**Experiment 3.1**

**3.1 a) Unary operator overloading using member function**

**Aim:** Unary operator overloading using member function

**Program:**

#include<iostream>

using namespace std;

class unary {

int x,y,z;

public:

void get(){

x=10;

y=-34;

z=-78;}

void show(){

cout<<"x= "<<x<<endl;

cout<<"y= "<<y<<endl;

cout<<"z= "<<z;}

void operator -(){

x=-x;

y=-y;

z=-z;}};

int main(){

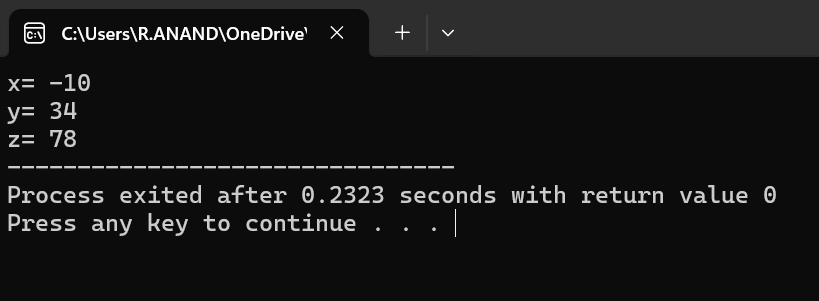
unary u;

u.get();

-u;

u.show();}

**output:**

****

**3.1 b) Binary operator overloading using member function**

Aim: Binary operator overloading using member function

**Program:**

#include<iostream>

using namespace std;

class complex{

int real,imag;

public:

void get(){

cout<<"enter real,imag values:";

cin>>real>>imag;}

void operator + (complex c2){

cout<<real+c2.real<<"+"<<imag+c2.imag<<"i";}};

int main(){

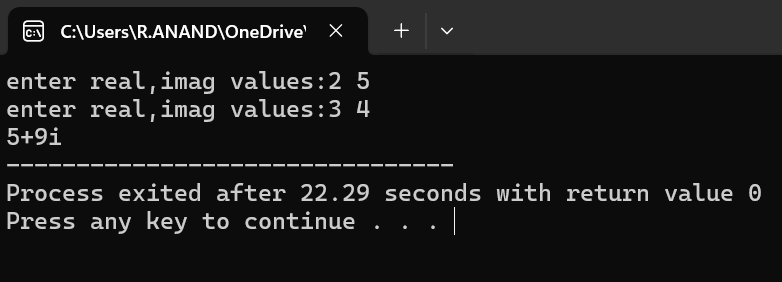
complex c1,c2;

c1.get();

c2.get();

c1+c2;

return 0;}

**Output:** ****

**Experiment 3.2**

**3.2 a) Unary operator overloading using friend function**

**Aim:** Unary operator overloading using friend function

**Program:**

#include<iostream>

using namespace std;

class unary {

int x,y,z;

public:

void get() {

x=10;

y=-34;

z=-78;}

void show() {

cout<<"x= "<<x<<endl;

cout<<"y= "<<y<<endl;

cout<<"z= "<<z;}

friend void operator -(unary &u);};

void operator -(unary &u){

u.x=-u.x;

u.y=-u.y;

u.z=-u.z;}

int mai(){

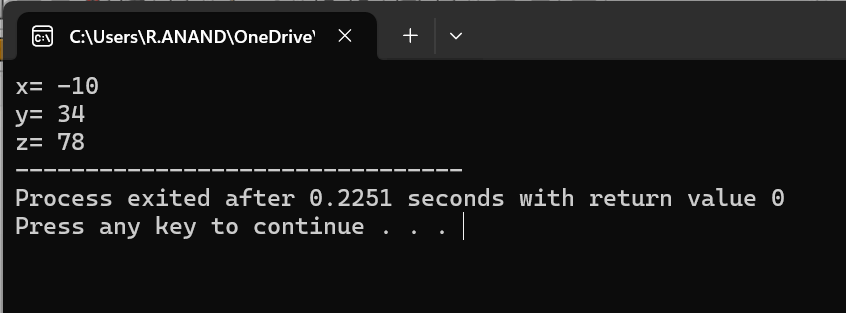
unary u;

u.get();

