Object-Oriented Programming

Class

A class is an expanded concept of a data structure: instead of holding only data, it can hold both data and functions.

An object is an instantiation of a class. In terms of variables, a class would be the type, and an object would be the variable. The format of the class:

```
class class_name {
access_specifier_1:
member1;
access_specifier_2:
member2;
...
} object_names;
```

Where class_name is a valid identifier for the class, object_names is an optional list of names for objects of this class. The body of the declaration can contain members, that can be either data or function declarations and optionally access specifiers. All is very similar to the declaration on data structures, except that we can now include also functions and members, but also this new thing called *access specifier*. An access specifier is one of the following three keywords: private, public or protected. These specifiers modify the access rights that the members following them acquire:

- private members of a class are accessible only from within other members of the same class or from their *friends*.
- protected members are accessible from members of their same class and from their friends, but also from members of their derived classes.
- Finally, public members are accessible from anywhere where the object is visible.

By default, all members of a class declared with the class keyword have private access for all its members. Therefore, any member that is declared before one other class specifier automatically has private access. For example:

```
class CRectangle {
                int x, y;
          public:
              void set_values (int,int);
               int area (void);
                  } rect;
```

Declares a class (i.e., a type) called CRectangle and an object (i.e., a variable) of this class called rect. This class contains four members: two data members of type int (member x and member y) with private access (because private is the default access level) and two member functions with public access: set_values() and area(), of which for now we have only included their declaration, not their definition.

Notice the difference between the class name and the object name: In the previous example, CRectangle was the class name (i.e., the type), whereas rect was an object of type CRectangle. It is the same relationship int and a have in the following declaration:

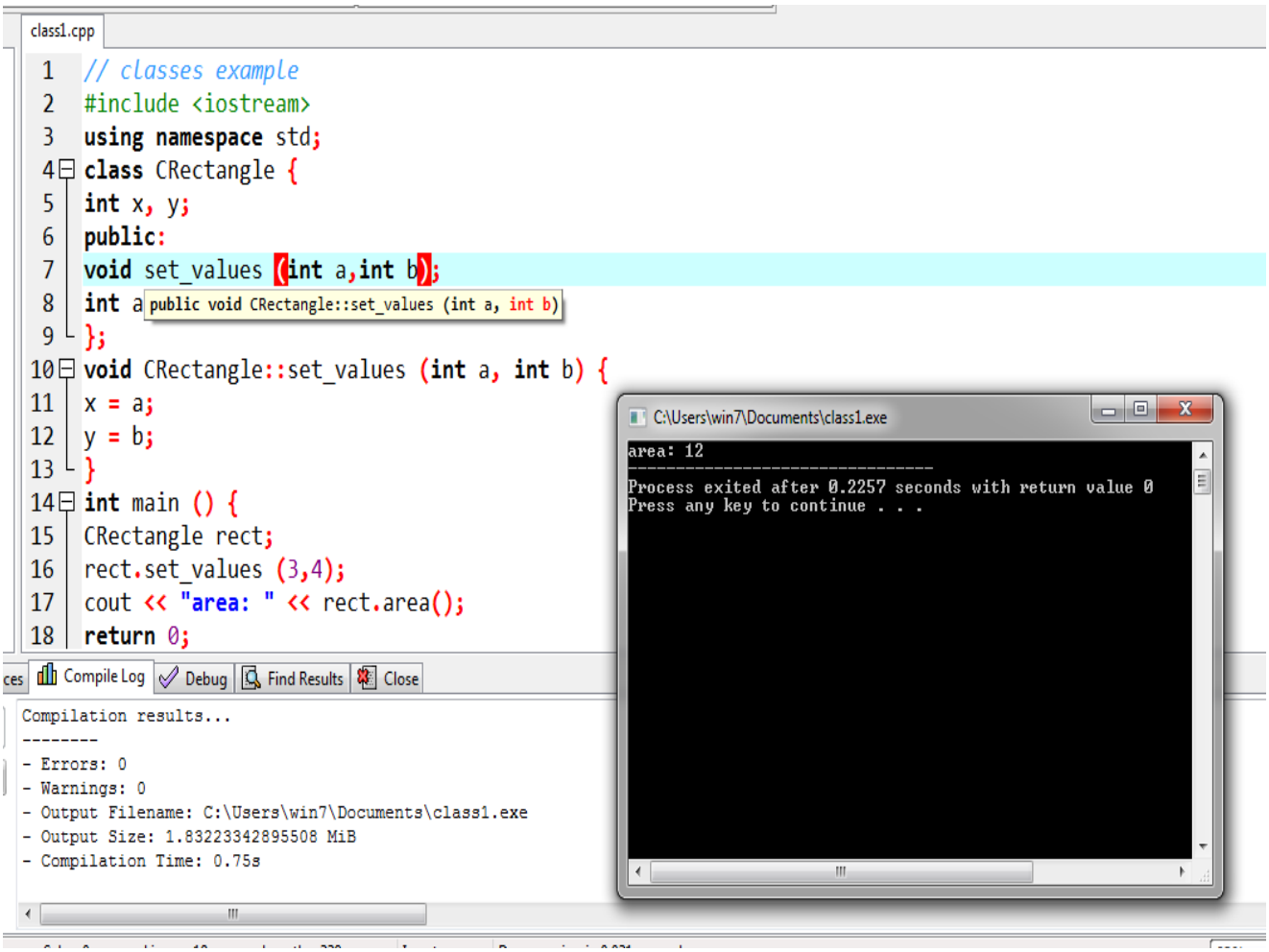
```
int a;
```

where int is the type name (the class) and a is the variable name (the object).

After the previous declarations of CRectangle and rect, we can refer within the body of the program to any of the public members of the object rect as if they were normal functions or normal variables, just by putting the object's name followed by a dot (.) and then the name of the member. All very similar to what we did with plain data structures before. For example:

```
rect.set_values (3,4);
myarea = rect.area();
```

The only members of rect that we cannot access from the body of our program outside the class are x and y, since they have private access and they can only be referred from within other members of that same class. Here is the complete example of class CRectangle:



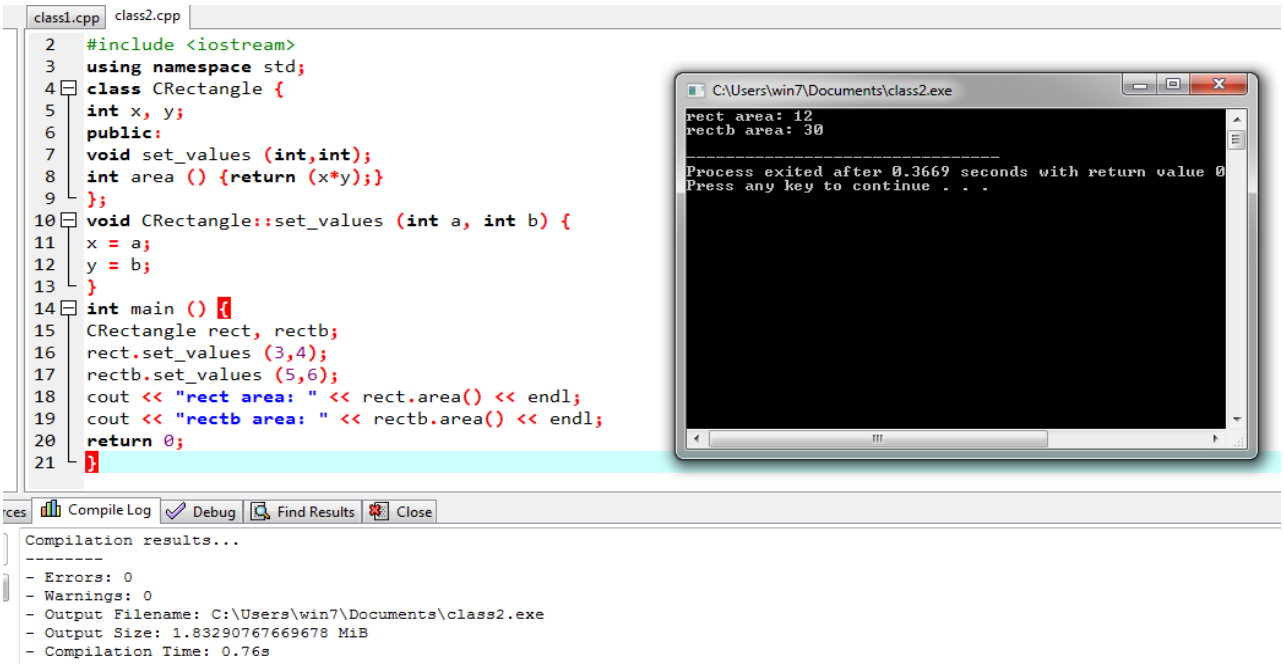
The most important new thing in this code is the operator of scope (::, two colons) included in the definition of set_values(). It is used to define a member of a class from outside the class definition itself. You may notice that the definition of the member function area() has been included directly within the definition of the CRectangle class given its extreme simplicity, whereas set_values() has only its prototype declared within the class, but its definition is outside it. In this outside declaration, we must use the operator of scope (::) to specify that we are defining a function that is a member of the class CRectangle and not a regular global function. The scope operator (::) specifies the class to which the member being declared belongs, granting exactly the same scope properties as if this function definition was directly included within the class definition. For example, in the function set_values() of the previous code, we have been able to use the variables x and y, which are private members of class CRectangle, which means they are only accessible from other members of their class.

