**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| ***S.no*** | ***Name of program*** | ***Page number*** | ***Signature and remarks*** |
| ***1.*** | *To implement Linear Search algorithm to an array.* | 2 – 3 |  |
| ***2.*** | *To sort an array and implement binary search algorithm to it.* | 4 – 6 |  |
| ***3.*** | *To implement the concepts of classes and function overloading.* | 7 - 8 |  |
| ***4.*** | *To implement the three types of function calls – Call by value , Call by address and Call by reference .* | 9 - 11 |  |
| ***5.*** | *To implement the concepts of Inline functions and Macro functions.* | 12 |  |
| ***6.*** | *To implement the concept of default function (arguments).* | 13 |  |
| ***7.*** | *To pass and return an object as an argument both by value and reference..* | 14 – 15 |  |
| ***8.*** | *To implement the concept of Constant member function and constant object.* | 16 – 17 |  |
| ***9.*** | *To implement the concept of Static member variable also known as class variable.* | 18 – 19 |  |
| ***10.*** | *To implement the concept of Static member function.* | 20 – 21 |  |
| ***11.*** | *To implement dynamic operations using Constructors and Destructors.* | 22 – 23 |  |
| ***12.*** | *To implement the concepts of dynamic memory allocation.* | 24 - 25 |  |
| ***13.*** | *To implement the concept of Single Inheritance.* | 26 – 27 |  |
| ***14.*** | *To implement the concepts of Multiple Inheritance in classes.* | 28 – 29 |  |
| ***15.*** | *To implement the concepts of Multilevel Inheritance in classes.* | 30 – 31 |  |
| ***16.*** | *To overload pre and post increment operators.(++).* | 32 – 33 |  |
| ***17.*** | *To overload addition, subtraction and multiplication binary operators.* | 34 - 39 |  |
| ***18.*** | *To implement the concept of a pure virtual function.* | 40 - 41 |  |

**Practical – 1**

***Aim –*** *To implement Linear Search algorithm to an array.*

***Code –***

#include<iostream>

using namespace std;

int main()

{

int arr[10], i, num, n, c=0, pos;

cout<<"Enter the array size : ";

cin>>n;

cout<<"Enter Array Elements : ";

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

{

cin>>arr[i];

}

cout<<"Enter the number to be searched : ";

cin>>num;

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

{

if(arr[i]==num)

{

c=1;

pos=i+1;

break;

}

}

if(c==0)

{

cout<<"Number not found..!!";

}

else

{

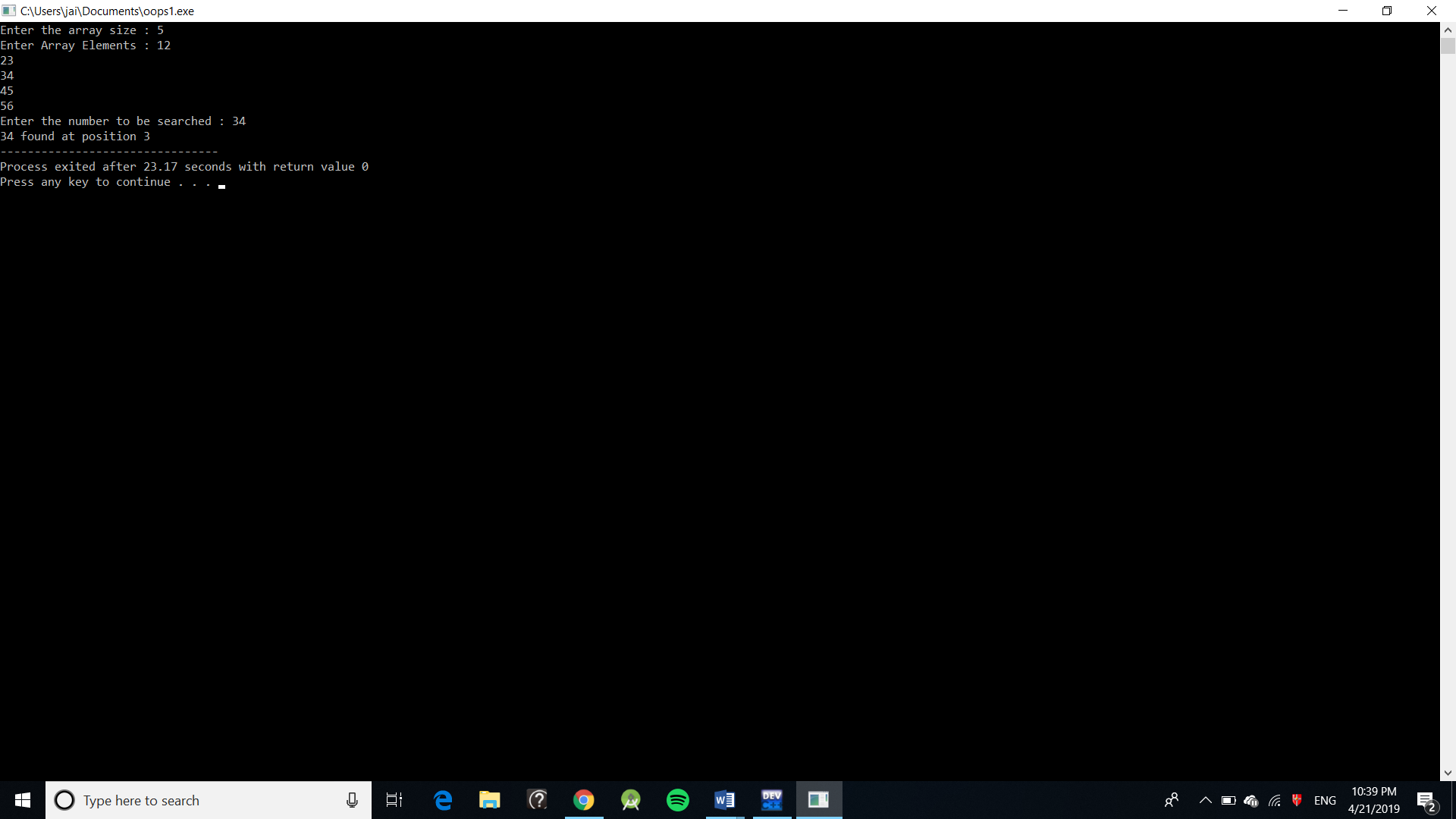
cout<<num<<" found at position "<<pos;

