

TPCT's
College of Engineering, Osmanabad

Laboratory Manual

OBJECT ORIENTED PROGRAMMING USING C++

For

Second Year Students

Manual Prepared by

Mr. T.K.Takbhate

Author COE, Osmanabad



TPCT's

College of Engineering

Solapur Road, Osmanabad

Department of Computer Science and Engineering

Vision of the Department:

To develop computer engineers with necessary analytical ability and human values who can creatively design, implement a wide spectrum of computer systems for welfare of the society.

Mission of the Department:

1. Preparing graduates to work on multidisciplinary platforms associated with their professional position both independently and in a team environment.
2. Preparing graduates for higher education and research in computer science and engineering enabling them to develop systems for society development.

College of Engineering

Technical Document

This technical document is a series of Laboratory manuals of Computer Science and Engineering Department and is a certified document of College of Engineering, Osmanabad. The care has been taken to make the document error-free. But still if any error is found. Kindly bring it to the notice of subject teacher and HOD.

Recommended by,

HOD

Approved by,

Principal

Copies:

1. Departmental Library
2. Laboratory
3. HOD
4. Principal

FOREWORD

It is my great pleasure to present this laboratory manual for Second year engineering students for the subject of Object Oriented Programming (Using C++)keeping in view the vast coverage required for visualization of concepts of OOP with simple language.

As a student, many of you may be wondering with some of the questions in your mind regarding the subject and exactly what has been tried is to answer through this manual.

As you may be aware that MGM has already been awarded with ISO 9000 certification and it is our endure to technically equip our students taking the advantage of the procedural aspects of ISO 9000 Certification.

Faculty members are also advised that covering these aspects in initial stage itself, will greatly relived them in future as much of the load will be taken care by the enthusiasm energies of the students once they are conceptually clear.

H.O.D.

CSE Dept.

LABORATORY MANUAL CONTENTS

This manual is intended for the Second year students of IT & CSE branches in the subject of OOP. This manual typically contains practical/Lab Sessions related OOP covering various aspects related the subject to enhanced understanding. Main aim of this course is to understand and solve logical & mathematical problems through C++ language. Strengthen knowledge of a procedural programming language. Further develop your skills in software development using a procedural language.

This course will also prepare students with the necessary programming background for Data Structures using C/C++ and Java programming courses.

We have made the efforts to cover various aspects of the subject covering these labs encompass the regular materials as well as some advanced experiments useful in real life applications. Programming aspects will be complete in it to make it meaningful, elaborative understandable concepts and conceptual visualization.

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus, as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Mr. T.K.Takbhate

SUBJECT INDEX

Lab No.	Index	Week involved
1	Develop programs to implement the concepts of classes and object, accessing members: e.g. <ol style="list-style-type: none"> a. Design an EMPLOYEE class to contain Data members: Employee_Number, Employee_Name, Basic_Salary, All_Allowances, IT, Net_Salary. Member functions: to read the data of an employee, to calculate Net_Salary and to print the values of all the data members. b. Design a STUDENT class c. Design Bank Account class. 	1-2
2	Design a program to Demonstrate concept of <ol style="list-style-type: none"> a. Arrays within a class, Arrays of Objects (e.g Develop program to calculate best of two class Test marks of six subjects of a student) b. Objects as Function Arguments, Returning Objects. (e.g. Complex number arithmetic) 	5-6
3	Design a C++ program for static members <ol style="list-style-type: none"> a. Design a class having static member function which has capability to display number of objects created. b. Design matrix and vector classes with static allocation and a friend function to do matrix-vector multiplication. 	7-8
4	Design a program to Demonstrate concept of constructor. Create a class called 'TIME' that has: <ol style="list-style-type: none"> a. three integer data members for hours, minutes and seconds b. constructor to initialize the object to zero c. constructor to initialize the object to some constant value d. member function to add two TIME objects e. member function to display time in HH:MM:SS format Write a main function to create two TIME objects, add them and display the result in HH:MM:SS format.	9-10
5	Design a program to demonstrate operator overloading for unary as well as binary operation. <ol style="list-style-type: none"> a. Design, develop and execute a program in C++ to create a class called DATE with methods to accept two valid dates in the form dd/mm/yy and to implement the following operations by overloading the operators + and -. After every operation, the results are to be displayed by overloading the operator <<. <ol style="list-style-type: none"> i. $\text{no_of_days} = d1 - d2$; where $d1$ and $d2$ are DATE objects, $d1 \geq d2$ and no_of_days is an integer. ii. $d2 = d1 + \text{no_of_days}$; where $d1$ is a DATE object and no_of_days is an integer. b. Design a class Complex which represents the Complex Number data type. Implement the following operations: 	10-11

	<ul style="list-style-type: none"> i. Constructor (including a default constructor which creates the complex number 0+0i). ii. Overloaded operator+ to add two complex numbers. iii. Overloaded operator* to multiply two complex numbers. iv. Overloaded << and >> to print and read Complex Numbers. 	
6	Design a program to demonstrate Single, multiple, multilevel, hybrid, hierarchical inheritance and Virtual base classes.	11-12
7	. Design a program to implement Array of pointers, pointer to functions, pointer to objects.	13-14
8	<p>Design a program to implement concept of Virtual functions:</p> <ul style="list-style-type: none"> a. Create a class called LIST with two pure virtual function store() and retrieve(). To store a value call store and to retrieve call retrieve function. Derive two classes stack and queue from it and override store and retrieve. b. Create a base class called 'SHAPE' having <ul style="list-style-type: none"> i. two data members of type double ii. member function get-data() to initialize base class data members iii. pure virtual member function display-area() to compute and display the area of the geometrical object. Derive two specific classes 'TRIANGLE' and 'RECTANGLE' from the base class. Using these three classes design a program that will accept dimension of a triangle / rectangle interactively and display the area. 	15-16
9	<p>File Handling</p> <ul style="list-style-type: none"> a. Develop a program to demonstrate Opening and Closing of file using constructors and open () function. b. Write a program to read the class object of student info such as name, age, gender, height and weight from the keyboard and to store them on a specified file using read() and write() functions. Again the same file is opened for reading and displaying the contents of the file on the screen. 	17-18
10	Develop a program to implement class and function template for stack and queue	19-20
11	Design a program to demonstrate the concepts of catching and throwing of an exception.	21-22
	Conduction of Viva-Voce Examinations	

1. Do's and Don'ts in the laboratory

- Make entry in the Log Book as soon as you enter the Laboratory.
- All the students should sit according to their roll numbers starting from their left to right.
- All the students are supposed to enter the terminal number in the log book.
- Do not change the terminal on which you are working.
- All the students are expected to get at least the algorithm of the program/concept to be implemented.
- Strictly observe the instructions given by the teacher/Lab Instructor.

2. Pre-lab (Introduction to C++)

Object oriented language was developed by Bjarne Stroustrup in 1979 at Bell Labs. As an enhancement to the C programming language and originally named "C with Classes". It was renamed to C++ in 1983. C++ is designed to be a statically typed general-purpose language that is as efficient and portable as C. C++ is designed to directly and comprehensively support multiple programming styles (procedural programming, data abstraction, object-oriented programming, and generic programming) C++ avoids features that are platform specific or not general purpose

The fundamental idea behind the object oriented language is to combine into a single unit of data and functions that operate on that data. Such unit is called an object. Object-oriented programming (OOP) is a programming paradigm that uses "objects" – data structures consisting of data fields and methods together with their interactions – to design applications and computer programs. Programming techniques may include features such as information hiding, data abstraction, encapsulation, modularity, polymorphism, and inheritance. It was not commonly used in mainstream software application development until the early 1990s.] Many modern programming languages now support OOP.

Some basic concept of object oriented programming language such as Class, object, data abstraction & Encapsulation, inheritance, polymorphism, dynamic binding, message passing

3. Lab Experiments:

Experiment No:-1

Aim:- Develop programs to implement the concepts of classes and object, accessing members: e.g. a. Design an EMPLOYEE class to contain Data members: Employee_Number, Employee_Name, Basic_Salary, All_Allowances, IT, Net_Salary. Member functions: to read the data of an employee, to calculate Net_Salary and to print the values of all the data members.

b. Design a STUDENT class

c. Design Bank Account class.

