queue[++rear]=ele;
    }
}
template<class T>
void Queue<T>::deletion()
{
    if(isempty())
```

```
     cout<<"\nSorry the queue is empty!";
     else
     cout<<"\nDeleted element of the queue is : "<<queue[front++];
}
template<class T>
void Queue<T>::display()
{
     if(isempty())
     cout<<"\nSorry the queue is empty!";
     else
     {
         cout<<"\nQueue elements are : ";
         for(inti=front;i<=rear;i++)
         {
                 cout<<"\t"<<queue[i];
         }
     }
}
int main()
{
    int ch;
    Queue <int> q(10);
    do
    {
        cout<<"\n 1.Insertion \n 2.Deletion \n 3.Display Front Element \n

         4. Display Rear Element      \n 5.Display Queue \n 6.Exit \n";
            cout<<"Enter your Choice:";
            cin>>ch;
            switch(ch)
            {        case 1:
                          q.insert();
                          break;
                     case 2:
                          q.deletion();
                          break;
                     case 3:
                          q.atfront();
                          break;
                     case 4:
                          q.atrear();
                          break;
```

```
                case 5:
                    q.display();
                    break;
                case 6:
                    exit(0);
                    break;
                default: cout<<"\nWrong Choice Entered!";

        }       }while(ch<=6);
            system("pause");
        return 0;

}
```

OUTPUT

```
1.Insertion

2.Deletion

3.Display Front Element

4.Display  Rear  Element

5.Display Queue

6.Exit

Enter your Choice:1

Enter the element to insert : 10

1.Insertion

2.Deletion

3.Display Front Element

4.Display  Rear  Element

5.Display Queue

6.Exit

Enter your Choice:1

Enter the element to insert : 20

1.Insertion

2.Deletion
```

```
 3.Display Front Element
 4.Display  Rear  Element
 5.Display Queue
 6.Exit
Enter your Choice:1
Enter the element to insert : 30
 1.Insertion
 2.Deletion
 3.Display Front Element
 4.Display  Rear  Element
 5.Display Queue
 6.Exit
Enter your Choice:1
Enter the element to insert : 40
 1.Insertion
 2.Deletion
 3.Display Front Element
 4.Display  Rear  Element
 5.Display Queue
 6.Exit
Enter your Choice:1
Enter the element to insert : 50
 1.Insertion
 2.Deletion
 3.Display Front Element
 4.Display Rear Element
 5.Display Queue

 6.Exit

Enter your Choice:5

Queue elements are :    10      20      30      40      50

 1.Insertion

 2. Deletion

 3. Display Front Element

 4.Display  Rear  Element

 5.Display Queue

 6.Exit

Enter your Choice:4

Rear element of the queue is : 50

 1.Insertion

 2. Deletion

 3. Display Front Element

 4.Display  Rear  Element

 5.Display Queue

 6.Exit

Enter your Choice:5

Queue elements are :    10      20      30      40      50

 1.Insertion

 2. Deletion

 3. Display Front Element

 4.Display  Rear  Element

 5.Display Queue

 6.Exit
```

```
Enter your Choice:2

Deleted element of the queue is : 10

 1.Insertion

 2.Deletion

 3.Display Front Element

 4.Display  Rear  Element

 5.Display Queue

 6.Exit

Enter your Choice:5

Queue elements are :    20      30      40      50

 1.Insertion

 2.Deletion

 3.Display Front Element

 4.Display  Rear  Element

 5.Display Queue

 6.Exit

Enter your Choice: 6
```

Conclusion: - Thus we implemented the concept of queue using class template.

Experiment No 17

Title: - Implementation of Exception Handling.

Aim: - To study & Implementation of Exception Handling.
Theory:

Exception handling:

Exception handling is a mechanism that separates code that detects and handles exceptional circumstances from the rest of your program. Note that an exceptional circumstance is not necessarily an error.

When a function detects an exceptional situation, you represent this with an object. This object is called an *exception object*. In order to deal with the exceptional situation you *throw the exception*. This passes control, as well as the exception, to a designated block of code in a direct or indirect caller of the function that threw the exception. This block of code is called a *handler*. In a handler, you specify the types of exceptions that it may process. The C++ run time, together with the generated code, will pass control to the first appropriate handler that is able to process the exception thrown. When this happens, an exception is *caught*. A handler may *rethrow* an exception so it can be caught by another handler.

The exception handling mechanism is made up of the following elements:

- try blocks
- catch blocks
- throw expressions

try blocks

You use a *try block* to indicate which areas in your program that might throw exceptions you want to handle immediately. You use a *function try block* to indicate that you want to detect exceptions in the entire body of a function.

try block syntax

```
                 .-----------.
                   V         |
>>-try--{--statements--}---handler-+-----------------------><
```

Function try block syntax

```
>>-try--+---------------------------+--function_body--- handler-+-><

       '--- member_initializer_list-'
```

catch blocks :

catch block syntax

>>-catch--(--*exception_declaration*--)--{--*statements*--} -------><

You can declare a handler to catch many types of exceptions. The allowable objects that a function can catch are declared in the parentheses following the catch keyword (the *exception_declaration*). You can catch objects

Program:

/*Program to implement concept of Exception Handling */

```
#include<iostream>

using namespace std;

int main()
{
  int a,b,x;
  cout<<"\n\t Enter values of a&b";
  cin>>a;
  cin>>b;
  x=a-b;

 try  {
            if(x!=0)
      {
            cout<<"\n\t Result (a/x)="<<a/x<<"\n";
      }
      else
      {
             throw(x);
      }
  }
catch(int i)
  {
            cout<<"\n\t Exception caught: x="<<x<<"\n";
  }
    cout<<"\n\t END";
  return 0;
}
```

OUTPUT

```
Enter Values of a & b
20 15
Result(a/x)=4
END

Enter Values of a & b
10 10
Exception caught:x=0
END
```

Conclusion:

Thus we have implement concept of Exception handling by using different clauses.

Experiment No 18

Title: - Implementation of different programs by using C++ libraries for graphics purpose.

Aim: - To study & implement different programs by using C++ libraries for graphics purpose .

Theory:

In C++ graph unit provides a complete set of library consisting of over 80 graphics routines that range from bit oriented routine to high level calls. In addition to the graphical constants and types. In order to C++ graphics routines the following hardware components should be installed in PC system.

- A graphics display controller
- A color monitor or any other display device.

For graph initialization and closing routines we use procedure and functions are given below,

Cleardevice:-This method is used to clear the graphics screen.It is also used to clear currently selected output device.

Syntax:-

```
void cleardevice(void);
```

Closegraph:-This method is used to shut down the graphics system.

Syntax:-

```
void closegraph(void);
```

Detectgraph:-This method is used to check the hardware and determine which graph driver and mode to use.

Syntax:-

```
void  detectgraph(void);
```

Getdrivername:-This method is used to return a string containing the name of the current driver.

Syntax:-

```
char getdrivername(void);
```

Getgraphmode:- This method is used to return current graphics mode.

Syntax:-

```
int getgraphmode(void);
```

Initgraph:- This method is used to initalize the graphics system and put the hardware into graphics mode.

Syntax:-

```
void far initgraph(int graphdriver, int graphmode,char  pathtodriver);
```

Line and Polygon selection routines :

There are different procedure and functions that are used to perform line and polygine selection.

Line:- This method is used to draw a line from the point (x1,y1) to (x2,y2).

Syntax:-

```
void  line(int x1, int y1, int x2, int y2);
```

Circle:- This method is used to draw a circle in the current color set by setcolor, using (x,y) as the center point.

Syntax:-

```
void  circle(int x,int y,int radius );
```

Drawpoly: - This method is used to draw the outline of a polygone using the current line style and color.

Syntax:-

```
void  drawpoly(int numpoint, int polypoint);
```

Ellipse: - This method is used to draw an ellipse arc using current color.

Syntax:-

```
void  ellipse(int x,int y,int stangle, int endangle, int xradius, int yradius);
```

Rectangle: - This method is used to draw a rectangle using line style ,thickness and drawing color.

Syntax:-

```
void rectangle(int left, int top, int right, int bottom);
```

Color selection routines: Following are the color constants used in C++,

Colors Name	Color Number
Black	0
Blue	1
Green	2
Cyan	3
Red	4
Magenta	5
Brown	6
Light gray	7

There are different procedure and functions that are used to perform the color selection operation in C++.

Setcolor: - This method is used to set current drawing color.

Syntax:- Void setcolor(int color);

Setbkcolor: - This method is used to set the current background color.

Syntax:-

Void setbkcolor(int color);

getcolor:- This method is used to return current drawing color.

Syntax:- int setcolor(void);

getbkcolor:- This method is used to return the current background color.

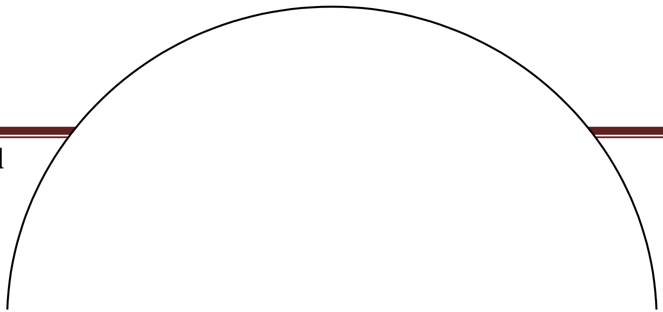
Syntax:- int getbkcolor(void);

PROGRAM:

//Program to draw Circle using graphics libraries.