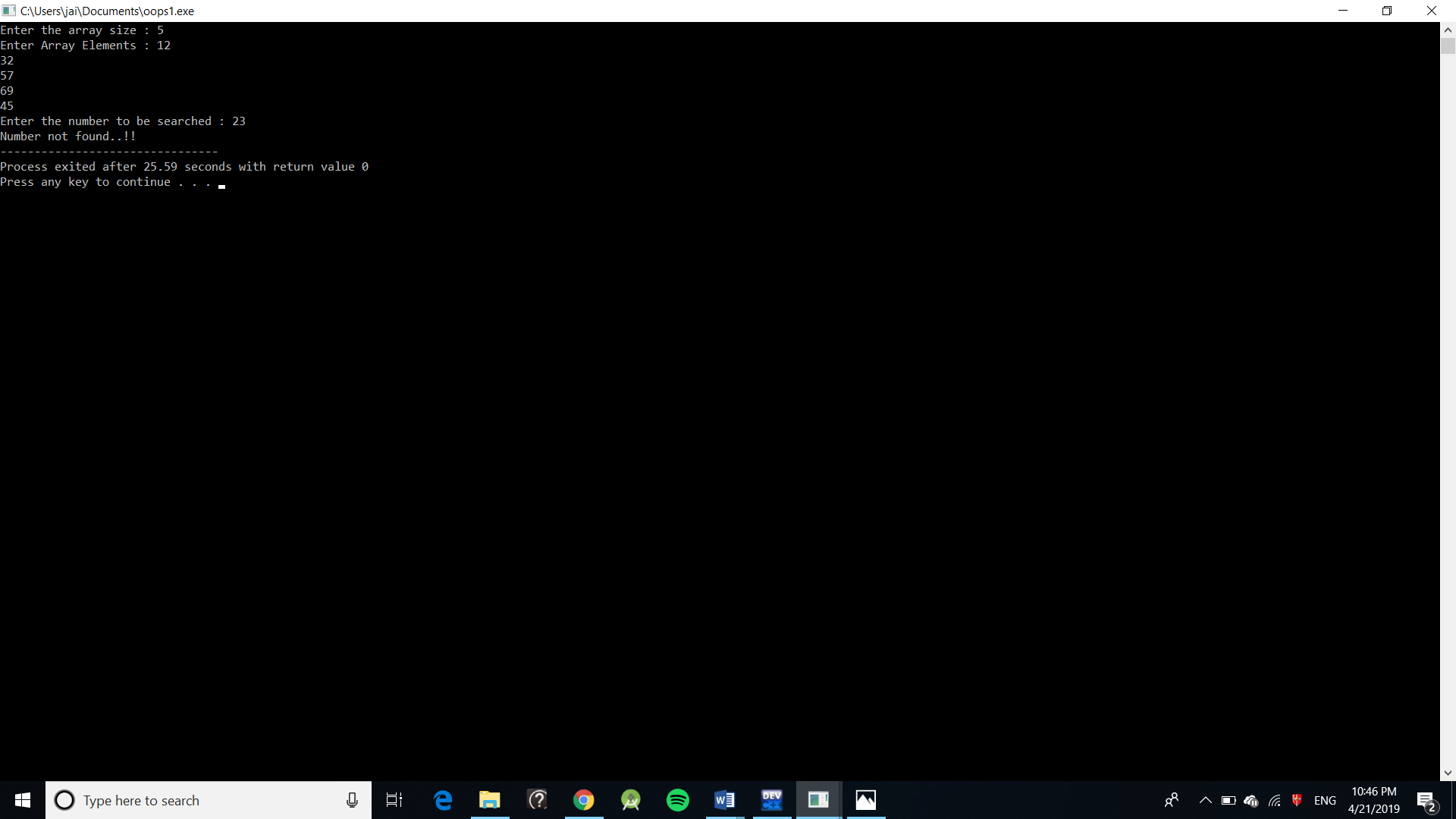
}

return 0;

}

***Output –***

****



**Practical – 2**

***Aim –*** *To sort an array and implement Binary Search algorithm to it.*

***Code –***

#include<iostream>

using namespace std;

int main()

{

int n, i, arr[50], search, first, last, middle;

cout<<"Enter total number of elements :";

cin>>n;

cout<<"Enter "<<n<<" number :";

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

{

cin>>arr[i];

}

cout<<"Enter a number to find :";

cin>>search;

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

{

for(int j=0;j<=n-i;j++)

{

if(arr[j]>arr[j+1])

