Unit 3 Classes and Objects

Introduction

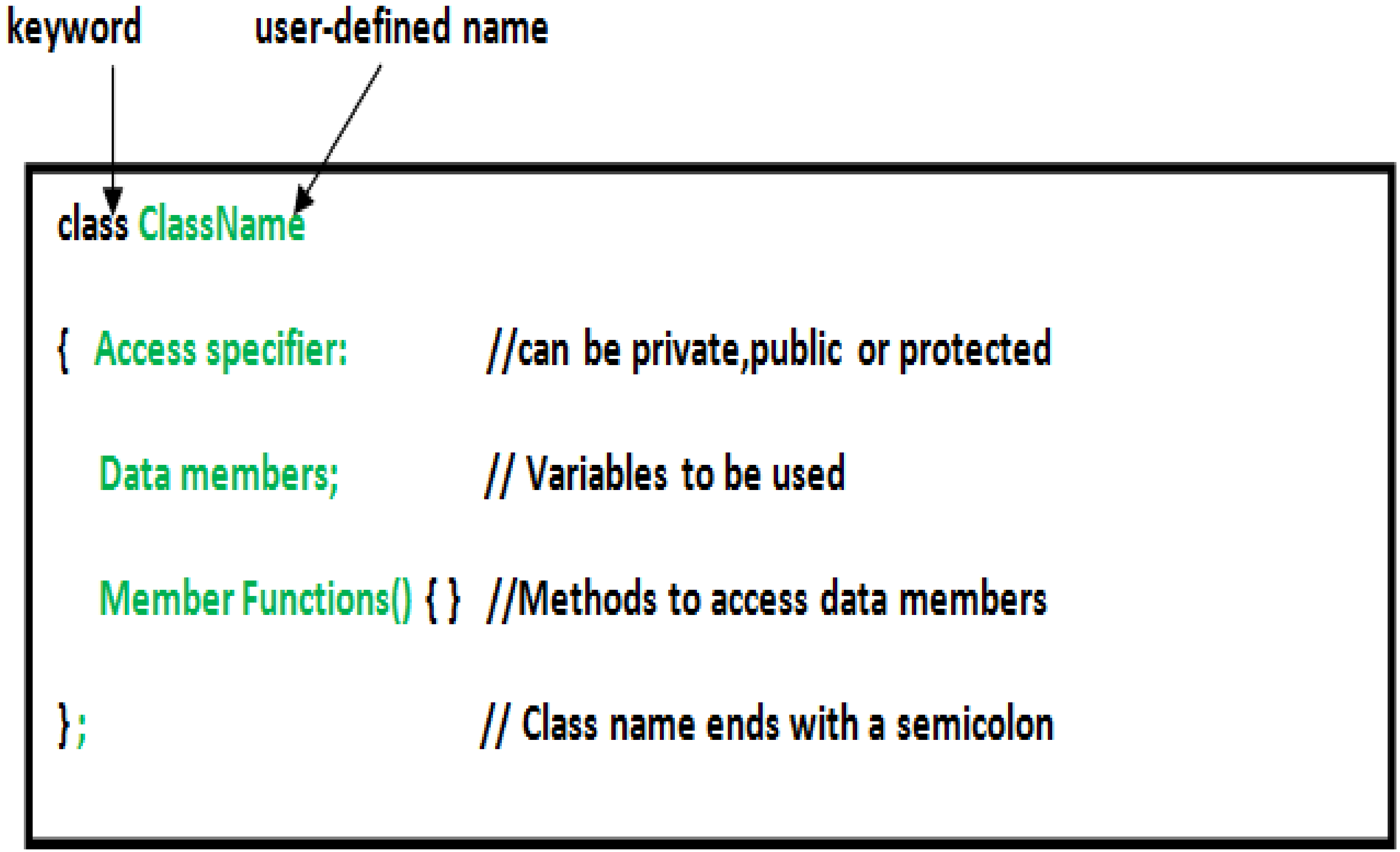
- A class is a blueprint(model) for the object. Object is an instance (example/variable)of a class
  - i.e., It is a template from which objects are created.
  - E.g., Sketch of a house and house.
- Once a class has been defined , we can create any number of objects of the same class.
- So, we can say that class is a user defied data type.

Contd..

- **Defining (Specifying) a class:**
  - A class is defined in C++ using keyword class followed by the name of class(a valid identifier).
  - The body of class is defined inside the curly brackets and terminated by a semicolon at the end.

Contd..

- **General format of defining class:**



```
class ClassName

{  Access specifier:          //can be private,public or protected

   Data members;              // Variables to be used

   Member Functions() { }     //Methods to access data members

};                            // Class name ends with a semicolon
```

Contd..

- The body of a class encapsulates data and functions called the class members.
- The data components of the class are called data members and the function components are called member functions.

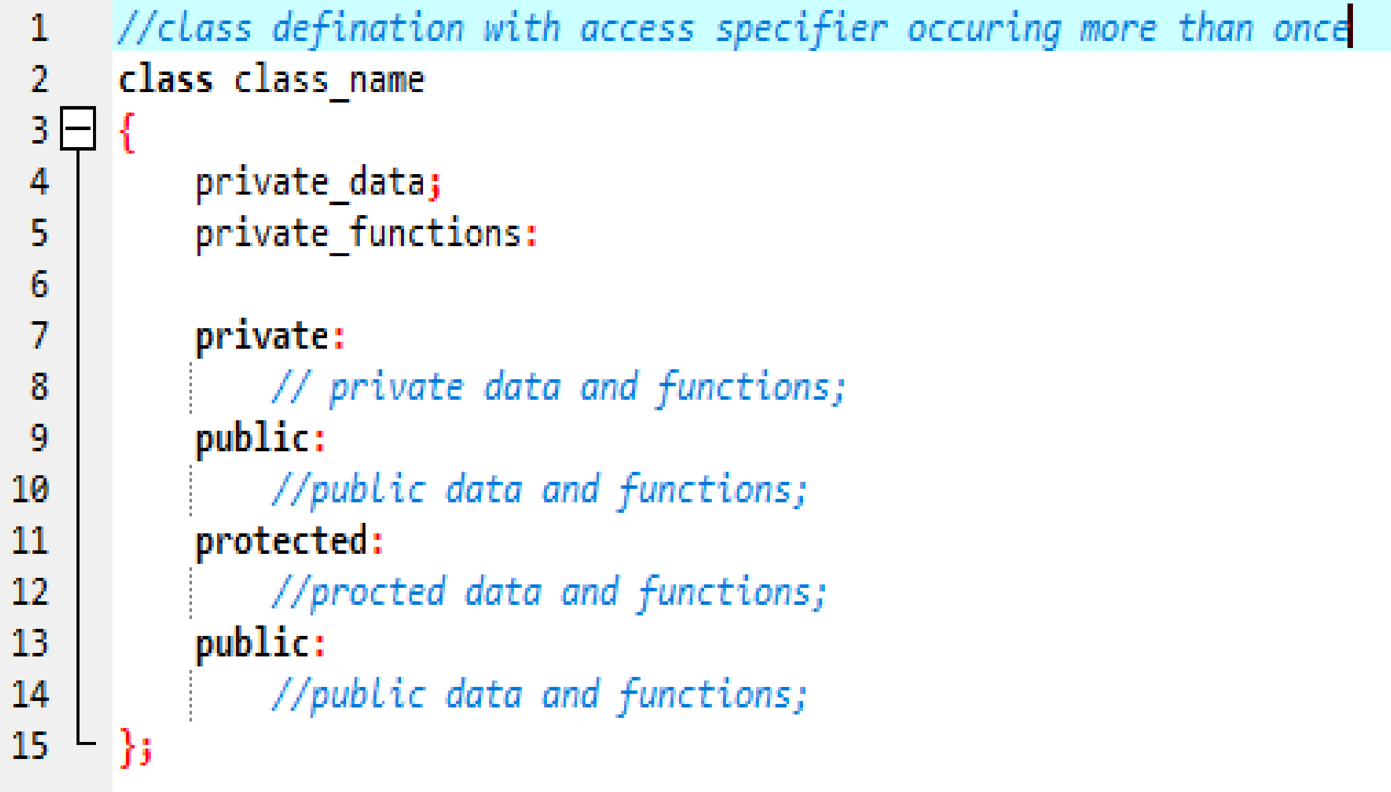
Contd..

- Private, Protected, Public is called visibility labels/access specifiers.
- In C++, data can be hidden by making it private.
- The members that are declared private can be accessed only from within the class.
- When access specifiers are not specified, members will be private by default.
- Public members can be accessed from outside the class also.
- Protected members can be accessed from the child class also.
- Usually within a class data are private and functions are public

Contd..

- The access specifiers can occur more than once in the class definition:

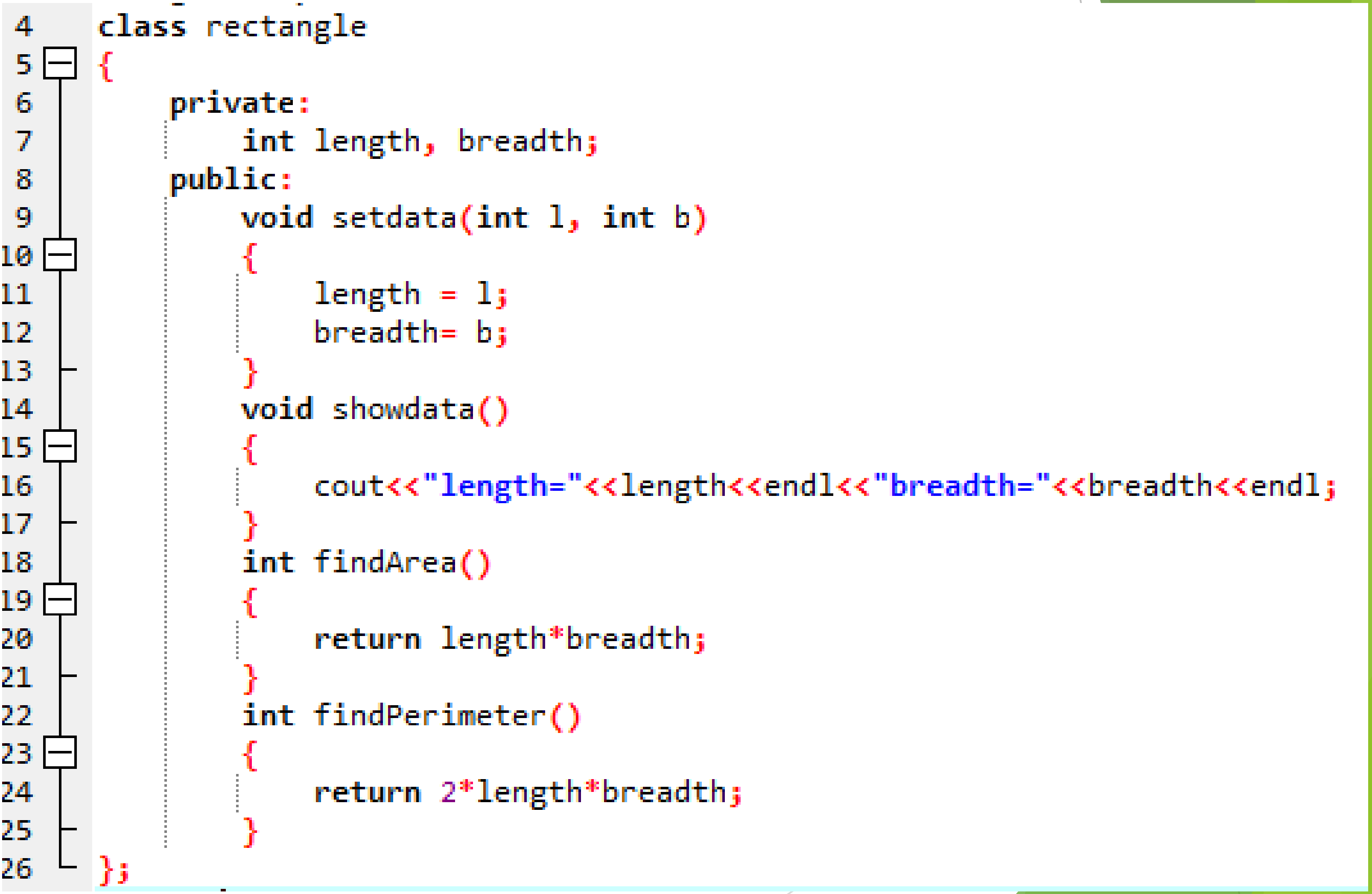
- E.g.,:



```
//class defination with access specifier occuring more than once
class class_name
{
    private_data;
    private_functions:

    private:
        // private data and functions;
    public:
        //public data and functions;
    protected:
        //procted data and functions;
    public:
        //public data and functions;
};
```

Contd..

