**PRACTICAL-6: CLASS AND OBJECTS, ENCAPSULATIONS**

**(private, public and protected)**

**AIM:**

To understand the class and objects, encapsulation (private, public and protected).

**THEORY:**

**Class** in C++ is the building block, that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object.

**Objects** is a real world entity, for example, chair, car, pen, mobile, laptop etc. Object is an instance of a class. All the members of the class can be accessed through object.

**Encapsulation** refers to the bundling of data, along with the methods that operate on that data, into a single unit.

**Access modifiers** are used to implement an important aspect of

Object-Oriented Programming known as Data Hiding.

**There are 3 types of access modifiers available in C++:**

1. Public [everyone can access outside the class]

2. Private [no one can access outside the class]

3. Protected[only child can access outside the class

**Q.1) Write a program to create a class area to calculate the area of a circle using private variables and public functions.**

**CODE:**

#include <iostream>

using namespace std;

class circle

{

private:

float radius;

public:

void set\_radius(float r)

{

radius = r;

}

float get\_radius()

{

return radius;

}

float area()

{

return 3.14 \* radius \* radius;

}

};

int main()

{

circle c;

float r;

cout << "Enter the radius of the circle: ";

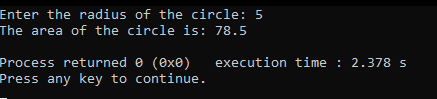
cin >> r;

c.set\_radius(r);

cout << "The area of the circle is: " << c.area() << endl;

}

**OUTPUT:**

****

**Fig1. Output of program showing area of cricle**

**PRACTICAL 7: INHERITANCE**

**AIM:**

To understand the inheritance.

**THEORY:**

**Inheritance** is a process in which one object acquires all the properties and behaviors of its parent object automatically.

**TYPES OF INHERITANCE:**

**Single Inheritance:** When a Derived Class to inherit properties and behavior from a single Base Class, it is called as single inheritance.

**Multi-level Inheritance:** A derived class is created from another derived class is called Multi Level Inheritance.

**Hierarchical Inheritance:** More than one derived classes are created from a single base class, is called Hierarchical Inheritance.

**Multipath Inheritance:** Multiple inheritance is a method of inheritance in which one derived class can inherit properties of base class in different paths. This inheritance is not supported in .NET Languages such as C#.

**Multiple Inheritance:** Multiple inheritances allows programmers to create classes that combine aspects of multiple classes and their corresponding hierarchies.

**Q.1) Write a program to perform a single inheritance to calculate area of circle.**

**CODE:**

#include <iostream>

using namespace std;

class Radius

{

public:

int radius;

void getRadius()

{

cout<<"Enter the radius of circle: ";

cin>>radius;

}

};

class Area : public Radius

{

public:

void getArea()

{

cout<<"Area of circle is: "<<3.14\*radius\*radius<<endl;

}

};

int main()

{

Area a;

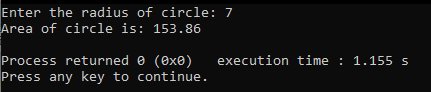
a.getRadius();

a.getArea();

return 0;

}

**OUTPUT:**



**Fig1. Output of program showing area of circle demonstrating single inheritance**

**Q.2) Write a program to perform multilevel inheritance student details, course details, marks.**

**CODE:-**

#include <iostream>

using namespace std;

class Student

{

private:

char name[50];

int id;

public:

int student()

{

cout<<"Enter student name: ";

cin.getline(name,50);

cout<<"Enter student id: ";

cin>>id;

}

void display\_student()

{

cout<<"\n\nStudent name: "<<name<<endl;

cout<<"Student id: "<<id;

}

};

class Marks : public Student

{

private:

int marks[5];

int total;

float avg;

public:

int student\_marks()

{

for(int i=0; i<5; i++)

{

cout<<"Enter marks of subject "<<i+1<<": ";

cin>>marks[i];

}

total=0;

for(int i=0; i<5; i++)

{

total=total+marks[i];

}

avg=total/5;

}

void display\_student\_marks()

{

cout<<"\nMarks: ";

for(int i=0; i<5; i++)

{

cout<<marks[i]<<" ";

}

cout<<"\nTotal: "<<total;

cout<<"\nAverage: "<<avg;

}

};

class Course : public Marks

{

private:

char course[50];

char branch[50];

int year;

public:

int student\_course()

{

cout<<"Enter course name: ";

cin>>course;

cout<<"Enter branch name: ";

cin>>branch;

cout<<"Enter year: ";

cin>>year;

return 0;

}

void display\_student\_course()

{

cout<<"\nCourse name: "<<course;

cout<<"\nBranch name: "<<branch;

cout<<"\nYear: "<<year;

}

};

int main()