{

int temp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

}

}

}

first = 0;

last = n-1;

middle = (first+last)/2;

while (first <= last)

{

if(arr[middle] < search)

{

first = middle + 1;

}

else if(arr[middle] == search)

{

cout<<search<<" found at location "<<middle+1<<"\n";

break;

}

else

{

last = middle - 1;

}

middle = (first + last)/2;

}

if(first > last)

{

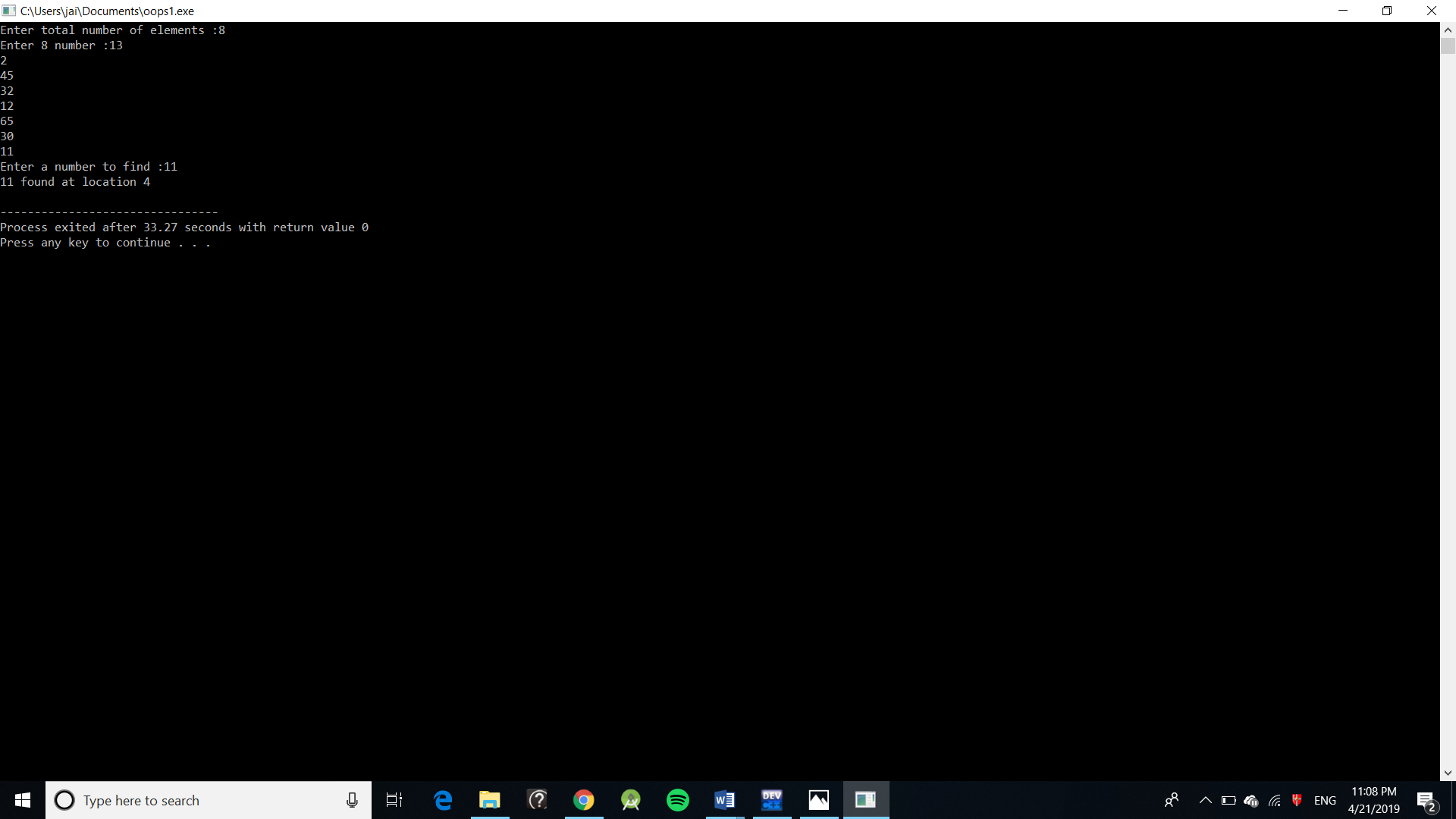
cout<<"Not found! "<<search<<" is not present in the list.";

}

return 0;

}

***Output -***

******

**Practical – 3**

***Aim –*** *To implement the concepts of classes and function overloading.*

***Code –***

#include<iostream>

using namespace std;

class ovr{

int a;

public:

int sum(int x, int y){

cout<<x+y<<endl;

}

double sum(double x, double y){

cout<<x+y<<endl;

}

double sum(double x, double y, double z){

cout<<x+y+z<<endl;

}

};

int main(){

ovr o1;

o1.sum(10,20);

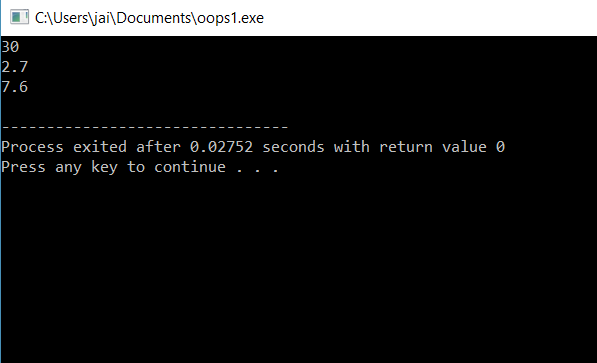
o1.sum(1.2 ,1.5);

o1.sum(1.4, 2.5, 3.7);

return 0;