- Example: Declaring a class rectangle



```
class rectangle
{
    private:
        int length, breadth;
    public:
        void setdata(int l, int b)
        {
            length = l;
            breadth= b;
        }
        void showdata()
        {
            cout<<"length="<<length<<endl<<"breadth="<<breadth<<endl;
        }
        int findArea()
        {
            return length*breadth;
        }
        int findPerimeter()
        {
            return 2*length*breadth;
        }
};
```

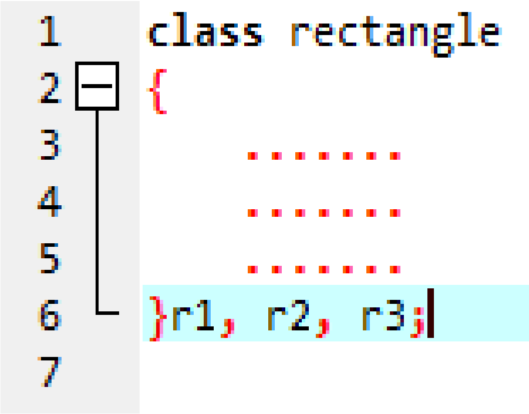
Contd..

- **Creating objects:**
  - The class declaration does not define any objects but only specifies what they will contain.
  - Once class has been declared, we can create variables(objects) of that type as
  - Syntax: class_name object_name; // instantiating object
    - E.g.,: rectangle r; // it creates a variable(object) r of type rectangle.
  - We can create any number of objects form the same class
    - E.g.,: rectangle r1, r2,….;

Contd..

Alternative way to create objects:

- Objects can also be created when a class defined by placing their names immediately after the closing brace.
- E.g..,



```
class rectangle
{
        ........
        ........
        ........
}r1, r2, r3;
```

- **Note:** size of the object is the sum of the size of the all the data members declared in the class.

Contd..

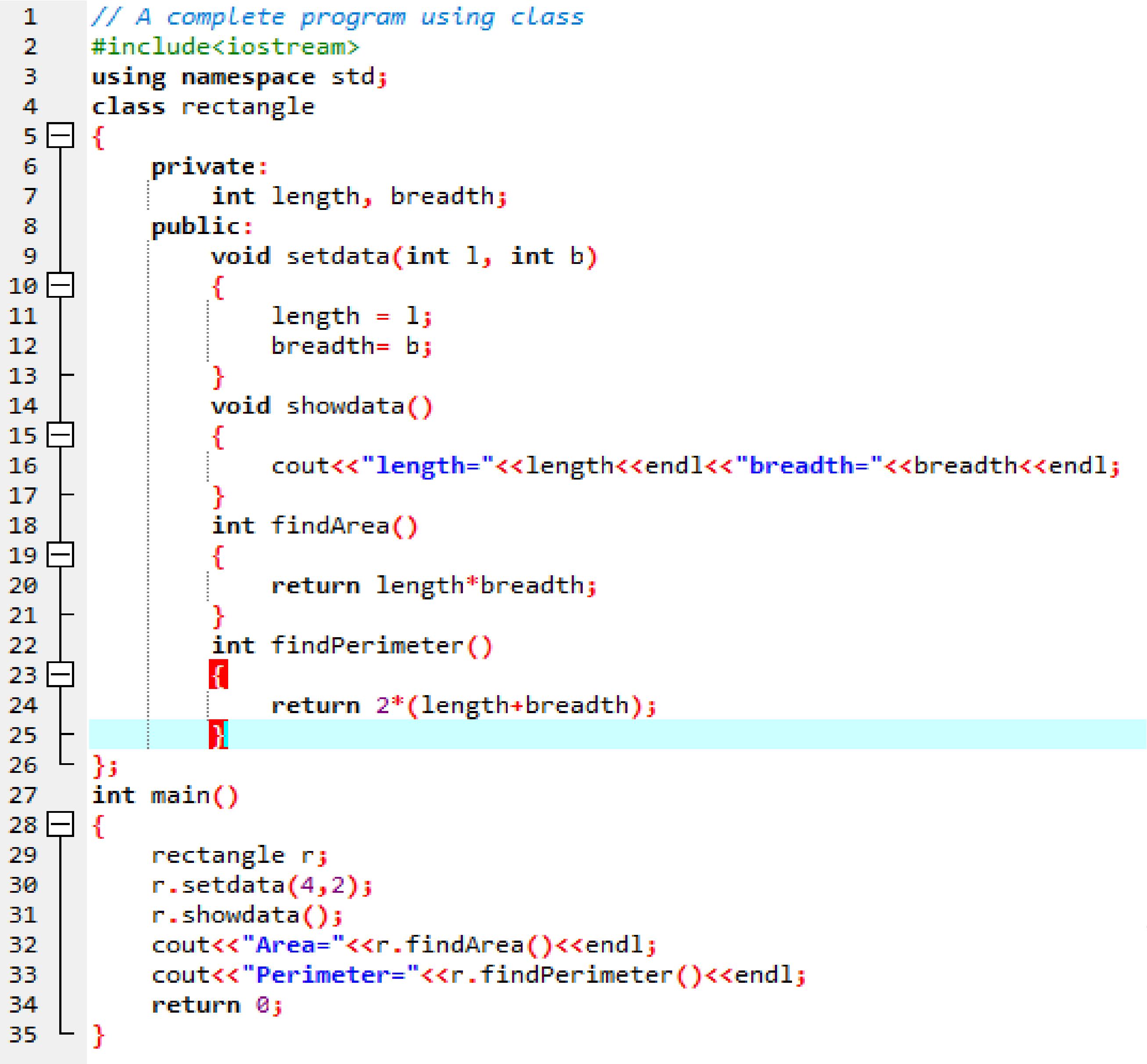
- **Accessing class members:**
  - When an objects of the class is created then the members are accessed using the '.' dot operator(also called member access operator) except the private members.

  - Syntax: object_name.data_member_name;

    object_name.function_member_name(argements….);

  - For example: r.setData(4, 2);

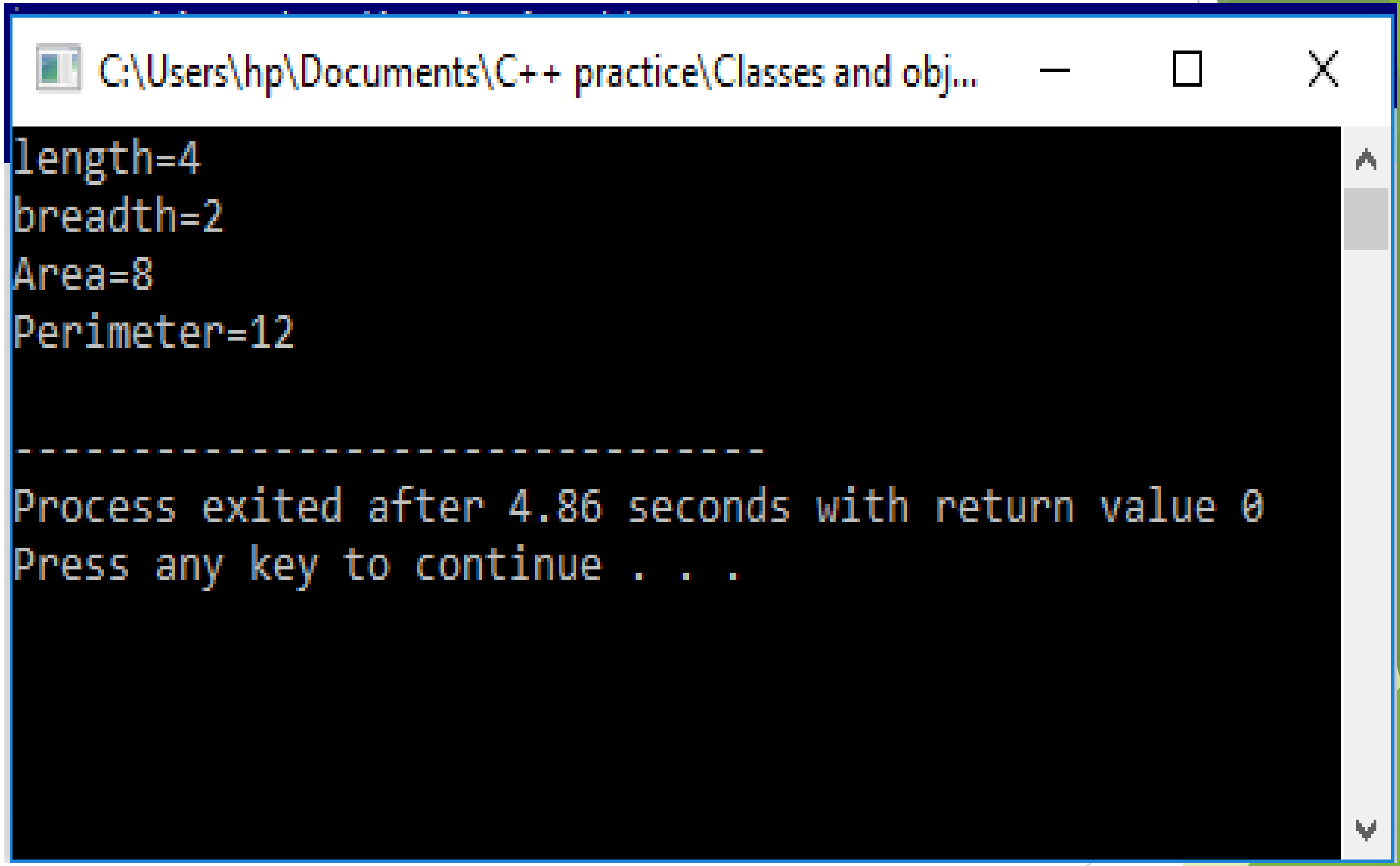
Example: Program to find the area and perimeter of rectangle



```cpp
// A complete program using class
#include<iostream>
using namespace std;
class rectangle
{
    private:
        int length, breadth;
    public:
        void setdata(int l, int b)
        {
            length = l;
            breadth= b;
        }
        void showdata()
        {
            cout<<"length="<<length<<endl<<"breadth="<<breadth<<endl;
        }
        int findArea()
        {
            return length*breadth;
        }
        int findPerimeter()
        {
            return 2*(length+breadth);
        }
};
int main()
{
    rectangle r;
    r.setdata(4,2);
    r.showdata();
    cout<<"Area="<<r.findArea()<<endl;
    cout<<"Perimeter="<<r.findPerimeter()<<endl;
    return 0;
}
```

Contd..