The only difference between defining a class member function completely within its class or to include only the class. This makes sense, since we have already

defined a member function to set values for those members within the object: the member function set_values(). Therefore, the rest of the program does not need to have direct access to them. Perhaps in a so simple example as this, it is difficult to see a utility in protecting those two variables, but in greater projects, it may be very important that values cannot be modified in an unexpected way (unexpected from the point of view of the object). One of the greater advantages of a class is that, like any other type, we can declare several objects of it. For example, following with the previous example of class CRectangle, we could have declared the object rectb in addition to the object rect: prototype and later its definition, is that in the first case the function will automatically be considered an inline member function by the compiler, while in the second it will be a normal (not-inline) class member function, which in fact supposes no difference in behavior.

Members x and y have private access (remember that if nothing else is said, all members of a class defined with keyword class have private access). By declaring them private we deny access to them from anywhere outside the class. This makes sense since we have already defined a member function to set values for those members within the object: the member function set_values(). Therefore, the rest of the program does not need to have direct access to them. Perhaps in a so simple example as this, it is difficult to see a utility in protecting those two variables, but in greater projects, it may be very important that values cannot be modified in an unexpected way (unexpected from the point of view of the object).

One of the greater advantages of a class is that, like any other type, we can declare several objects of it. For example, following the previous example of class CRectangle, we could have declared the object rectb in addition to the object rect:



In this concrete case, the class (type of the objects) to which we are talking about is CRectangle, of which there are two instances or objects: rect and rectb. Each one of them has its own member variables and member functions.

Notice that the call to rect.area() does not give the same result as the call to rectb.area(). This is because each object of class CRectangle has its own variables x and y, as they, in some way, have also their own function members set_value() and area() that each uses its object's own variables to operate.

That is the basic concept of *object-oriented programming*: Data and functions are both members of the object. We no longer use sets of global variables that we pass from one function to another as parameters, but instead we handle objects that have their own data and functions embedded as members. Notice that we have not had to give any parameters in any of the calls to rect.area or rectb.area. Those member functions directly used the data members of their respective objects rect and rectb.

Q1: Write an O.O.P for finding the value of Y:

$$Y = 1/2^{1} + 2/2^{2} + 3/2^{3} + \ldots + n/2^{n}$$

Q2 : If you have an int a[5][5] .Write an O.O.P for finding :

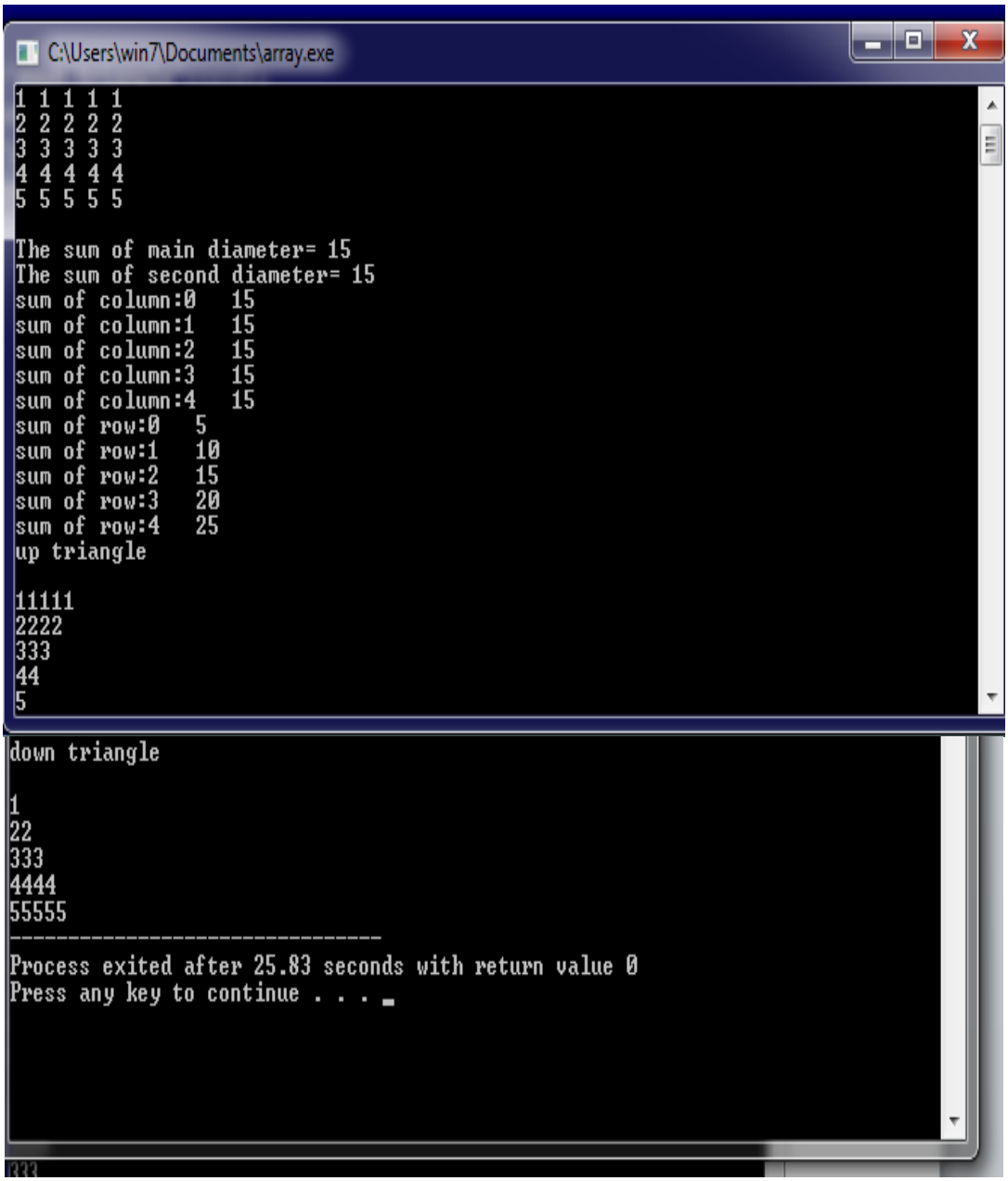
1. The sum of main diameter of a.
2. The sum of second diameter of a.
3. The sum of each column of a.
4. The sum of each row of a.
5. Print the elements of Up triangle of a.
6. Print the elements of Down triangle of a.

The solution of question2:

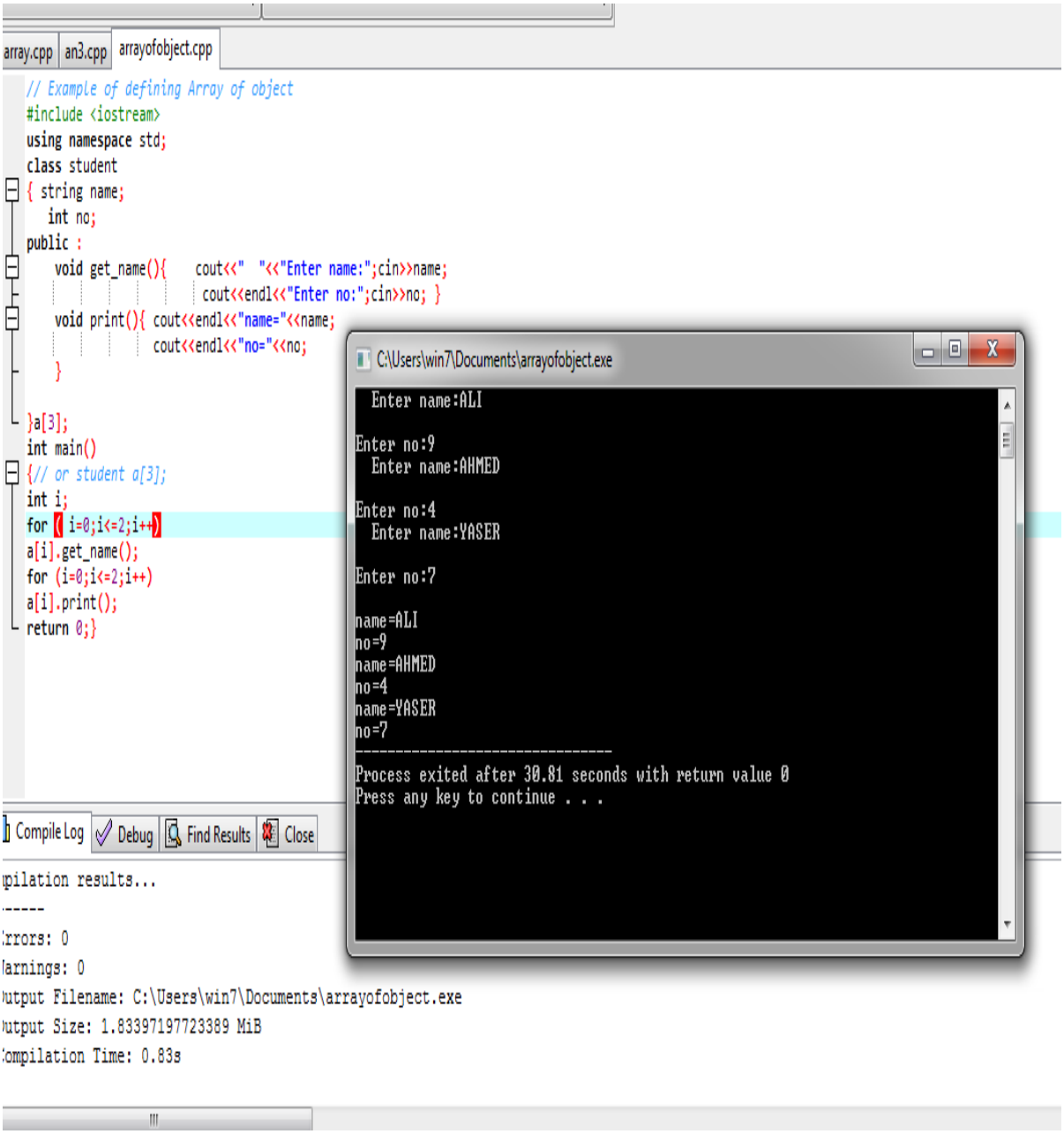
```
//If you have an int a[5][5] .Write an O.O.P for finding :
//1. The sum of main diameter of a.
//2. The sum of second diameter of a.
//3. The sum of each column of a.
//4. The sum of each row of a.
//5. Print the elements of Up triangle of a.
//6. Print the elements of  Down triangle of a.
#include <iostream>
using namespace std;
class array_op
{int x[5][5];
public :
      void get_x(){for (int i=0;i<=4;i++)
                  for (int j=0;j<=4;j++)cin>>x[i][j];}
   void main_di(){
  int sum=0;
  for (int i=0;i<=4;i++) sum=sum+x[i][i];
  cout<<endl<<"The sum of main diameter= "<<sum;
  }
  void second_di()
{ int sum=0;for (int i=0;i<=4;i++)
                  for (int j=0;j<=4;j++)
                              if (i+j==4)sum=sum+x[i][j];
                              cout<<endl<<"The sum of second diameter=
"<<sum;
}

void sum_row(){int sum=0;for (int i=0;i<=4;i++)
                  {for (int j=0;j<=4;j++)
                              sum=sum+x[i][j];
                              cout<<endl<<"sum of row:"<<i<<"   "<<sum;
                              sum=0;
                              }
                        }
```

```
        void sum_column(){int sum=0;for (int i=0;i<=4;i++)
           {for (int j=0;j<=4;j++)
                    sum=sum+x[j][i];
                    cout<<endl<<"sum of column:"<<i<<"   "<<sum;
                    sum=0;
                    }
                }

    void top_tri(){cout<<endl<<"up triangle"<<endl;
                for (int i=0;i<=4;i++)
                {cout<<"\n";
                            for (int j=i;j<=4;j++)cout <<x[i][j];}


    }
void down_tri(){cout<<endl<<"down triangle"<<endl;
                for (int i=0;i<=4;i++)
                {cout<<"\n";
                            for (int j=0;j<=i;j++)cout <<x[i][j];}


    }

};
int main()
{array_op obj1;
obj1.get_x();
obj1.main_di();
obj1.second_di();
obj1.sum_column();
obj1.sum_row();
obj1.top_tri();
obj1.down_tri();
}
```

The output :


C:\Users\win7\Documents\array.exe
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
The sum of main diameter= 15
The sum of second diameter= 15
sum of column:0 15
sum of column:1 15
sum of column:2 15
sum of column:3 15
sum of column:4 15
sum of row:0 5
sum of row:1 10
sum of row:2 15
sum of row:3 20
sum of row:4 25
up triangle
11111
2222
333
44
5
down triangle
1
22
333
4444
55555
Process exited after 25.83 seconds with return value 0
Press any key to continue . . .

EXAMPLE OF ARRAY OF OBJECT



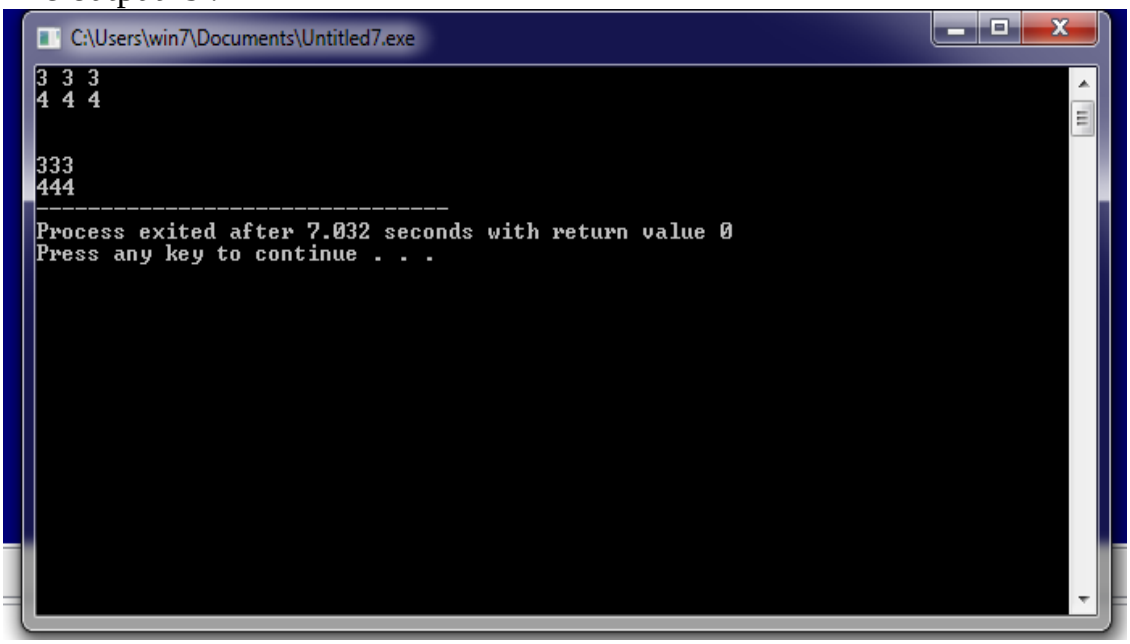
// If you have two separated classes ,each class has one dimention array a[3],b[3] ,Write oop for combing these arrays into c[3][2] .

// using friend function

```
#include <iostream>
using namespace std;
        class A;
```

```
        class B {
                int L[3];
                public:
        void get_b(int l[3]){int i;
                        for (i=0;i<=2;i++)L[i]=l[i];}
                       friend void mer(A a, B b);
        };
        class A {   int s[3];
                public:
                void get_a(int m[3]){int i;
                        for (i=0;i<=2;i++)s[i]=m[i];}

                friend void mer(A a, B b);
        };
void  mer(A a, B b)
        { int i,j;int c[3][2];
          for (i=0;i<=2;i++) {
          c[i][0]=a.s[i];
          c[i][1]=b.L[i];}
          cout<<endl;
        for (i=0;i<=1;i++)
        {
cout<<"\n";
                for (j=0;j<=2;j++)cout<<c[j][i] ;
        }
}

        int main()
        {A aa;B bb;
      int a1[3],a2[3],i;
      for (i=0;i<=2;i++) cin>>a1[i];
       for (i=0;i<=2;i++)cin >>a2[i];
     aa.get_a(a1);
          bb.get_b(a2);
    mer(aa,bb);

return 0;
        }
```

The output is :

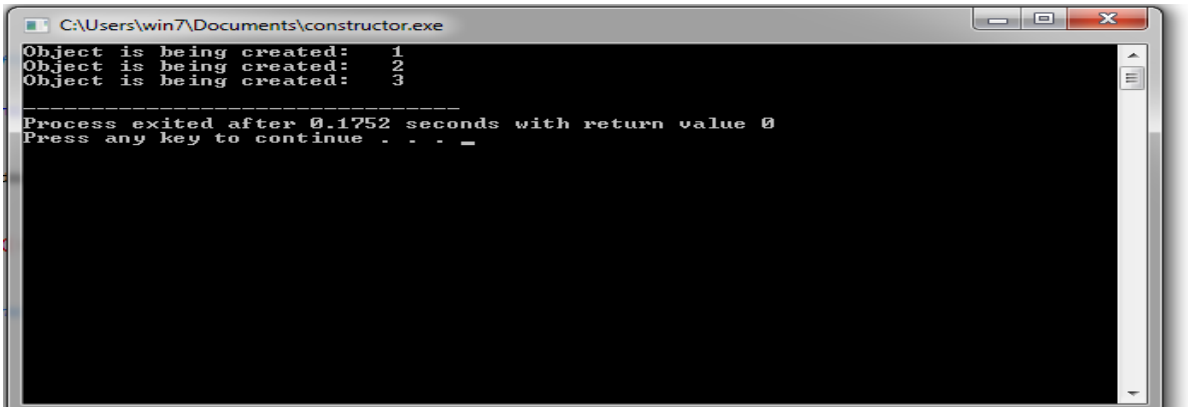


Constructor and Destructor

A **constructor** is a special member function of a class that is executed whenever we create new objects of that class.
A constructor will have exact same name as the class and it does not have any return type at all, not even void. Constructors can be very useful for setting initial values for certain member variables.Following example explains the concept of constructor:

```
#include  <iostream>
using namespace std;
int a=1;
class Line
{ public:
Line(); // This is the constructor
};
// Member functions definitions including constructor
Line::Line(void)
{cout << "Object is being created:   " <<a<<endl;
 a++;
}
```

```
// Main function for the program
int main( )
{
Line line; //create object1
Line aa;   //create object 2
Line ff;    //create object3
return 0;
}
```