}

***Output -***



**Practical – 4**

***Aim –*** *To implement the three types of function calls – Call by value, Call by address and Call by reference.*

***Code –***

1. ***Call by Value: -***

#include<iostream>

using namespace std;

void swap(int x, int y) {

int temp;

temp = x;

x = y;

y = temp;

cout<<"1st number now:"<<x<<endl;

cout<<"2nd number now:"<<y;

return;

}

int main(void)

{

int a,b;

cout<<"Enter 1st number:";

cin>>a;

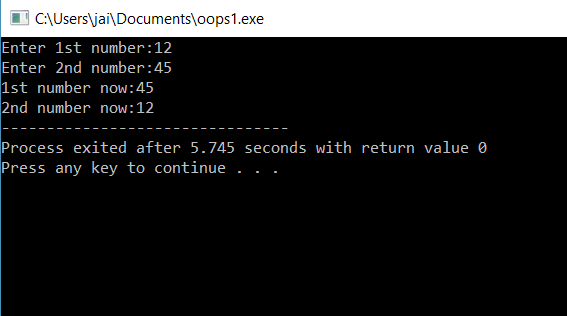
cout<<"Enter 2nd number:";

cin>>b;

swap(a,b);

}

***Output: -***



1. ***Call by Address: -***

#include<iostream>

using namespace std;

void swap(int \*,int \*);

int main (void)

{

int a,b;

a=10;

b=20;

cout<<"Before calling swap function: a="<<a<<" b="<<b;

swap (&a,&b);

cout<<"\nAfter calling swap function: a="<<a<<" b="<<b;

}

void swap(int \*x,int \*y)

{

int temp;

temp=\*x;

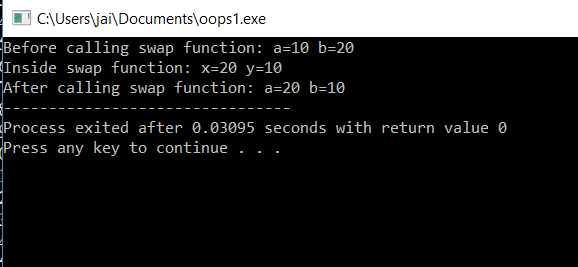
\*x=\*y;

\*y=temp;

cout<<"\nInside swap function: x="<<\*x<<" y="<<\*y;

}

***Output: -***



1. ***Call by Reference: -***

#include <iostream>

using namespace std;

void swap(int &x, int &y);

int main () {

int a = 100;

int b = 200;

cout << "Before swap, value of a :" << a << endl;

cout << "Before swap, value of b :" << b << endl;

swap(a, b);

cout << "After swap, value of a :" << a << endl;

cout << "After swap, value of b :" << b << endl;

return 0;

}

void swap(int &x, int &y) {

int temp;

temp = x;

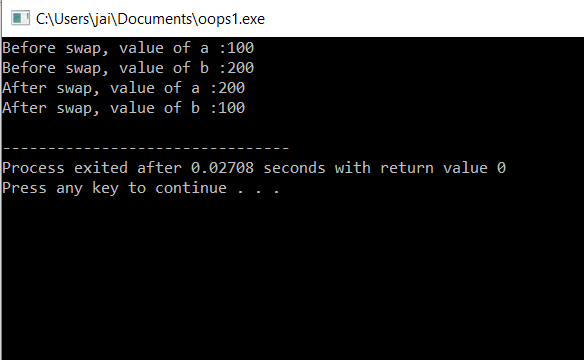
x = y;

y = temp;

return;

}

***Output: -***



**Practical – 5**

***Aim –*** *To implement the concepts of Inline functions and Macro functions.*

***Code –***

#include<iostream>

using namespace std;

#define CUBE(b) b\*b\*b

inline int cube(int a)

{

return a\*a\*a;

}

int main()

{

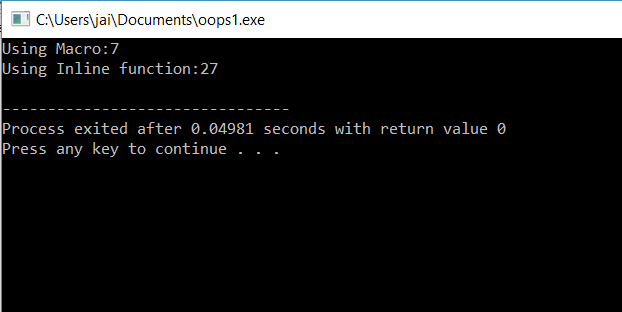
cout<<"Using Macro:"<<CUBE(1+2)<<endl;

cout<<"Using Inline function:"<<cube(1+2)<<endl;

return 0;

}

***Output –***



**Practical – 6**

***Aim –*** *To implement the concept of default function (arguments).*

***Code –***

#include<iostream>

using namespace std;

int sum(int x, int y, int z=0, int w=0)

{

return (x + y + z + w);

}

int main()

{

cout << sum(41, 45) << endl;