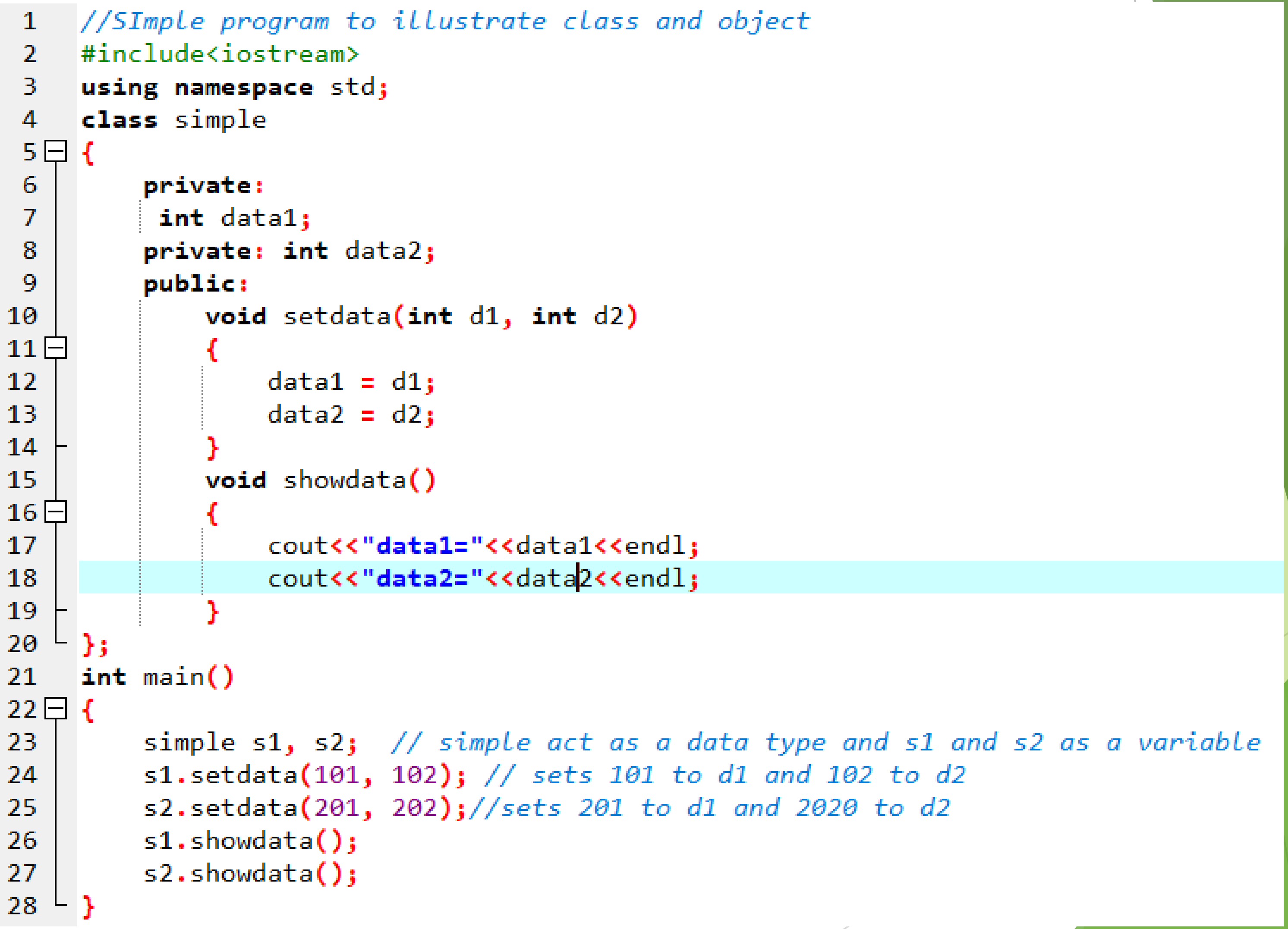
- **Output:**



```
length=4
breadth=2
Area=8
Perimeter=12


--------------------------------
Process exited after 4.86 seconds with return value 0
Press any key to continue . . .
```

Contd..

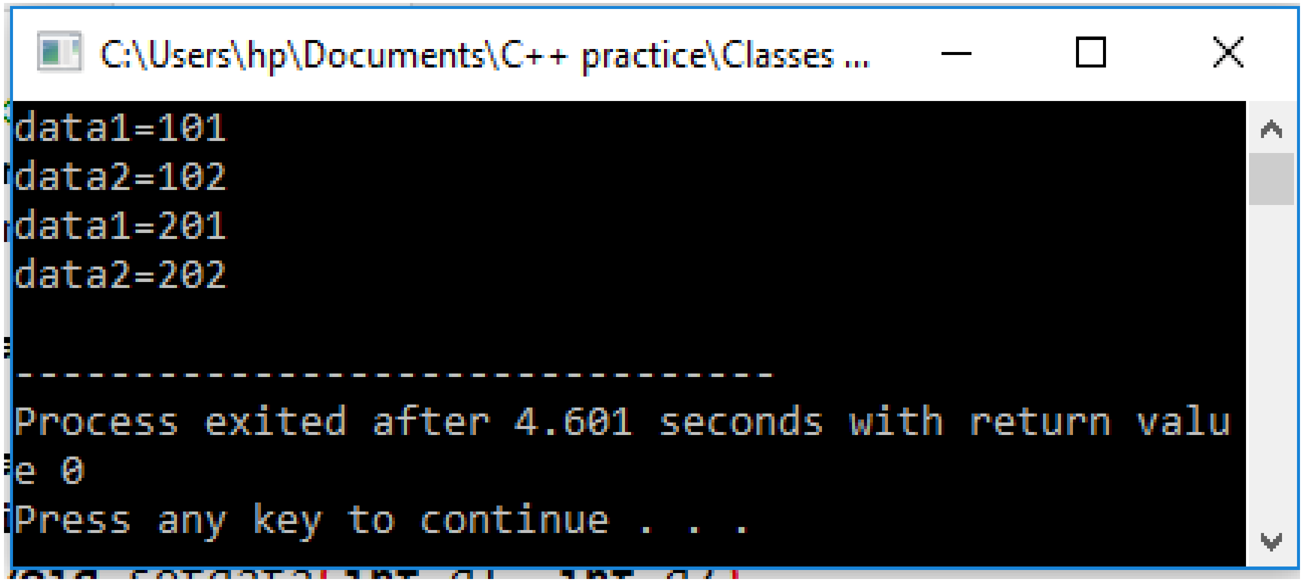
- **Example2**: simple program to set and show the data by using the concept of class



```cpp
//SImple program to illustrate class and object
#include<iostream>
using namespace std;
class simple
{
    private:
     int data1;
    private: int data2;
    public:
        void setdata(int d1, int d2)
        {
            data1 = d1;
            data2 = d2;
        }
        void showdata()
        {
            cout<<"data1="<<data1<<endl;
            cout<<"data2="<<data2<<endl;
        }
};
int main()
{
    simple s1, s2;  // simple act as a data type and s1 and s2 as a variable
    s1.setdata(101, 102); // sets 101 to d1 and 102 to d2
    s2.setdata(201, 202);//sets 201 to d1 and 2020 to d2
    s1.showdata();
    s2.showdata();
}
```

Contd..

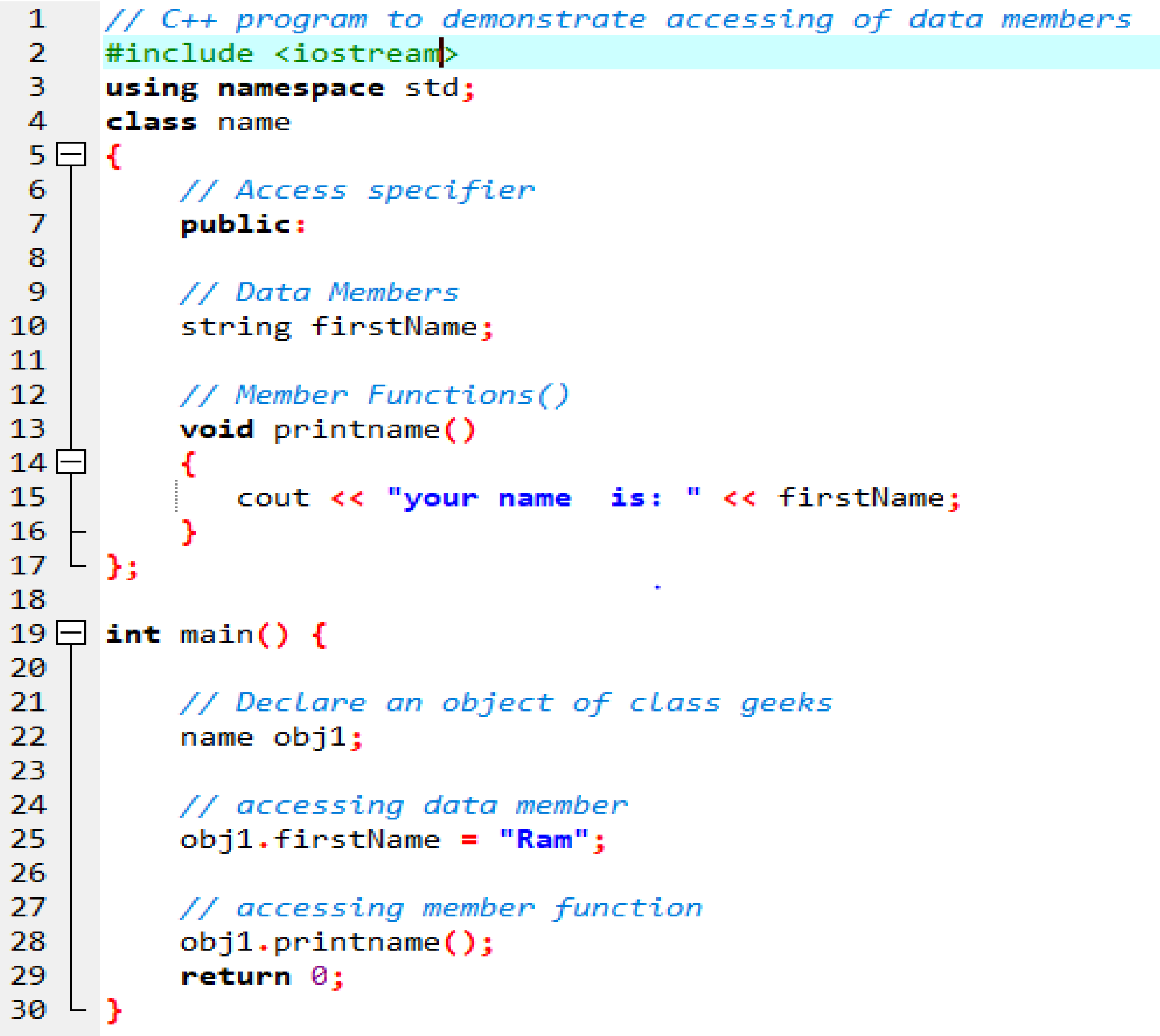
- **Output:**



```
data1=101
data2=102
data1=201
data2=202


--------------------------------
Process exited after 4.601 seconds with return valu
e 0
Press any key to continue . . .
```

Contd..

