**Practical : 1**

**Aim :**  WAP to create a Message class with a constructor that takes a single string with a default value. Create a private member string, and in the constructor simply assign the argument string to your internal string. Create two overloaded member functions called print( ): one that takes no arguments and simply prints the message stored in the object, and one that takes a string argument, which it prints in addition to the internal message.

**Program :**

#include<iostream>

using namespace std;

class Message

{

private :

char s[100];

public :

Message(char n[])

{

cout << "Enter the string : " << n;

}

void print()

{

cout << endl << "no arguments";

}

void print(char n[])

{

cout << endl << "my name is : " << n;

}

};

int main()

{

Message o("Hello ");

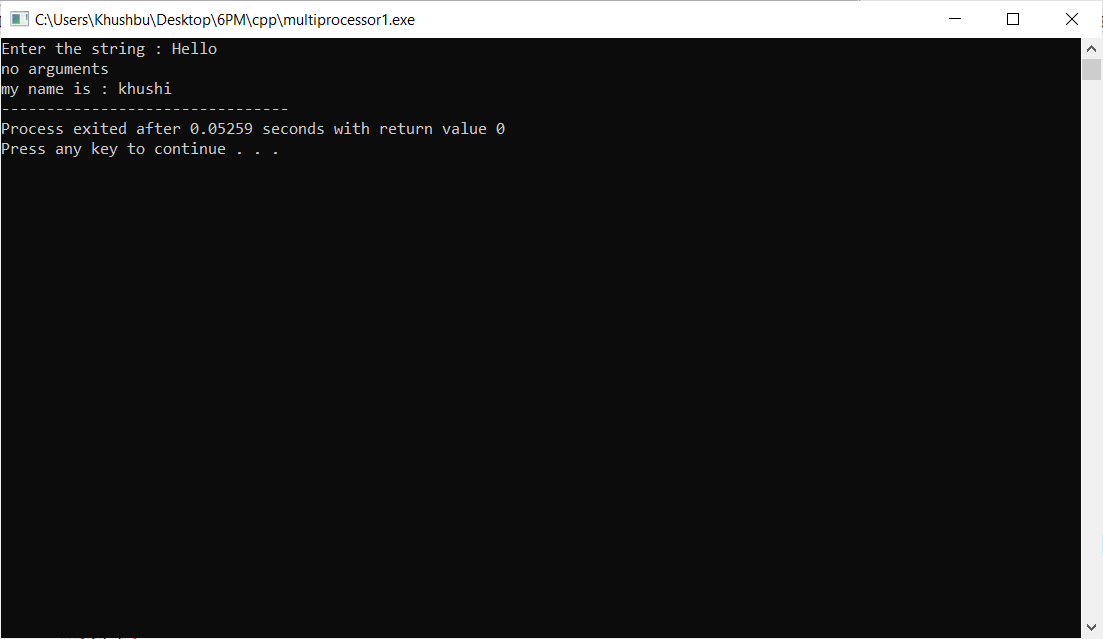
o.print();

o.print("khushi");

return 0;

};

**Output :**

****

**Practical : 3**

**Aim :** WAP which illustrate the use of Method Overriding concept.

**Program :**

#include<iostream>

using namespace std;

class Parent

{

public :

void overriding()

{

cout << "hello";

}

};

class Child : public Parent

{

public:

void overriding()

{

Parent :: overriding();

cout << endl << "my name is khushi..";

}

};

int main ()