```
#include <graphics.h>

#include <stdlib.h>

#include <iostream.h>

#include <conio.h>

const int radius = 100;
```

```
class sample
 {  private:
  int midx,midy;
  public:
  void getdata();
  void draw_circle();
};
void sample::getdata()
{midx = getmaxx()/2;
            midy = getmaxy()/2;
}
void sample::draw_circle()
{circle(midx,midy,radius);
}
void main(void)
{         sample obj;
            int gdriver =DETECT,gmode;//auto detection
            initgraph(&gdriver,&gmode,"\\tc\\bgi");
            obj.getdata();
            obj.draw_circle();
            getch();
            cleardevice();//clears a graphics screen
            closegraph();
 }
```



OUTPUT

Conclusion: - Thus we implemented program which draws circle using the methods of C++ graphics libraries.

Program for Exercise

1 Program to find sum of two numbers.
2 Program to find area and circumference of circle.
3 Program to find the simple interest.
4 Program to convert temperature from degree centigrade to Fahrenheit.
5 Program to calculate sum of 5 subjects & find percentage.
6 Program to show swap of two no‟s without using third variable.
7 Program to reverse a given number.
8 Program to find gross salary.
9 Program to print a table of any number.
10 Program to find greatest in 3 numbers.
11 Program to show the use of conditional operator.
12 Program to find that entered year is leap year or not.
13 Program to find whether given no is even or odd.
14 Program to shift inputed data by two bits to the left.
15 Program to use switch statement. Display Monday to Sunday.
16 Program to display arithmetic operator using switch case.
17 Program to display first 10 natural no & their sum.
18 Program to print stars Sequence1.
19 Program to print stars Sequence2.
20 Program to print star Sequences3.
21 Program to print Fibonacci series up to 100.
22 Program to find factorial of a number.
23 Program to find whether given no is a prime no or not.
24 Program to display sum of series 1+1/2+1/3+…........ +1/n.
25 Program to display series and find sum of 1+3+5+ .... +n.
26 Program to use bitwise AND operator between the two integers.
27 Program to add two number using pointer.
28 Program to show sum of 10 elements of array & show the average.
29 Program to find the maximum no in an array.
30 Program to display matrix.
31 Program to find sum of two matrices.
32 Program to find subtraction of two matrices.
33 Program to find multiplication of two matrices.
34 Program to find transpose of a matrix.
35 Program to find the maximum number in array using pointer.
36 Program to reverse a number using pointer.
37 Program to show input and output of a string.
38 Program to find square of a number using functions.
39 Program to swap two numbers using functions.
40 Program to find factorial of a number using functions.
41 Program to show table of a number using functions.
42 Program to show call by value.
43 Program to show call by reference.
44 Program to find largest of two numbers using functions.
45 Program to find factorial of a number using recursion.
46 Program to find whether a string is palindrome or not.

Program for Exercise

1. Write a program to find out the gross amount from the given basic pay.
   Gross = Basic + DA + HRA
   DA & HRA can be calculated as follows: -
   If Basic Pay is greater than or equal to 8000 DA is 20% of Basic Pay & HRA is 25% of Basic Pay, otherwise DA is 15% of Basic Pay & HRA is 20% of Basic Pay.
   Using three functions input, calculate and display.

2. Raising a number n to a power p is the same as multiplying n by itself p times. Write a function called power ( ) that takes a double value for n and an int value for p, and returns the result as double value. Use a default argument of 2 for p, so that if this argument is omitted, the number will be squared. Write a main ( ) function that gets values from the user to test this function.

3. Write a program to print a triangles like the followings using function overloading: -

```
(a) 1
        1 2
        1 2 3
        1 2 3 4
        1 2 3 4 5
(b) 1
        2 3
        4 5 6
        7 8 9 10
        11 12 13 14
(c)         1
        1 1 1
        1 1 1 1 1
        1 1 1 1 1 1 1
        1 1 1 1 1 1 1 1 1
(d) 1
        1 2 1
        1 2 3 2 1
        1 2 3 4 3 2 1
        1 2 3 4 5 4 3 2 1
```

4. A point on the two dimensional plane can be represented by two numbers: an X coordinate and a Y coordinate. For example, (4,5) represents a point 4 units to the right of the origin along the X axis and 5 units up the Y axis. The sum of two points can be defined as a new point whose X coordinate is the sum of the X coordinates of the points and whose Y coordinate is the sum of their Y coordinates.
   Write a program that uses a structure called point to model a point. Define three points, and have the user input values to two of them. Than set the third point equal to the sum of the other two, and display the value of the new point. Interaction with the program might look like this:

```
Enter coordinates for P1: 3 4
Enter coordinates for P2: 5 7
Coordinates of P1 + P2 are: 8, 11
```

5. Write a program for Multiplication of the two matrixes and store it in another matrix. Display the new matrix using three functions one for input, second for multiplication with array as argument and display for output.

6. Create the equivalent of a four function calculator. The program should request the user to enter a number, an operator, and another number. It should then carry out the specified arithmetical operation: adding, subtracting, multiplying, or dividing the two numbers. (It should use a switch statement to select the operation). Finally it should display the result. When it finishes the calculation, the program should ask if the user wants to do another calculation. The response can be „Y‟ or „N‟. Some sample interaction with the program might look like this.

```
Enter first number, operator, second number: 10/ 3

Answer = 3.333333

Do another (Y/ N)? Y

Enter first number, operator, second number 12 + 100

Answer = 112

Do another (Y/ N)? N
```

7. Write a program to calculate the income tax of „n‟ number of employees using class as structure. Use a separate inline function to calculate the tax. Rate of tax is 20% of basic salary if it is less than 9000 otherwise tax is 25% of basic salary.

8. A phone number, such as (91) 120-4370000, can be thought of as having three parts: the country code (91), city code (120) and the number (4370000). Write a program that uses a structure to store these three parts of a phone number separately. Call the structure phone. Create two structure variables of type phone. Initialize one, and have the user input a number for the other one. Then display both numbers. The interchange might look like this:

```
Enter your area code, exchange, and number: (91) 124-4513999

My number is (91) 120-4370000

Your number is (91) 124-4513999
```

9. Create two classes DM and DB which store the value of distances. DM stores distances in metres and centimeters and DB in feet and inches. Write a program that can read values for the class objects and add one object of DM with another object of DB.

   Use a friend function to carry out the addition operation. The object that stores the results maybe a DM object or DB object, depending on the units in which the results are required. The display should be in the format of feet and inches or metres and cenitmetres depending on the object on display.

10. Create a class rational which represents a numerical value by two double values- NUMERATOR & DENOMINATOR. Include the following public member Functions:
    • constructor with no arguments (default).
    • constructor with two arguments.
    • void reduce( ) that reduces the rational number by eliminating the highest common factor between the numerator and denominator.
    • Overload + operator to add two rational number.

    Overload >> operator to enable input through cin.

    • Overload << operator to enable output through cout.

    Write a main ( ) to test all the functions in the class.

11. Consider the following class definition
    class father {

    protected :int age;

    public;

    father (int x) {age = x;}

    virtual void iam ( )

    { cout<< "I AM THE FATHER, my age is : "<< age<< end1:}

    };

    Derive the two classes son and daughter from the above class and for each, define iam ( ) to write our similar but appropriate messages. You should also define suitable constructors for these classes. Now, write a main ( ) that creates objects of the three classes and then calls iam ( ) for them.

    Declare pointer to father. Successively, assign addresses of objects of the two derived classes to this pointer and in each case, call iam ( ) through the pointer to demonstrate polymorphism in action.

12. Write a program that creates a binary file by reading the data for the students from the terminal. The data of each student consist of roll no., name (a string of 30 or lesser no. of characters) and marks.

13. A hospital wants to create a database regarding its indoor patients. The information to store include

    a) Name of the patient

    b) Date of admission

    c) Disease

    d) Date of discharge

    Create a structure to store the date (year, month and date as its members). Create a base class to store the above information. The member function should include functions to enter information and display a list of all the patients in the database. Create a derived class to store the age of the patients. List the information about all, to store the age of the patients. List the information about all the pediatric patients (less than twelve years in age).

14. Make a class **Employee** with a name and salary. Make a class **Manager** inherit from **Employee**. Add an instance variable, named department, of type string. Supply a method to **toString**that prints the manager‟s name, department and salary. Make a class **Executive** inherit from **Manager**. Supply a method **to String** that prints the string **"Executive"** followed by the information stored in the **Manager** superclass object. Supply a test program that tests these classes and methods.

15. Imagine a tollbooth with a class called toll Booth. The two data items are a type unsigned int to hold the total number of cars, and a type double to hold the total amount of money collected. A constructor initializes both these to 0. A member function called payingCar ( ) increments the car total and adds 0.50 to the cash total. Another function, called nopayCar ( ), increments the car total but adds nothing to the cash total. Finally, a member function called displays the two totals.
    Include a program to test this class. This program should allow the user to push one key to count a paying car, and another to count a nonpaying car. Pushing the ESC key should cause the program to print out the total cars and total cash and then exit.

16. Write a function called reversit ( ) that reverses a string (an array of char). Use a for loop that swaps the first and last characters, then the second and next to last characters and so on. The string should be passed to reversit ( ) as an argument.
    Write a program to exercise reversit ( ). The program should get a string from the user, call

    reversit ( ), and print out the result. Use an input method that allows embedded blanks. Test the program with Napoleon‟s famous phrase, "Able was I ere I saw Elba)".

17. Create some objects of the string class, and put them in a Deque-some at the head of the Deque and some at the tail. Display the contents of the Deque using the forEach ( ) function and a user written display function. Then search the Deque for a particular string, using the first That ( ) function and display any strings that match. Finally remove all the items from the Deque using the getLeft ( ) function and display each item. Notice the order in which the items are displayed: Using getLeft ( ), those inserted on the left (head) of the Deque are removed in "last in first out" order while those put on the right side are removed in "first in first out" order. The opposite would be true if getRight ( ) were used.

18. Create a base class called shape. Use this class to store two double type values that could be used to compute the area of figures. Derive two specific classes called triangle and rectangle from the base shape. Add to the base class, a member function get_data ( ) to initialize base class data members and another member function display_area ( ) to compute and display the area of figures. Make display_area ( ) as a virtual function and redefine this function in the derived classes to suit their requirements.
    Using these three classes, design a program that will accept dimensions of a triangle or a

    rectangle interactively and display the area.

    Remember the two values given as input will be treated as lengths of two sides in the case of

    rectangles and as base and height in the case of triangles and used as follows:

    Area of rectangle = x * y

    Area of triangle = ½ * x * y

19. Write a program to create the card class and group an array of 52 such objects together in an array, thus creating a deck of cards. Display the cards in the deck, shuffle it , and then display it again.

20. Write a menu driven program for stack implementation using inheritance concept of class.

21. Write a single function to find out min value of different data type.

22. Write program for sorting and searching single array for different data type using class concept.

23. Write a program to two different type of argument. The data type must decide at run time.

24. Write a program for array bound checking using exception handling with class concept.

25. Write a program using constructor to handle divide by zero exception.

26. Write a program to write employee detail in the file.

27. Write a program to search give employee detail in the file.

28. Write a program to read a file and print the number of vowels and number of words in the file. Assume that a word is a sequence of letters ending with a blank, or a tab, or an end of line marker or end of file or punctuation symbols such as “,”, ”.”, ”!” and “?”.

Laboratory Manual 2023-24

Sub:- **Operating System-I (OS-I)**

S.Y.B.Tech (CSE)

Prepared By

Prof. Karande A.A.

G.K. Gujar Memorial Charitable Trust's,
Dr.Ashok Gujar Technical Institute's
Dr. Daulatrao Aher College of Engineering, Karad.
Vidyanagar Ext. Banawadi, Tal. Karad 415124, Dist. Satara, Maharashtra INDIA

Prepared By:- Prof. Karande A.A.	Approved By: Head of Department

EXPERIMENT LIST

Department: Computer Science & Engg. **Academic Year:** 2023-24

Class: Second Year (S.Y.B.Tech. CSE) **Semester:** IV

Subject: Operating System-I **Semester Duration:** 6 months

Sr. No.	Experiment Name
1.	Installation of Windows operating System and Linux operating System
2.	Execute general purpose utilities in Linux.
3.	Execute process, file & directory related commands in Linux.
4.	To study of various UNIX editors such as vi, ed, ex and EMACS.
5.	Write C programs to simulate Producer – Consumer Problem.
6.	Write C programs to simulate CPU scheduling algorithms: FCFS, SJF, and Round Robin.
7.	Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.
8.	Write a C program to simulate page replacement algorithms.
9.	Write a C program to simulate the following memory allocation techniques a) Worst-fit b) Best-fit c) First-fit
10.	Write a C program to simulate the following file organization techniques a) Single level directory b) Two level directory c) Hierarchical

Experiment No 1

TITLE:

Installation of Windows operating System and Linux operating System

DESCRIPTION:

- **Installing or upgrading Windows**

To start the Windows install or upgrade process, you need to configure your computer to boot from a CD or DVD before booting to the hard drive. Changing the boot process forces the computer to look for the Windows installation disc before booting from the hard drive.

1. Open the CMOS setup.

- How to enter the BIOS or CMOS setup.