{

Course c;

c.student();

c.student\_marks();

c.student\_course();

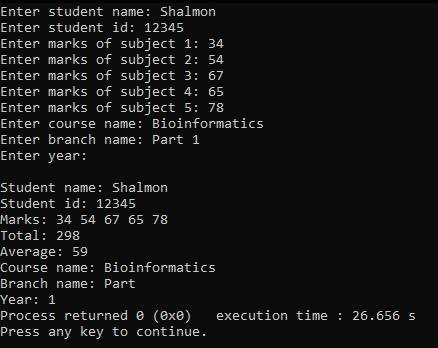
c.display\_student();

c.display\_student\_marks();

c.display\_student\_course();

}

**OUTPUT:**

****

**Fig2. Output of program showing student information demonstrating multilevel inheritance**

**PRACTICAL 8: Polymorphism, Virtual Function, Friend Function**

**AIM:**

To understand the Polymorphism, Virtual Function, Friend Function.

**THEORY:**

The word **polymorphism** means having many forms. The ability of a message to displayed in more than one form.

A **virtual function** is a member function that you expect to be redefined in derived classes.

**Friend function** is given the same access as methods to private and protected data.

**TYPES OF POLYMORPHISM:**

**Compile time polymorphism:** This type of polymorphism is achieved by function overloading or operator overloading.

**Runtime polymorphism:** This type of polymorphism is achieved by function overriding.

**Q.1) Write a program to perform addition of two and three numbers using method overloading.**

**CODE:**

#include <iostream>

using namespace std;

class Addition

{

public:

//function to add two numbers

int add(int a, int b)

{

return a+b;

}

//function to add three numbers

int add(int a, int b, int c)

{

return a+b+c;

}

};

int main()

{

Addition obj;

int a, b, c;

cout<<"Enter two numbers: ";

cin>>a>>b;

cout<<"Sum of two numbers is: "<<obj.add(a,b)<<endl;

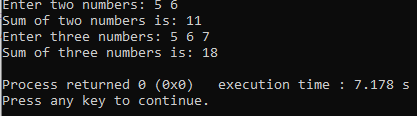
cout<<"Enter three numbers: ";

cin>>a>>b>>c;

cout<<"Sum of three numbers is: "<<obj.add(a,b,c)<<endl;

}

**OUTPUT:**



**Fig1. Output showing addition of two and three numbers using method overloading**

**Q.2) Write a program to demonstrate method overriding using virtual function.**

**CODE:**

#include <iostream>

using namespace std;

class Base

{

public:

virtual void print()

{

cout << "print base class on compile" << endl;

}

void display()

{

cout << "display base class on compile" << endl;

}

};

class Derived : public Base

{

public:

void print()

{

cout << "print derived class on runtime" << endl;

}

void display()

{

cout << "display derived class on runtime" << endl;

}

};

int main()

{

Base\* bptr;

Derived d;

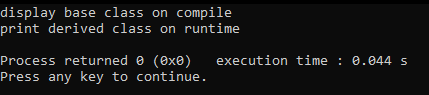
bptr = &d;

bptr->display();

bptr->print();

}

**OUTPUT:**

****

**Fig2. Output of program showing method overriding using virtual function**

**PRACTICAL 9: CONSTRUCTOR AND DESTRUCTOR**

**AIM:**

To understand constructor and destructor.

**THEORY:**

A **constructor** is a special type of member function of a class which initializes objects of a class.

**Syntax:**

class Wall { public: // create a constructor Wall() { // code } };

**DESTRUCTOR:**

Destructors are members functions in a class that delete an object.

**Syntax:**

~constructor-name();

**TYPES OF CONSTRUCTORS:**

**Default constructor** is the constructor which doesn't take any argument. It has no parameter.

**Syntax:**

class\_name(parameter1, parameter2, ...)

{

// constructor Definition

}

**Parameterized Constructors :** These are the constructors with parameter. Using this Constructor you can provide different values to data members of different objects, by passing the appropriate values as argument.

**Syntax:** name\_of\_class

**Copy Constructors:** These are special type of Constructors which takes an object as argument, and is used to copy values of data members of one object into other object. We will study copy constructors in detail later.

**Syntax:** Class\_name a,b; b = a;

**Q.1) Write a program to demonstrate constructor and destructor**

**CODE:**

#include <iostream>

using namespace std;

class Simple

{

public:

Simple()

{

cout << "Constructor called" << endl;

}

~Simple()

{

cout << "Destructor called" << endl;

}

};

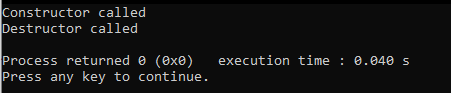
int main()

{

Simple s;

}

**OUTPUT:**

****

**Fig1. Output of program demonstrating constructors and destructors**