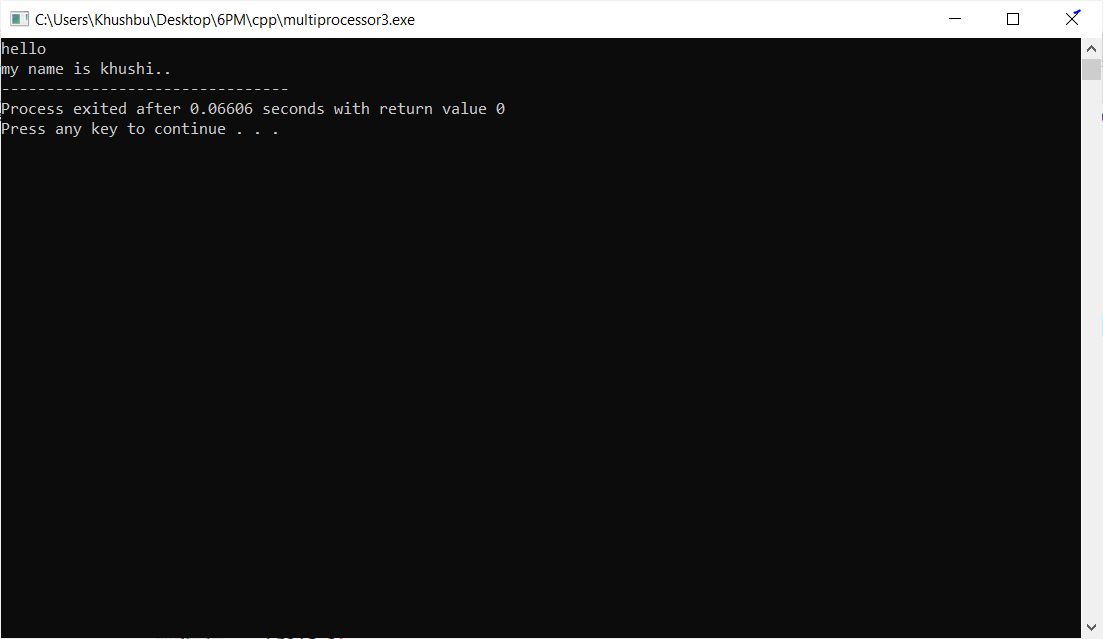
{

Child o;

o.overriding();

return 0;

}

**Output :**

**Practical : 4**

**Aim :** WAP to demonstrate usage of Virtual Function.

**Program :**

#include <iostream>

using namespace std;

class Parent {

public:

virtual void print() {

cout << "parent class" << endl;

}

};

class Child : public Parent {

public:

void print() {

cout << "child class" << endl;

}

};

int main() {

Child c;

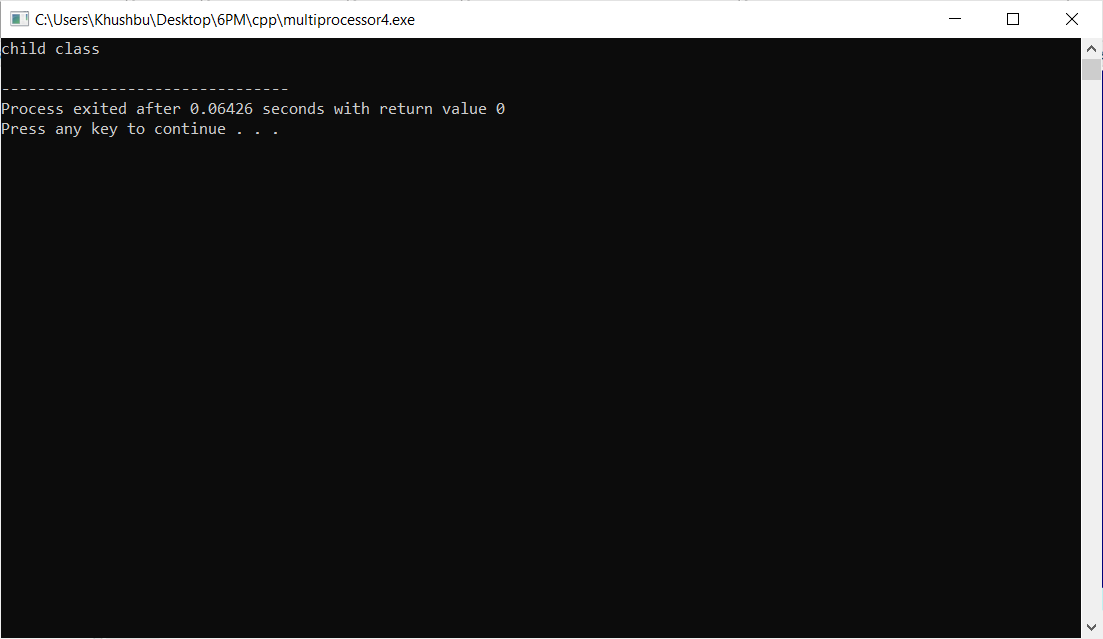
Parent\* p = &c;

p->print();

return 0;

}

**Output :**

****

**Practical : 5**

**Aim :** WAP to create an Abstract Class Shape. Make two sub-class Circle and Triangle which implements a method from Shape class called getArea().