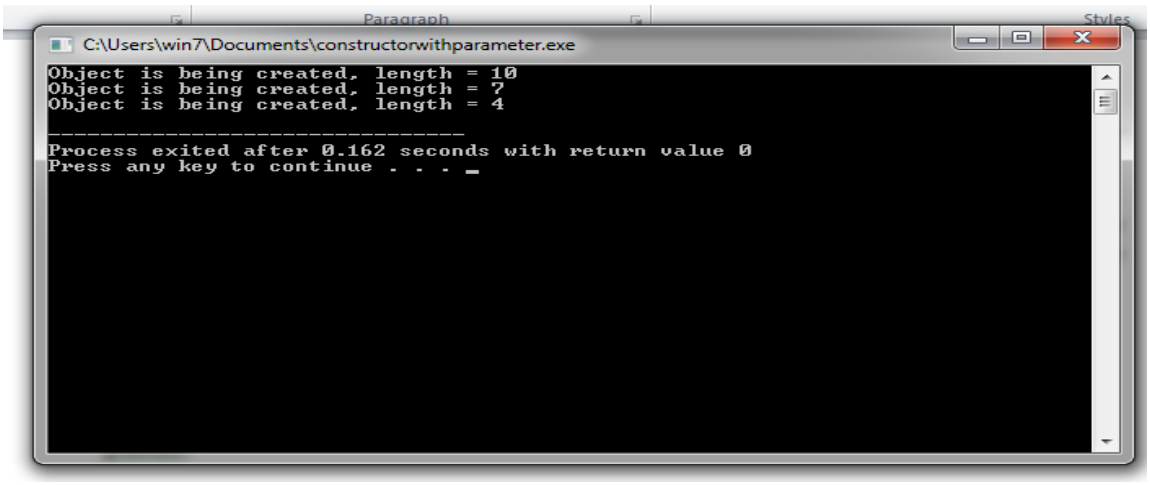
Parameterized Constructor

A default constructor does not have any parameter, but if you need, a constructor can have parameters. This helps you to assign initial value to an object at the time of its creation as shown in the following example:

Example 1

```
#include <iostream>
using namespace std;
class Line
{public:
Line(double len); // This is the constructor
double length;
};
// Member functions definitions including constructor
Line::Line( double len)
{
cout << "Object is being created, length = " << len << endl;
length = len; }
// Main function for the program
int main( )
{
```

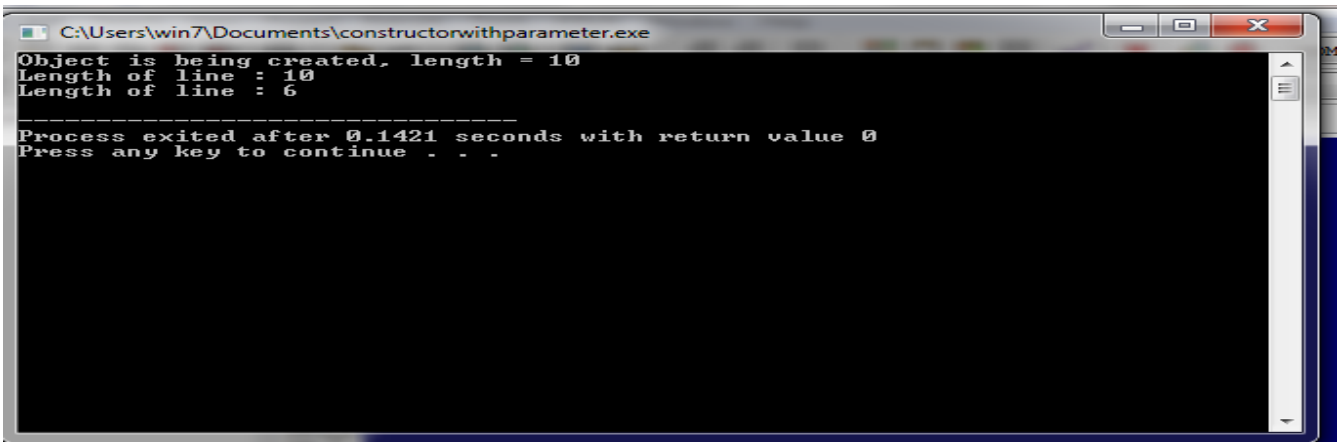
```
Line line1(10.0);//create object 1
Line line2(7.0);//create object 2
Line line3(4);//create object 3
return 0;
}
```



Example 2

```
#include <iostream>
using namespace std;
class Line
{
public:
void setLength( double len );
double getLength( void );
Line(double len); // This is the constructor
private:
double length;
};
// Member functions definitions including constructor
Line::Line( double len)
{
cout << "Object is being created, length = " << len << endl;
length = len;
}
void Line::setLength( double len )
{
```

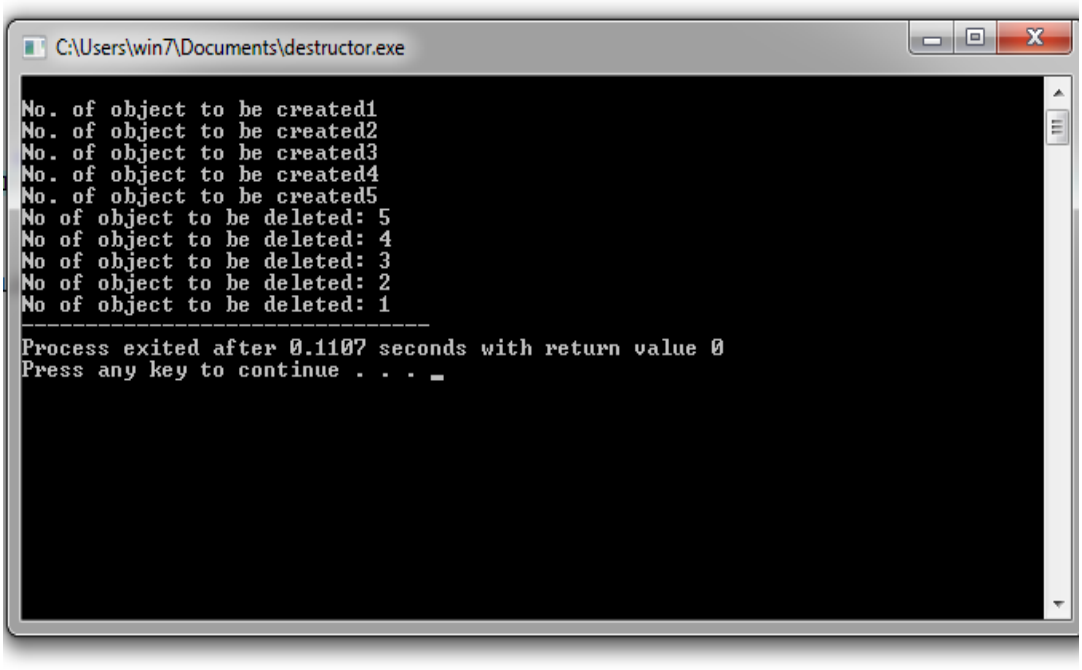
```
length = len;
}
double Line::getLength( void )
{
return length;
}
// Main function for the program
int main( )
{

Line line(10.0);
// get initially set length.
cout << "Length of line : " << line.getLength() <<endl;
// set line length again
line.setLength(6.0);
cout << "Length of line : " << line.getLength() <<endl;
return 0;
}
```



The Class Destructor

A **destructor** is a special member function of a class that is executed whenever an object of its class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class. A destructor will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters. Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc. Following example explains the concept of destructor:

```
#include <iostream>
using namespace std;
int a=1;
class Line
{
public:

Line(){cout<<endl<<"No. of object to be created"<<a;a=a+1;} // This is the
constructor declaration
~Line(){a=a-1;cout<<endl<<"No of object to be deleted: "<<a;} // This is the
destructor: declaration
};
int main( )
{
Line line1;Line line2; Line line3;
Line line4;Line line5;
return 0;
}
```

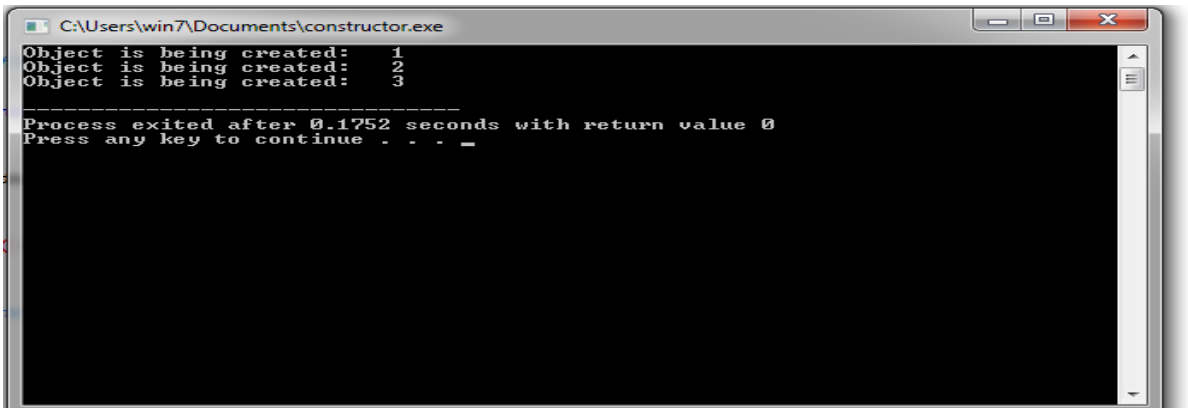


Constructor and Destructor

A **constructor** is a special member function of a class that is executed whenever we create new objects of that class.