2. Change the computer's boot order. Set the CD, DVD, or disc drive as the first boot device if you are trying to boot from a disc. Or, set the first boot device to your USB drive if you're trying to boot from a USB thumb drive. If the drive is not shown, keep the disc is inserted and reboot the computer. With the disc in the drive, BIOS should recognize and include it in the list.

3. Save the settings change and exit BIOS.

Once you have updated the boot order, you can begin the Windows installation process.

4. Place the Windows disc in the CD/DVD drive or USB thumb drive into the back of the computer.

5. Turn on or restart the computer. As the computer starts up, it should detect the installation disc or drive and show a message similar to *Press any key to boot from CD*. Press any key on the keyboard to have the computer boot from the Windows disc or drive.

6. After the Windows install begins, there are several prompts that you need to answer. Select either **Yes** or the appropriate option to install Windows.

7. When asked which partition to install Windows onto, select the main partition, usually the C: drive or one labeled "Unallocated partition". If upgrading Windows, select the existing installation of Windows on the hard drive.

8. You may be asked if you want to erase all contents on the hard drive, then install Windows. We recommend you choose this option, as it also formats the hard drive to allow the Windows operating system to be installed.

9. The computer may need to restart several times during the Windows install process. The restarts are normal and if prompted to restart, select the **Yes** option.

10. When the install process is nearly complete, the Windows configuration option screens are shown. On these screens, you may be asked to select the time zone you live in, your preferred language, and the account's name you use to access Windows. Select the appropriate options and enter the appropriate information on each configuration screen.

The Windows install process is completed when the computer prompts you to log in or when it loads into Windows.

➢ Installation of Linux operating System

Linux is the base of many of open source operating systems designed to replace Windows and Mac OS. It is free to download and install on any computer. Because it is open source, there are a variety of different versions, or distributions, available developed by different groups. To prevent hacking attempts, many organizations keep their Linux operating systems private. Many others make their variations of Linux available publicly so the whole world can benefit at large.

Step 1: Download the ISO file.

Step 2: Boot your system with Bootable DVD / USB drive.

To start the installation click on "Install Ubuntu"

Step 3: Check Install Prerequisite

Step 4: Select the Installation Type

Step 5: Select your respective Time Zone

Step 6: Select your respective Keyboard Layout

Step 7: Set the Hostname of your system and User credentials that will be used after installation.

Installation has started. Once the installation is completed, it will ask to restart the Machine.
Click on "Restart Now"
Step 8: Login Screen after reboot.

Use the same user and its credentials that we have set during the installation. We will get below screen after entering the credentials.
Ubuntu Installation is Completed Now.

Conclusion: - Installation of Windows operating System and Linux operating System

Experiment No 2

TITLE:
Execute general purpose utilities in Linux.

DESCRIPTION:
UNIX is now one of the most commonly used Operating systems used for various purposes such as Personal use, Servers, Smartphone's, and many more. It was developed in the 1970's at AT & T Labs by two famous personalities Dennis M. Ritchie and Ken Thompson.

- The most important part of the Linux is Linux Kernel which was first released in early 90's by Linus Torvalds. There are several Linux distros available (most are open-source and free to download and use) such as Ubuntu, Debian, Fedora, Kali, Mint, Gentoo, Arch and much more.
- Now coming to the Basic and most usable commands of Linux/Unix part. (Please note that all the Linux/Unix commands are run in the terminal of a Linux system. Terminal is like command prompt as that of in Windows OS)
- Linux/Unix commands are **case-sensitive** i.e Hello is different from hello**.**

Commands:

Sr no	Commands	Syntax	Description
1	Who	$ who	The '$ who' command displays all the users who have logged into the system currently
2	pwd	$ pwd	he '$pwd' command stands for 'print working directory' and as the name says,it displays the directory in which we are currently
3	mkdir	$ mkdir newfolder	The '$ mkdir' stands for 'make directory' and it creates a new directory.
4	rmdir	$ rmdir newfolder	The '$ rmdir' command deletes any directory we want to delete and you can remember it by its names 'rmdir' which stands for 'remove directory'.
5	cd	$ cd newfolder	he '$ cd' command stands for 'change directory' and it changes your current directory to the 'newfolder' directory.
6	ls	$ls	The 'ls' command simply displays the contents of a directory.
7	touch	$ touch example	The '$ touch' command creates a file(not directory) and you can simple add an extension such as .txt after it to make it a Text File.

8	cp	$ cp /home/harssh/file.txt /home/harssh/new/	This ‘$ cp ‘ command stands for ‘copy’ and it simply copy/paste the file wherever you want to.
9	mv	$ mv /home/harssh/file.txt /home/harssh/new	The ‘$ mv’ command stands for ‘move’ and it simply move a file from a directory to another directory.
10	rm	$ rm file.txt	The ‘$ rm ‘ command for remove and the ‘-r’ simply recursively deletes file.

11	chmod	$ chmod +w file.txt
		$ chmod +r file.txt
		$ chmod +x file.txt
			he ‘$ chmod’ command stands for change mode command

12	cal	$ cal	The ‘$ cal’ means calendar and it simply display calendar on to your screen.
13	man	$ man pwd	he ‘$ man’ command stands for ‘manual’ and it can display the in-built manual for most of the commands that we ever need.
14	passwd	$ passwd	The ‘$ passwd’ command simply changes the password of the user.In above case ‘harssh’ is the user.
15	clear	$ clear	The ‘$ clear’ command is used to clean up the terminal so that you can type with more accuracy
16	date	$ date	Used to check the date and time
17	echo	$echo “text”	Used to print the message on the screen.
18	lp	$lp filename	Used to take printouts
19	uptime	$uptime	Tells you how long the computer has been running since its last reboot or power-off.
20	hostname	$ hostname	Displays and set system host name

Conclusion: - Hence we studied general purpose utilities command in Linux

Experiment No 3

TITLE:
Execute process, file & directory related commands in Linux.

DESCRIPTION:

- **Process related commands in Linux:-**

Linux provides us utility called **ps** for viewing information related with the processes on a system which stands as abbreviation for **"Process Status".** ps command is used to list the currently running processes and their PIDs along with some other information depends on different options. It reads the process information from the virtual files in **/proc** file-system. /proc contains virtual files, this is the reason it's referred as a virtual file system.

Commands:

Sr no	Commands	Syntax	Description
1	ps	$ps	The ps command is used to view currently running processes on the system.
2	wait	$wait	Waits until all background processes are completed and then exits.
3	sleep	$sleep 30	Used to execute commands after certain amount of time by sleeping for given seconds.
4	exit	$exit	Use the quit the shell
5	kill	$kill 0	Used to stop execution of particular process by sending an intemipt signal to the process.

Options of ps commands:

$ps -f	Full listing showing PPID of each process.
$ps -u username	Displays processes of user 'username'
$ps -a	Processes of all users
$ps -e	Processes including user and system processes.

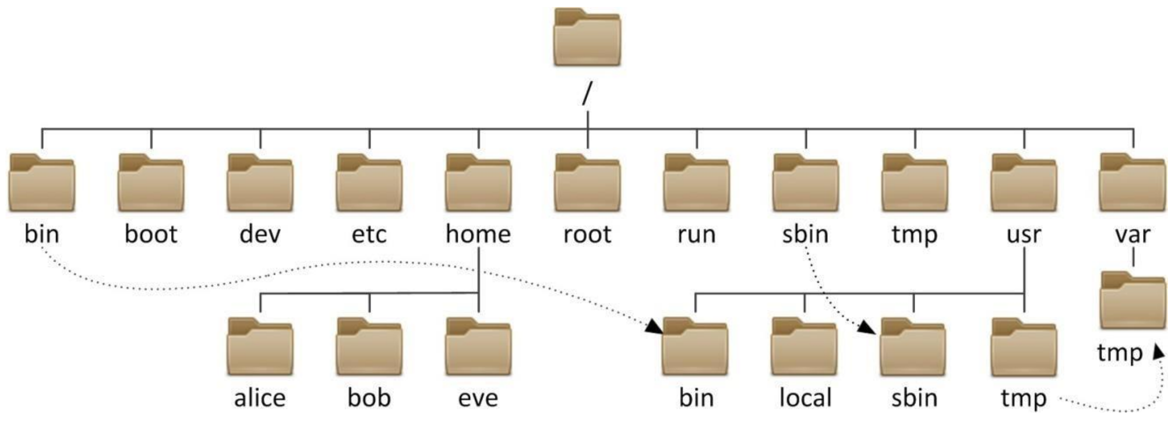
Options of kill commands:

$kill 0	Kills all the processes on the terminal except the login shell by special argument '0'
$ki11 120 230 234	Kills three processes with pid 120 230 234
$kill -9 0	Kills all processes including login shell
$kill -9 $$	Kills login shell

➢ File & directory related commands in Linux:

Unix uses a hierarchical structure for organizing files and directories. This structure is called as a directory tree. The tree has a single root node, the slash character (/), and all other directories are contained below it. When user first log in to the UNIX server, the specified directory is called as Home directory.

Directory Structure in Unix/Linux:



Directory manipulation commands are as follows:

1. mkdir 2. cd 3. rmdir 4. pwd

1. **'mkdir' command:**

   It is used to create a new directory in a current directory.
   $mkdir <directory named Example:
   $mkdir CM5I
   User can create more than one directory in a single command.
   $mkdir subjectl subject2
   If user wants to create directory on the specific path then syntax is:
   $mkdir path/Directory name

2. **'cd' command:**

   The 'cd' command is used to change directory. You can use it to change to any directory by specifying a valid absolute or relative path.
   The syntax is as given below
   $cd <directory name>

   Example
   $ cd Directory name
   $cd CM5I
   $cd .

Example:

```
$cd ..
$ cd /
```

Example:

```
$ cd /
```

To come out from current working directory.
It changes to root directory.

3.’rmdir’ command

It is used to delete/ remove a directory. If the parent directory having subdirectories then first all subdirectories will be deleted then the parent directory is deleted.

```
$rmdir <directory named
```

4.‘pwd’ command

pwd stands for Present Working Directory. This is most used Linux command to see the specific Unix Directory on which the user is working on.

$pwd

diff command:

This command is used to show difference between two text files. It also tells which line in one has to be changed to make the two files identical.

Syntax:

```
$diff filename1 filename2
```

The options of the result should be like this —

a	-Added the text to file
c	-Changes are made in the file
d	-Deletion operation is performed
<	-Lines from the first file
>	-Lines from the second file

```
$cat file1.txt
I need to go to the shop.
I need to buy some mangoes. When I get home, I'll wash the cat.

$cat file2.txt
I need to go to the shop.
I need to buy some mangoes.
Oh yeah, I also need to buy cheese. When I get home, I'll wash the cat.
```

Use the diff command to compare both files.

```
$ diff fiIe1.txt file2.txt
```

The above command should give the result as shown below — 2a3

```
> Oh yeah, I also need to buy cheese.
```

From the output, 2a3 means “After line 2 in the first file, a line needs to be added: line 3 from the second file”.

comm command:

This command compares two sorted files. It compares each line of first file with its corresponding line in the second file and generates three column output.
The first column lists the lines only in first file.
The second column lists the lines only in second file. The third column lists the lines in both files.
Syntax:
$comm filenamel filename2 Example:

$cat studentl Harsh
Sujay Smith
$comm studentl student2

$cat student2 Harsh
Niket Ashutosh
Sujay
Smith
Niket Ashutosh
Harsh

pr command:

pr - convert text files for printing
The pr command does minor formatting of files on the terminal screen or for a printer. For example, if you have a long list of names in a file, you can format it onscreen into two or more columns.
The syntax is:
pr option(s) filename(s)
pr changes the format of the file only on the screen or on the printed copy; it doesn't modify the original file.

dir command:

It is used to list all files and directories.

Conclusion: - Hence we studied process, file & directory related commands in Linux

Experiment No 4

TITLE:

To study of various UNIX editors such as vi, ed, ex and EMACS.

DESCRIPTION:

Editor is a program that allows user to see a portions a file on the screen and modify characters and lines by simply typing at the current position. UNIX supports variety of Editors. They are:
ed, ex, vi, EMACS

➢ **Vi-**

vi is stands for "visual". vi is the most important and powerful editor. vi is a full screen editor that allows user to view and edit entire document at the same time. vi editor was written in the University of California, at Berkley by Bill Joy, who is one of the co-founder of Sun Microsystems.

Features of vi:
It is easy to learn and has more powerful features.
It works great speed and is case sensitive. vi has powerful undo functions and has 3 modes:
1. Command mode- In command mode, no text is displayed on the screen.
2. Insert mode- In Insert mode, it permits user to edit insert or replace text.
3. Escape or ex mode- In escape mode, it displays commands at command line.

Moving the cursor with the help of h, l, k, j, I, etc

➢ **EMACS Editor-**

Motion Commands:
M-> Move to end of file
M-< Move to beginning of file
C-v Move forward a screen
M –v Move backward a screen
C –n Move to next line
C-p Move to previous line
C-a Move to the beginning of the line
C-e Move to the end of the line
C-f Move forward a character
C-b Move backward a character
M-f Move forward a word
M-b Move backward a word

Deletion Commands:
DEL delete the previous character
C –d delete the current character
M –DEL delete the previous word
M-d delete the next word
C-x DEL deletes the previous sentence
M-k delete the rest of the current sentence

C-k deletes the rest of the current line
C-xu undo the lasted it change

Search and Replace in EMACS:
y- Change the occurrence of the pattern
n- Don"t change the occurrence, but look for the other q Don"t change. Leave query replace completely
! - Change this occurrence and all others in the file

Conclusion: - Hence we studied of various UNIX editors such as vi, ed, ex and EMACS

Experiment No 5

TITLE:

Write C programs to simulate Producer – Consumer Problem.

DESCRIPTION:

Producer-consumer problem is a common paradigm for cooperating processes. A producer process produces information that is consumed by a consumer process. One solution to the producer-consumer problem uses shared memory. To allow producer and consumer processes to run concurrently, there must be available a buffer of items that can be filled by the producer and emptied by the consumer. This buffer will reside in a region of memory that is shared by the producer and consumer processes. A producer can produce one item while the consumer is consuming another item. The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

PROGRAM

```
#include<stdio.h>
void main()
{
int buffer[10], bufsize, in, out, produce, consume, choice=0;
in = 0;
out = 0;
bufsize = 10;
while(choice !=3)
{
printf("\n1. Produce \t 2. Consume \t3. Exit");
printf("\nEnter your choice: ");
scanf("%d",
&amp;choice);
switch(choice) {
case 1: if((in+1)%bufsize==out)
printf("\nBuffer is Full");
else
{

}
Break;
printf("\nEnter the value: ");
scanf("%d", &amp;produce);
buffer[in] = produce;
in = (in+1)%bufsize;

case 2: if(in == out)
```

```
        printf("\nBuffer is Empty");
```

```
} } }

else
{

}
break;
consume = buffer[out];
printf("\nThe consumed value is %d", consume);
out = (out+1)%bufsize;
```

OUTPUT

```
1. Produce
2. Consume
3. Exit
Enter your choice: 2
Buffer is Empty

1. Produce
 2. Consume
3. Exit
Enter your choice: 1
Enter the value: 100

1. Produce
 2. Consume
 3. Exit
Enter your choice: 2
The consumed value is 100

1. Produce
2. Consume
3. Exit
Enter your choice: 3
```

Conclusions: Hence we studied producer consumer problem.

Experiment No 6

TITLE:

Write C programs to simulate CPU scheduling algorithms: FCFS, SJF, and Round Robin.

DESCRIPTION:

FCFS

To calculate the average waiting time using the FCFS algorithm first the waiting time of the first process is kept zero and the waiting time of the second process is the burst time of the first process and the waiting time of the third process is the sum of the burst times of the first and the second process and so on. After calculating all the waiting times the average waiting time is calculated as the average of all the waiting times. FCFS mainly says first come first serve the algorithm which came first will be served first.

ALGORITHM:

```
Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process name and the burst time
Step 4: Set the waiting of the first process as ‗0‘and its burst time as its turnaround time
Step 5: for each process in the Ready Q calculate
        a) Waiting time (n) = waiting time (n-1) + Burst time (n-1)
        b) Turnaround time (n)= waiting time(n)+Burst time(n)
Step 6: Calculate
        a) Average waiting time = Total waiting Time / Number of process
        b) Average Turnaround time = Total Turnaround Time / Number of process
Step 7: Stop the process
```

SOURCE CODE:

```
#include<stdio.>
#include<conio.h>
main()
{
int bt[20], wt[20], tat[20], i, n;
float wtavg, tatavg;
clrscr();
printf("\nEnter the  number of processes --");
 scanf("%d", &n);
for(i=0;i<n;i++)
{
printf("\nEnter Burst Time for Process %d -- ", i);
scanf("%d", &bt[i]);
}
wt[0] = wtavg = 0;
tat[0] = tatavg = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
tat[i] = tat[i-1] +bt[i];
wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++)
printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]);
printf("\nAverage Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n);
getch();
}
```

INPUT

```
Enter the number of processes --          3
Enter Burst Time for Process 0 --         24
Enter Burst Time for Process 1 --         3
Enter Burst Time for Process 2 --         3
```

OUTPUT

```
PROCESS        BURST TIME        WAITING TIME        TURNAROUND
                                                     TIME
P0             24                0                   24
P1             3                 24                  27
P2             3                 27                  30
Average Waiting Time--    17.000000
Average Turnaround Time --                27.000000
```

SJF:

To calculate the average waiting time in the shortest job first algorithm the sorting of the process based on their burst time in ascending order then calculate the waiting time of each process as the sum of the bursting times of all the process previous or before to that process.

ALGORITHM:

```
Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process id and accept the
CPU burst time
Step 4: Start the Ready Q according the shortest Burst time by sorting
according to lowest to highest burst time.
Step 5: Set the waiting time of the first process as ‗0‘ and its turnaround time as
its burst time.
Step 6: Sort the processes names based on their Burt time
Step 7: For each    process in the ready queue, calculate
    a)    Waiting time(n)= waiting time (n-1) + Burst time (n-1)
    b)    Turnaround time (n)= waiting time(n)+Burst time(n)
Step 8: Calculate
    c)    Average waiting time = Total waiting Time / Number of process
    d)    Average Turnaround time = Total Turnaround Time / Number of
process
Step 9: Stop the process
```

SOURCE CODE :

```
#includ<stdio.h>
#include<conio.h>
main()
{
int p[20], bt[20], wt[20], tat[20], i, k, n, temp; float
wtavg,tatavg;
clrscr();
printf("\nEnter the number of processes -- ");
scanf("%d", &n);
for(i=0;i<n;i++)
{
p[i]=i;
printf("Enter Burst Time for Process %d -- ", i);
scanf("%d", &bt[i]);

}
for(i=0;i<n;i++)
for(k=i+1;k<n;k++)
if(bt[i]>bt[k])
{
temp=bt[i];bt[i]=bt[k]; bt[k]=temp;

temp=p[i];p[i]=p[k]; p[k]=temp;
}
wt[0] = wtavg = 0;
tat[0] = tatavg = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
 tat[i] = tat[i-1] +bt[i];
 wtavg = wtavg + wt[i];
 tatavg = tatavg + tat[i];
}
printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++)
          printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);
          printf("\nAverage Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n);
 getch();
}
```

INPUT

```
Enter the number of processes --                4
Enter Burst Time for Process 0 --               6
Enter Burst Time for Process 1 --               8
Enter Burst Time for Process 2 --               7
Enter Burst Time for Process 3 --               3
```

OUTPUT

```
PROCESS                         BURST        WAITING          TURNARO
                                TIME         TIME             UND TIME
```

```
P3                      3           0           3
P0                      6           3           9
P2                      7           9           16
P1                      8           16          24
Average Waiting Time --             7.000000
Average Turnaround Time --          13.000000
```

ROUND ROBIN:

To aim is to calculate the average waiting time. There will be a time slice, each process should be executed within that time-slice and if not it will go to the waiting state so first check whether the burst time is less than the time-slice. If it is less than it assign the waiting time to the sum of the total times. If it is greater than the burst-time then subtract the time slot from the actual burst time and increment it by time-slot and the loop continues until all the processes are completed.

ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue and time quantum (or) timeslice

Step 3: For each process in the ready Q, assign the process id and accept the CPU bursttime

Step 4: Calculate the no. of time slices for each process where

No. of timeslice for process (n) = burst time process (n)/time slice

Step 5: If the burst time is less than the time slice then the no. of time slices =1.

Step 6: Consider the ready queue is a circular Q, calculate

a) Waiting time for process (n) = waiting time of process(n-1)+ burst time ofprocess(n-1) + the time difference in getting the CPU from process(n-1)
b) Turnaround time for process(n) = waiting time of process(n) + burst time ofprocess(n)+ the time difference in getting CPU from process(n).

Step 7: Calculate

a) Average waiting time = Total waiting Time / Number of process
b) Average Turnaround time = Total Turnaround Time / Number of process

Step8: Stop the process

SOURCE CODE