**Program :**

#include <iostream>

using namespace std;

class Shape {

public:

virtual int getArea() = 0;

void setWidth(int w) {

width = w;

}

void setHeight(int h) {

height = h;

}

protected:

int width;

int height;

};

class Circle: public Shape {

public:

int getArea() {

return (3.14 \* height\*height);

}

};

class Triangle: public Shape {

public:

int getArea() {

return (width \* height)/2;

}

};

int main(void) {

Circle c;

Triangle Tri;

c.setHeight(5);

cout << "Total Circle area: " << c.getArea() << endl;

Tri.setWidth(5);

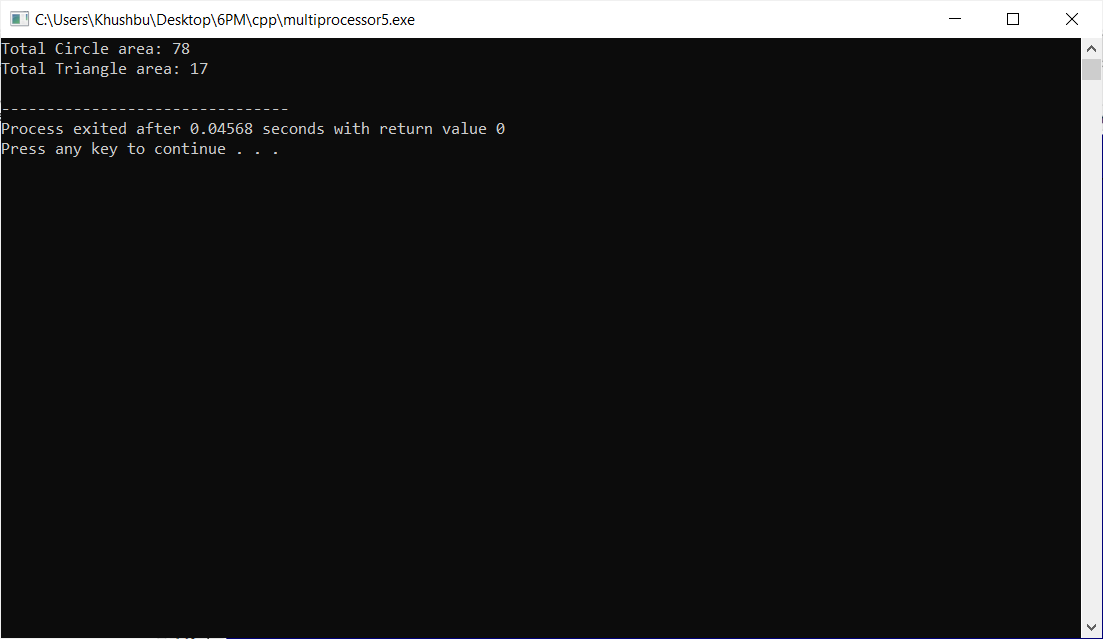
Tri.setHeight(7);

cout << "Total Triangle area: " << Tri.getArea() << endl;

return 0;

}

**Output :**

****

**Practical : 6**

**Aim :** WAP for unary increment (++) and decrement (--) operator overloading.

**Program :**

#include<iostream>

using namespace std;

class A

{

private:

int n;

public:

void setNum(int x)

{

n=x;

}

void getNum()

{

cout << "value of n is: " << n;

}

void operator++ ()

{

n=++n;

}

void operator-- ()

{

n=--n;

}

};

int main()

{

A a;

a.setNum(16);

++a;

cout << "After increment - ";

a.getNum();

cout << endl;

--a;

cout << "After decrement - ";