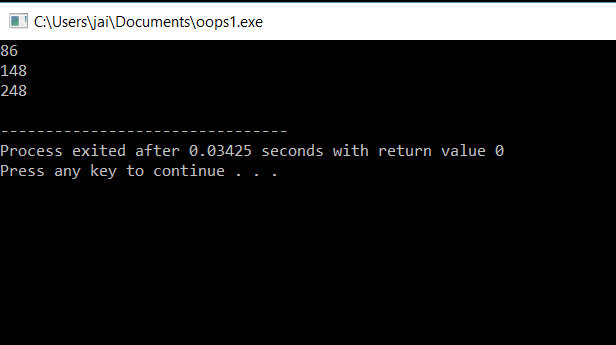
cout << sum(41, 45, 62) << endl;

cout << sum(41, 45, 62, 100) << endl;

return 0;

}

***Output –***



**Practical – 7**

***Aim –*** *To pass and return an object as an argument both by value and reference.*

***Code –***

#include <bits/stdc++.h>

using namespace std;

class test {

public:

int a;

test add(test tx, test ty)

{

test t;

t.a=0;

t.a = t.a + tx.a + ty.a;

return t;

}

};

int main()

{

test t1, t2, t3;

t1.a = 50;

t2.a = 100;

t3.a = 0;

cout << "Initial Values \n";

cout << "Value of object 1: " << t1.a << ", \nobject 2: " << t2.a << ", \nobject 3: " << t3.a

<< "\n";

t3 = t3.add(t1, t2);

cout << "New values \n";

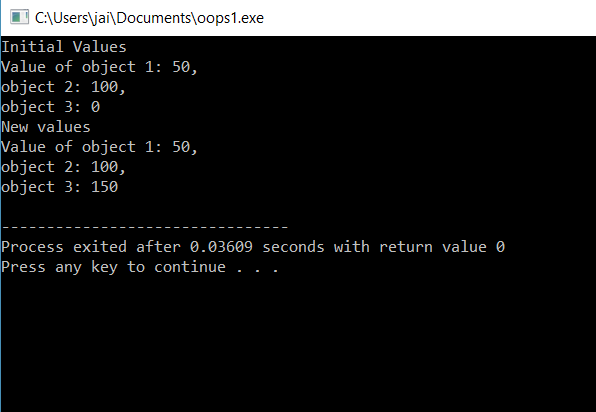
cout << "Value of object 1: " << t1.a << ", \nobject 2: " << t2.a << ", \nobject 3: " << t3.a

<< "\n";

return 0;

}

***Output –***



**Practical – 8**

***Aim –*** *To implement the concept of Constant member function and constant object.*

***Code –***

#include<iostream>

using namespace std;

class Test {

int value;

public:

Test(int v = 0)

{

value = v;

}

int getValue() const

{

return value;

}

};

int main() {

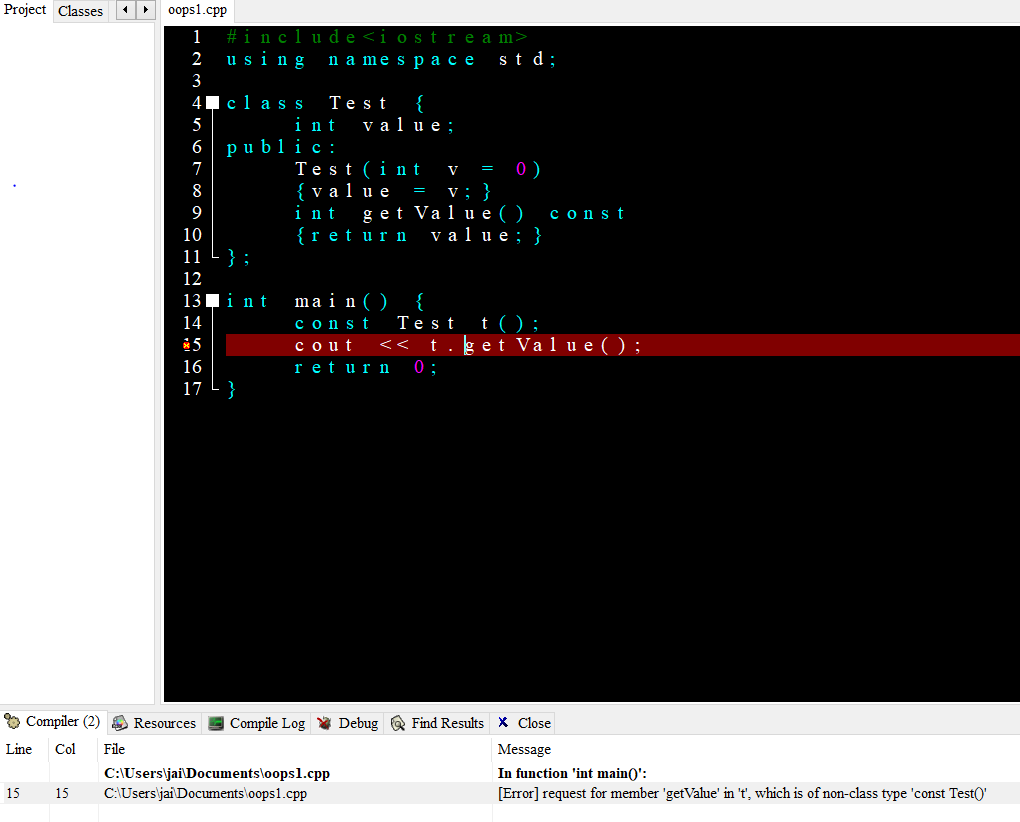
const Test t();

cout << t.getValue();

return 0;