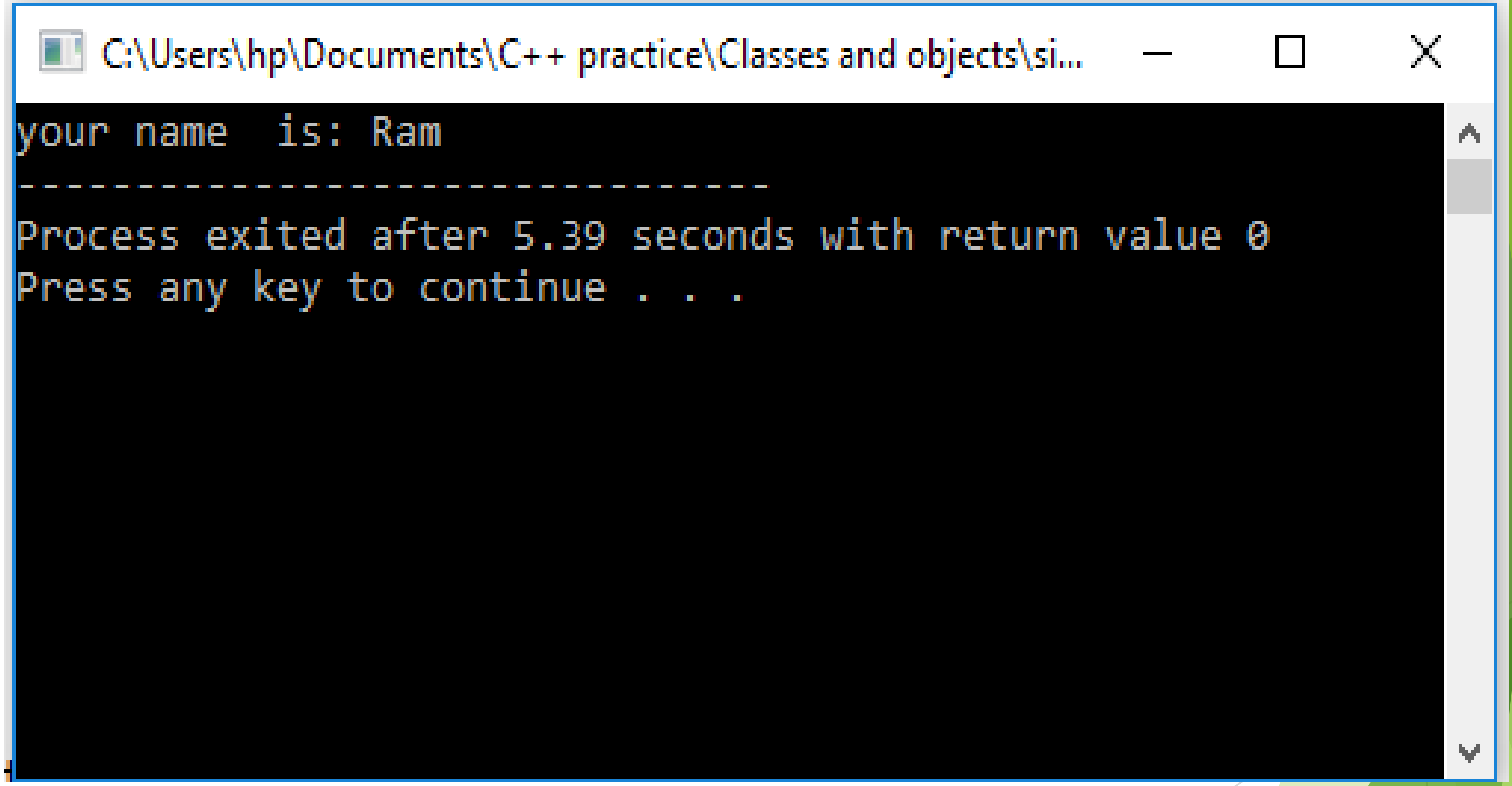
- **Example3:** Program to demonstrate the class and object



```cpp
// C++ program to demonstrate accessing of data members
#include <iostream>
using namespace std;
class name
{
    // Access specifier
    public:

    // Data Members
    string firstName;

    // Member Functions()
    void printname()
    {
       cout << "your name  is: " << firstName;
    }
};

int main() {

    // Declare an object of class geeks
    name obj1;

    // accessing data member
    obj1.firstName = "Ram";

    // accessing member function
    obj1.printname();
    return 0;
}
```

Contd..

- **Output:**

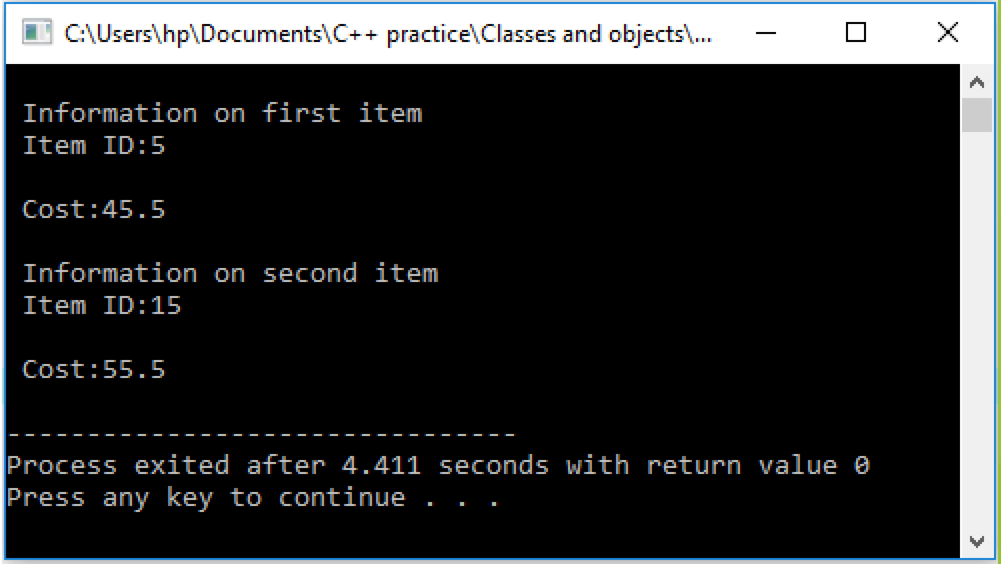


C:\Users\hp\Documents\C++ practice\Classes and objects\si...

```
your name  is: Ram
--------------------------------
Process exited after 5.39 seconds with return value 0
Press any key to continue . . .
```

Contd..

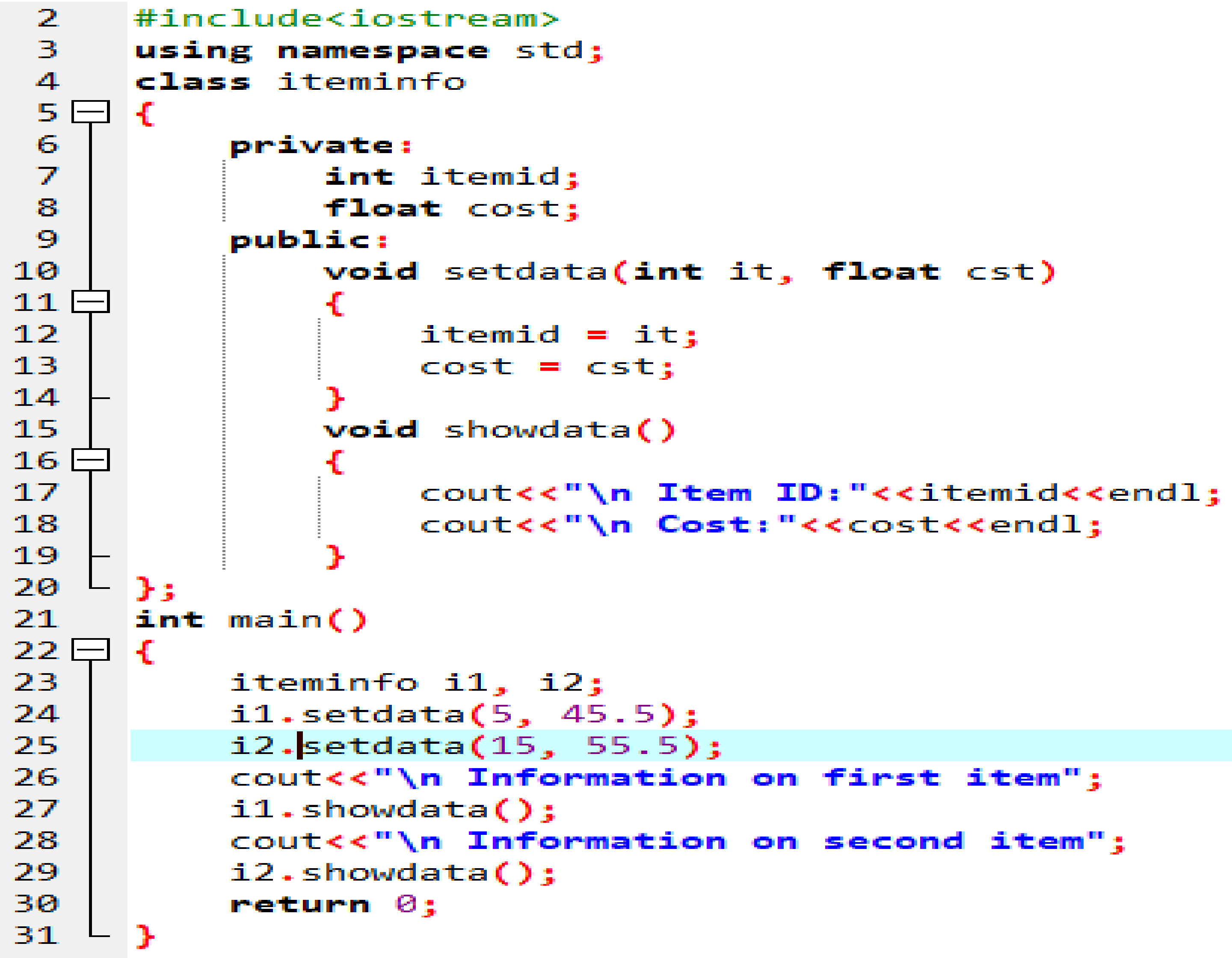
- **Class work1**: Write a program designing a class to represent Item information. Include the following members
  - Data members
    - Itemid
    - cost
  - Member functions
    - Setdata
    - showdata



```

 Information on first item
 Item ID:5

 Cost:45.5

 Information on second item
 Item ID:15

 Cost:55.5

--------------------------------
Process exited after 4.411 seconds with return value 0
Press any key to continue . . .
```

Contd..