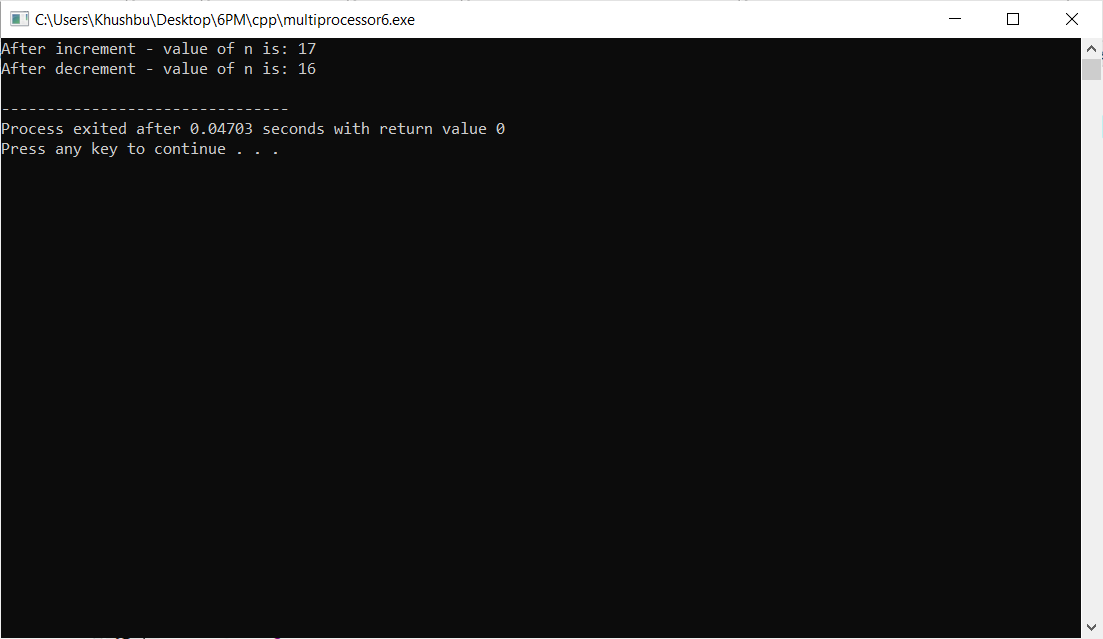
a.getNum();

cout << endl;

return 0;

}

**Output :**

****

**Practical : 7**

**Aim :** WAP to add two objects using binary plus (+) operator overloading.

**Program :**

#include<iostream>

using namespace std;

class A

{

private:

int n;

public:

void setNum(int x)

{

n=x;

}

void getNum(void)

{

cout << "Number is: " << n;

}

A operator +(A &obj)

{

A x;

x.n=this->n + obj.n;

return (x);

}

};

int main()

{

A n1,n2,sum;

n1.setNum(100);

n2.setNum(200);

sum=n1+n2;

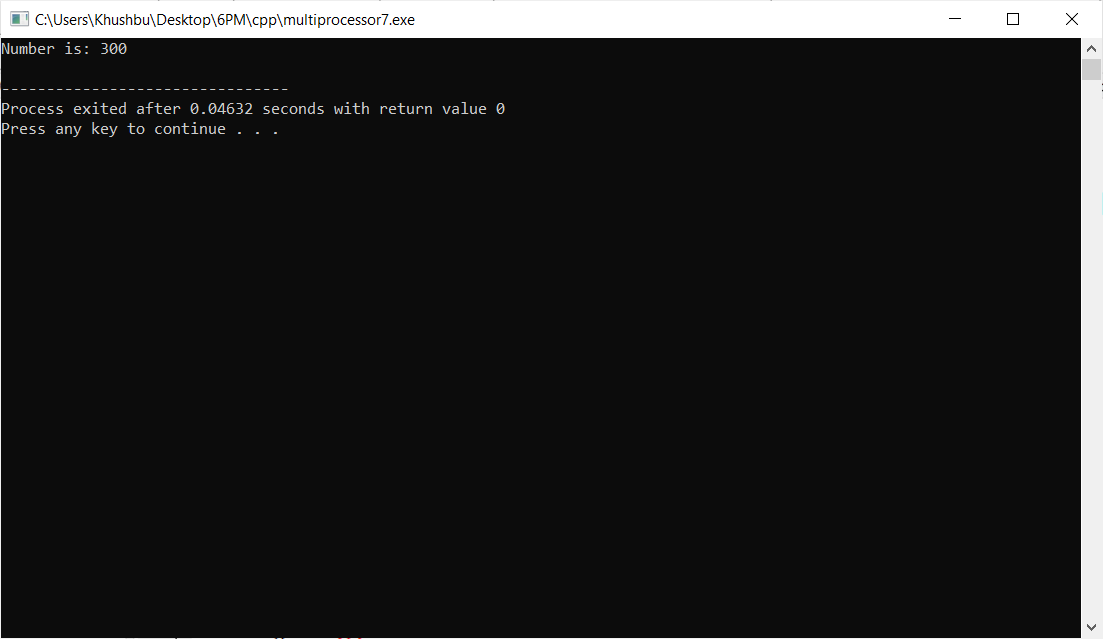
sum.getNum();