-u;

u.show();}

**Output:**

****

**3.2 b) Binary operator overloading using friend function**

**Aim:** Binary operator overloading using friend function

**Program:**

#include<iostream>

using namespace std;

class complex{

int real,imag;

public:

void get(){

cout<<"enter real,imag values:";

cin>>real>>imag;}

friend void operator + (complex &c1,complex &c2);};

void operator + (complex &c1,complex &c2){

cout<<c1.real+c2.real<<"+"<<c1.imag+c2.imag<<"i";}

int main(){

complex c1,c2;

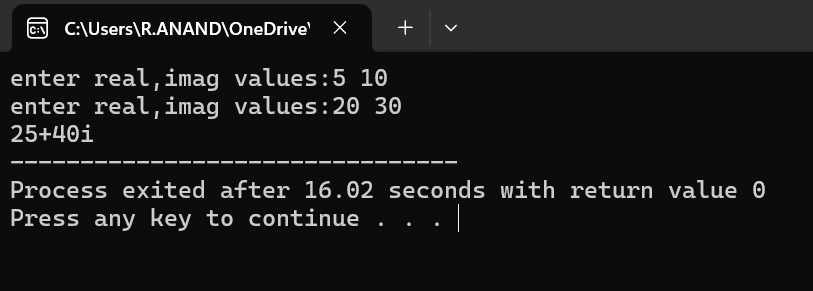
c1.get();

c2.get();

c1+c2;//c1.operator + (c2)

return 0;}

**Output:**

****

**Experiment 3.3**

**3.3 a)Write a c++ program on single inheritance**

**Aim:** ToWrite a c++ program on single inheritance

**Program:**

#include<iostream>

using namespace std;

class person{

protected:

string name;

int age;

string gender;

void getp(){

cout<<"Enter name,age and gender of the person :";

cin>>name>>age>>gender;}

void showp(){

cout<<"Name :"<<name<<endl;

cout<<"Age :"<<age<<endl;

cout<<"Gender :"<<gender<<endl;}};

class student:public person{

private:

int rollno;

float perc;

public:

void getst(){

getp();

cout<<"Enter Rollno :";

cin>>rollno;

cout<<"Enter percentage :";

cin>>perc;}

void showst(){

showp();

cout<<"Roll-no :"<<rollno<<endl;

cout<<"Percentage :"<<perc<<endl;}};

int main(){

student s1,s2;

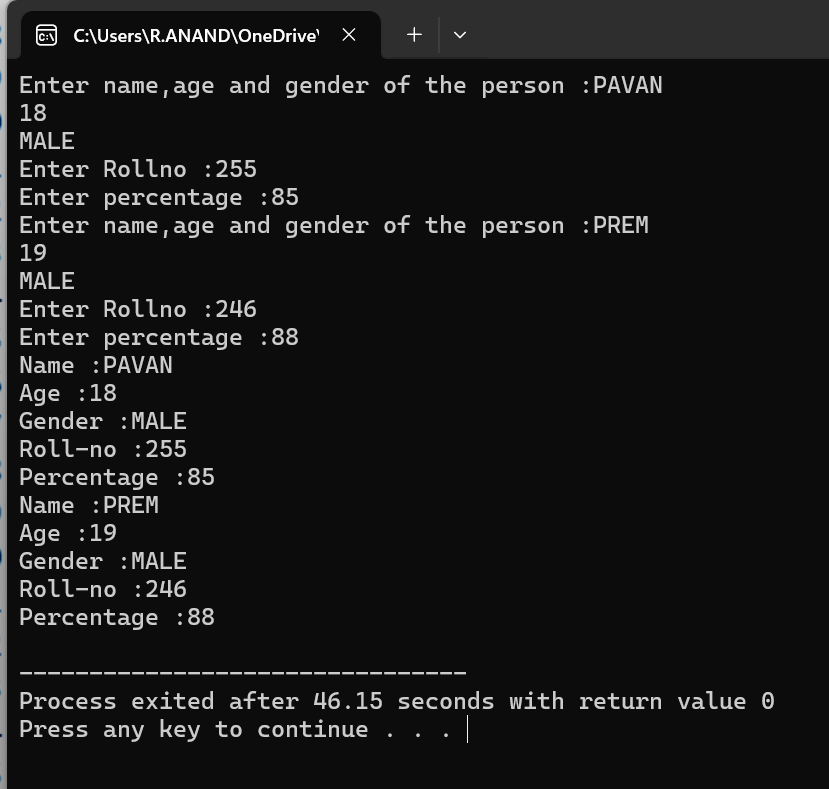
s1.getst();

s2.getst();