A constructor will have exact same name as the class and it does not have any return type at all, not even void. Constructors can be very useful for setting initial values for certain member variables.Following example explains the concept of constructor:

```
#include  <iostream>
using namespace std;
int a=1;
class Line
{ public:
Line(); // This is the constructor
};
// Member functions definitions including constructor
Line::Line(void)
{cout << "Object is being created:   " <<a<<endl;
 a++;
}
// Main function for the program
int main( )
{
Line line; //create object1
Line aa;   //create object 2
Line ff;    //create object3
return 0;
}
```

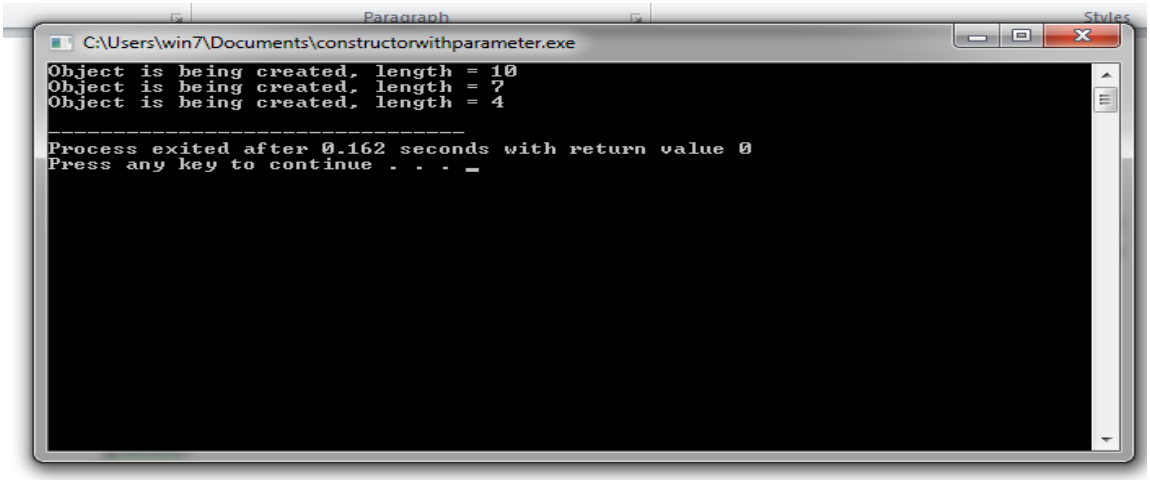


Parameterized Constructor

A default constructor does not have any parameter, but if you need, a constructor can have parameters. This helps you to assign initial value to an object at the time of its creation as shown in the following example:

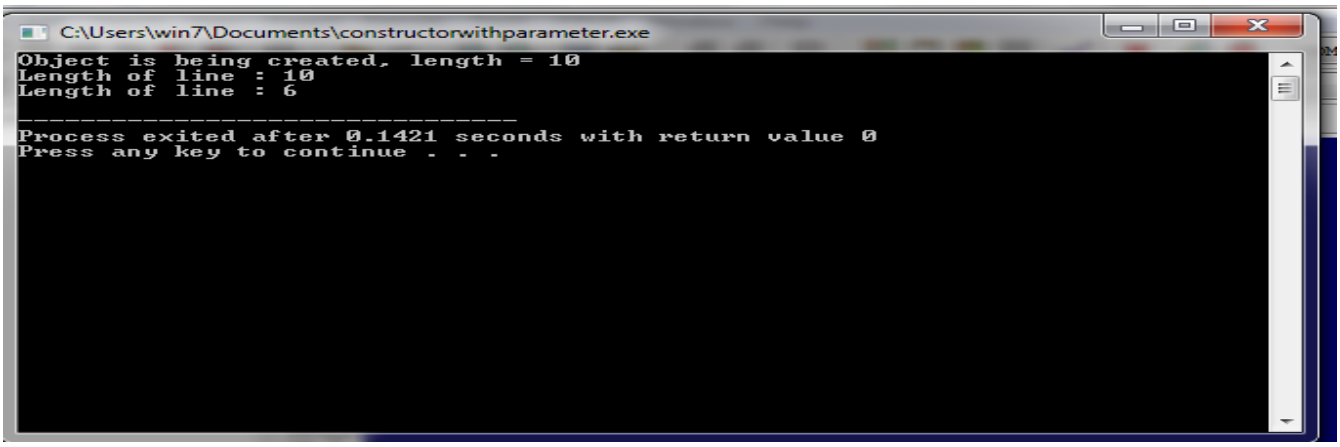
Example 1

```
#include <iostream>
using namespace std;
class Line
{public:
Line(double len); // This is the constructor
double length;
};
// Member functions definitions including constructor
Line::Line( double len)
{
cout << "Object is being created, length = " << len << endl;
length = len; }
// Main function for the program
int main( )
{
Line line1(10.0);//create object 1
Line line2(7.0);//create object 2
Line line3(4);//create object 3
return 0;
}
```



Example 2

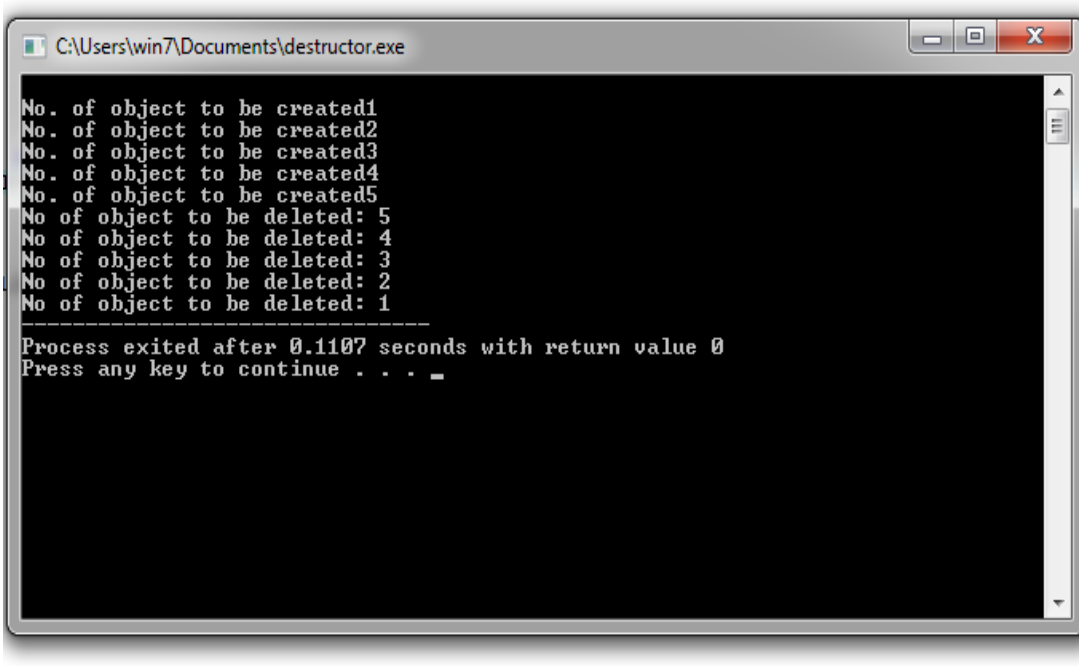
```
#include <iostream>
using namespace std;
class Line
{
public:
void setLength( double len );
double getLength( void );
Line(double len); // This is the constructor
private:
double length;
};
// Member functions definitions including constructor
Line::Line( double len)
{
cout << "Object is being created, length = " << len << endl;
length = len;
}
void Line::setLength( double len )
{
length = len;
}
double Line::getLength( void )
{
return length;
}
// Main function for the program
int main( )
{

Line line(10.0);
// get initially set length.
cout << "Length of line : " << line.getLength() <<endl;
// set line length again
line.setLength(6.0);
cout << "Length of line : " << line.getLength() <<endl;
return 0;
}
```