```
#include<stdio.h>
main()
{
inti,j,n,bu[10],wa[10],tat[10],t,ct[10],max;float awt=0,att=0,temp=0;
clrscr();
printf("Enter  the  no  of  processes  -- ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nEnter  Burst  Time  for process %d -- ", i+1);
scanf("%d",&bu[i]);
ct[i]=bu[i];
}
printf("\nEnter the size of time slice -- ");
scanf("%d",&t);
max=bu[0]; for(i=1;i<n;i++)
if(max<bu[i]) max=bu[i];
for(j=0;j<(max/t)+1;j++)
for(i=0;i<n;i++)
if(bu[i]!=0)
 if(bu[i]<=t)
 {
 tat[i]=temp+bu[i];
temp=temp+bu[i];
bu[i]=0;
 }
 Else
  {
  bu[i]=bu[i]-t;
 temp=temp+t;
 }
for(i=0;i<n;i++)
{
wa[i]=tat[i]-ct[i];
att+=tat[i];
awt+=wa[i];
}
printf("\nThe Average Turnaround time is -- %f",att/n);
printf("\nThe Average Waiting time is -- %f ",awt/n);
printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");
for(i=0;i<n;i++)
printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);
getch();
}
```

INPUT:

```
Enter the no of processes – 3
Enter Burst Time  for process
1 – 24
  Enter Burst Time for process
2 - 3
 Enter Burst Time for process
3 – 3
  Enter the size of time slice –
3
```

OUTPUT:

```
PROCESS        BURST TIME        WAITING TIME     TURNAROUNDTIME
   1               24                 6                 30
   2               3                  4                  7
   3               3                  7                 10
The Average Turnaround time is -15.666667
 The Average Waiting time is: 5.666667
```

Conclusion: - Hence we implement C programs to simulate CPU scheduling algorithms: FCFS, SJF, and Round Robin.

Experiment No 7

TITLE:

Write a C program to simulate Bankers algorithm for the purpose of deadlock Avoidance.

DESCRIPTION:

In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock. Deadlock avoidance is one of the techniques for handling deadlocks. This approach requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process. Banker's algorithm is a deadlock avoidance algorithm that is applicable to a system with multiple instances of each resource type.

PROGRAM

```
 #include<stdio.h>
int main()
{
int n, r, i, j, k;
n = 5;
r = 3;
int alloc[5][3] = { { 0, 0, 1 },{ 3, 0, 0 },{ 1, 0, 1 },{ 2, 3, 2 },{ 0, 0, 3 } };

int max[5][3] = { { 7, 6, 3 }, { 3, 2, 2 },{ 8, 0, 2 },{ 2, 1, 2 },{ 5, 2, 3 } }; // P0 // MAX Matrix

int avail[3] = { 2, 3, 2 }; // These are Available Resources

int f[n], ans[n], ind = 0;
for (k = 0; k &lt; n; k++)
{
f[k] = 0;
}
int need[n][r];
for (i = 0; i &lt; n; i++)
{
for (j = 0; j &lt; r; j++)
need[i][j] = max[i][j] - alloc[i][j];

}
int y = 0;
for (k = 0; k &lt; 5; k++) {
for (i = 0; i &lt; n; i++) {
if (f[i] == 0)
{
int flag = 0;
for (j = 0; j &lt; r; j++)
{
if (need[i][j] &gt; avail[j])
{
flag = 1;
break;
}
}
if (flag == 0)
{
ans[ind++] = i;
for (y = 0; y &lt; r; y++)
avail[y] += alloc[i][y];
f[i] = 1;
}
}
}
}
printf(&quot;Th SAFE Sequence is as follows\n&quot;);
for (i = 0; i &lt; n - 1; i++)
printf(&quot; P%d -&gt;&quot;, ans[i]);
printf(&quot; P%d&quot;, ans[n - 1]);
return (0);
}
```

Output:

```
The Safe sequence is as follows:
P1-&gt;P3-&gt;P4-&gt;P0-&gt;P2
```

Conclusion: - Hence we implement C program to simulate Bankers algorithm for the purpose of deadlock avoidance.

Experiment No 8

TITLE:

Write a C program to simulate page replacement algorithms.

DESCRIPTION:

Page replacement is basic to demand paging. It completes the separation between logical memory and physical memory. With this mechanism, an enormous virtual memory cans be provided for programmers on a smaller physical memory. There are many different page-replacement algorithms. Every operating system probably has its own replacement scheme. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. If the recent past is used as an approximation of the near future, then the page that has not been used for the longest period of time can be replaced. This approach is the Least Recently Used (LRU) algorithm. LRU replacement associates with each page the time of that page&#39;s last use. When a page must be replaced, LRU chooses the page that has not been used for the longest period of time. Least frequently used (LFU) page-replacement algorithm requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count. Optimal page replacement algorithm has the lowest page-fault rate of all algorithms and will never suffer from Belady&#39;s anomaly. The basic idea is to replace the page that will not be used for the longest period of time. Use of this page-replacement algorithm guarantees the lowest possible page fault rate for a fixed number of frames. Unfortunately, the optimal page- replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.

PROGRAM

a) <u>Fifo Page Replacement Algorithm</u>

```
#include<stdio.h>
int main()
{
int incomingStream[] = {4, 1, 2, 4, 5};
int pageFaults = 0;
int frames = 3;
int m, n, s, pages;

pages = sizeof(incomingStream)/sizeof(incomingStream[0]);
printf(&quot;Incoming \t Frame 1 \t Frame 2 \t Frame 3&quot;);
int temp[frames];
for(m = 0; m &lt; frames; m++)
{
temp[m] = -1;
}
for(m = 0; m &lt; pages; m++)
{
```

```
s = 0;
for(n = 0; n &lt; frames; n++)
{
if(incomingStream[m] == temp[n])
{
s++;
pageFaults--;
}
}
pageFaults++;
if((pageFaults &lt;= frames) &amp;&amp; (s == 0))
{
temp[m] = incomingStream[m];
}
else if(s == 0)
{
temp[(pageFaults - 1) % frames] = incomingStream[m];
}
printf(&quot;\n&quot;);
printf(&quot;%d\t\t\t&quot;,incomingStream[m]);
for(n = 0; n &lt; frames; n++)
{
if(temp[n] != -1)
printf(&quot; %d\t\t\t&quot;, temp[n]);
else
printf(&quot; - \t\t\t&quot;);
}
}
printf(&quot;\nTotal Page Faults:\t%d\n&quot;, pageFaults);
return 0;
}
```

OUTPUT:-

```
Incoming Frame 1 Frame 2 Frame 3
4 4 - -
1 4 1 -
2 4 1 2
4 4 1 2
5 5 1 2
Total Page Faults: 4
```

b) LRU:

```
#include<stdio.h>
int main()
{
int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];
printf(&quot;Enter no of pages:&quot;);
scanf(&quot;%d&quot;,&amp;n);
printf(&quot;Enter the reference string:&quot;);
for(i=0;i&lt;n;i++)
scanf(&quot;%d&quot;,&amp;p[i]);
printf(&quot;Enter no of frames:&quot;);
scanf(&quot;%d&quot;,&amp;f);
q[k]=p[k];
printf(&quot;\n\t%d\n&quot;,q[k]);
c++;
k++;
for(i=1;i&lt;n;i++)
{
c1=0;
for(j=0;j&lt;f;j++)
{
if(p[i]!=q[j])
c1++;
}
if(c1==f)
{
c++;
if(k&lt;f)
{
q[k]=p[i];
k++;
for(j=0;j&lt;k;j++)
printf(&quot;\t%d&quot;,q[j]);
printf(&quot;\n&quot;);
}
else
{
for(r=0;r&lt;f;r++)
{
c2[r]=0;
for(j=i-1;j&lt;n;j--)
{
if(q[r]!=p[j])
c2[r]++;
else
break;
}
}
for(r=0;r&lt;f;r++)
b[r]=c2[r];
for(r=0;r&lt;f;r++)
{
for(j=r;j&lt;f;j++)
{
if(b[r]&lt;b[j])
{
t=b[r];
b[r]=b[j];
b[j]=t;
}
}
}
for(r=0;r&lt;f;r++)
{
if(c2[r]==b[0])
q[r]=p[i];
printf(&quot;\t%d&quot;,q[r]);
}
printf(&quot;\n&quot;);
}
}
}
printf(&quot;\nThe no of page faults is %d&quot;,c);
}
```

OUTPUT:

```
Enter no of pages:10
Enter the reference string: 7 5 9 4 3 7 9 6 2 1

Enter no of frames:3
7
7 5
7 5 9
4 5 9
4 3 9
4 3 7
9 3 7
```

```
9 6 7
9 6 2
1 6 2
The no of page faults is 10
```

c) LFU:

```
#include<stdio.h>
#include<conio.h>
main()
{
int rs[50], i, j, k, m, f, cntr[20], a[20], min, pf=0;
clrscr();
printf(&quot;\nEnter number of page references -- &quot;);
scanf(&quot;%d&quot;,&amp;m);
printf(&quot;\nEnter the reference string -- &quot;);
for(i=0;i&lt;m;i++)
scanf(&quot;%d&quot;,&amp;rs[i]);
printf(&quot;\nEnter the available no. of frames -- &quot;);
scanf(&quot;%d&quot;,&amp;f);
for(i=0;i&lt;f;i++)
{
cntr[i]=0; a[i]=-1;
}
printf("\nThe Page Replacement Process is – \n");
for(i=0;i&lt;m;i++)
{
for(j=0;j&lt;f;j++)
if(rs[i]==a[j])
{
cntr[j]++;
break;
if(j==f)
{ min = 0;
for(k=1;k&lt;f;k++)
if(cntr[k]&lt;cntr[min])
min=k;
a[min]=rs[i]; cntr[min]=1;

pf++;
}
printf(&quot;\n&quot;);
for(j=0;j&lt;f;j++)
printf(&quot;\t%d&quot;,a[j]);
if(j==f)
}
printf("\tPF No. %d",pf);}
printf(&quot;\n\n Total number of page faults -- %d&quot;,pf);
getch();
}
```

INPUT

```
Enter number of page references -- 10
Enter the reference string -- 1 2 3 4 5 2 5 2 5 1 4 3
Enter the available no. of frames -- 3
```

OUTPUT

```
The Page Replacement Process is –
1 -1 -1 PF No. 1
1 2 -1 PF No. 2
1 2 3 PF No. 3
4 2 3 PF No. 4
5 2 3 PF No. 5
5 2 3
5 2 3
5 2 1 PF No. 6
5 2 4 PF No. 7
5 2 3 PF No. 8
Total number of page faults – 8
```

d) <u>OPTIMAL</u>

```
#include<stdio.h>
int n;
main()
{
int seq[30],fr[5],pos[5],find,flag,max,i,j,m,k,t,s;
int count=1,pf=0,p=0;
float pfr; clrscr();
printf(&quot;Enter maximum limit of the sequence: &quot;);
scanf(&quot;%d&quot;,&amp;max);
printf(&quot;\nEnter the sequence: &quot;);
for(i=0;i&lt;max;i++)
scanf(&quot;%d&quot;,&amp;seq[i]);
printf(&quot;\nEnter no. of frames: &quot;);
scanf(&quot;%d&quot;,&amp;n);
fr[0]=seq[0];
pf++;
printf(&quot;%d\t&quot;,fr[0]);
i=1;
while(count&lt;n)
```

```
{
flag=1; p++;
for(j=0;j&lt;i;j++)
{
if(seq[i]==seq[j]) flag=0;
}
if(flag!=0)
{
fr[count]=seq[i];
printf(&quot;%d\t&quot;,fr[count]);
count++;
pf++;
}
i++;
}
printf(&quot;\n&quot;);
for(i=p;i&lt;max;i++)
{
flag=1;
for(j=0;j&lt;n;j++)
{
if(seq[i]==fr[j])
flag=0;
}
if(flag!=0)
{
for(j=0;j&lt;n;j++)
{
m=fr[j];
for(k=i;k&lt;max;k++)
{
if(seq[k]==m)
{
pos[j]=k;
break;
}
else
pos[j]=1;
}
}
for(k=0;k&lt;n;k++)
{
if(pos[k]==1)
flag=0;
}
if(flag!=0)
s=findmax(pos);
if(flag==0)
{
for(k=0;k&lt;n;k++)
{
if(pos[k]==1)
{
s=k;
break;
}
}
}
fr[s]=seq[i];
for(k=0;k&lt;n;k++)
printf(&quot;%d\t&quot;,fr[k]);
pf++;
printf(&quot;\n&quot;);
}
}
pfr=(float)pf/(float)max;
printf(&quot;\nThe no. of page faults are %d&quot;,pf);
printf(&quot;\nPage fault rate %f&quot;,pfr);
getch();
}
int findmax(int a[])
{
int max,i,k=0;
max=a[0];
for(i=0;i&lt;n;i++)
{
if(max&lt;a[i])
{
max=a[i];
k=i;
}
}
return k;

}
```

INPUT

Enter number of page references -- 10

Enter the reference string -- 1 2 3 4 5 2 5 2 5 1 4 3
Enter the available no. of frames – 3

OUTPUT

```
The Page Replacement Process is –
1 -1 -1 PF No. 1
1 2 -1 PF No. 2
1 2 3 PF No. 3
4 2 3 PF No. 4
5 2 3 PF No. 5
5 2 3
5 2 3
5 2 1 PF No. 6
5 2 4 PF No. 7
5 2 3 PF No. 8
Total number of page faults – 8
```

Conclusion: Hence we studied page replacement algorithm in detail.

Experiment No 9

TITLE:

Write a C program to simulate the following memory allocation techniques
a) Worst-fit b) Best-fit c) First-fit

DESCRIPTION:

One of the simplest methods for memory allocation is to divide memory into several fixed-sized partitions. Each partition may contain exactly one process. In this multiple-partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process. The operating system keeps a table indicating which parts of memory are available and which are occupied. Finally, when a process arrives and needs memory, a memory section large enough for this process is provided. When it is time to load or swap a process into main memory, and if there is more than one free block of memory of sufficientsize, then the operating system must decide which free block to allocate. Best-fit strategy chooses the block that is closest in size to the request. First-fit chooses the first available block that is large enough. Worst-fit chooses the largest available block.

PROGRAM

a)WORST-FIT

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp;
static int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - First Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
{
ff[i]=j;
break;
}
}
}
frag[i]=temp;
bf[ff[i]]=1;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}
```

INPUT

```
Enter the number of blocks: 3
Enter the number of files: 2

Enter the size of the blocks:-
Block 1: 5
Block 2: 2
Block 3: 7

Enter the size of the files:-
File 1: 1
File 2: 4
```

OUTPUT

```
File No      File Size     Block No       Block Size      Fragment
1                1           1                5               4
2                4           3                7               3
```

b) Best-fit

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
static int bf[max],ff[max];
clrscr();
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
if(lowest>temp)
{
ff[i]=j;

lowest=temp;
}}}
frag[i]=lowest;
bf[ff[i]]=1;
lowest=10000;
}
printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
for(i=1;i<=nf && ff[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}
```

```
INPUT
Enter the number of blocks: 3
Enter the number of files: 2

Enter the size of the blocks:-
Block 1: 5
Block 2: 2
Block 3: 7

Enter the size of the files:-
File 1: 1
File 2: 4

OUTPUT
File No       File Size      Block No      Block Size     Fragment
1              1               2              2              1
2              4               1              5              1
```

c) First-fit

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
static int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - Worst Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{

for(j=1;j<=nb;j++)
{
if(bf[j]!=1) //if bf[j] is not allocated
{
temp=b[j]-f[i];
if(temp>=0)
if(highest<temp)
{
ff[i]=j;
highest=temp;
}}}
frag[i]=highest;
bf[ff[i]]=1;
highest=0;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}
```

INPUT

```
Enter the number of blocks: 3
Enter the number of files: 2

Enter the size of the blocks:-
Block 1: 5
Block 2: 2
Block 3: 7

Enter the size of the files:-
File 1: 1
File 2: 4
```

OUTPUT

```
File No         File Size       Block No        Block Size      Fragment
1                1                3               7                 6
2                4                1               5                 1
```

Conclusion: Hence we implement C program to simulate the following memory allocation techniques
a) Worst-fit b) Best-fit c) First-fit

Experiment No 10

TITLE:

Write a C program to simulate the following file organization techniques
a) Single level directory b) Two level directory c) Hierarchical

DESCRIPTION:

The directory structure is the organization of files into a hierarchy of folders. In a single-level directory system, all the files are placed in one directory. There is a root directory which has all files. It has a simple architecture and there are no sub directories. Advantage of single level directory system is that it is easy to find a file in the directory. In the two-level directory system, each user has own user file directory (UFD). The system maintains a master block that has one entry for each user. This master block contains the addresses of the directory of the users. When a user job starts or a user logs in, the system's master file directory (MFD) is searched. When a user refers to a particular file, only his own UFD is searched. This effectively solves the name collision problem and isolates users from one another. Hierarchical directory structure allows users to create their own subdirectories and to organize their files accordingly. A tree is the most common directory structure. The tree has a root directory, and every file in the system has a unique path name. A directory (or subdirectory) contains a set of files or subdirectories.