s1.showst();

s2.showst();

return 0;}

**Output:** ****

**3.3 b) Write a c++ program on Multiple inheritance**

**Aim:** To Write a c++ program on Multiple inheritance

**Program:**

#include<iostream>

using namespace std;

class Area{

protected:

void calc\_area(int l,int b){

cout<<"The Area is :"<<l\*b<<endl;}};

class perimeter{

protected:

void calc\_peri(int l,int b){

cout<<"The perimeter is :"<<2\*(l+b)<<endl;}};

class rectangle:public Area,public perimeter{

public:

int l,b;

void result(){

cout<<"Enter length and breadth :";

cin>>l>>b;

calc\_area(l,b);

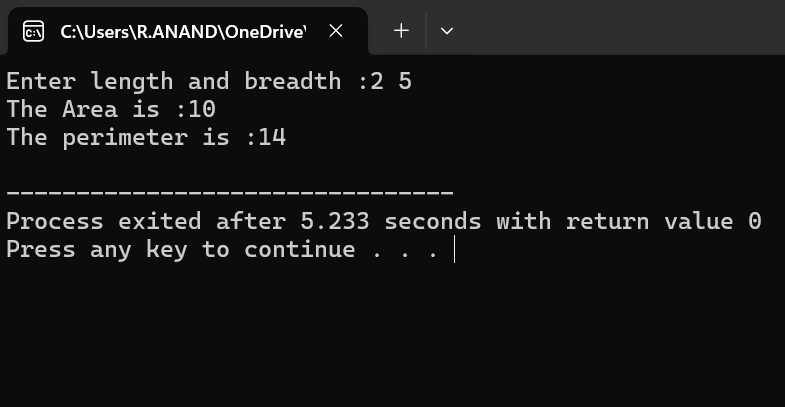
calc\_peri(l,b);}};

int main(){

rectangle r;

r.result();

return 0;}

**Output: **

**3.3 c) Write a c++ program on Multi-level inheritance**

**Aim:** To Write a c++ program on Multi-level inheritance

**Program:**

#include<iostream>

using namespace std;

class student{

protected:

int rollno;

string name;

void getst(){

cout<<"Enter Rollno and Name :";

cin>>rollno>>name;}

void showst(){

cout<<"Name :"<<name<<endl;

cout<<"Rollno :"<<rollno<<endl;}};

class marks:public student{

protected:

int m1,m2,m3,m4,m5;

void getm(){

getst();

cout<<"Enter Five subjects - marks :";

cin>>m1>>m2>>m3>>m4>>m5}};

class percentage:public marks{

public:

float perc;

void showp(){

getm();

perc=((float)(m1+m2+m3+m4+m5)/500)\*100;

showst();

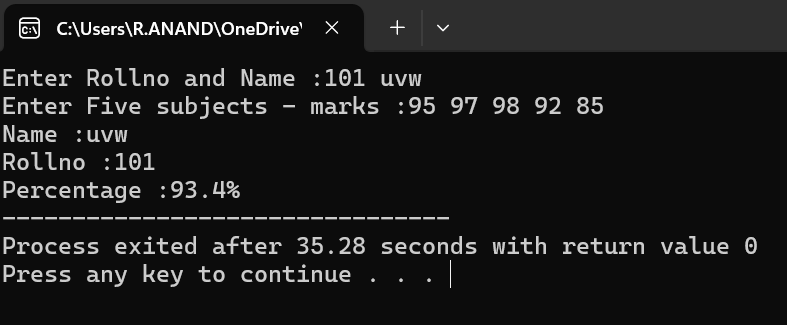
cout<<"Percentage :"<<perc<<"%";}};

int main(){

percentage p;

p.showp();

return 0;}

**Output:** ****

**3.3 d) Write a c++ program on Hierarchical inheritance**

**Aim:** To Write a c++ program on Hierarchical inheritance

**Program:**

#include<iostream>

using namespace std;

class TV {

public:

int inches;

int price;

void gettv() {

cout << "Enter Price and Size of TV : ";

cin >> price >> inches; }

void showtv() {

cout << "\nLCDTV-Details" << endl;

cout << "Size of the TV : " << inches << " inches" << endl;

cout << "Price : " << price << "/-" << endl; }};

class manfac : public TV {

public:

string date;

int warranty;

void getm() {

gettv();

cout << "Enter Date and Warranty of TV : ";

cin >> date >> warranty; }

void showm() {

showtv();

cout << "Date-of-manufacture : " << date << endl;

cout << "Warranty : " << warranty << " years" << endl; }};

class product : public TV {

public:

string brand;

string country;

void getp() {

gettv();

cout << "Enter Brand and Country : ";

cin >> brand >> country; }

void showp() {

showtv();

cout << "Brand : " << brand << endl;

cout << "Country : " << country << endl; }};

int main() {

manfac m;

product p;

cout << "Enter Manufacturer Details:\n";

m.getm();

cout << "\nEnter Product Details:\n";

p.getp();

cout << "\n--- Manufacturer Information ---\n";