Objective:- At the end of this experiment, students should be able to understand following points:

1. Class and object.
2. Crating class and object.
3. Defining function in different ways.
4. Concept of Access Specifier like private, public etc.

Theory:- An Overview about Objects and Classes

In object-oriented programming language C++, the data and functions (procedures to manipulate the data) are bundled together as a self-contained unit called an *object*. A *class* is an extended concept similar to that of *structure* in C programming language, this class describes the data properties alone. In C++ programming language, *class* describes both the properties (data) and behaviors (functions) of objects. *Classes* are not *objects*, but they are used to instantiate *objects*.

A typical C++ program would contain four sections as shown in the Fig. 2.1. These sections may be placed in separate code files and then compiled independently.

Include Header Files
Class Declaration
Member Function Defination
Main Function Program

Fig:-2.1

Code:

b)

```
#include<iostream.H>
#include<conio.h>
class student
{
    private:
        char name[20],regd[10],branch[10];
        int sem;
    public:
        void input();
        void display();
};
void student::input()
{
    cout<<"Enter Name:";
    cin>>name;
    cout<<"Enter Regdno.:";
    cin>>regd;
    cout<<"Enter Branch:";
    cin>>branch;
    cout<<"Enter Sem:";
    cin>>sem;
}
void student::display()
{
    cout<<"\nName:"<<name;
    cout<<"\nRegdno.:"<<regd;
    cout<<"\nBranch:"<<branch;
    cout<<"\nSem:"<<sem;
}
int main()
{
    student s;
    s.input();
    s.display();
}
```

```
}
```

c)

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

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

```
class Bank
```

```
{
```

```
    char name[20];
```

```
    int acc_no;
```

```
    char acc_type[20];
```

```
    int bal;
```

```
    public:
```

```
    void getdata();
```

```
    void putdata();
```

```
    void create();
```

```
    void withdraw();
```

```
};
```

```
void Bank::getdata()
```

```
{
```

```
    cout<<"enter name";
```

```
    cin>>name;
```

```
    cout<<"enter account number";
```

```
    cin>>acc_no;
```

```
    cout<<"enter balance";
```

```
    cin>>bal;
```

```
    cout<<"enter account type";
```

```
    cin>>acc_type;
```

```
}
```

```
void Bank::putdata()
```

```
{
```

```
    cout<<"customer name="<<name<<endl;
```

```
    cout<<"account number="<<acc_no<<endl;
```

```
    cout<<"account type="<<acc_type<<endl;
```

```

        cout<<"balance="<<bal;
    }
void Bank::create()
{
    float amount;
    cout<<"enter ammount";
    cin>>amount;
    bal=bal+amount;
    cout<<"Your account is created by"<<amount<<"rupees"<<endl;
    cout<<"So,Your avaiable balance is "<<bal<<endl;
}
void Bank::withdraw()
{

    float ammount1;
    cout<<"Enter Ammount";
    cin>>ammount1;
    bal=bal-ammount1;
    cout<<"Your Account is debited by"<<ammount1<<"rupees"<<endl;
    cout<<"So,your available balance is "<<bal<<endl;
}
void main()
{

    clrscr();
    int choice;
    Bank b;
    b.getdata();
    b.create();
    b.withdraw();
    /*cout<<"1.create"<<endl;
    cout<<"2.withraw"<<endl;
    cout<<"enter choice";
    cin>>choice;
    switch(choice)

```

```
{
    case 1:b.create();
    break;
    case 2:b.withdraw();
    break;
    default:
    cout<<"wrong operation";
    break;
} */
b.putdata();
getch();
}
```

Result:-

b.

```
Enter Name:tushar
Enter Regdno.:1233
Enter Branch:cse
Enter Sem:3
```

```
Name:tushar
Regdno.:1233
Branch:cse
Sem:3
```

c.

```
enter nameRahul
enter account number12345598943
enter balance30000
enter account typecurrent
enter ammount200
Your account is created by200rupees
So,Your avaiable balance is 30200
Enter Ammount300
Your Account is debited by300rupees
So,your available balance is 29900
customer name=Rahul
account number=-7201
account type=current
balance=29900_
```

Experiment No:-2

Aim:- Design a program to Demonstrate concept of

a. Arrays within a class, Arrays of Objects (e.g Develop program to calculate best of two

class Test marks of six subjects of a student)

b. Objects as Function Arguments, Returning Objects. (e.g. Complex number arithmetic)

Objective:-

An array is an object that contains collections of other objects. The Array class contains a number of methods specifically designed to ease the creation and manipulation of arrays within Objective-C programs. Unlike some object oriented programming languages (C# being one example), the objects contained in an array do not all have to be of the same type.

Theory:-

a) Firstly array of object program more than information are stored in one student. So firstly maximum size can declared before program . program to input name,rollno,total,percentage marks of total of a student and calculate total, percentage of all subjects. How to calculate total percentage in C++ programming. First you have to create student class with four datamember such as name,rollno,total,percentage and two member function getdetails() , putdetails().The getdetails method Is defined the information about the student and putdetails method is defined information is print the output screen.

So last main() function is called and array of objects creation .Enter details of student in getdetails function called and other putdetails function is called the output is print the screen in array size.

b) The first method is called pass by value a copy of the object is passed.

To the function any change made to the object inside the function do not affect the object used to the call the function.

The second method is called pass by reference when an address of the object is passed the called the function works directly on the actual address object used in the call.

First is to declare header file then create a class and member function then two value declare we use a function puttime() is display the time is hour and minute. Then we can calculate the minute and hour of the two object we use the formula minute is first object and minute is second object $hour = \text{minute} / 60$ $minute = \text{minute} \% 60$ hour is first object and hour is second object. The time is a class declare three object first object, second object and third object. Object is the first object then pass by the value (2,45) second object pass by the value(3,30)and then third is the addition of the first object and second object. Using the puttime() function display the details of all the three object.

Code:

a) Array of object

```
#include <iostream>
#include<conio.h>
#define MAX 10
class student
{
private:
    char name[30];
    int rollNo;
    int total;
    float perc;
public:
    //member function to get student's details
    void getDetails(void);
    //member function to print student's details
    void putDetails(void);
};
//member function definition, outside of the class
void student::getDetails(void)
{
    cout << "Enter name: " ;
    cin >> name;
```

```

    cout << "Enter roll number: ";
    cin >> rollNo;
    cout << "Enter total marks outof 500: ";
    cin >> total;
    perc=(float)total/500*100;
}
//member function definition, outside of the class
void student::putDetails(void){
    cout << "Student details:\n";
    cout << "Name:"<< name << ",Roll Number:" << rollNo << ",Total:" <<
total << ",Percentage:" << perc;
}
int main()
{
    student std[MAX];    //array of objects creation
    int n,loop;
    cout << "Enter total number of students: ";
    cin >> n;

    for(loop=0;loop< n; loop++){
        cout << "Enter details of student " << loop+1 << ":\n";
        std[loop].getDetails();
    }

    cout << endl;

    for(loop=0;loop< n; loop++){
        cout << "Details of student " << (loop+1) << ":\n";
        std[loop].putDetails();
    }

    return 0;
}

```

b) Object as Function argument

```
#include<iostream.h>
#include<conio.h>
class time
{
    int hour;
    int minute;
public:
    void gettime(int h,int m)
    {
        hour=h;
        minute=m;
    }
    void puttime()
    {
        cout<<"Hour="<<hour<<endl;
        cout<<"Minute="<<minute<<endl;
    }
    void sum(time,time);
};
void time::sum(time t1,time t2)
{
    minute=t1.minute+t2.minute;
    hour=minute/60;
    minute=minute%60;
    hour=hour+t1.hour+t2.hour;
}
int main()
{
    clrscr();
    time T1,T2,T3;
    T1.gettime(2,45);
    T2.gettime(3,30);

    T3.sum(T1,T2);
```

```
cout<<"T1=";  
T1.puttime();
```

```
cout<<"T2=";  
T2.puttime();
```

```
cout<<"T3=";  
T3.puttime();  
getch();  
return 0;
```

Result:

a.

```
Enter total number of students: 3
Enter details of student 1:
Enter name: pooja
Enter roll number: 54
Enter total marks outof 500: 420
Enter details of student 2:
Enter name: jyoti
Enter roll number: 61
Enter total marks outof 500: 430
Enter details of student 3:
Enter name: megha
Enter roll number: 14
Enter total marks outof 500: 430
```

b.

```
T1=Hour=2
Minute=45
T2=Hour=3
Minute=30
T3=Hour=6
Minute=15
-
```

Experiment No:-3

Aim:- Design a C++ program for static members

- a. Design a class having static member function which has capability to display number of objects created.
- b. Design matrix and vector classes with static allocation and a friend function to do matrix-vector multiplication

Objective:- We can define class members static using static keyword. When we declare a member of a class as static it means no matter how many objects of the class are created, there is only one copy of the static member.