}

***Output –***



**Practical – 9**

***Aim –*** *To implement the concept of Static member variable also known as class variable.*

***Code –***

#include<iostream>

using namespace std;

class Bank{

int acc\_num,balance;

int time =1;

static double rate;

public:

int getInput(){

cout<<"Enter Account Number:";

cin>>acc\_num;

cout<<"Enter Balance:";

cin>>balance;

}

int showOutput(){

cout<<"Account Number:"<<acc\_num<<endl<<"Balance after 1 year (Rate = "<<rate<<"P.A ):"<<balance;

}

int calcIntrest();

static int mod();

};

int Bank::calcIntrest(){

int intrest = (balance \* rate \* time)/100;

balance = balance + intrest;

}

double Bank::rate =1.5;

int main(){

Bank b1;

b1.getInput();

b1.calcIntrest();

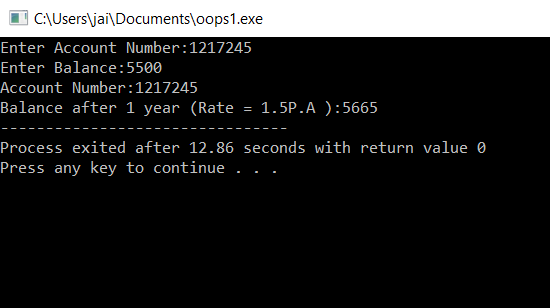
b1.calcIntrest();

b1.showOutput();

return 0;

}

***Output –***



**Practical – 10**

***Aim –*** *To implement the concept of Static member function.*

***Code –***

#include <iostream>

using namespace std;

class Box {

public:

static int objectCount;

Box(double l = 2.0, double b = 2.0, double h = 2.0) {

cout <<"Constructor called." << endl;

length = l;

breadth = b;

height = h;

objectCount++;

}

double Volume() {

return length \* breadth \* height;

}

static int getCount() {

return objectCount;

}

private:

double length;

double breadth;

double height;

};

int Box::objectCount = 0;

int main(void) {

cout << "Inital Stage Count: " << Box::getCount() << endl;

Box Box1(3.3, 1.2, 1.5);

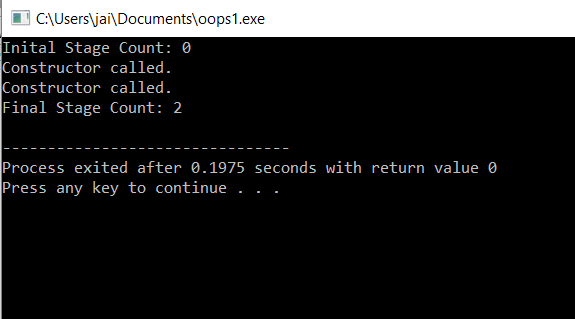
Box Box2(8.5, 6.0, 2.0);

cout << "Final Stage Count: " << Box::getCount() << endl;

return 0;

}

***Output –***



**Practical – 11**

***Aim –*** *To implement dynamic operations using Constructors and Destructors.*

***Code –***

#include<iostream>

#include<string.h>

using namespace std;

class str

{

char \*name;

int len;

public:

str(char \*s)

{

len=strlen(s);

name=new char[len+ 1];

strcpy(name,s);

}

int com(str &ss2)

{

int k;

k=strcmp(name,ss2.name);

if(k==0)

cout<<"\nBoth persons have same name";

else

cout<<"\nBoth persons have different name";

}

void put()

{

cout<<"\nName of person : "<<name;

}

~str()

{

cout<<"\nRelease memory allocated to '"<<name<<"'";

delete[] name;

}

};

int main(void)

{

str obj1 ("Uzumaki Naruto");

str obj2("Uchiha Sasuke");

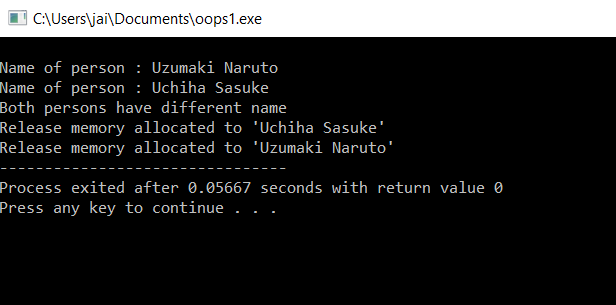
obj1.put();

obj2.put();

obj1.com(obj2);

}

***Output –***



**Practical – 12**

***Aim –*** *To implement the concepts of dynamic memory allocation.*

***Code –***

#include <iostream>

#include <new>

using namespace std;

int main ()

{

int i,n;

int \* p;

cout << "How many numbers would you like to type? ";

cin >> i;

p= new (nothrow) int[i];

if (p == NULL)

cout << "Error: memory could not be allocated";

else{

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

{ cout << "Enter number: ";

cin >> p[n];

}

cout << "You have entered: ";

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

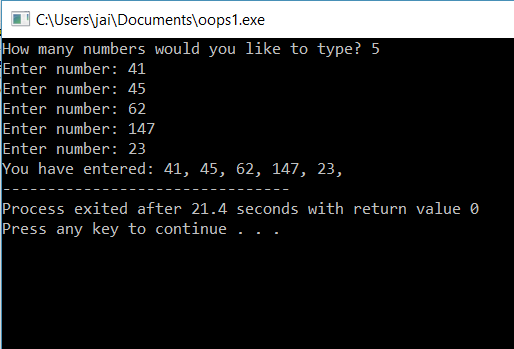
cout << p[n] << ", ";

delete[] p;