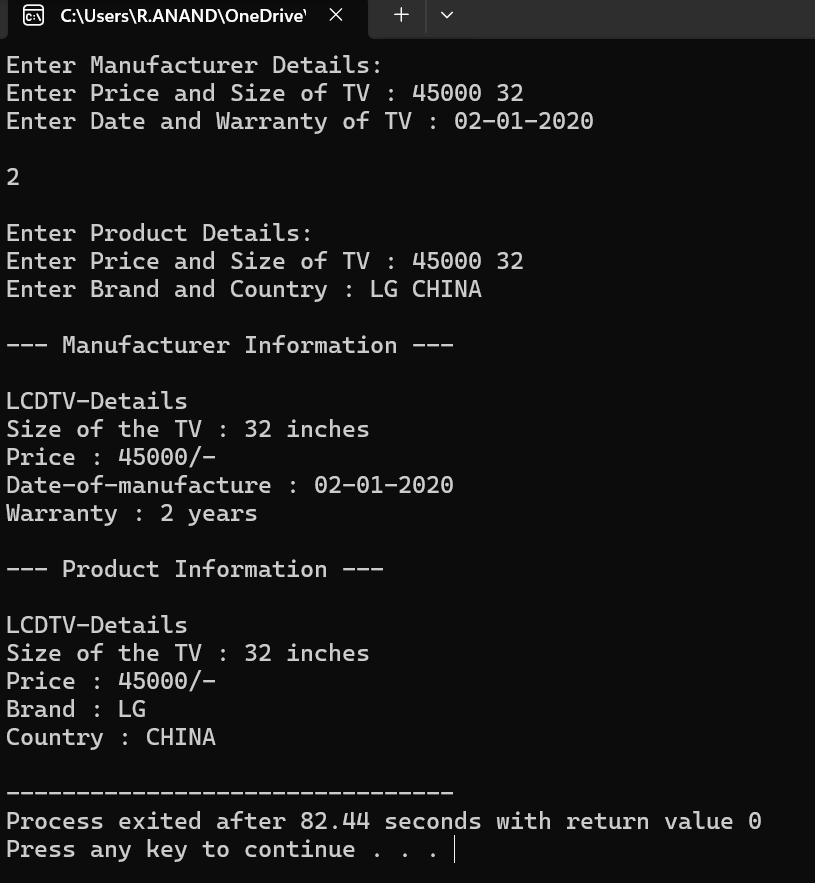
m.showm();

cout << "\n--- Product Information ---\n";

p.showp();

return 0;}

**Output:**

****

**3.3 e) Write a c++ program on Hybrid inheritance**

**Aim:** To Write a c++ program on Hybrid inheritance

**Program:**

#include<iostream>

using namespace std;

class student{

protected:

int rollno;

string name;

void getst(){

cout<<"Enter Rollno and name of the student :";

cin>>rollno>>name;}

void showst(){

cout<<"Name :"<<name<<endl<<"Roll-no :"<<rollno<<endl;}};

class marks:public student{

protected:

int m1,m2,m3,m4,m5;

void getm(){

getst();

cout<<"Enter five subjects marks :";

cin>>m1>>m2>>m3>>m4>>m5;}};

class SABL{

protected:

int tech;

int non\_tech;

void getp(){

cout<<"Enter Marks obtained for techincal and non-technical events in SABL :";

cin>>tech>>non\_tech;}};

class percentage:public marks , public SABL{

public:

float per;

void cal\_percentage(){

getm();

getp();

per =((float)(m1 + m2 + m3 + m4 + m5 + tech + non\_tech) / 600) \* 100;

showst();