A static member is shared by all objects of the class. All static data is initialized to zero when the first object is created, if no other initialization is present. We can't put it in the class definition but it can be initialized outside the class as done in the following example by redeclaring the static variable, using the scope resolution operator :: to identify which class it belongs to.

Theory:-

a)Static data member:

These are the instance variable or class variable. The memory for these elements are allocated when object is created first time. These variables are bind with class name & object name. The static variable defines the global data for more than one object.

In above program class "Circle" is defined, it contains private data member i.e. "radius" and one static data member i.e. "PI".

It also contains public member function i.e. "set()" & "area()". "set()" member function is used to get the radius of the circle. "area()" member function calculate the area of the circle. In above program the static variable is initialize out side of the class. In whole program the value of static variable is not change.

In main function the object of class is created and the set() & area() member functions are access with the help of class object.

b)Static member function:

In above program the class "A" is define, it contain the "x" private data member & "y" is the static data member & fun1() member function to get value of "x" & "y". It also contain the Amz() static member function which can access only the static member of the class

In above program the object of class is created to access the data member and member function, by using the object Fun1() is accessed.

The static member function is access with the help of class name i.e. A::Amz();

Code:

a) Static data member

```
#include<iostream.h>
#include<conio.h>
class circle
{
    float r;
    static float pi;
public:
    void set(float x)
    {
        r=x;
    }
    float find()
    {
        return pi*r*r;
    }
};

float circle::pi=3.147;

void main()
{
    clrscr();
    circle c1;
    c1.set(1.3);
    float a=c1.find();
    cout<<"area of circle="<<a<<endl;
    getch();
}
```

b) Static member function

```
#include<iostream.h>
#include<conio.h>
class A
{
    int x;
    static int y;
```

```
public:
    void fun1()
    {
        cout<<"x="<<x;
        cout<<"y="<<y;
    }
    static void fun2()
    {
        cout<<"y="<<y;
    }
};

int A::y=5;

void main()
{
    clrscr();
    A::fun2();
    getch();
}
```

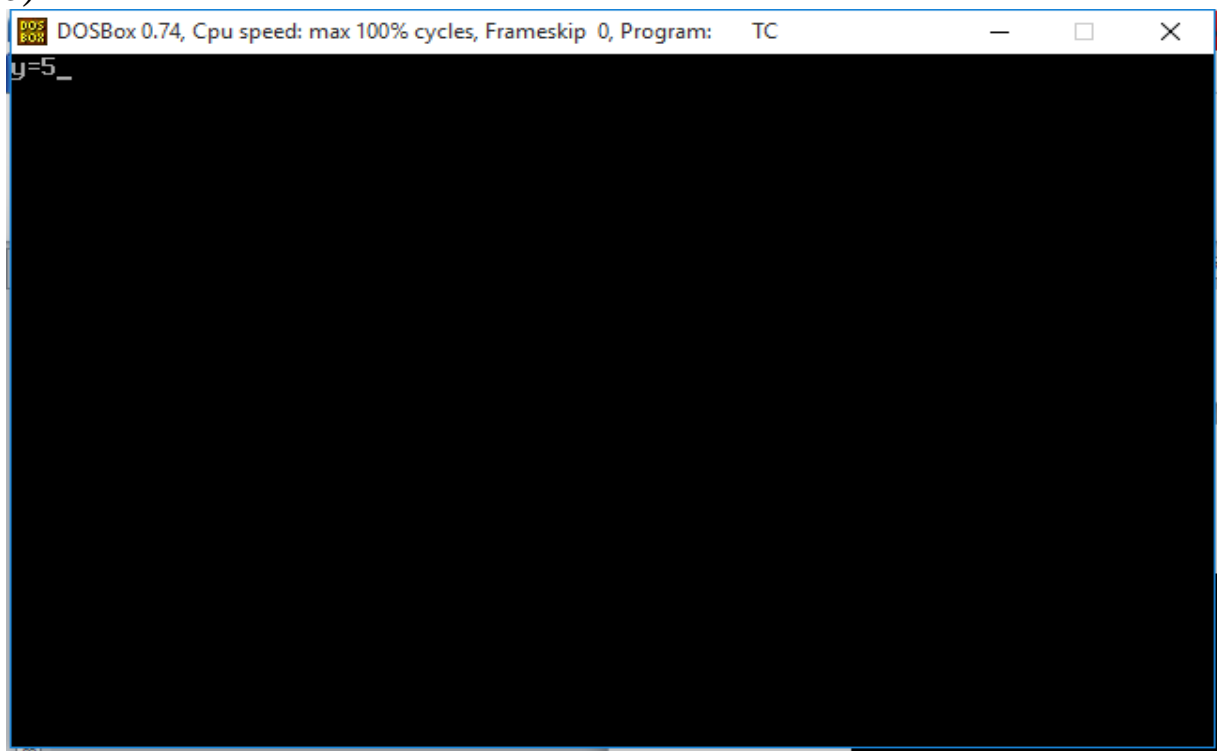

Result:-

a)

```
area of circle=5.31843
```

A screenshot of a DOSBox window. The window title bar reads "DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC". The main area of the window is black, with the text "area of circle=5.31843" displayed in white at the top left. A small white cursor is visible on the line below the text.

b)



Experiment No:-4

Aim:- Design a program to Demonstrate concept of constructor. Create a class called 'TIME' that has:

- a. three integer data members for hours, minutes and seconds
- b. constructor to initialize the object to zero
- c. constructor to initialize the object to some constant value
- d. member function to add two TIME objects
- e. member function to display time in HH:MM:SS format

Write a main function to create two TIME objects, add them and display the result in HH:MM:SS format.

Objective:-

The constructor name must be equal to class name.

The constructor is declared in public section of a class.

The constructor is defined without any return-type not even void.

Constructor by calling the constructor explicitly and implicitly.

Theory:-

This program is concept of constructor. First create a class TIME .it has three data members for hours, minutes, seconds.then after default constructor is called and initialize the hours, minutes and seconds is declared zero.

Next one parameterized constructor is called with parameter and initialize the some constant value is defined.

Then again additional constructor in a member function is passing with two objects. After the main () function two objects are declared is implicitly called with value, and add the two TIME objects. And last one display () function call HH:MM:SS format in main () function.

Code:-

```
#include<iostream.h>
#include<conio.h>
class TIME
{
int hours;
int minutes;
```

```

int seconds;
public:
TIME()
{
hours=00;
minutes=00;
seconds=00;
}
void display()
{
cout<<hours<<":"<<minutes<<":"<<seconds<<endl;
}
TIME(int x,int y,int z)
{
hours=x;
minutes=y;
seconds=z;
}
TIME(TIME t1,TIME t2)
{
seconds=t1.seconds+t2.seconds;
minutes=seconds/60;
seconds=seconds%60;
minutes=minutes+t1.minutes+t2.minutes;

minutes=t1.minutes+t2.minutes;
hours=minutes/60;
minutes=minutes%60;
hours=hours+t1.hours+t2.hours;
}

};
void main()
{
clrscr();

```

```
TIME t;
cout<<"a) First initialize three data members Hour,Minutes,Seconds"<<endl;
cout<<"b) constructor initialize the object is zero"<<endl;
t.display();
cout<<"c) constructor to initialize the some object consyant value"<<endl;
TIME t1(12,00,00);
t1.display();
TIME t2(11,34,50);
TIME t3(11,34,50);
TIME t4(t2,t3);
t4.display();
getch();
}
```

Result:-

```
a) First initialize three data members Hour,Minutes,Seconds  
b) constructor initialize the object is zero  
0:0:0  
c) constructor to initialize the some object consyant value  
12:0:0  
23:8:40  
-
```

Experiment No:-5

Aim:- Design a program to demonstrate operator overloading for unary as well as binary operation.

a. Design, develop and execute a program in C++ to create a class called DATE with methods to accept two valid dates in the form dd/mm/yy and to implement the following operations by overloading the operators + and -. After every operation, the results are to be displayed by overloading the operator <<.

i. $\text{no_of_days} = d1 - d2$; where $d1$ and $d2$ are DATE objects, $d1 \geq d2$ and no_of_days is an integer.

ii. $d2 = d1 + \text{no_of_days}$; where $d1$ is a DATE object and no_of_days is an integer.

b. Design a class Complex which represents the Complex Number data type. Implement the following operations:

i. Constructor (including a default constructor which creates the complex number $0+0i$).

ii. Overloaded operator+ to add two complex numbers.

iii. Overloaded operator* to multiply two complex numbers.

iv. Overloaded << and >> to print and read Complex Numbers.