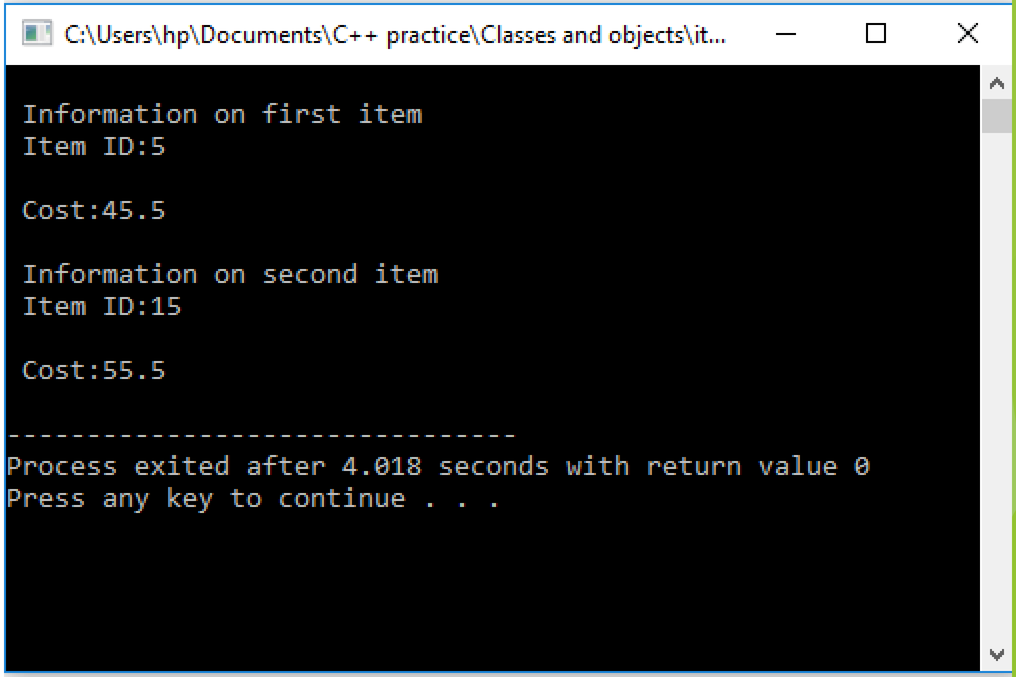
- **Program:**



```
#include<iostream>
using namespace std;
class iteminfo
{
    private:
        int itemid;
        float cost;
    public:
        void setdata(int it, float cst)
        {
            itemid = it;
            cost = cst;
        }
        void showdata()
        {
            cout<<"\n Item ID:"<<itemid<<endl;
            cout<<"\n Cost:"<<cost<<endl;
        }
};
int main()
{
    iteminfo i1, i2;
    i1.setdata(5, 45.5);
    i2.setdata(15, 55.5);
    cout<<"\n Information on first item";
    i1.showdata();
    cout<<"\n Information on second item";
    i2.showdata();
    return 0;
}
```

Contd..

- **Class work2**: Write a program designing a class to represent Item information. Include the following members
  - Data members
    - Itemid
    - cost
  - Member functions
    - showdata

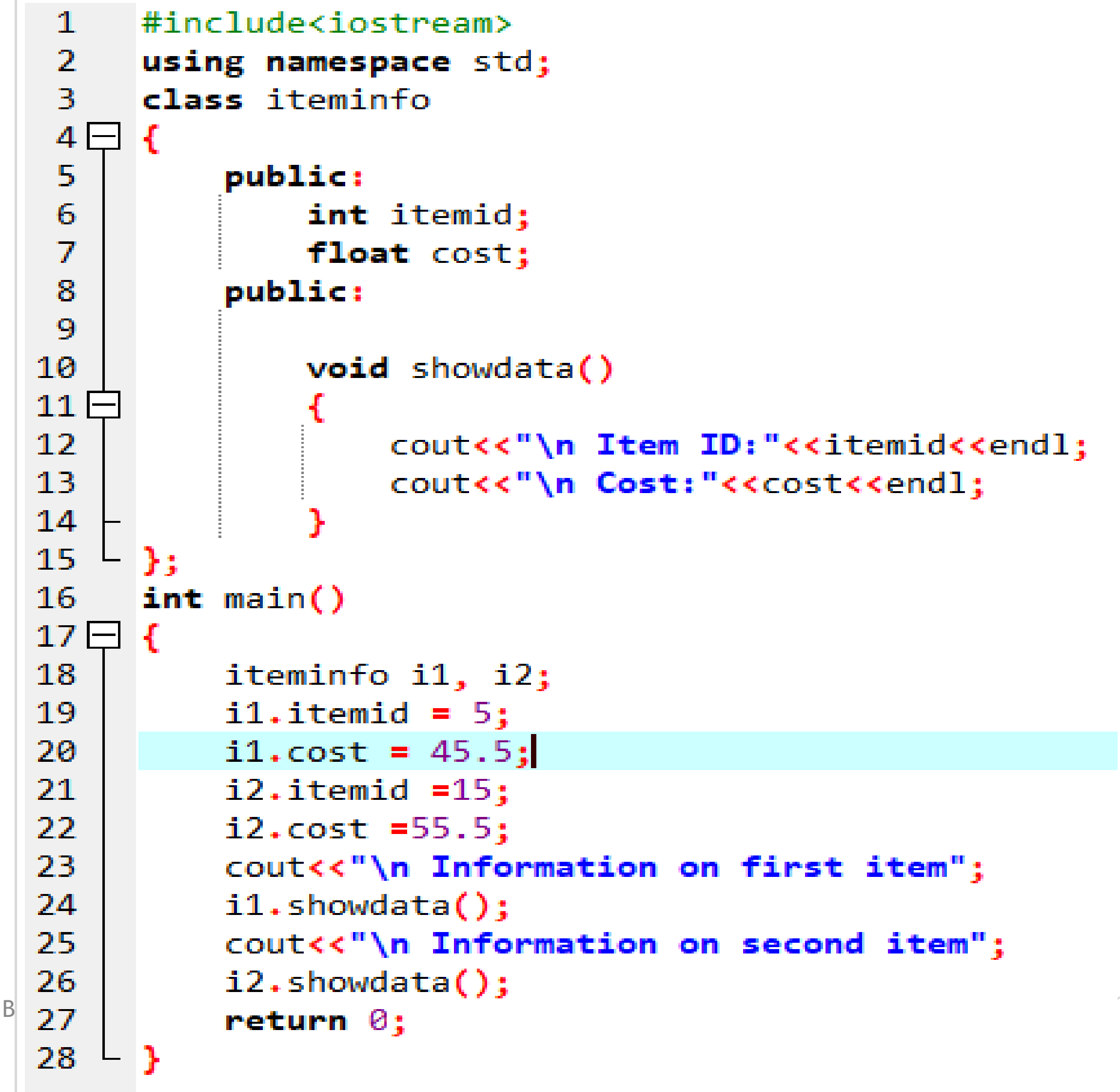


C:\Users\hp\Documents\C++ practice\Classes and objects\it...

```
 Information on first item
 Item ID:5

 Cost:45.5

 Information on second item
 Item ID:15

 Cost:55.5

--------------------------------
Process exited after 4.018 seconds with return value 0
Press any key to continue . . .
```

Contd..

- **Program:**



```
#include<iostream>
using namespace std;
class iteminfo
{
    public:
        int itemid;
        float cost;
    public:

        void showdata()
        {
            cout<<"\n Item ID:"<<itemid<<endl;
            cout<<"\n Cost:"<<cost<<endl;
        }
};
int main()
{
    iteminfo i1, i2;
    i1.itemid = 5;
    i1.cost = 45.5;
    i2.itemid =15;
    i2.cost =55.5;
    cout<<"\n Information on first item";
    i1.showdata();
    cout<<"\n Information on second item";
    i2.showdata();
    return 0;
}
```

Defining Member function

- The data member of a class is declared within the body of class. However, the member functions can be defined in one of the two places:
  - Inside the class defining: Definition and declaration is placed in the same place i.e., inside the class definition.
  - Outside the class definition:
    - Declaration is placed inside the class definition and
    - But the definition is provided outside of the class by using scope resolution operator as:

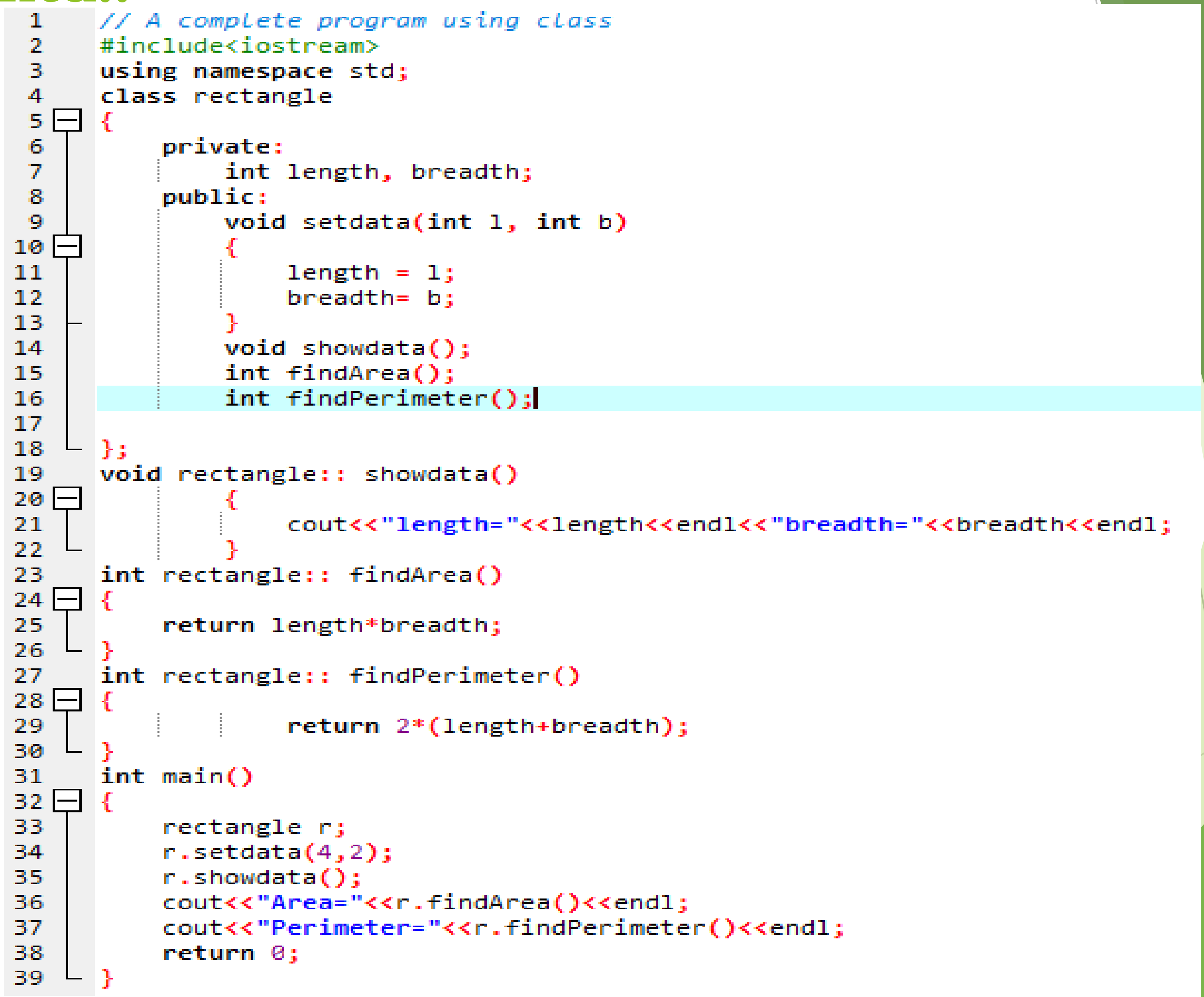
Contd..