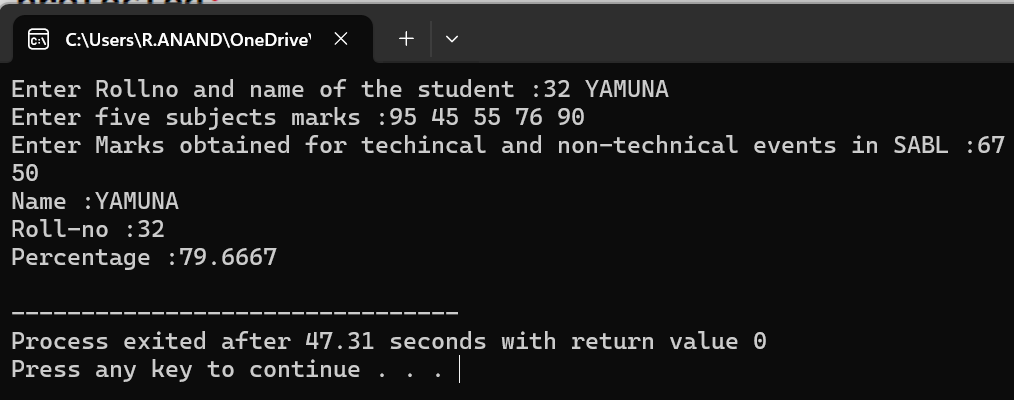
cout<<"Percentage :"<<per<<endl; }};

int main(){

percentage p;

p.cal\_percentage();

return 0;}

**Output:** ****

**Experiment 3.4**

**3.4) Write a c++ program on Order of execution of constructors and destructors in inheritance.**

**Aim:** To Write a c++ program on Order of execution of constructors and destructors in inheritance**.**

**Program:**

#include<iostream>

using namespace std;

class A{

public:

A(){

cout<<"Class A constructor is invoked"<<endl;}

~A(){

cout<<"Class A destructor is invoked"<<endl;}};

class B:public A{

public:

B(){

cout<<"Class B constructor is invoked"<<endl;}

~B(){

cout<<"Class B destructor is invoked"<<endl;}};

class C:public A{

public:

C(){

cout<<"Class C constructor is invoked"<<endl;}

~C(){

cout<<"Class C destructor is invoked"<<endl;}};

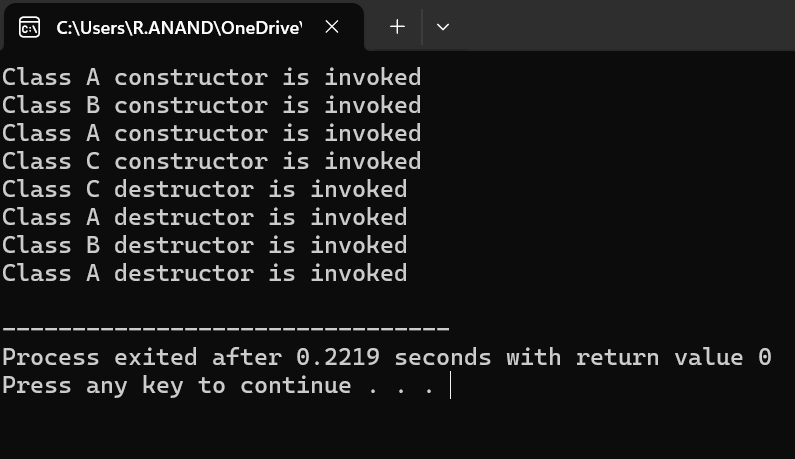
int main(){

B b;

C c;

return 0;}

**Output:**

****

**Experiment 3.5**

**3.5 Illustrate the use of object as a class member,pointer to a class,this pointer and Virtual Base Class**

**Aim:** To Illustrate the use of object as a class member,pointer to a class,this pointer and Virtual Base Class

**Program:**

1. **Object as a class member**

**Program:**

#include<iostream>

using namespace std;

class Birth{

public:

int dd, mm, yy;

void get\_date() {

cout << "Enter Date of Birth (dd mm yyyy): ";

cin >> dd >> mm >> yy; }

void show\_date() {

cout << "Date of Birth: " << dd << "-" << mm << "-" << yy << endl; }};

class student{

public:

char name[20];