}

}

***Output –***



**Practical – 13**

***Aim –*** *To implement the concept of Single Inheritance.*

***Code –***

#include <iostream>

using namespace std;

class base

{

public:

int x;

void getdata()

{

cout << "Enter the value of x = "; cin >> x;

}

};

class derive : public base

{

private:

int y;

public:

void readdata()

{

cout << "Enter the value of y = "; cin >> y;

}

void product()

{

cout << "Product = " << x \* y;

}

};

int main()

{

derive a;

a.getdata();

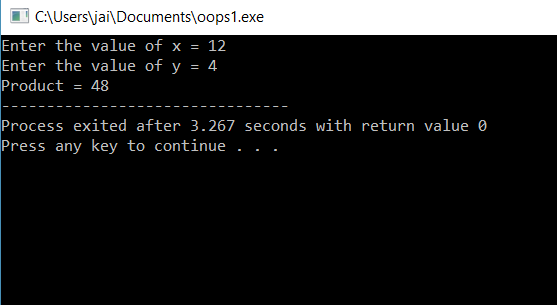
a.readdata();

a.product();

return 0;

}

***Output –***



**Practical – 14**

***Aim –*** *To implement the concepts of Multiple Inheritance in classes.*

***Code –***

#include <iostream>

using namespace std;

class Vehicle {

public:

Vehicle()

{

cout << "This is a Vehicle" << endl;

}

};

class FourWheeler {

public:

FourWheeler()

{

cout << "This is a 4 wheeler Vehicle" << endl;

}

};

class Car: public Vehicle, public FourWheeler {

};

int main()

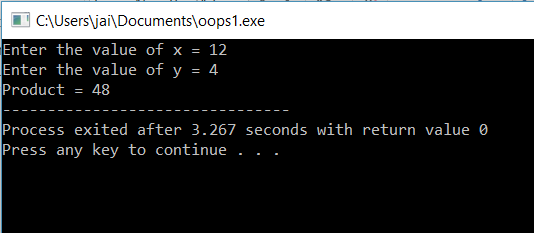
{

Car obj;

return 0;

}

***Output –***



**Practical – 15**

***Aim –*** *To implement the concepts of Multilevel Inheritance in classes.*

***Code –***

#include <iostream>

using namespace std;

class Vehicle

{

public:

Vehicle()

{

cout << "This is a Vehicle" << endl;

}

};

class fourWheeler: public Vehicle

{ public:

fourWheeler()

{

cout<<"Objects with 4 wheels are vehicles"<<endl;

}

};

class Car: public fourWheeler{

public:

Car()

{

cout<<"Car has 4 Wheels"<<endl;

}

};

int main()

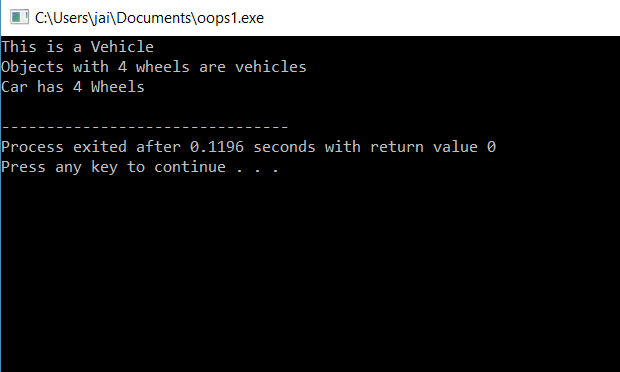
{

Car obj;

return 0;

}

***Output –***



**Practical – 16**

***Aim –*** *To overload pre and post increment operators (++).*

***Code –***

#include<iostream>

using namespace std;

class test{

int a,b;

public: int getData(){

cin>>a;

}

test operator++(){

test temp;

a=a+1;

temp.a=a;

return temp;

}

test operator++(int){

test temp;

temp.a=a;

a=a+1;

return temp;

}

int showData(){

cout<<”New Data:”<<a;

}

};

int main(){

test t1,t2;

cout<<"Enter Data:";

t1.getData();

int var;

cout<<"Press 1 for Pre Increment"<<endl<<"Press 2 for Post increment"<<endl;

cin>>var;

if(var ==1)

t2=++t1;

else

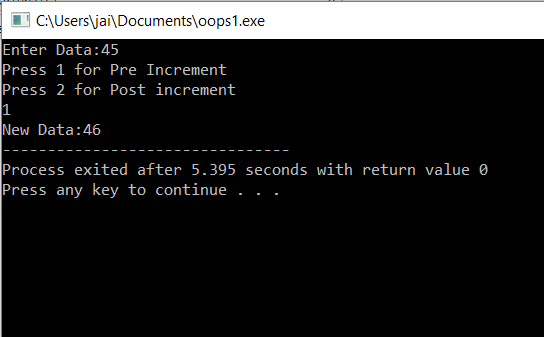
t2=t1++;

t2.showData();

return 0;

}

***Output –***



**Practical – 17**

***Aim –*** *To overload addition, subtraction and multiplication binary operators.*

***Code –***

#include<iostream>

#include<conio.h>

#include<process.h>

using namespace std;

class demo

{

float a,b;

public:

void getdata();

void display();

demo operator +(demo);

demo operator -(demo);

demo operator \*(demo);

demo operator /(demo);

};

void demo::getdata()

{

cout<<"Enter values of a and b:";

cin>>a>>b;

}

void demo::display()