- Defining member function Outside the class definition
  - Declaration is placed inside the class definition and
  - But the definition is provided outside of the class by using scope resolution operator as:
  - Syntax:

```
Return_type  class_name::function_name(argument declaration)
          {
               //Function body
          }
```

Contd..

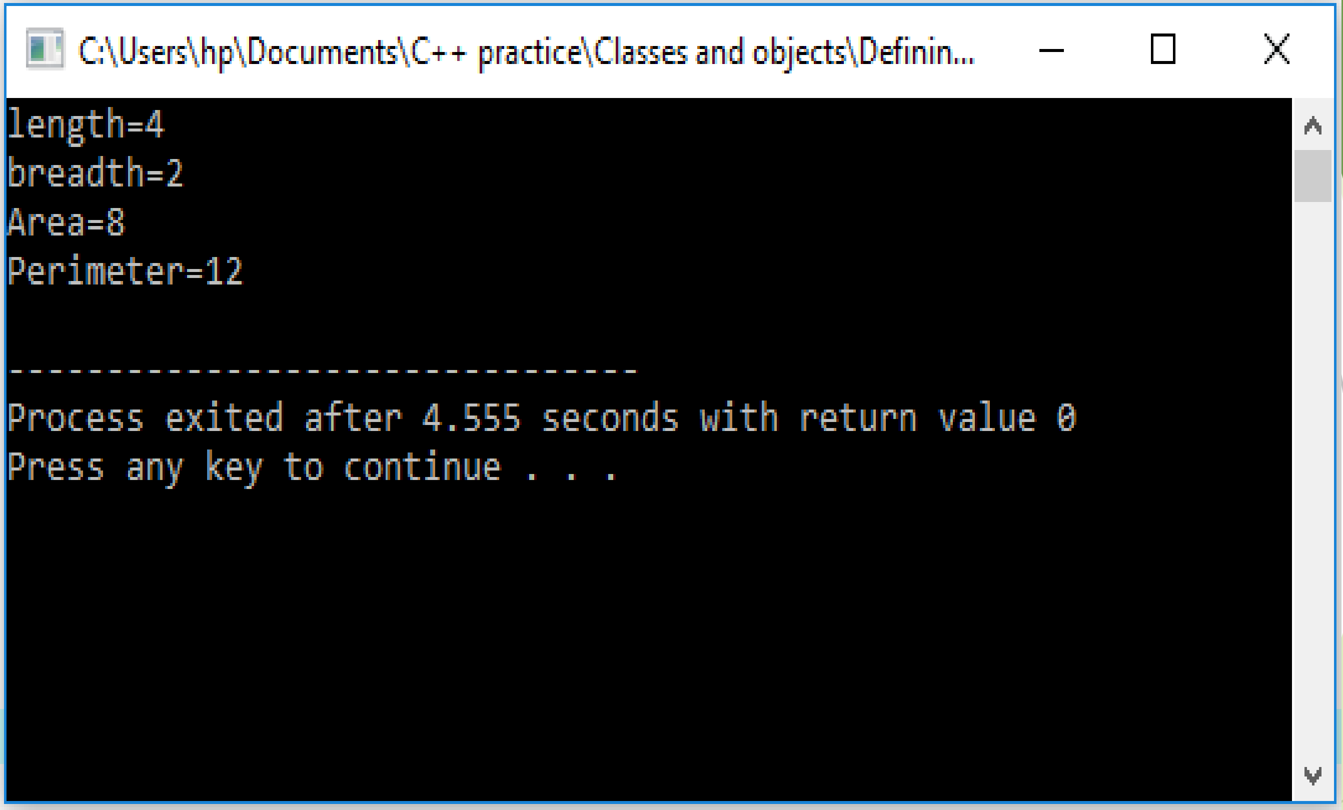
Example:



```
// A complete program using class
#include<iostream>
using namespace std;
class rectangle
{
    private:
        int length, breadth;
    public:
        void setdata(int l, int b)
        {
            length = l;
            breadth= b;
        }
        void showdata();
        int findArea();
        int findPerimeter();

};
void rectangle:: showdata()
        {
            cout<<"length="<<length<<endl<<"breadth="<<breadth<<endl;
        }
int rectangle:: findArea()
{
    return length*breadth;
}
int rectangle:: findPerimeter()
{
            return 2*(length+breadth);
}
int main()
{
    rectangle r;
    r.setdata(4,2);
    r.showdata();
    cout<<"Area="<<r.findArea()<<endl;
    cout<<"Perimeter="<<r.findPerimeter()<<endl;
    return 0;
}
```

Contd..

- **Output:**



C:\Users\hp\Documents\C++ practice\Classes and objects\Definin...

```
length=4
breadth=2
Area=8
Perimeter=12

--------------------------------
Process exited after 4.555 seconds with return value 0
Press any key to continue . . .
```

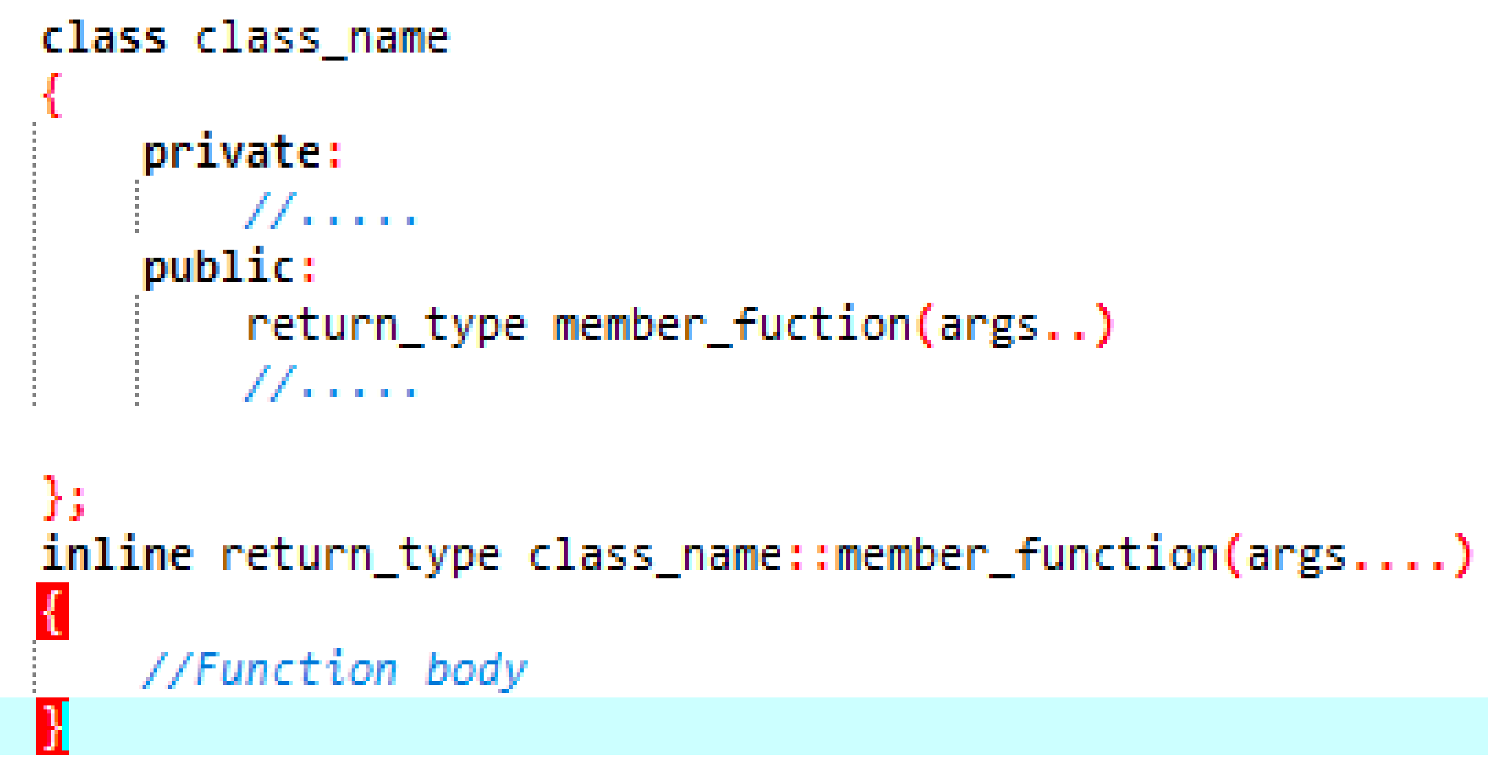
Making outer function inline

- In C++, all the member functions defined inside the class are treated as inline where as the function defined outside the class are not inline.

- The member function defined outside the class can be made inline too. It is made inline by prefixing the keyword inline to the function definition as follows:

Contd..