cout << endl;

return 0;

}

**Output :**

****

**Practical : 8**

**Aim :** WAP to add two distances using binary plus (+) operator overloading.

**Program :**

#include<iostream>

using namespace std;

class Distance

{

private:

int feet,inches;

public:

void setDistance(void)

{

cout << "Enter feet: ";

cin >>feet;

cout << "Enter inches: ";

cin >>inches;

}

void getDistance(void)

{

cout << "Feet:" << feet << "\t" << "Inches:" << inches << endl;

}

Distance operator+(Distance &dist1)

{

Distance temp;

temp.inches= inches + dist1.inches;

temp.feet = feet + dist1.feet + (temp.inches/12);

temp.inches=temp.inches%12;

return temp;

}

};

int main()

{

Distance d1,d2,d3;

cout << "Enter first distance:" << endl;

d1.setDistance();

cout << endl;

cout << "Enter second distance:" << endl;

d2.setDistance();

cout << endl;

d3=d1+d2;

cout << "Total Distance:" << endl;

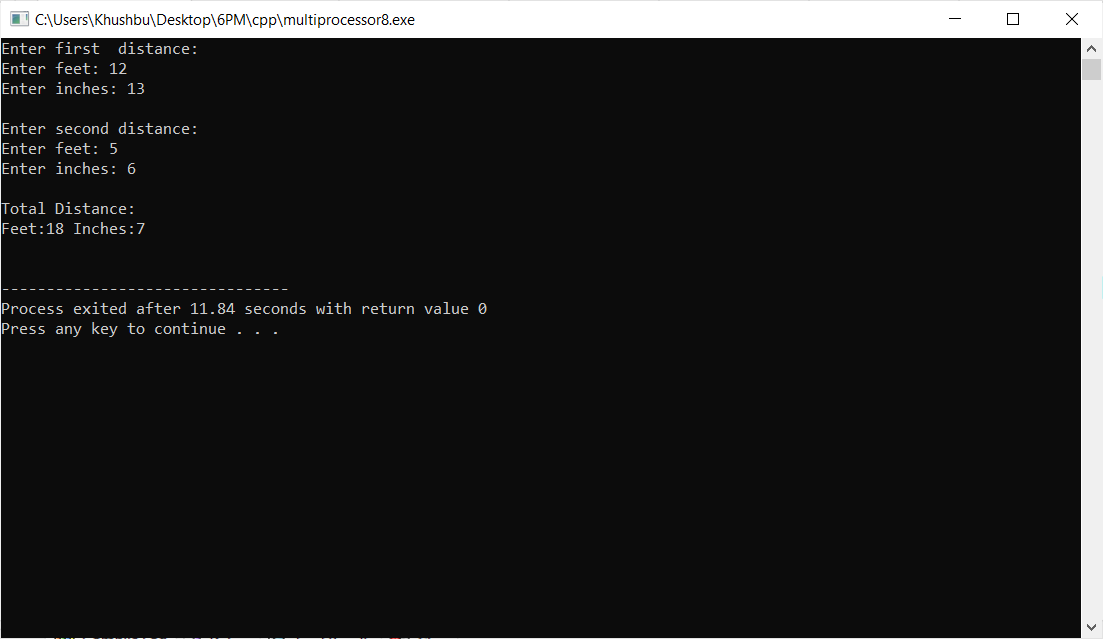
d3.getDistance();

cout << endl;

return 0;

}

**Output :**

****