Birth dob;

char gender;

void get\_details() {

cout << "Enter Name: ";

cin >> name;

cout << "Enter Gender (M/F): ";

cin >> gender;

dob.get\_date(); }

void print\_details() {

cout << "\n--- Student Details ---" << endl;

cout << "Name: " << name << endl;

cout << "Gender: " << gender << endl;

dob.show\_date(); }};

int main() {

student s;

s.get\_details();

s.print\_details();

return 0;}

1. **pointer to a class**

**Program:**

#include<iostream>

using namespace std;

class base{

public:

void show(){

cout<<"Show() of base class"<<endl;}};

class derived : public base{

public:

void print(){

cout<<"Print() of derived class"<<endl;}};

int main(){

derived d,\*dptr;

dptr=&d;

dptr->print();

dptr->show();}

1. **This pointer**

**Program:**

#include<iostream>

using namespace std;

class rectangle{

private:

int l,b;

public:

void input(int l,int b){

this->l=l;

this->b=b;}

void area(){

cout<<"area of rectangle:"<<l\*b<<endl;}};

int main(){

rectangle r;

r.input(10,20);

r.area();}

1. **Virtual Base Class.**

**Program:**

#include<iostream>

using namespace std;

class A{

public:

void showA(){

cout<<"showA() of class A"<<endl; }};

class B:virtual public A{

public:

void showB(){

cout<<"showB() of class B"<<endl; }};

class C:virtual public A{

public:

void showC(){

cout<<"showC() of class C"<<endl;} };

class D:public B,public C{

public:

void showD(){

cout<<"showD() of class D"<<endl;}};

int main(){

D d;

d.showD();

d.showC();

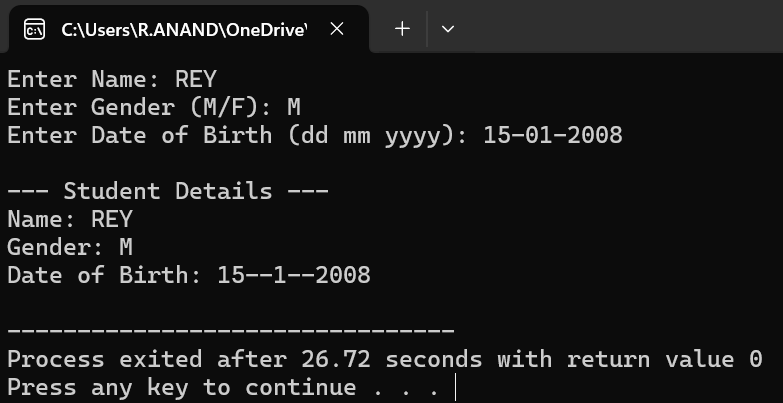
d.showB();