Objective:-

To define the Rational class to represent rational numbers

- To understand in general how an operator can be overloaded in C++ .
- To know how to overload the relational operators and arithmetic operators.
- To know how to overload the shorthand operators.
- To know how to overload the array subscript operator [].
- To know how to overload the unary operators .
- To know how to overload the prefix and postfix ++ and -- operators .
- To know how to overload the stream insertion and extraction operators << and >> .
- To define operator functions to perform object conversion .
- To overload the = operator to perform a deep copy.

Theory:-

a) We develop a program in c++ to create a class called DATE .class DATE accept the in the formate of dd/mm/yy .

Create a class DATE with data members i.e dd,mm,yy,days and also cteate the member functions.

The putdata() function is used to assign the date.we assign the value to dd,mm,yy variable i.e d,m,y Which is given in the argument of the putdata() function.

Then we use the calculate the no of days in the given two date i.e. no_of_days=d1-d2; where d1 and d2 are DATE class objects and d1 >=d2 and no_of _days are integer no.

For that we use the uniry operator - to overload the function.in this function we create the object of DATE class i.e. d3 which is local object the we calculate the different between two date

```
d3.dd=d1.dd-d2.dd;
d3.mm=d1.mm-d2.mm;
d3.yy=d1.yy-d2.yy;
days=d3.dd+(d3.mm*30)+(d3.yy*365);
```

then return the d3 object.

Then we calculate the d2=d1+no_of_days ,where d1 is a date object and no_of_days is an integer .For that we use the unary + operator overloading function.In this function we also create the local object of DATE class i.e. d3 .we use following operation to calculate the numbers of days.

```
D3.dd=d1.dd+(days%30);
If(d3.dd>30)
{
    D3.mm=d3.dd/30;
    D3.dd=d3.dd%30;
}
D3.mm=d3.mm+(days/30);
D3.yy=d1.yy+(days/365);
```

Then return the d3 object to main function.

Then we create the display function to display the date of object in the formate of dd/mm/yy.

All these functions are includeing the in the class.

Create the main function.in main function create the four object of DATE class.

Call the putdata() function with argument 21,02,18 of first object of DATE i.e. d1.the also call the putdata() function with argument 22,04,18 of first object of DATE i.e. d1 .

Call the uniry - operator function to calculate the no of days.we use

$D3=d2-d1;$

Call the uniry + operator function to calculate the date of second object .we use

$D3=d2+d1;$

Call the display function to display the date of object of all four object.

b) Binary operators overloaded by means member function take one formal argument which is the value to the right of the operator.

In overloading of binary operator, left hand operand is used to invoke the operator function and write an operator is passed as an argument as explicitly.

In operator overloading program one class is given, whose name is complex .In class two datamember 'x' and 'y' .this are the float type datamember.In public section one default constructor is given and another constructor is parameterized constructor. Arguments in parameterized constructor are formal. These formal arguments are "real" and "img" .here value of formal parameter assign to actual parameter, actual parameter are 'x' and 'y'. After that in program to overload binary operator "complex operator +()" is used. Which return the object so its returntype is class that is "complex".

"Void display()" is used here for display function which is declared in class.

Outside the class binary operator is defined .in that definition temporary object "temp" is created. This temporary object return the value. Display function is also defined outside the class for display the value of 'x' and 'y'.In main function three object created 'c1','c2','c3'.

In 'c1' object these add two values 2.5 and 3.5.and in 'c2' object these add two values 3.5 and 4.5. In 'c3' object addition of two object 'c1' and 'c2'.after that display the value of 'c1','c2'and 'c3'.

Code:a)

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

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

```
class date
```

```
{
```



```

int dd,mm,yy,days;
public:
    void putdata(int d,int m,int y)
    {
        dd=d;
        mm=m;
        yy=y;
    }
    date operator -(date d1)
    {

        date d3;
        d3.dd=dd-d1.dd;
        d3.mm=mm-d1.mm;
        d3.yy=yy-d1.yy;
        days=d3.dd+d3.mm*30+d3.yy*365;
        return d3;
    }
    date operator +(date d1)
    {
        date d3;
        cout<<"\ndays="<<days;
        d3.dd=d1.dd+(days%30);
        if(d3.dd>30)
        {
            d3.mm=d3.dd/30;
            d3.dd=d3.dd%30;
        }
        d3.mm=d3.mm+(days/30);
        d3.yy=d1.yy+(days/356);
        return d3;
    }
    void display()
    {
        cout<<"\nDate is : "<<dd<<"/"<<mm<<"/"<<yy;
    }

```

```

        }

};

void main()
{
    clrscr();
    date d1,d2,d3,d4;
    d1.putdata(21,02,18);
    d2.putdata(22,04,18);
    d1.display();
    d2.display();

    d3=d2-d1;
    d4=d2+d1;
    d3.display();
    d4.display();
    getch();
}

```

b)

```

#include<iostream.h>
#include<conio.h>
class complex
{
    private:
    float x;
    float y;
    public:
    complex()
    {

    }
    complex (float real, float img)
    {
        x=real;
        y=img;
    }
}

```

```

        complex operator+(complex);
        void display(void);
};
complex complex::operator+(complex c)
{
    complex temp;
    temp.x=x+c.x;
    temp.y=y+c.y;
    return temp;
}
void complex::display(void)
{
    cout<<x<<" +j" <<y<<endl;
}
int main()
{
    clrscr();
    complex c1,c2,c3;
    c1=complex(2.5,3.5);
    c2=complex(1.6,2.7);
    c3=c1+c2;

    cout<<"c1=";
    c1.display();

    cout<<"c2=";
    c2.display();

    cout<<"c3=";
    c3.display();
    getch();
    return 0;
}

```

Result:-

a.

```
Date is :21/2/18  
Date is :22/4/18  
days=61  
Date is :1/2/0  
Date is :22/4/18_
```

b.

```
c1=2.5+j3.5  
c2=1.6+j2.7  
c3=4.1+j6.2
```

Experiment No:-6

Aim:- Design a program to demonstrate Single, multiple, multilevel, hybrid, hierarchical inheritance and Virtual base classes.

Objective:- One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the base class, and the new class is referred to as the derived class.

Theory:-

Single inheritance

- The derivation of class from only one base class and one derived class is called as single inheritance.
- Single inheritance is the ability of a derived class to inherit the member functions and variables of the existing base class.
- The declaration of a singly derived class is as that same of an ordinary class.
- A derived class consist of following components.
 - 1.The keyword class.
 - 2.The name of derived class.
 - 3.A single colon.
 - 4.Type of derivation.(private,public, protected)
 - 5.The name of base class or parent class.

In single inheritance program there are two header file are used. Assume the base class i.e student. Store the data members are name and course name. In public section there are two member functions are used i.e getstudent() and printstudent().

Assume the derived class i.e mark and base class is accessed with visibility mode.

- Class mark: public student

In derived class i.e in mark class there are three variables are declared.

Data members used as follows:

- `Int sub1,sub2,sub3;`

And two member functions are defined in derived class. i.e `getmarks()` and `findresult()`. In `getmarks` member function we already read three subjects marks. i.e `sub1,sub2,sub3` which is integer type. In `findresult()` function we have to use if else loop, which decides the students are “pass” or “fail”. In `main()` function we have to create an object of mark class. i.e `stud` is an object which is used to access member function.