The Class Destructor

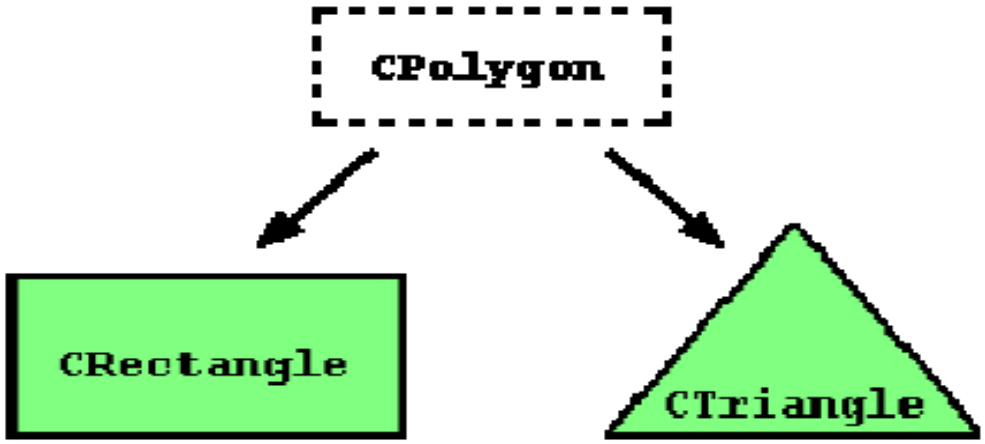
A **destructor** is a special member function of a class that is executed whenever an object of its class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class. A destructor will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters. Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc. Following example explains the concept of destructor:

```
#include <iostream>
using namespace std;
int a=1;
class Line
{
public:

Line(){cout<<endl<<"No. of object to be created"<<a;a=a+1;} // This is the
constructor declaration
~Line(){a=a-1;cout<<endl<<"No of object to be deleted: "<<a;} // This is the
destructor: declaration
};
int main( )
{
Line line1;Line line2; Line line3;
Line line4;Line line5;
return 0;
}
```



Inheritance between classes

A key feature of C++ classes is inheritance. Inheritance allows to create classes which are derived from other classes so that they automatically include some of its "parent's" members, plus it is own. For example, we are going to suppose that we want to declare a series of classes that describe polygons like our CRectangle, or like CTriangle. They have certain common properties, such as both can be described by means of only two sides: height and base.
This could be represented in the world of classes with a class CPolygon from which we would derive the two other ones: CRectangle and CTriangle.



The class CPolygon would contain members that are common for both types of polygon. In our case: width and height. And CRectangle and CTriangle would be its derived classes, with specific features that are different from one type of polygon to the other. Classes that are derived from others inherit all the accessible members of the base class. That means that if a base class includes a member A and we derive it to another class with another member called B, the derived class will contain both members A and B.

In order to derive a class from another, we use a colon (:) in the declaration of the derived class using the following format:

class derived_class_name: public base_class_name { /*...*/ };

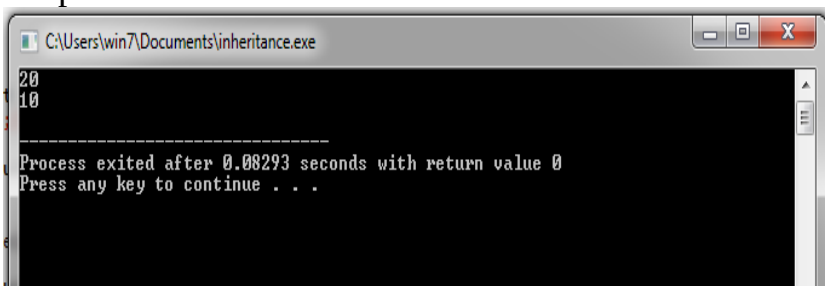
Where deriveed_class_name is the name of the derived class_and base_class_name is the name of the class on which it is based. The public access specifier may be replaced by anyone of the other access specifies protected and private. This access specifier describes the minimum access level for the members that are inherited from the base class.

EXAMPLE:

```
// derived classes
#include <iostream>
using namespace std;
class CPolygon {
protected:
int width, height;
public:
void set_values (int a, int b)
{ width=a; height=b;}
};
class CRectangle: public CPolygon {
public:
int area ()
{ return (width * height); }
};
class CTriangle: public CPolygon {
public:
int area ()
{ return (width * height / 2); }
};
int main () {
CRectangle rect;
```

```
CTriangle trgl;
rect.set_values (4,5);
trgl.set_values (4,5);
cout << rect.area() << endl;
cout << trgl.area() << endl;
return 0;
}
```

Output:



Private Inheritance

Consider the following classes:

```
class A { /*.....*/);
class C: private A
{ /*
 .
 .
.
.
 */
}
```

All the public parts of class A and all the protected parts of class A, become private members/parts of the derived class C in class C. No private member of class A can be accessed by class C. To do so, you need to write public or private functions in the Base class.

A public function can be accessed by any object, however, private function can be used only within the class hierarchy that is class A and class C and friends of these classes in the above cases.

Public Inheritance

Consider the following classes:
Consider the following classes:
class A{/*.....*/};
class E: public A
{ /*
:
:
:
};
Now, all the public parts of class A become public in class E and protected part of A become protected in E.

Example:
Consider the following classes:
class E: protected A
{ /*
.
.
.
*/
};

Now, all the public and protected parts of class A become protected in class E.
No private member of class A can be accessed by class E.

We can summarize the different access types according to who can access them in the following way:

Access	public	protected	private
members of the same class	yes	yes	yes
members of derived classes	yes	yes	no
not members	yes	no	no

This public keyword after the colon (:) denotes the maximum access level for all the members inherited from the class that follows it (in this case CPolygon). Since public is the most accessible level, by specifying this keyword the derived class will inherit all the members with the same levels they had in the base class.
If we specify a more restrictive access level like protected, all public members of the base class are inherited as protected in the derived class. Whereas if we specify the most restricting of all access levels: private, all the base class members are inherited as private. For example, if daughter was a class derived from mother that we defined as:

class daughter: protected mother;

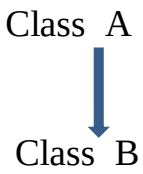
This would set protected as the maximum access level for the members of daughter that it inherited from mother. That is, all members that were public in mother would become protected in daughter. Of course, this would not restrict daughter to declare its own public members. That maximum access level is only set for the members inherited from mother.
If we do not explicitly specify any access level for the inheritance, the compiler assumes private for class**.**

What is inherited from the base class?

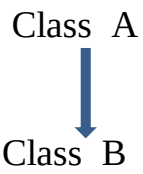
In principle, a derived class inherits every member of a base class **except:**

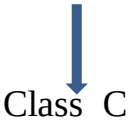
- Its constructor and its destructor.
- Its friends**.**

Although the constructors and destructors of the base class are not inherited themselves, its default constructor (i.e., its constructor with no parameters) and its destructor are always called when a new object of a derived class is created or destroyed.
Types of inheritance:
1-Single inheritance

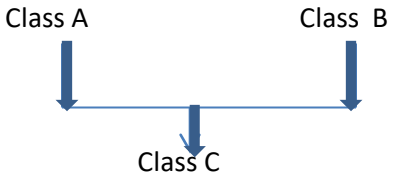


2- Multilevel inheritance

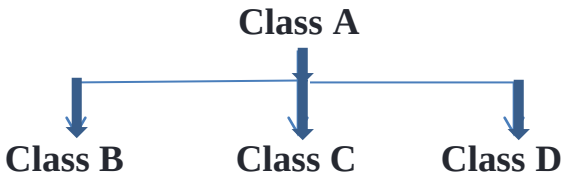




3- Multiple inheritances



4. Hierarchical inheritance



Example of Multilevel Inheritance

```
#include <iostream>
using namespace std;
class A
{    public: void display(){  cout<<"Base class content."; }
};
class B : public A{};
class C : public B{};
int main()
{   C obj; obj.display();
```

```
    return 0;
}
```

Multiple inheritances

In C++ it is perfectly possible that a class inherits members from more than one class. This is done by simply separating the different base classes with commas in the derived class declaration. For example, if we had a specific class to print on screen (COutput) and we wanted our classes CRectangle and CTriangle to also inherit its members in addition to those of CPolygon we could write:

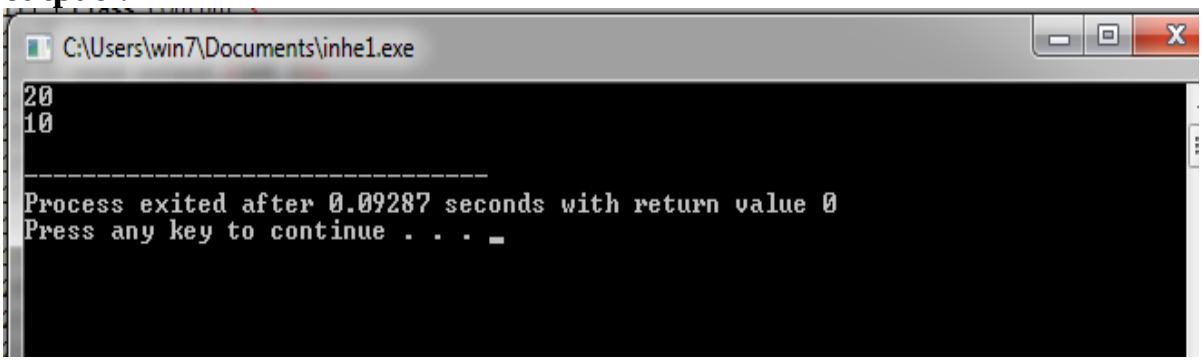
class CRectangle: public CPolygon, public COutput
class CTriangle: public CPolygon, public COutput;