d.showA();}

**Outputs:**

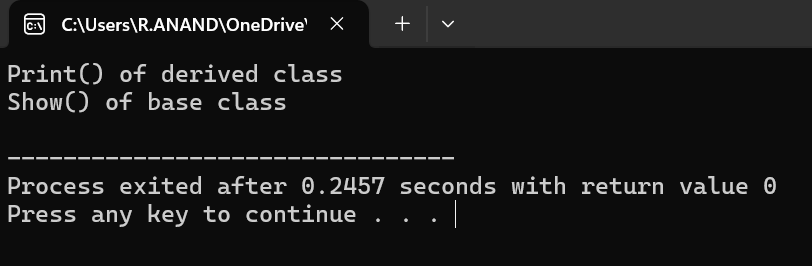
1. **Object as a class member**

**Output:**

****

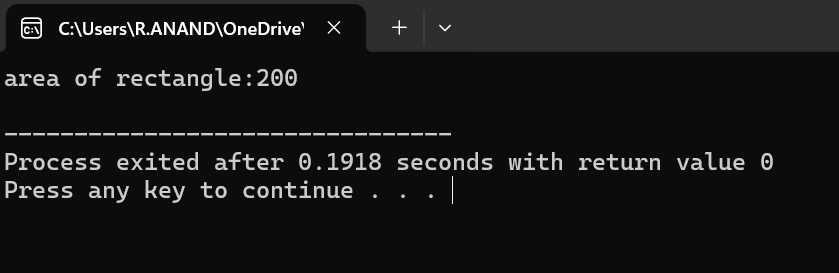
1. **pointer to a class**

**Output :**

****

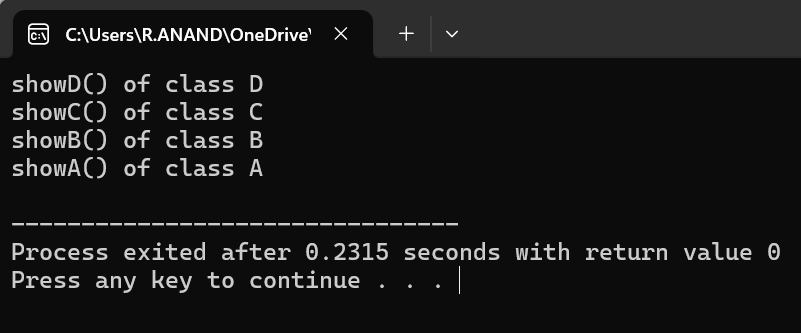
1. **This pointer**

**Output 3:**

****

1. **Virtual Base Class**

**Output 4:**

****

**Expermiment 3.6**

**3.6)**  **illustrate the concept of virtual functions in C++**

**Aim:.** To illustrate the concept of virtual functions in C++ and demonstrate how they enable runtime polymorphism using base class pointers.

**Program:**

#include<iostream>

using namespace std;

class Base {

public:

virtual void print() {

cout << "print() of base class" << endl;}};

class Derived : public Base {

public:

void print() {

cout << "print() of the derived class" << endl;}};

int main() {

Base b, \*bptr;

Derived d;

bptr = &b;

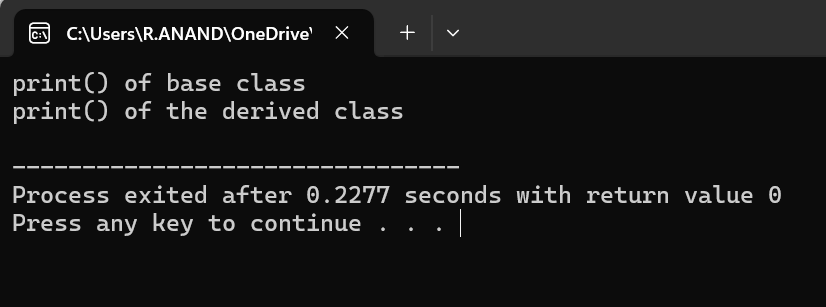
bptr->print();

bptr = &d;

bptr->print();

return 0;}

**Output:**



**Experiment 3.7**

**3.7)** **Implement pure virtual function and calculate the area of different shapes by using abstract class.**

**Aim:** To Implement pure virtual function and calculate the area of different shapes by using abstract class.

**Program:**

#include<iostream>

using namespace std;

class shapes{

public:

virtual void area()=0;};

class rectangle : public shapes{

public:

int l,b;

rectangle(){

l=12;

b=4;}

void area(){

cout<<"Area of the Rectangle is: "<<l\*b<<endl;}};

class square : public shapes{

public:

int a;

square(){

a=10;}

void area(){

cout<<"Area of the Square is: "<<a\*a<<endl;}};

class triangle : public shapes{

public:

int b,h;

triangle(){

b=12;

h=4;}

void area(){

cout<<"Area of the Triangle is: "<<0.5\*(b\*h)<<endl;}};

class circle : public shapes{

public:

int r;

float pi;

circle(){

r=6;

pi=3.14}

void area(){

cout<<"Area of the Circle is: "<<pi\*r\*r<<endl;}};

int main{

shapes \*ptr;

rectangle r;

ptr = &r;

ptr->area();

square s;

ptr = &s;

ptr->area();

triangle t;

ptr = &t;

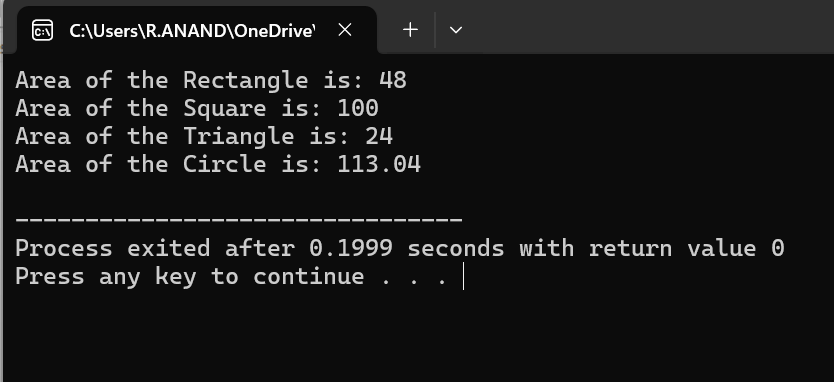
ptr->area();

circle c;

ptr = &c;

ptr->area();}

**Output:**

****