Syntax of accessing member function using object is as follows:

- `Object name. member_function();`

In this program we create object for four member functions.

i.e. for below functions

1.`getstudent();`

2.`printstudent();`

3.`getmark();`

4.`findresult();`

Multilevel inheritance:

The derivation of a class from another derived class is called as multilevel.

The derivation of a class which is derived class is called as multilevel inheritance.

The above relationship base class_intermediate class_Derived class is known as multilevel inheritance path.

From above fig. It is understood that firstly three class are define.then First student class is Base class then another second class test is Intermediate ,and last result is Derived class.

Base class student in two member function are define i.e `getnumber()` and `putnumber()`.then access the base class student is class test. Then using in test class two function i.e `getmark()`and `putmark()`.then access the intermediate test class.the result class in using total of subjects and use the `display()`function.

Then using `main()` function in create object of result and pass the `getnumber()`, `putnumber()`, `getmark()`,`putmark()`,`display()` function is called and the result is printed.

Code:

```
a)
#include<iostream.h>
#include<conio.h>
class student
{
    char name[10];
    char course[10];
public:
    void getstudent()
    {
        cout<<"Input name and course";
        cin>>name>>course;
    }
    void printstudent()
    {
        cout<<name<<endl;
        cout<<course<<endl;
    }
};
class mark:public student
{
    int sub1,sub2,sub3;
public:
    void getmark()
    {
        cout<<"Input marks"<<endl;
        cin>>sub1>>sub2>>sub3;
    }
    void findresult()
    {
        if(sub1<40||sub2<40||sub3<40)
            cout<<"Fail";
        else
            cout<<"Pass";
    }
};
int main()
{
    clrscr();
    mark stud;
```

```
    stud.getstudent();
    stud.printstudent();
    stud.getmark();
    stud.findresult();
    getch();
    return 0;
}
```

b)

```
#include<iostream.h>
#include<conio.h>
class student
{
protected:
    int rollno;
public:
    void getstudent(int a)
    {
        rollno=a;
    }
    void putstudent()
    {
        cout<<rollno<<endl;
    }
};
class test:public student
{
protected:
    float sub1,sub2;
public:
    void getmark(float x,float y)
    {
        sub1=x;
        sub2=y;
    }
    void putmark()
    {
        cout<<"mark of subject 1"<<sub1<<endl;
        cout<<"mark of subject 2"<<sub2<<endl;
    }
}
```



```
};
class result:public test
{
    float total;
public:
    void display()
    {
        total=sub1+sub2;
        putstudent();
        putmark();
        cout<<"Total="<<total<<endl;
    }
};
int main()
{
    clrscr();
    result stud;
    stud.getstudent(111);
    stud.getmark(75.00,59.50);
    stud.display();
    getch();
    return 0;
}
```

Result:-
a)

```
Input name and course
tushar
C++
tushar
C++
Input marks
45 56 78
Pass_
```

b)

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
111
mark of subject 175
mark of subject 259.5
Total=134.5
```

Experiment No:-7

Aim:- Design a program to implement Array of pointers, pointer to functions, pointer to objects.

Objective:- Unlike normal pointers, a function pointer points to code, not data. Typically a function pointer stores the start of executable code.

Unlike normal pointers, we do not allocate de-allocate memory using function pointers.

A function's name can also be used to get functions' address. For example, in the below program, we have removed address operator '&' in assignment. We have also changed function call by removing *, the program still works.

Theory:-

Array of pointer

Pointer are the variables that hold address. Not only can pointers store address of a single variable, it can also store address of cells of an array.

Suppose, pointer needs to point to the fourth element of an array, that is, hold address of fourth array element in above case.

Since ptr points to the third element in the above example, ptr + 1 will point to the fourth element.

You may think, ptr + 1 gives you the address of next byte to the ptr. But it's not correct.

This is because pointer ptr is a pointer to an int and size of int is fixed for a operating system (size of int is 4 byte of 64-bit operating system). Hence, the address between ptr and ptr + 1 differs by 4 bytes.

If pointer ptr was pointer to char then, the address between ptr and ptr + 1 would have differed by 1 byte since size of a character is 1 byte.

Pointer to functions

Function Pointer Syntax

The syntax for declaring a function pointer might seem messy at first, but in most cases it's really quite straight-forward once you understand what's going on. Let's look at a simple example:

```
void (*foo) (int);
```

In this example, foo is a pointer to a function taking one argument, an integer, and that returns void. It's as if you're declaring a function called "*foo", which takes an int and returns void; now, if *foo is a function, then foo must be a pointer to a function. (Similarly, a declaration like int *x can be read as *x is an int, so x must be a pointer to an int.)

The key to writing the declaration for a function pointer is that you're just writing out the declaration of a function but with (*func_name) where you'd normally just put func_name.

To initialize a function pointer, you must give it the address of a function in your program. The syntax is like any other variable:

Pointers to Objects

A variable that holds an address value is called a pointer variable or simply pointer.

Pointer can point to objects as well as to simple data types and arrays.

sometimes we dont know, at the time that we write the program , how many objects we want to creat. when this is the case we can use new to creat objects while the program is running. new returns a pointer to an unnamed objects. lets see the example of student that wiil clear your idea about this topic

Code:a)

```
#include <iostream.h>
#include<conio.h>
int main()
{
    float arr[5];
    float *ptr;
    cout << "Displaying address using arrays: " << endl;
    for (int i = 0; i < 5; ++i)
    {
        cout << "&arr[" << i << "] = " << &arr[i] << endl;
    }
    // ptr = &arr[0]
    ptr = arr;
    cout<<"\nDisplaying address using pointers: "<< endl;
    for (int i = 0; i < 5; ++i)
    {
        cout << "ptr + " << i << " = "<< ptr + i << endl;
    }

    return 0;
}
```

b)

```
#include <iostream.h>
#include<conio.h>
void my_int_func(int x)
{
    Cout<<"x="<<x;
}

int main()
{
    void (*foo)(int);
    /* the ampersand is actually optional */
    foo = &my_int_func;
    getch();
    return 0;
}
```

c)

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

class student
{
private:
    int rollno;
    string name;
public:
    student():rollno(0),name("")
    {}
    student(int r, string n): rollno(r),name (n)
    {}
    void get()
    {
        cout<<"enter roll no";
        cin>>rollno;
```

```
        cout<<"enter name";
        cin>>name;
    }
    void print()
    {
        cout<<"roll no is "<<rollno;
        cout<<"name is "<<name;
    }
};
void main ()
{
    student *ps=new student;
    (*ps).get();
    (*ps).print();
    delete ps;
}
```

Result:-

a.

```
&arr[1] = 0x8f42ffe4
&arr[2] = 0x8f42ffe8
&arr[3] = 0x8f42ffec
&arr[4] = 0x8f42fff0

Displaying address using pointers:
ptr + 0 = 0x8f42ffe0
ptr + 1 = 0x8f42ffe4
ptr + 2 = 0x8f42ffe8
ptr + 3 = 0x8f42ffec
ptr + 4 = 0x8f42fff0
Displaying address using arrays:
&arr[0] = 0x8f42ffe0
&arr[1] = 0x8f42ffe4
&arr[2] = 0x8f42ffe8
&arr[3] = 0x8f42ffec
&arr[4] = 0x8f42fff0

Displaying address using pointers:
ptr + 0 = 0x8f42ffe0
ptr + 1 = 0x8f42ffe4
ptr + 2 = 0x8f42ffe8
ptr + 3 = 0x8f42ffec
ptr + 4 = 0x8f42fff0
```

c.

```
enter roll no 12
enter name tushar
roll no is 12name is tushar_
```

Experiment No:-8

Aim:- Design a program to implement concept of Virtual functions:

- a. Create a class called LIST with two pure virtual function store() and retrieve(). To store a value call store and to retrieve call retrieve function. Derive two classes stack and queue from it and override store and retrieve.
- b. Create a base class called 'SHAPE' having
 - i. two data members of type double
 - ii. member function get-data() to initialize base class data members
 - iii. pure virtual member function display-area() to compute and display the area of the geometrical object. Derive two specific classes 'TRIANGLE' and 'RECTANGLE' from the base class. Using these three classes design a program that will accept dimension of a triangle / rectangle interactively and display the area.