{

cout<<"a="<<a<<"tb="<<b;

}

demo demo::operator +(demo d1)

{

demo d2;

d2.a=a+d1.a;

d2.b=b+d1.b;

return d2;

}

demo demo::operator -(demo d1)

{

demo d2;

d2.a=a-d1.a;

d2.b=b-d1.b;

return d2;

}

demo demo::operator \*(demo d1)

{

demo d2;

d2.a=a\*d1.a;

d2.b=b\*d1.b;

return d2;

}

demo demo::operator /(demo d1)

{

demo d2;

d2.a=a/d1.a;

d2.b=b/d1.b;

return d2;

}

int main()

{

int ch;

demo d1,d2,d3;

cout<<"First Object:\n";

d1.getdata();

cout<<"\nSecond Object:\n";

d2.getdata();

cout<<"\n\nOperator Overloadig Menu";

cout<<"\n\n1.Addition\n2.Subtraction\n3.Multiplication\n4.Division\n5.Exit";

cout<<"\n\nEnter your choice(1-5):";

cin>>ch;

switch(ch)

{

case 1: d3=d1+d2;

cout<<"\nThird Object:\n";

d3.display();

break;

case 2: d3=d1-d2;

cout<<"\nThird Object:\n";

d3.display();

break;

case 3: d3=d1\*d2;

cout<<"\nThird Object:\n";

d3.display();

break;

case 4: d3=d1/d2;

cout<<"\nThird Object:\n";

d3.display();

break;

case 5: exit(0);

break;

default: cout<<"Wrong Choice!!!Press any key to exit";

}

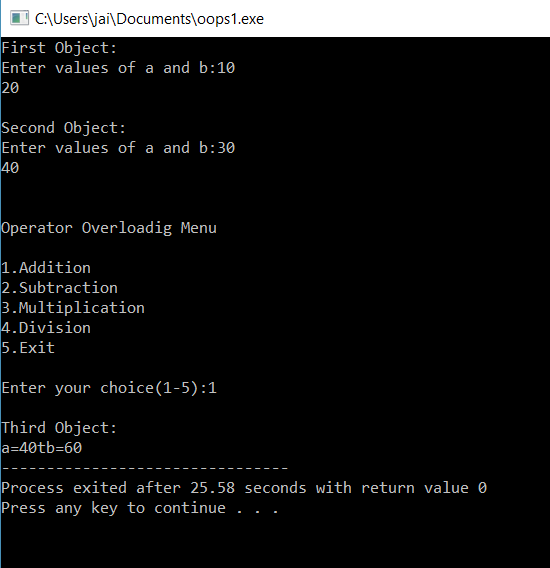
getch();

return 0;

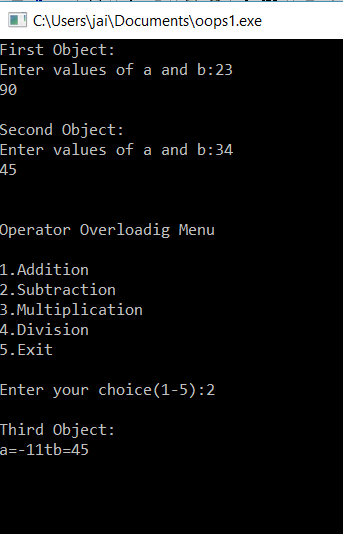
}

***Output –***

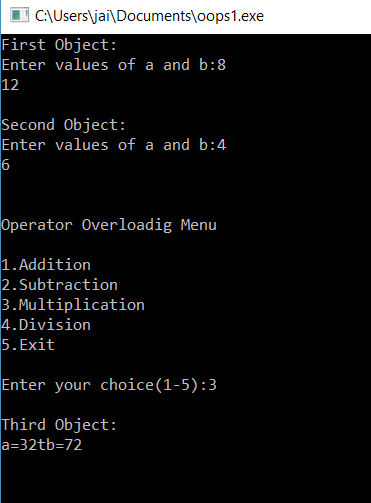
1. *Addition: -*



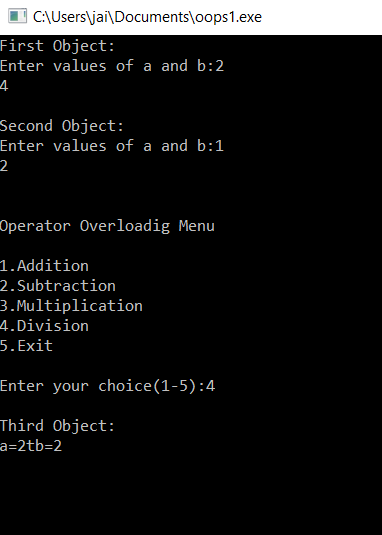
1. *Subtraction: -*



1. *Multiplication: -*



1. *Division: -*



**Practical – 18**

***Aim –*** *To implement the concept of a pure virtual function.*

***Code –***

#include<iostream>

using namespace std;

class BaseClass

{

public:

virtual void Display1()=0;

virtual void Display2()=0;

void Display3()

{

cout<<"\n\tThis is Display3() method of Base Class";

}

};

class DerivedClass : public BaseClass

{

public:

void Display1()

{

cout<<"\n\tThis is Display1() method of Derived Class";

}

void Display2()

{

cout<<"\n\tThis is Display2() method of Derived Class";

}

};

int main(void)

{

DerivedClass D;

D.Display1();

D.Display2();

D.Display3();

}

***Output –***