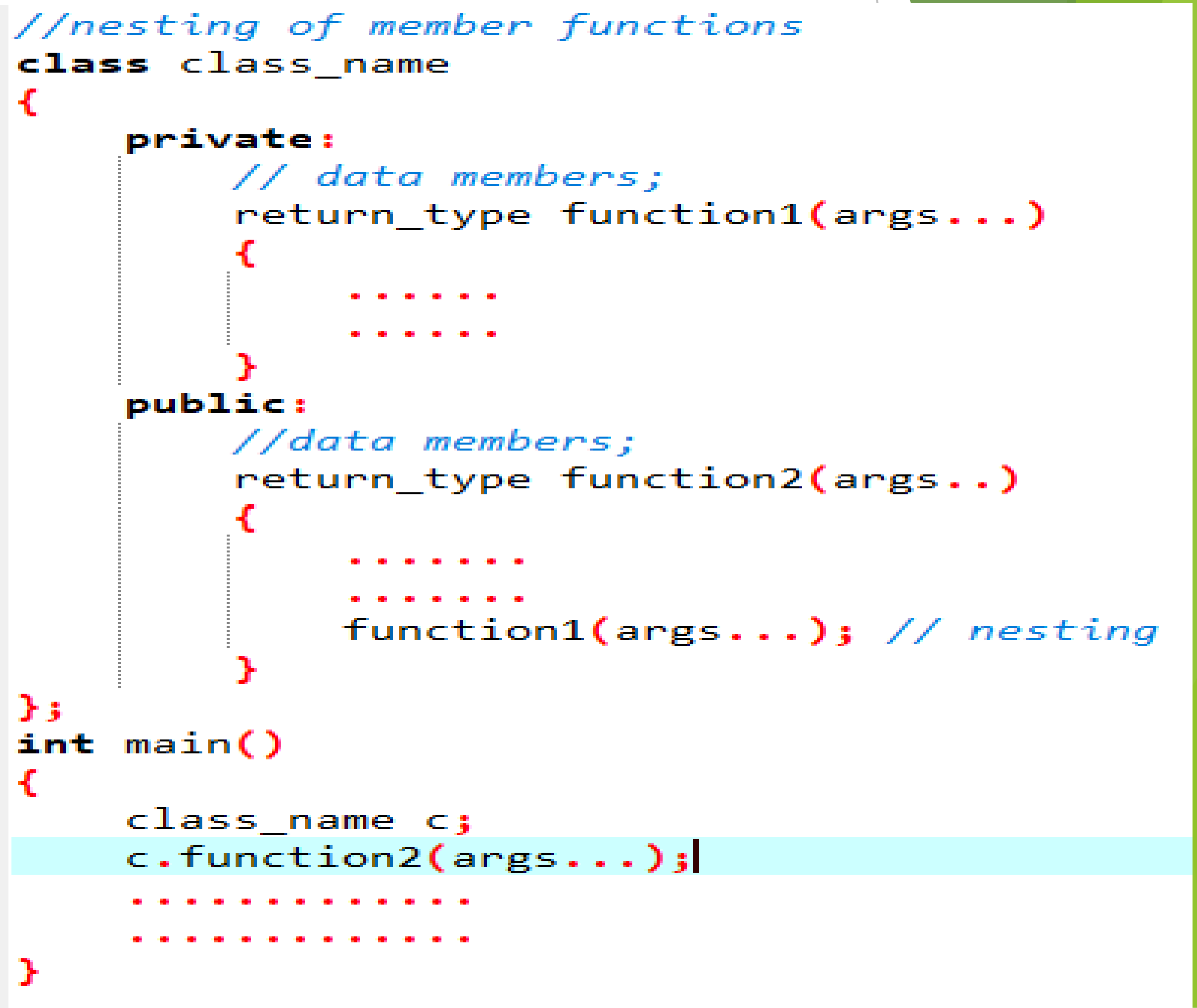
- **Syntax:**



```
class class_name
{
    private:
        //.....
    public:
        return_type member_fuction(args..)
        //.....

};
inline return_type class_name::member_function(args....)
{
    //Function body
}
```

Nesting of member Functions

- A member function can be called by using its name from another member function of the same class. This is known as nesting of member functions.

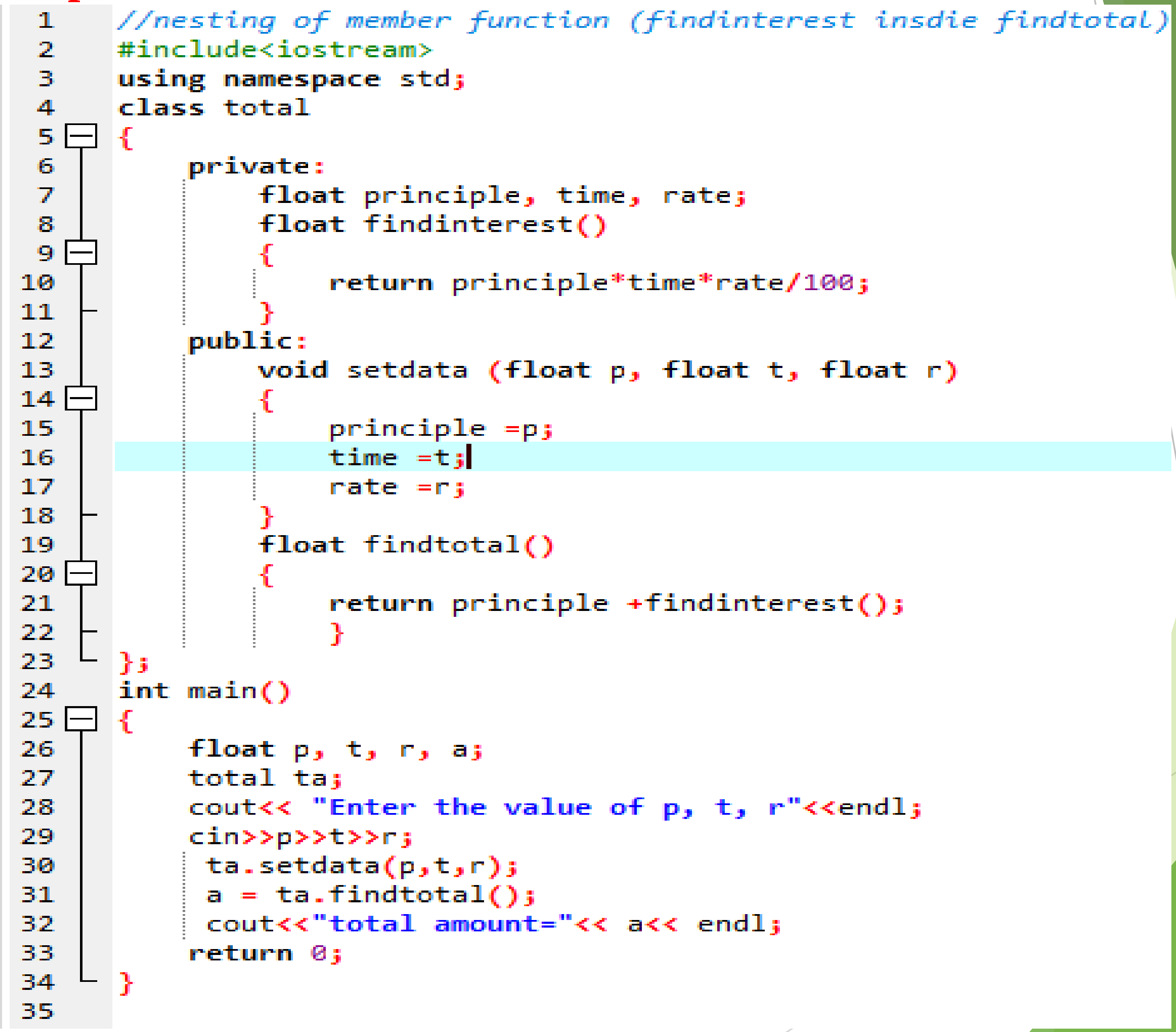


```
//nesting of member functions
class class_name
{
    private:
        // data members;
        return_type function1(args...)
        {
            ......
            ......
        }
    public:
        //data members;
        return_type function2(args..)
        {
            .......
            .......
            function1(args...); // nesting
        }
};
int main()
{
    class_name c;
    c.function2(args...);
    ..............
    ..............
}
```

By: Rhishav Poudyal

Contd..

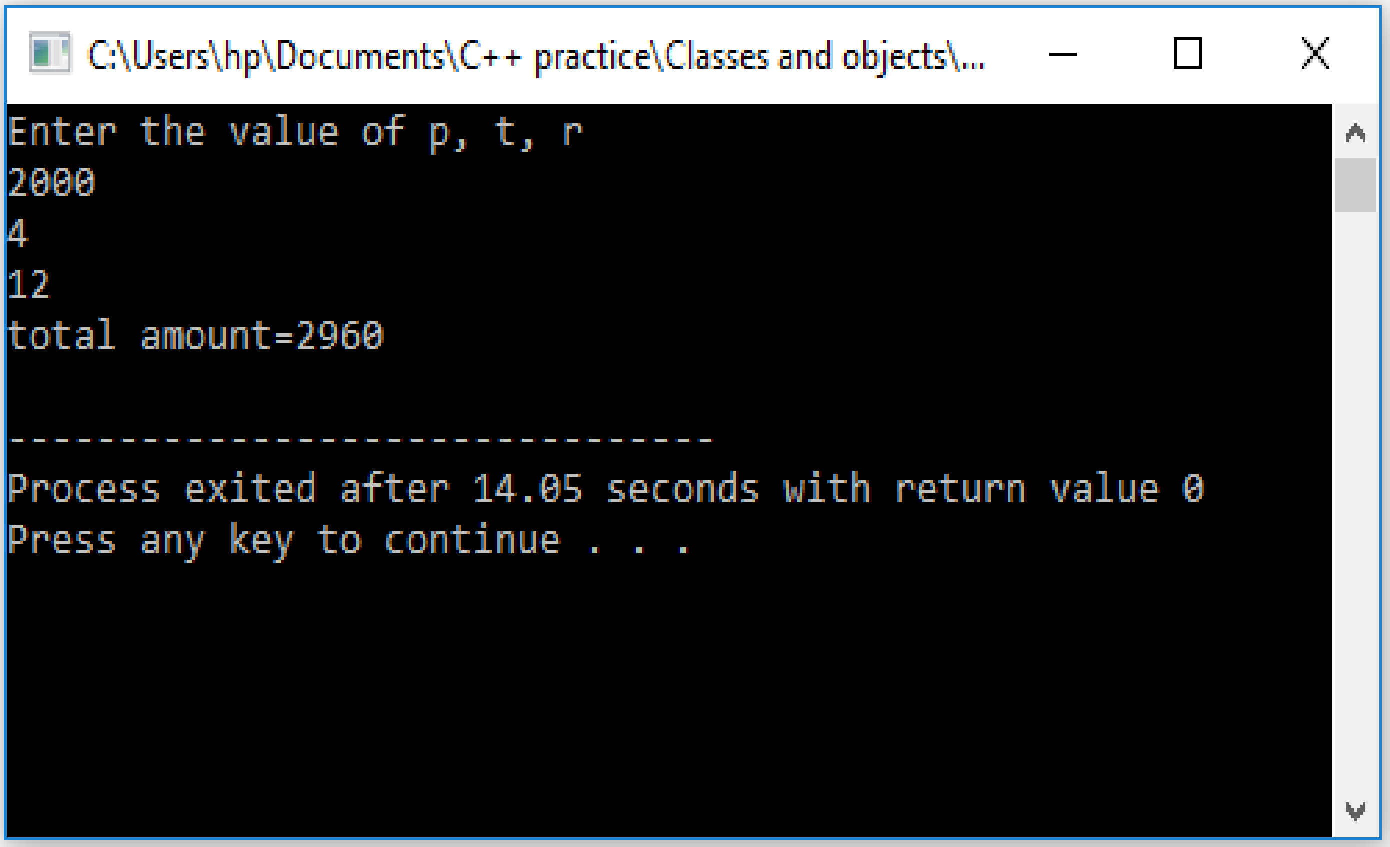
- **Example:**



```
//nesting of member function (findinterest insdie findtotal)
#include<iostream>
using namespace std;
class total
{
    private:
        float principle, time, rate;
        float findinterest()
        {
            return principle*time*rate/100;
        }
    public:
        void setdata (float p, float t, float r)
        {
            principle =p;
            time =t;
            rate =r;
        }
        float findtotal()
        {
            return principle +findinterest();
            }
};
int main()
{
    float p, t, r, a;
    total ta;
    cout<< "Enter the value of p, t, r"<<endl;
    cin>>p>>t>>r;
     ta.setdata(p,t,r);
     a = ta.findtotal();
     cout<<"total amount="<< a<< endl;
    return 0;
}

```

Contd..