Objective:- Pure virtual Functions are virtual functions with no definition. They start with virtual keyword and ends with = 0. Here is the syntax for a pure virtual function,

```
virtual void f() = 0;
```

Theory:- Sometimes implementation of all function cannot be provided in a base class because we don't know the implementation. Such a class is called abstract class. For example, let Shape be a base class. We cannot provide implementation of function draw() in Shape, but we know every derived class must have implementation of draw(). Similarly an Animal class doesn't have implementation of move() (assuming that all animals move), but all animals must know how to move. We cannot create objects of abstract classes.

A pure virtual function (or abstract function) in C++ is a virtual function for which we don't have implementation, we only declare it. A pure virtual function is declared by assigning 0 in declaration. See the following example.

```
// An abstract class
class Test
{
    // Data members of class
public:
    // Pure Virtual Function
    virtual void show() = 0;
```



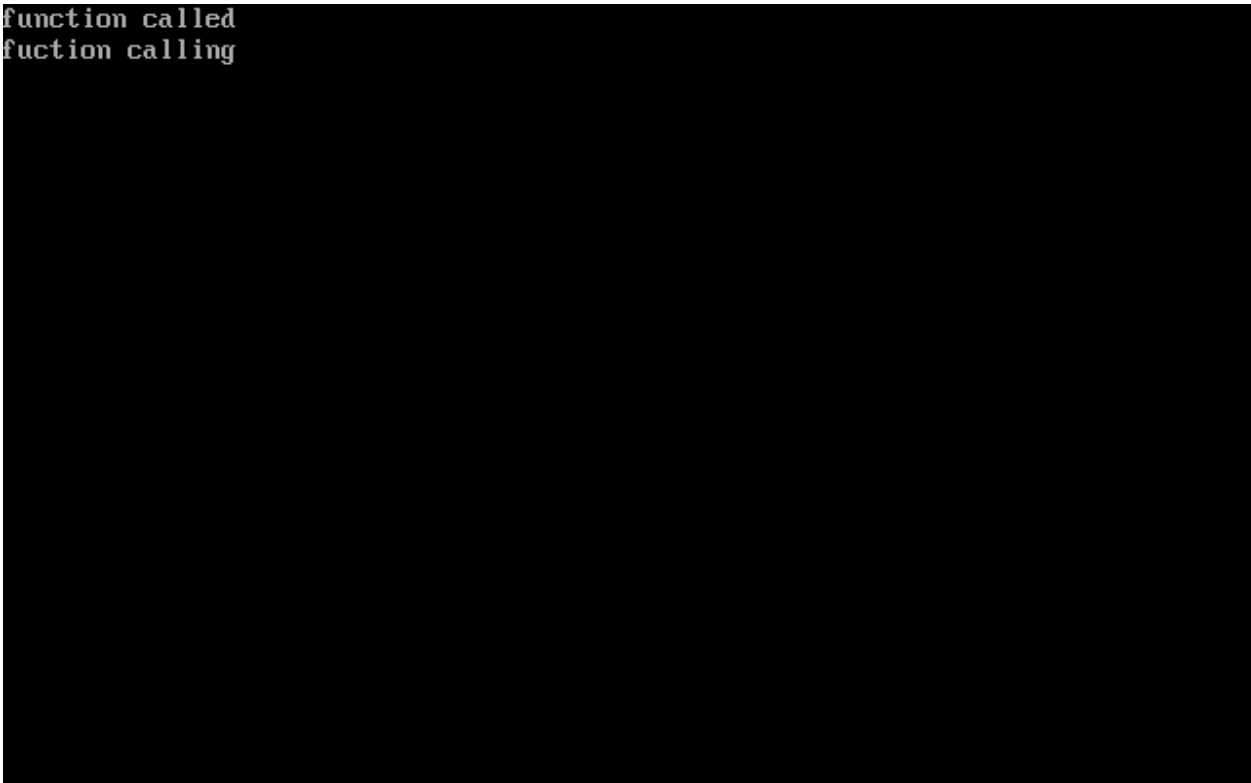
```

    /* Other members */
};
Code:
#include<iostream.h>
#include<conio.h>
class LIST
{
    public:
    virtual void store()=0;
    virtual void retrieve()=0;
};
class B:public LIST
{
    public:
    void store()
    {
        cout<<"function called"<<endl;
    }
    void retrieve()
    {
        cout<<"fuction calling"<<endl;
    }
};
void main()
{
    clrscr();
    B obj;
    obj.store();
    obj.retrieve();
    getch();
}

```

Result:-

```
function called  
function calling
```



Experiment No:-9

Aim:- File Handling

- a. Develop a program to demonstrate Opening and Closing of file using constructors and open () function.
- b. Write a program to read the class object of student info such as name, age, gender, height and weight from the keyboard and to store them on a specified file using read() and write() functions. Again the same file is opened for reading and displaying the contents of the file on the screen.

Objective:- For permanent storage. The transfer of input - data or output - data from one computer to another can be easily done by using files.

For read and write from a file you need another standard C++ library called fstream

Theory:- So far, we have been using the iostream standard library, which provides cin and cout methods for reading from standard input and writing to standard output respectively.

This tutorial will teach you how to read and write from a file. This requires another standard C++ library called fstream, which defines three new data types

Opening a File

A file must be opened before you can read from it or write to it. Either ofstream or fstream object may be used to open a file for writing. And ifstream object is used to open a file for reading purpose only.

Following is the standard syntax for open() function, which is a member of fstream, ifstream, and ofstream objects.

```
void open(const char *filename, ios::openmode mode);
```

Here, the first argument specifies the name and location of the file to be opened and the second argument of the open() member function defines the mode in which the file should be opened.

Closing a File

When a C++ program terminates it automatically flushes all the streams, release all the allocated memory and close all the opened files. But it is always a good

practice that a programmer should close all the opened files before program termination.

Following is the standard syntax for `close()` function, which is a member of `fstream`, `ifstream`, and `ofstream` objects.

```
void close();
```

Writing to a File

While doing C++ programming, you write information to a file from your program using the stream insertion operator (`<<`) just as you use that operator to output information to the screen. The only difference is that you use an `ofstream` or `fstream` object instead of the `cout` object.

Reading from a File

You read information from a file into your program using the stream extraction operator (`>>`) just as you use that operator to input information from the keyboard. The only difference is that you use an `ifstream` or `fstream` object instead of the `cin` object.