Example:

```
// multiple inheritance
#include <iostream>
using namespace std;
class CPolygon {
protected:
int width, height;
public:
void set_values (int a, int b)
{ width=a; height=b;}
};
class COutput {
public:
void output (int i);
};
void COutput::output (int i) {
cout << i << endl;
}
class CRectangle: public CPolygon, public COutput {
public:
int area ()
{ return (width * height); }
};
```

```
class CTriangle: public CPolygon, public COutput {
public:
int area ()
{ return (width * height / 2); }
};
int main () {
CRectangle rect;
CTriangle trgl;
rect.set_values (4,5);
trgl.set_values (4,5);
rect.output (rect.area());
trgl.output (trgl.area());
return 0;}
```

output :



Questions:

Q1. If you have 2 inherited Classes A-> B .Class A contains (a1) string of 5 small letters , class B contains (b1) string of 5 small letters .Write an O.O.P for merge a1 with b1 into c1 ,then convert c1 to capital letters .

Q2. If you have 3 inherited classes A->B->C each class has an array of integer numbers a[3] ,b[3],c[3] .Write an O.O.P for combining all three digits into one number.

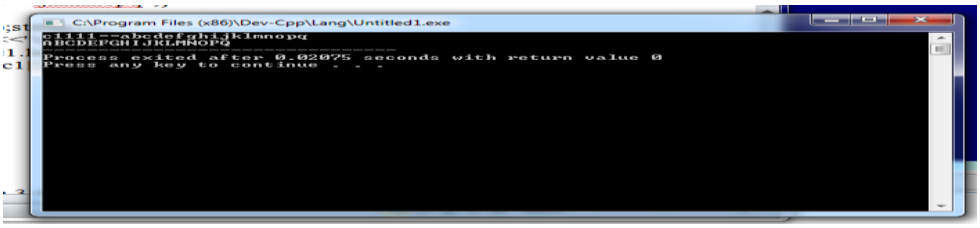
e.g. a[0]=8,b[0]=6,c[0]=2 the result=862

Q3. If you have 3-inherited classes A->B->C each one has an array a[4],b[4],c[4] respectively . Write an O.O.P for finding d [4], where each element of d[i] = the biggest no. of a[i], b[i], c[i], and so on.

Questions:

Q1. If you have 2 inherited Classes A-> B .Class A contains (a1) string of 5 small letters , class B contains (b1) string of 5 small letters .Write an O.O.P for merge a1 with b1 into c1 ,then convert c1 to capital letters .

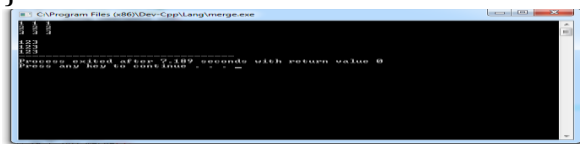
```
//Q1.  If you have 2 inherited Classes A-> B .Class A contains (a1)  string of 5 small
//      letters , class B contains (b1) string of 5 small letters .Write an O.O.P for
//      merge a1 with b1 into c1 ,then convert c1 to capital letters .

#include <iostream>
#include <cstring>
#include <cctype>
using namespace std;
class A
{ public : string a1;
void get_a(){ a1="abcdefgh";};
};
class B :public A
{ public : string b1;
void get_b(){ b1="ijklmnopq";}
void merge()
{get_a();get_b();string c1;
c1=a1+b1;cout<<"c1111=="<<c1<<"\n";
for (int i=0;i<=c1.length();i++)
{c1[i]=toupper(c1[i]); cout<<c1[i];}
}
};
int main()
{B c;
c.merge();
}
```



Q2. If you have 3 inherited classes A->B->C each class has an array of integer numbers a[3] ,b[3],c[3] .Write an O.O.P for combining all three digits into one number.
e.g. a[0]=8,b[0]=6,c[0]=2 the result=862

sol:

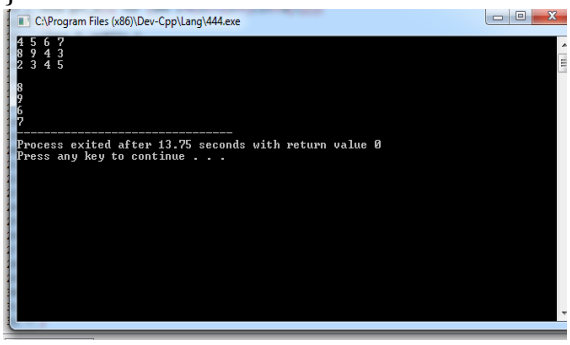
```
//Q2. If you have 3 inherited classes A->B->C each class has an array of integer numbers a[3] ,b[3],c[3] .
//Write an O.O.P for combining all three digits into one number. e.g. a[0]=8,b[0]=6,c[0]=2 the result=862
#include <iostream>
#include <cstring>
#include <cctype>
using namespace std;
class A
{ public : int a[3];
void get_a(){ for (int i=0;i<=2;i++)cin>>a[i];};
};
class B :public A
{ public : int b[3];
void get_b(){for (int i=0;i<=2;i++)cin>>b[i];}};
class C :public B
{public: int c[3];
void get_c(){ for (int i=0;i<=2;i++)cin>>c[i];}

void merge()
{get_a();get_b();get_c();
for (int i=0;i<=2;i++)
{int res=a[i]*100+b[i]*10+c[i];cout<<"\n"<<res;
}}
};
int main()
{C obj;
obj.merge();
}
```



Q3. If you have 3-inherited classes A->B->C each one has an array a[4],b[4],c[4] respectively . Write an O.O.P for finding d [4], where each element of d[i] = the biggest no. of a[i], b[i], c[i], and so on.

Sol:

```
//Q3. If you have 3-inherited classes A->B->C each one has an array a[4],b[4],c[4] respectively .
// Write an O.O.P for finding d [4], where each element of d[i] = the biggest no. of a[i], b[i], c[i], and so on.
#include <iostream>
using namespace std;
class A
{ public : int a[4];
void get_a(){ for (int i=0;i<=3;i++)cin>>a[i];};
};
class B :public A
{ public : int b[4];
void get_b(){for (int i=0;i<=3;i++)cin>>b[i];}};
class C :public B
{public: int c[4];
void get_c(){ for (int i=0;i<=3;i++)cin>>c[i];}
void biggest()
{int d[4];get_a();get_b();get_c();
for (int i=0;i<=3;i++)
{if (a[i]>b[i] && a[i]>c[i]) d[i]=a[i];
if (b[i]>a[i] && b[i]>c[i]) d[i]=b[i];
if (c[i]>a[i] && c[i]>b[i]) d[i]=c[i];
}
for (int i=0;i<=3;i++)cout <<"\n"<<d[i];
}
};
int main()
{C obj;
obj.biggest();
}
```



More Examples of Inheritance:

Q. What is the result of following program and mention the type of inheritance and draw it?

```
#include <iostream>
        using namespace std;
        class student
        {
           public:
           int rno , m1 , m2 ;
           void get()
           {
              rno = 15, m1 = 10, m2 = 10;
           }
        };
        class sports
        {
           public:
           int sm;
           void getsm()
           {
              sm = 10;
           }
        };
        class statement:public student,public sports
        {
           int tot,avg;
           public:
           void display()
           {
              tot = (m1 + m2 + sm);
              avg = tot / 3;
              cout << tot <<"   ";
              cout << avg <<"    ";
           }
        };
        int main()
        {
```

```
        statement obj;
        obj.get();
        obj.getsm();
        obj.display();
}
```

Output: 30 10 type? draw ?

Q. If you have four classes A-> B->C->D . Each class has one words , write an O.O.P using inheritance to combine sentence. E.g Class A has the word "these", class B has the word "strings" ,class C has "are" and class D has "concatenated.

Q1. If you have 3 classes A, B, C .Class A contains a number in Decimal system, Class B contains a number in Binary system, class C contains a number in Octal system, Write an O.O.P program to convert the three values in different system to decimal system.