PROGRAM

1. SINGLE LEVEL DIRECTORY ORGANIZATION

```
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir;
void main()
{
int i,ch;
char f[30];
clrscr();
dir.fcnt = 0;
printf("\nEnter name of directory -- ");
scanf("%s", dir.dname);
while(1)
{
printf("\n\n 1. Create File\t2. Delete File\t3. Search File \n 4. Display Files\t5. Exit\nEnter your choice --
");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\n Enter the name of the file -- ");
scanf("%s",dir.fname[dir.fcnt]);
dir.fcnt++;
break;
case 2: printf("\n Enter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is deleted ",f);
strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);
break;
}
}
if(i==dir.fcnt)
printf("File %s not found",f);
else
dir.fcnt--;
break;
case 3: printf("\n Enter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is found ", f);
break;
}
}
if(i==dir.fcnt)
printf("File %s not found",f);
break;
case 4: if(dir.fcnt==0)
printf("\n Directory Empty");
else
{
printf("\n The Files are -- ");
for(i=0;i<dir.fcnt;i++)
printf("\t%s",dir.fname[i]);
}
break;
default: exit(0);
}
}
getch();
}
```

OUTPUT:

```
Enter name of directory -- CSE

1. Create File
2. Delete File
3. Search File
4. Display Files
5. Exit
Enter your choice – 1
Enter the name of the file -- A

1. Create File
2. Delete File
3. Search File
4. Display Files
5. Exit
Enter your choice – 1
Enter the name of the file -- B

1. Create File
2. Delete File
3. Search File
4. Display Files
 5. Exit
Enter your choice – 1
Enter the name of the file -- C

1. Create File
2. Delete File
 3. Search File
4. Display Files
5. Exit
Enter your choice – 4
The Files are -- A B C

1. Create File
 2. Delete File
3. Search File
4. Display Files
 5. Exit
Enter your choice – 3
Enter the name of the file – ABC
File ABC not found

1. Create File
 2. Delete File
3. Search File
4. Display Files
5. Exit
Enter your choice – 2
Enter the name of the file – B
File B is deleted

1. Create File
2. Delete File
3. Search File
4. Display Files
5. Exit
 Enter your choice – 5
```

2. TWO LEVEL DIRECTORY ORGANIZATION

```
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir[10];
void main()
{
int i,ch,dcnt,k;
char f[30], d[30];
clrscr();
dcnt=0;
while(1)
{
printf("\n\n 1. Create Directory\t 2. Create File\t 3. Delete File");
printf("\n 4. Search File \t \t 5. Display \t 6. Exit \t Enter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\n Enter name of directory -- ");
scanf("%s",  dir[dcnt].dname);
dir[dcnt].fcnt=0;
dcnt++;
printf("Directory created");
break;
case 2: printf("\n Enter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",dir[i].fname[dir[i].fcnt]);
dir[i].fcnt++;
printf("File created");
break;
```

```
}
if(i==dcnt)
printf("Directory %s not found",d);
break;
case 3: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is deleted ",f);
dir[i].fcnt--;
strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
goto jmp;
}
}
printf("File %s not found",f);
goto jmp;
}
}
printf("Directory %s not found",d);
jmp : break;
case 4: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{

if(strcmp(d,dir[i].dname)==0)
{
printf("Enter the name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is found ",f);
goto jmp1;
}
}
printf("File %s not found",f);
goto jmp1;
}
}
printf("Directory %s not found",d);
jmp1: break;
case 5: if(dcnt==0)
printf("\nNo Directory's ");
else
{
printf("\nDirectory\tFiles");
for(i=0;i<dcnt;i++)
{
printf("\n%s\t\t",dir[i].dname);
for(k=0;k<dir[i].fcnt;k++)
printf("\t%s",dir[i].fname[k]);
}
}
break;
default:exit(0);
}
}
getch();
}
```

OUTPUT:

```
1. Create Directory
2. Create File
3. Delete File
4. Search File
5. Display
6. Exit
 Enter your choice -- 1
Enter name of directory -- DIR1
Directory created

1. Create Directory
2. Create File
3. Delete File
4. Search File
5. Display
6. Exit
 Enter your choice -- 1
Enter name of directory -- DIR2
Directory created

1. Create Directory
2. Create File
 3. Delete File
```

```
4. Search File
5. Display
6. Exit
Enter your choice -- 2
Enter name of the directory – DIR1
Enter name of the file -- A1
File created

1. Create Directory
 2. Create File
3. Delete File
4. Search File
 5. Display
 6. Exit
Enter your choice -- 2
Enter name of the directory – DIR1
Enter name of the file -- A2
File created

1. Create Directory
2. Create File
 3. Delete File
4. Search File
 5. Display
6. Exit
Enter your choice -- 2
Enter name of the directory – DIR2
Enter name of the file -- B1
File created

1. Create Directory
 2. Create File
 3. Delete File
4. Search File
5. Display
 6. Exit
Enter your choice -- 5
Directory Files
DIR1 A1 A2
DIR2 B1

1. Create Directory
2. Create File
3. Delete File
4. Search File
5. Display
6. Exit
```

```
Enter your choice -- 4
Enter name of the directory – DIR
Directory not found

1. Create Directory
2. Create File
3. Delete File
4. Search File
5. Display
6. Exit
Enter your choice -- 3
Enter name of the directory – DIR1
Enter name of the file -- A2
File A2 is deleted

1. Create Directory
2. Create File
3. Delete File
4. Search File
5. Display
6. Exit
Enter your choice – 6
```

3. HIERARCHICAL DIRECTORY ORGANIZATION

```
#include<stdio.h>
#include<graphics.h>
struct tree_element
{
char name[20];
int x,y,ftype,lx,rx,nc,level;
struct tree_element *link[5];
};
typedef struct tree_element
node; void main()
{
int gd=DETECT,gm;
node *root;
root=NULL;
clrscr();
create(&root,0,"root",0,639,320);
clrscr();
initgraph(&gd,&gm,"c:\\tc\\BGI");
display(root);
getch();
closegraph();
}
```

```
create(node **root,int lev,char *dname,int lx,int rx,int x)
{
int i,gap;
if(*root==NULL)
{
(*root)=(node *)malloc(sizeof(node));
printf("Enter name of dir/file(under %s) :",dname);
fflush(stdin);
gets((*root)->name);
printf("enter 1 for Dir/2 forfile :");
scanf("%d",&(*root)->ftype);
(*root)->level=lev;
(*root)->y=50+lev*50;
(*root)->x=x;
(*root)->lx=lx;
(*root)->rx=rx;
for(i=0;i<5;i++)
(*root)->link[i]=NULL;
if((*root)->ftype==1)
{
printf("No of sub directories/files(for %s):",(*root)->name); scanf("%d",&(*root)->nc);
if((*root)->nc==0)
gap=rx-lx;
else gap=(rx-lx)/(*root)->nc;
for(i=0;i<(*root)->nc;i++)
create(&((*root)->link[i]),lev+1,(*root)->name,lx+gap*i,lx+gap*i+gap,lx+gap*i+gap/2);
}
else (*root)->nc=0;
}
}
display(node *root)
{
int i;
settextstyle(2,0,4);
settextjustify(1,1);
setfillstyle(1,BLUE);
setcolor(14); if(root!=NULL)
{
for(i=0;i<root->nc;i++)
{
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
}
if(root->ftype==1) bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0); else
fillellipse(root->x,root->y,20,20);
outtextxy(root->x,root->y,root->name); for(i=0;i<root->nc;i++)
{
display(root->link[i]);
}
}
}
```

OUTPUT:

```
Enter Name of dir/file (under root): ROOT
Enter 1 for Dir / 2 For File : 1
No of subdirectories / files (for ROOT) :2
Enter Name of dir/file (under ROOT):USER 1
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for USER 1):1
Enter Name of dir/file (under USER 1):SUBDIR
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for SUBDIR):2
Enter Name of dir/file (under USER 1):
JAVA Enter 1 for Dir /2 for file:1
No of subdirectories /files (for JAVA): 0
Enter Name of dir/file (under SUBDIR):VB
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for VB): 0

Enter Name of dir/file (under ROOT):USER2
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for USER2):2
Enter Name of dir/file (under ROOT):A
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under USER2):SUBDIR 2
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for SUBDIR 2):2

Enter Name of dir/file (under SUBDIR2):PPL
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for PPL):2
Enter Name of dir/file (under PPL):B
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under PPL):C
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under SUBDIR):AI
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for AI): 2
Enter Name of dir/file (under AI):D
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under AI):E
```

Conclusion: Hence we studied file organization techniques.