Code:

```
#include <fstream.h>
#include <iostream.h>
int main ()
{
    char data[100];

    // open a file in write mode.
    ofstream outfile;
    outfile.open("afile.dat");

    cout << "Writing to the file" << endl;
    cout << "Enter your name: ";
    cin.getline(data, 100);

    // write inputted data into the file.
    outfile << data << endl;
```

```
cout << "Enter your age: ";
cin >> data;
cin.ignore();

// again write inputted data into the file.
outfile << data << endl;

// close the opened file.
outfile.close();

// open a file in read mode.
ifstream infile;
infile.open("afile.dat");

cout << "Reading from the file" << endl;
infile >> data;

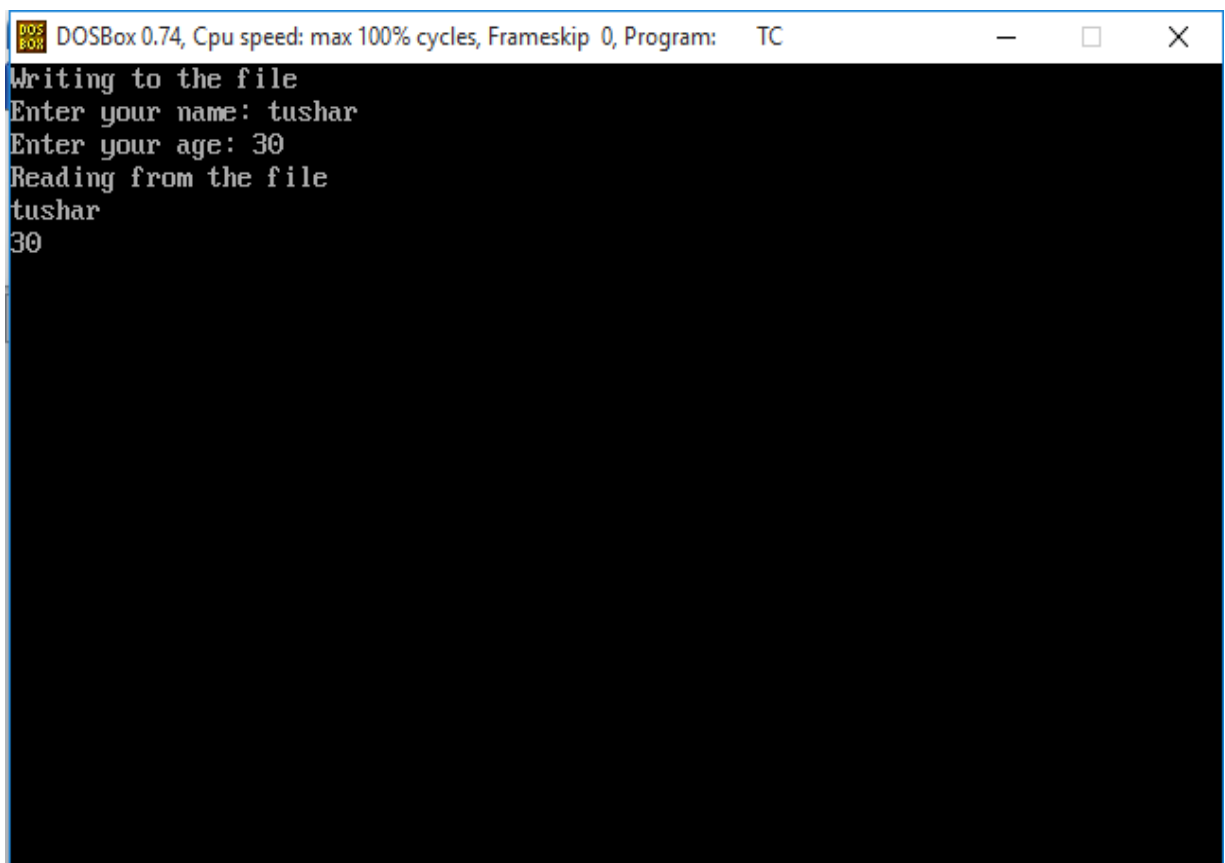
// write the data at the screen.
cout << data << endl;

// again read the data from the file and display it.
infile >> data;
cout << data << endl;

// close the opened file.
infile.close();

return 0;
}
```

Result:-



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Writing to the file
Enter your name: tushar
Enter your age: 30
Reading from the file
tushar
30
```

Experiment No:-10

Aim:- Develop a program to implement class and function template for stack and queue

Objective:- Template is simple and yet very powerful tool in C++. The simple idea is to pass data type as a parameter so that we don't need to write same code for different data types

Theory:-

Templates are expanded at compiler time. This is like macros. The difference is, compiler does type checking before template expansion. The idea is simple, source code contains only function/class, but compiled code may contain multiple copies of same function/class.

Many C++ programs use common data structures like stacks, queues and lists. A program may require a queue of customers and a queue of messages. One could easily implement a queue of customers, then take the existing code and implement a queue of messages. The program grows, and now there is a need for a queue of orders. So just take the queue of messages and convert that to a queue of orders (Copy, paste, find, replace????). Need to make some changes to the queue implementation? Not a very easy task, since the code has been duplicated in many places. Re-inventing source code is not an intelligent approach in an object oriented environment which encourages re-usability. It seems to make more sense to implement a queue that can contain any arbitrary type rather than duplicating code. How does one do that? The answer is to use type parameterization, more commonly referred to as templates.

C++ templates allow one to implement a generic Queue<T> template that has a type parameter T. T can be replaced with actual types, for example, Queue<Customers>, and C++ will generate the class Queue<Customers>. Changing the implementation of the Queue becomes relatively simple. Once the changes are implemented in the template Queue<T>, they are immediately reflected in the classes Queue<Customers>, Queue<Messages>, and Queue<Orders>.

Templates are very useful when implementing generic constructs like vectors, stacks, lists, queues which can be used with any arbitrary type. C++ templates provide a way to re-use source code as opposed to inheritance and composition which provide a way to re-use object code.

C++ provides two kinds of templates: class templates and function templates. Use function templates to write generic functions that can be used with arbitrary types. For example, one can write searching and sorting routines which can be used with any arbitrary type. The Standard Template Library generic algorithms have been implemented as function templates, and the containers have been implemented as class templates.

Code:

```
#include <iostream>
```

```

#include <string>
using namespace std;

template <class T>
class Stack
{
public:
    Stack();
    void push(T i);
    T pop();
private:
    int top;
    T st[100];
};

template <class T>
Stack<T>::Stack()
{
    top = -1;
}

template <class T>
void Stack<T>::push(T i)
{
    st[++top] = i;
}

template <class T>
T Stack<T>::pop()
{
    return st[top--];
}

int main ()
{
    Stack<int> int_stack;
    Stack<string> str_stack;
    int_stack.push(67);
    str_stack.push("Hello");
    str_stack.push("Codezclub");
}

```



```
    cout << int_stack.pop() << endl;  
    cout << str_stack.pop() << endl;  
    cout << str_stack.pop() << endl;  
    return 0;  
}
```

Result:-

```
67  
Codezclub  
Hello  
  
Process returned 0
```

Experiment No:-11

Aim:- Design a program to demonstrate the concepts of catching and throwing of an exception.

Objective:- throw – A program throws an exception when a problem shows up. This is done using a throw keyword.

catch – A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.

Theory:-

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: try, catch, and throw.

throw – A program throws an exception when a problem shows up. This is done using a throw keyword.

catch – A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.

try – A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

Assuming a block will raise an exception, a method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch as follows –

```
try {  
    // protected code  
} catch( ExceptionName e1 ) {  
    // catch block  
} catch( ExceptionName e2 ) {  
    // catch block  
} catch( ExceptionName eN ) {
```

```
// catch block
}
```

You can list down multiple catch statements to catch different type of exceptions in case your try block raises more than one exception in different situations.

Throwing Exceptions

Exceptions can be thrown anywhere within a code block using throw statement. The operand of the throw statement determines a type for the exception and can be any expression and the type of the result of the expression determines the type of exception thrown.

Following is an example of throwing an exception when dividing by zero condition occurs –

```
double division(int a, int b) {
    if( b == 0 ) {
        throw "Division by zero condition!";
    }
    return (a/b);
}
```

Catching Exceptions

The catch block following the try block catches any exception. You can specify what type of exception you want to catch and this is determined by the exception declaration that appears in parentheses following the keyword catch.

```
try {
    // protected code
} catch( ExceptionName e ) {
    // code to handle ExceptionName exception
}
```

Above code will catch an exception of ExceptionName type. If you want to specify that a catch block should handle any type of exception that is thrown in a try block, you must put an ellipsis, ..., between the parentheses enclosing the exception declaration as follows –

```
try {
    // protected code
} catch(...) {
    // code to handle any exception
}
```

The following is an example, which throws a division by zero exception and we catch it in catch block.

Code:

```
#include <iostream>
#include <conio.h>
using namespace std;

int main()
{
    int a,b;
    cout << "Enter 2 numbers: ";
    cin >> a >> b;
    try
    {
        if (b != 0)
        {
            float div = (float)a/b;
            if (div < 0)
                throw 'e';
            cout << "a/b = " << div;
        }
        else
            throw b;
    }
    catch (int e)
    {
        cout << "Exception: Division by zero";
    }
    catch (char st)
    {
        cout << "Exception: Division is less than 1";
    }
    catch(...)
```