Q2. If you have two inherited classes A->B. Class A contains an array a [10] of type characters. Class B contains array b [10] of type character. Write an O.O.P program for printing the odd letter several times until print one letter of b[10] and print the even letter several times until print one letter of a[10] in between.

e.g a[10]=[A B C D E F G H I K],B[10]= [L M N O P Q R S T V]

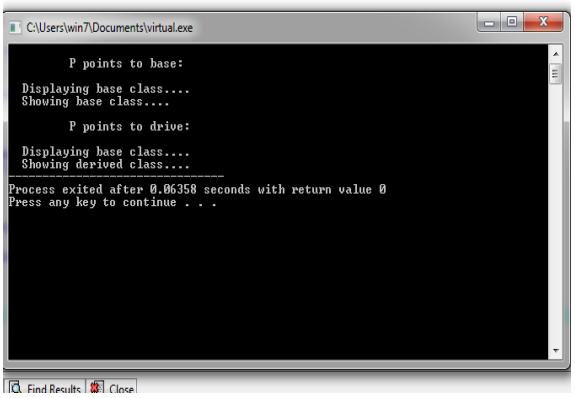
Result= B L D N F O H Q S K V

Virtual Function

A virtual function is a function that is declared as virtual in a base class and redefined in one or more derived classes. A class that contains one or more virtual functions is called a polymorphic class. A virtual function is declared as virtual inside the base class by preceding its declaration with the keyword virtual. However, when a virtual function is redefined by a derived class, the keyword virtual need not be repeated.

A virtual function is a special form of member function that is declared within a base class and redefined by a derived class. The keyword virtual is used to create a virtual function, precede the function's declaration in the base class. If a class includes a virtual function and if it gets inherited, the virtual class redefines a virtual function to go with its own need. Example of virtual function:

```
#include <iostream>
using namespace std;
class b { public:
virtual void show()
{cout<<"\n  Showing base class....";}
void display(){cout<<"\n  Displaying base class...." ;}};
class d:public b { public:
void display(){ cout<<"\n  Displaying derived class....";}
void show(){cout<<"\n  Showing derived class....";}
};
int main()
{b B; b *ptr;
cout<<"\n\t P points to base:\n" ; ptr=&B; ptr->display();
ptr->show();
cout<<"\n\n\t P points to drive:\n"; d D; ptr=&D; ptr->display();
ptr->show();
}
```

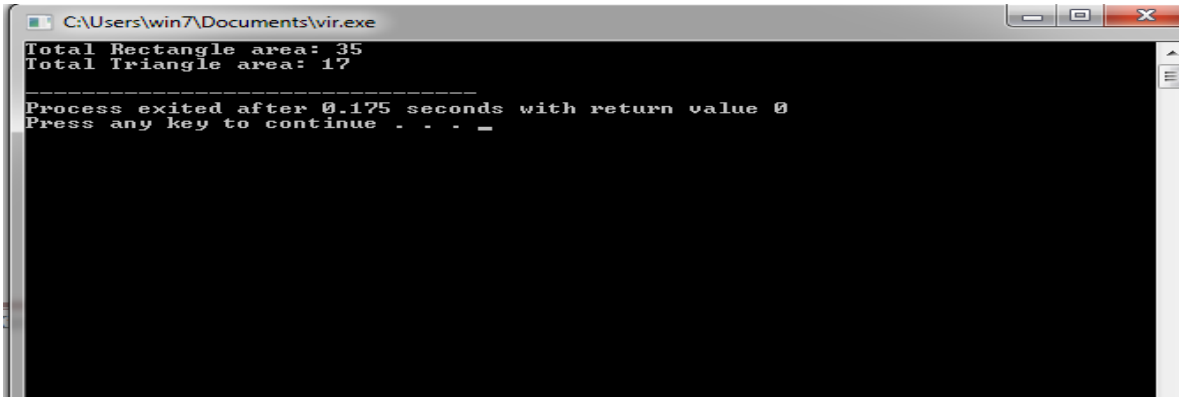


Example:

Finding the area of rectangle and triangle using virtual function

```
#include <iostream>
using namespace std;
// Base class
class Shape {   public:     // pure virtual function providing interface framework.
     virtual int getArea() = 0;
     void setWidth(int w) {  width = w;  }
       void setHeight(int h) { height = h;}
  protected:  int width;    int height; };
// Derived classes
class Rectangle: public Shape {
  public:   int getArea() {  return (width * height);  }};
class Triangle: public Shape {
  public:
    int getArea() { return (width * height)/2;  }};
int main(void) {   Rectangle Rect; Triangle  Tri;
  Rect.setWidth(5);
  Rect.setHeight(7);
  // Print the area of the object.
  cout << "Total Rectangle area: " << Rect.getArea() << endl;
  Tri.setWidth(5);  Tri.setHeight(7);
    // Print the area of the object.
  cout << "Total Triangle area: " << Tri.getArea() << endl;
  return 0;
}
```

Output:



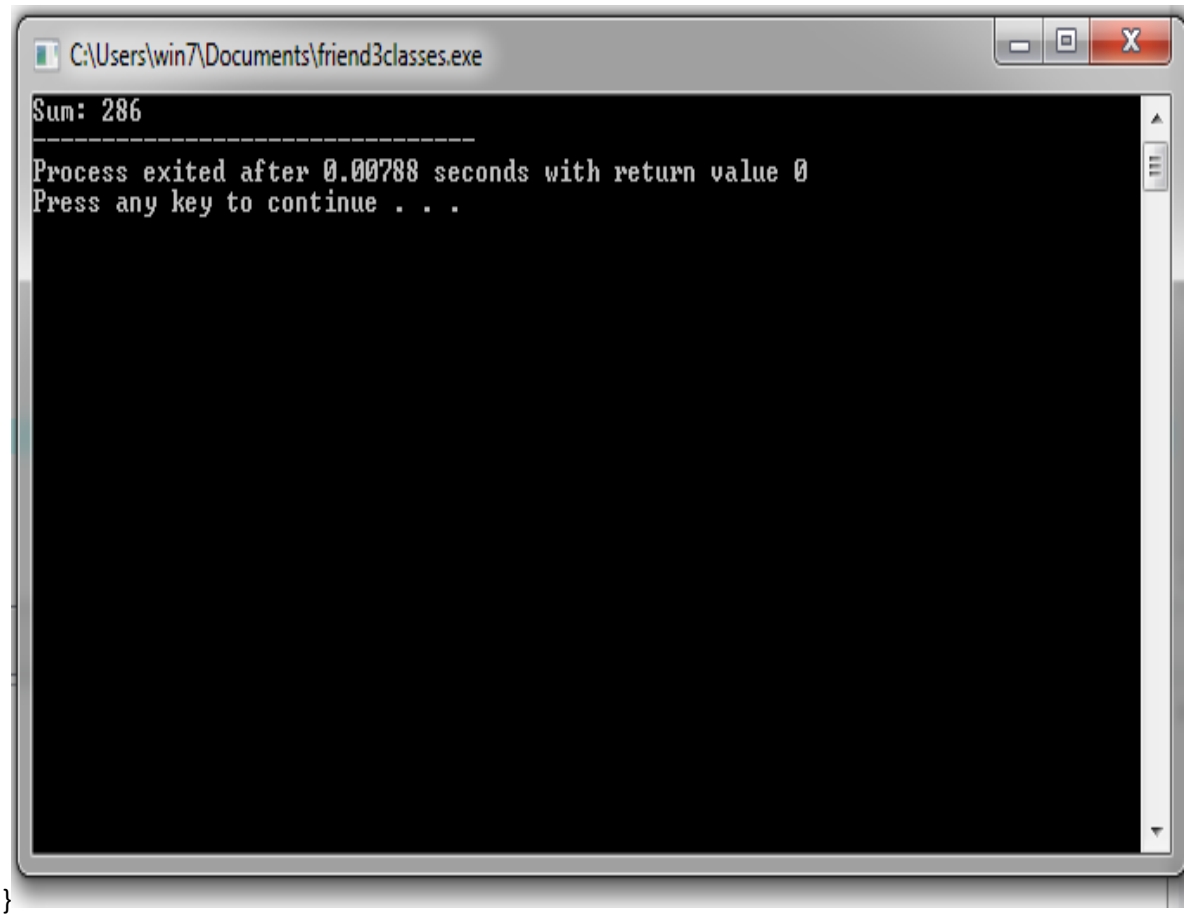
Q1. Write an O.O.P for find the area of circle ,square ,and triangle using Virtual function.

Q2. Write an O.O.P for finding 2^3, 3^4, and 5^5 using virtual function.

```
// If you have 3 separated (A,B,C) classes ,each class has one value
//(a,b,c)

// Write an O.O.P for combining this 3 digit to produce number consist
//of 3 digits

// e.g. a=3,b=4,c=9 number = 943

#include <iostream>

using namespace std;

// forward declaration

class B;

class C;

class A {

private:

int a;

public:

void get_numA(int b) {a=b; }

// friend function declaration

friend int add(A, B,C);

};
```

```
class B {
private:
int b;
public:
void get_numB(int a) {b=a; }
// friend function declaration
friend int add(A , B, C);
};
class C {
private:
int c;
public:
void get_numC(int b) {c=b; }
// friend function declaration
friend int add(A, B,C);
};
// Function add() is the friend function of classes A and B
// that accesses the member variables numA and numB
int add(A objectA, B objectB, C objectC)
{
```

```
return (objectA.a + objectB.b*10+objectC.c*100);
}
int main()
{
A objectA; objectA.get_numA(6);
B objectB; objectB.get_numB(8);
C objectC; objectC.get_numC(2);
cout<<"Sum: "<< add(objectA, objectB,objectC);
return 0;
```




```
}
```