```
{  
    cout << "Exception: Unknown";  
}  
getch();  
return 0;  
}
```

Result:-

Enter 2 numbers: 8 5

a/b = 1.6

Enter 2 numbers: 9 0

Exception: Division by zero

Enter 2 numbers: -1 10

Exception: Division is less than 1

4.Quiz

Object Oriented Programming using C++ Quiz Questions

1. Which of the following type of class allows only one object of it to be created?
 - A. Virtual class
 - B. Abstract class
 - C. Singleton class
 - D. Friend class

Answer: Option C

2. Which of the following is not a type of constructor?
 - A. Copy constructor
 - B. Friend constructor
 - C. Default constructor
 - D. Parameterized constructor

Answer: Option B

3. Which of the following statements is correct?
 - A. Base class pointer cannot point to derived class.
 - B. Derived class pointer cannot point to base class.
 - C. Pointer to derived class cannot be created.
 - D. Pointer to base class cannot be created.

Answer: Option B

4. Which of the following is not the member of class?

- A. Static function
- B. Friend function
- C. Const function
- D. Virtual function

Answer: Option B

5. Which of the following concepts means determining at runtime what method to invoke?

- A. Data hiding
- B. Dynamic Typing
- C. Dynamic binding
- D. Dynamic loading

Answer: Option C

6. Which of the following term is used for a function defined inside a class?

- A. Member Variable
- B. Member function
- C. Class function
- D. Classic function

Answer: Option B

7. Which of the following concept of oops allows compiler to insert arguments in a function call if it is not specified?

- A. Call by value

- B.** Call by reference
- C.** Default arguments
- D.** Call by pointer

Answer: Option C

8. How many instances of an abstract class can be created?

- A.** 1
- B.** 5
- C.** 13
- D.** 0

Answer: Option D

9. Which of the following cannot be friend?

- A.** Function
- B.** Class
- C.** Object
- D.** Operator function

Answer: Option C

10. Which of the following concepts of OOPS means exposing only necessary information to client?

- A.** Encapsulation

- B. Abstraction
- C. Data hiding
- D. Data binding

Answer: Option C

Explanation:

11. Why reference is not same as a pointer?
- A. A reference can never be null.
 - B. A reference once established cannot be changed.
 - C. Reference doesn't need an explicit dereferencing mechanism.
 - D. All of the above.

Answer: Option D

12. *cout* is a/an _____ .
- A. operator
 - B. function
 - C. object
 - D. macro

Answer: Option C

13. Which of the following concepts provides facility of using object of one class inside another class?
- A. Encapsulation
 - B. Abstraction

C. Composition

D. Inheritance

Answer: Option C

14. How many types of polymorphisms are supported by C++?

A. 1

B. 2

C. 3

D. 4

Answer: Option B

Explanation:

The two main types of polymorphism are run-time (implemented as inheritance and virtual functions), and compile-time (implemented as templates).

15. Which of the following is an abstract data type?

A. int

B. double

C. string

D. Class

Answer: Option D

16. Which of the following concepts means adding new components to a program as it runs?

A. Data hiding

B. Dynamic typing

C. Dynamic binding

D. Dynamic loading

Answer: Option **D**

17. Which of the following statement is correct?

A. A constructor is called at the time of declaration of an object.

B. A constructor is called at the time of use of an object.

C. A constructor is called at the time of declaration of a class.

D. A constructor is called at the time of use of a class.

Answer: Option **A**

18. Which of the following correctly describes overloading of functions?

A. Virtual polymorphism

B. Transient polymorphism

C. Ad-hoc polymorphism

D. Pseudo polymorphism

Answer: Option **C**

19. Which of the following approach is adapted by C++?

A. Top-down

B. Bottom-up

C. Right-left

D. Left-right

Answer: Option **B**

20. Which of the following is correct about function overloading?

- A.** The types of arguments are different.
- B.** The order of argument is different.
- C.** The number of argument is same.
- D.** Both A and B.

Answer: Option **D**

Explanation:

21. Which of the following is correct about class and structure?

- A.** class can have member functions while structure cannot.
- B.** class data members are public by default while that of structure are private.
- C.** Pointer to structure or classes cannot be declared.
- D.** class data members are private by default while that of structure are public by default.

Answer: Option **D**

22. Which of the following concepts means wrapping up of data and functions together?

- A.** Abstraction
- B.** Encapsulation
- C.** Inheritance
- D.** Polymorphism

Answer: Option **B**

23. Which of the following concepts means waiting until runtime to determine which function to call?

- A. Data hiding
- B. Dynamic casting
- C. Dynamic binding
- D. Dynamic loading

Answer: Option C

24. How "Late binding" is implemented in C++?

- A. Using C++ tables
- B. Using Virtual tables
- C. Using Indexed virtual tables
- D. Using polymorphic tables

Answer: Option B

25. Which of the following operator is overloaded for object `cout`?

- A. `>>`
- B. `<<`
- C. `+`
- D. `=`

Answer: Option B

26. Which of the following is the correct class of the object `cout`?

A. *iostream*

B. *istream*

C. *ostream*

D. *ifstream*

Answer: Option C

27. Which of the following cannot be used with the keyword *virtual*?

A. class

B. member functions

C. constructor

D. destructor

Answer: Option C

28. Which of the following functions are performed by a constructor?

A. Construct a new class

B. Construct a new object

C. Construct a new function

D. Initialize objects

Answer: Option D

29. Which of the following problem causes an exception?

A. Missing semicolon in statement in *main()*.

- B. A problem in calling function.
- C. A syntax error.
- D. A run-time error.

Answer: Option D

30. Which one of the following options is correct about the statement given below? The compiler checks the type of reference in the object and not the type of object.

- A. Inheritance
- B. Polymorphism
- C. Abstraction
- D. Encapsulation

Answer: Option B

31. Which of the following ways are legal to access a class data member using this pointer?

- A. *this->x*
- B. *this.x*
- C. **this.x*
- D. **this-x*

Answer: Option A

32. Which of the following is a mechanism of static polymorphism?

- A. Operator overloading
- B. Function overloading
- C. Templates
- D. All of the above

Answer: Option **D**

33. Which of the following is correct about the statements given below?

All operators can be overloaded in C++.

We can change the basic meaning of an operator in C++.

- A.** Only I is true.
- B.** Both I and II are false.
- C.** Only II is true.
- D.** Both I and II are true.

Answer: Option **B**

34. What happens if the base and derived class contains definition of a function with same prototype?

- A.** Compiler reports an error on compilation.
- B.** Only base class function will get called irrespective of object.
- C.** Only derived class function will get called irrespective of object.
- D.** Base class object will call base class function and derived class object will call derived class function.

Answer: Option **D**

35. Which of the following are available only in the class hierarchy chain?

- A.** Public data members
- B.** Private data members

- C. Protected data members
- D. Member functions

Answer: Option C

36. Which of the following is not a type of inheritance?

- A. Multiple
- B. Multilevel
- C. Distributive
- D. Hierarchical

Answer: Option C

37. Which of the following keyword is used to overload an operator?

- A. *overload*
- B. *operator*
- C. *friend*
- D. *override*

Answer: Option B

38. What will happen if a class is not having any name?

- A. It cannot have a destructor.
- B. It cannot have a constructor.
- C. It is not allowed.

D Both A and B.
.

Answer: Option **D**

39. Which inheritance type is used in the class given below?

```
class A : public X, public Y  
{
```

A Multilevel inheritance
.

B Multiple inheritance
.

C Hybrid inheritance
.

D Hierarchical Inheritance
.

Answer: Option **B**

40. Which one of the following is correct about the statements given below?

All function calls are resolved at compile-time in Procedure Oriented Programming.

All function calls are resolved at compile-time in OOPS.

A. Only II is correct.

B. Both I and II are correct.

C. Only I is correct.

D. Both I and II are incorrect.

Answer: Option **C**

41. Which of the following is an invalid visibility label while inheriting a class?

A. *public*

B. *private*

C. *protected*

D. *friend*

Answer: Option **D**

5. Conduction of VIVA-VOCE Examinations:

Teacher should conduct oral exams of the students with full preparation. Normally the objective questions with guess are to be avoided. To make it meaningful, the questions should be such that depth of the student in the subject is tested. Oral Exams are to be conducted in co-cordial situation. Teachers taking oral exams should not have ill thoughts about each other & courtesies should be offered to each other in case of opinion, which should be critically suppressed in front of the students.

6.Evaluation and marking system:

Basic honesty in the evaluation and marking system is essential and in the process impartial nature of the evaluator is required in the exam system. It is a primary responsibility of the teacher to see that right students who really put their effort & intelligence are correctly awarded.

The marking pattern should be justifiable to the students without any ambiguity and teacher should see that students are faced with just circumstance.