- **Output:**



```
Enter the value of p, t, r
2000
4
12
total amount=2960


--------------------------------
Process exited after 14.05 seconds with return value 0
Press any key to continue . . .
```

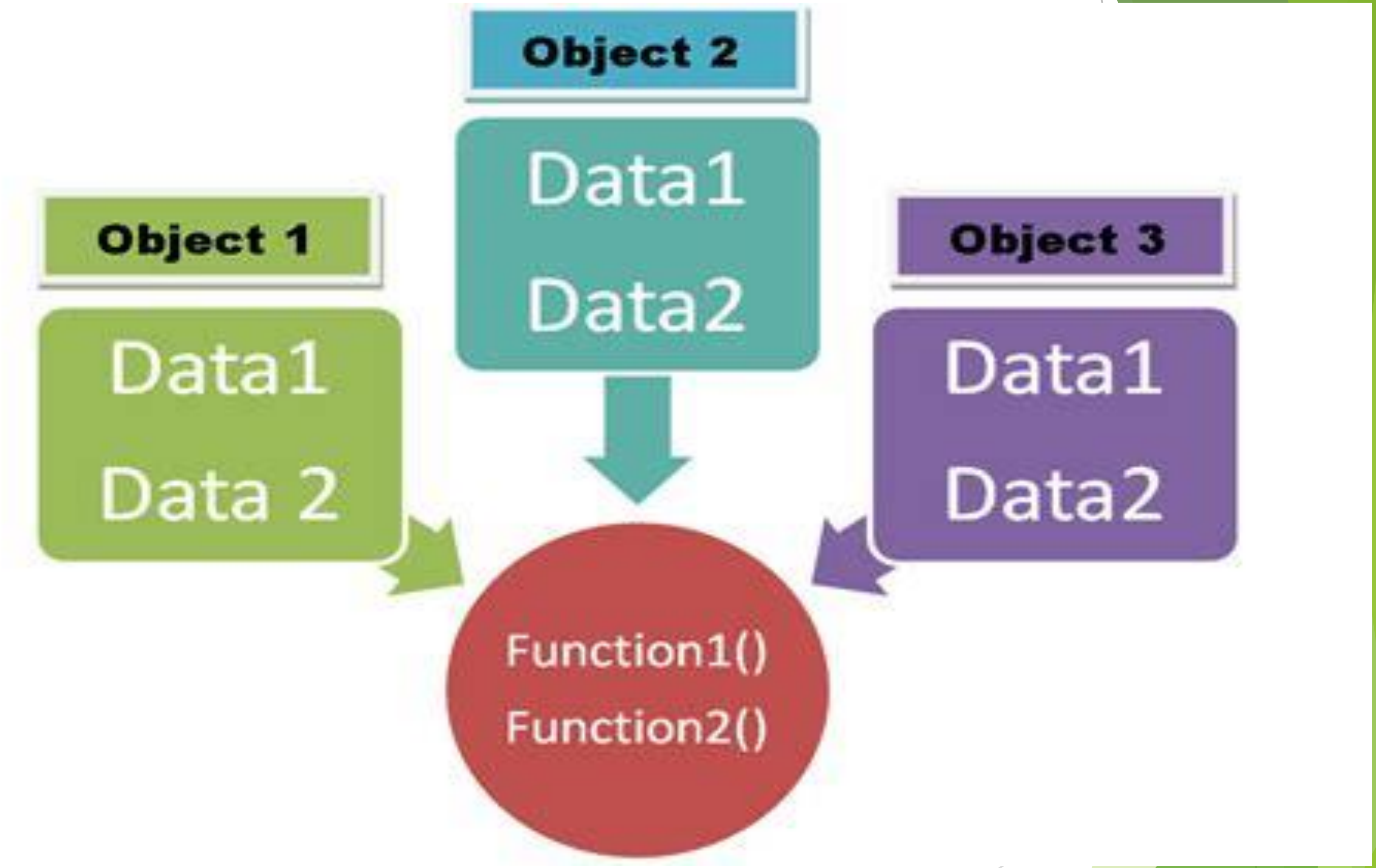
Memory Allocation For Objects

- For each object, the memory space for data members is allocated separately because the data members will hold different data values for different objects.

- However, all the objects in a given class use the same member functions.

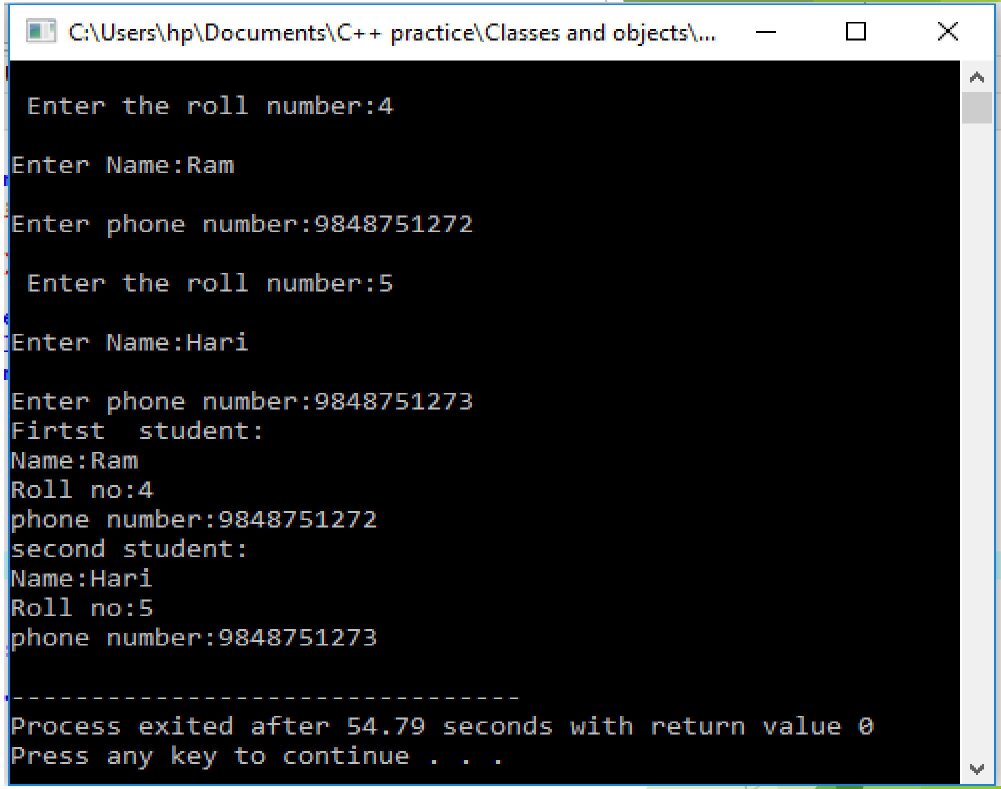
- Hence, the member functions are created and placed in memory only once when they are defined as a part of class specification and no separate space is allocated for member functions when objects are created.

Contd..


Object 2
Data1
Data2
Object 1
Data1
Data 2
Object 3
Data1
Data2
Function1()
Function2()

Contd..

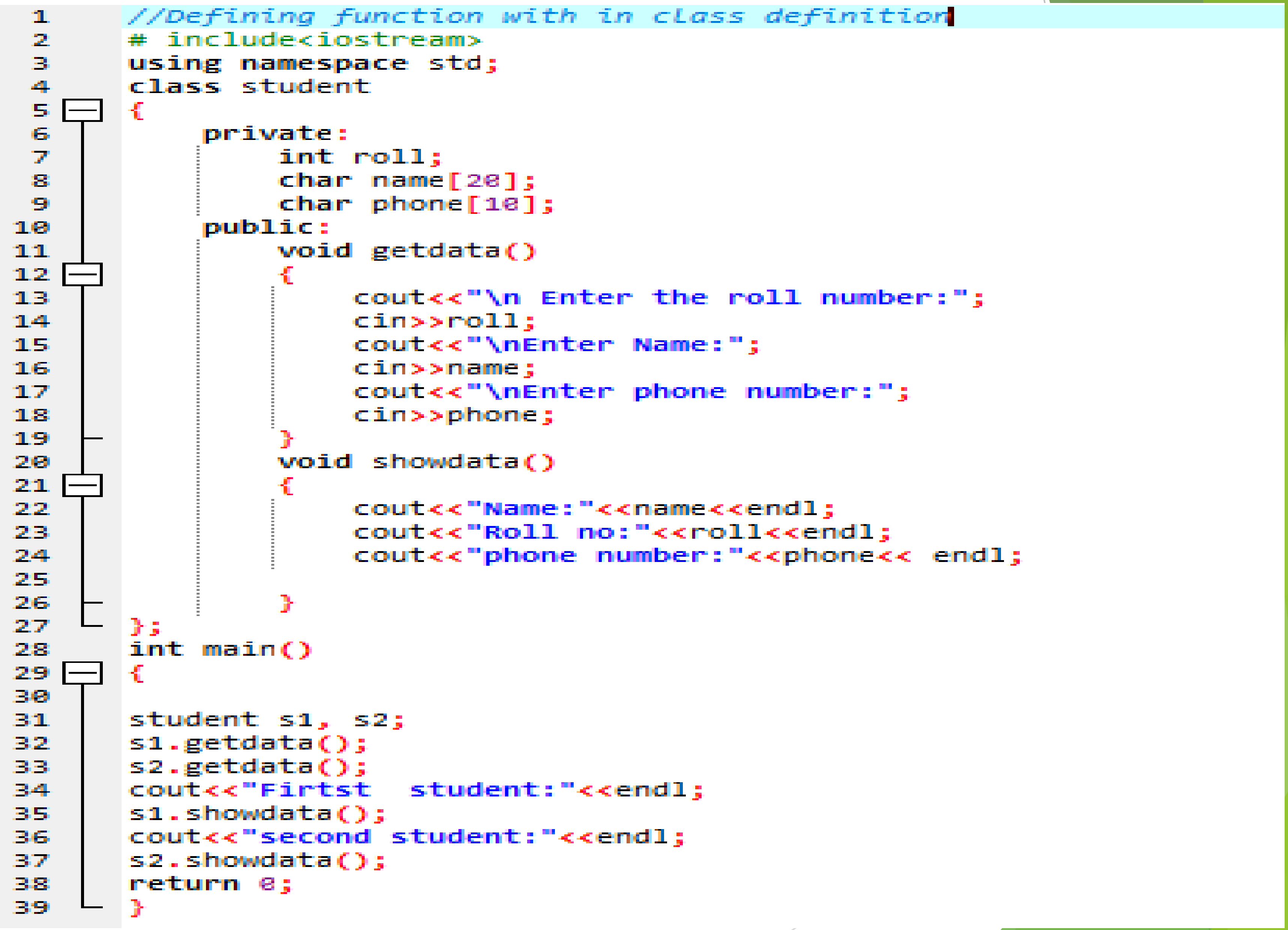
- **Class work**: Write a program designing a class to represent student. Include the following members
  - Data members
    - Roll
    - Name
    - phone
  - Member functions
    - Getdata
    - showdata



```
 Enter the roll number:4

Enter Name:Ram

Enter phone number:9848751272

 Enter the roll number:5

Enter Name:Hari

Enter phone number:9848751273
Firtst  student:
Name:Ram
Roll no:4
phone number:9848751272
second student:
Name:Hari
Roll no:5
phone number:9848751273

--------------------------------
Process exited after 54.79 seconds with return value 0
Press any key to continue . . .
```

- Define member function inside class
- Define member function outside class
- Make the functions inline also

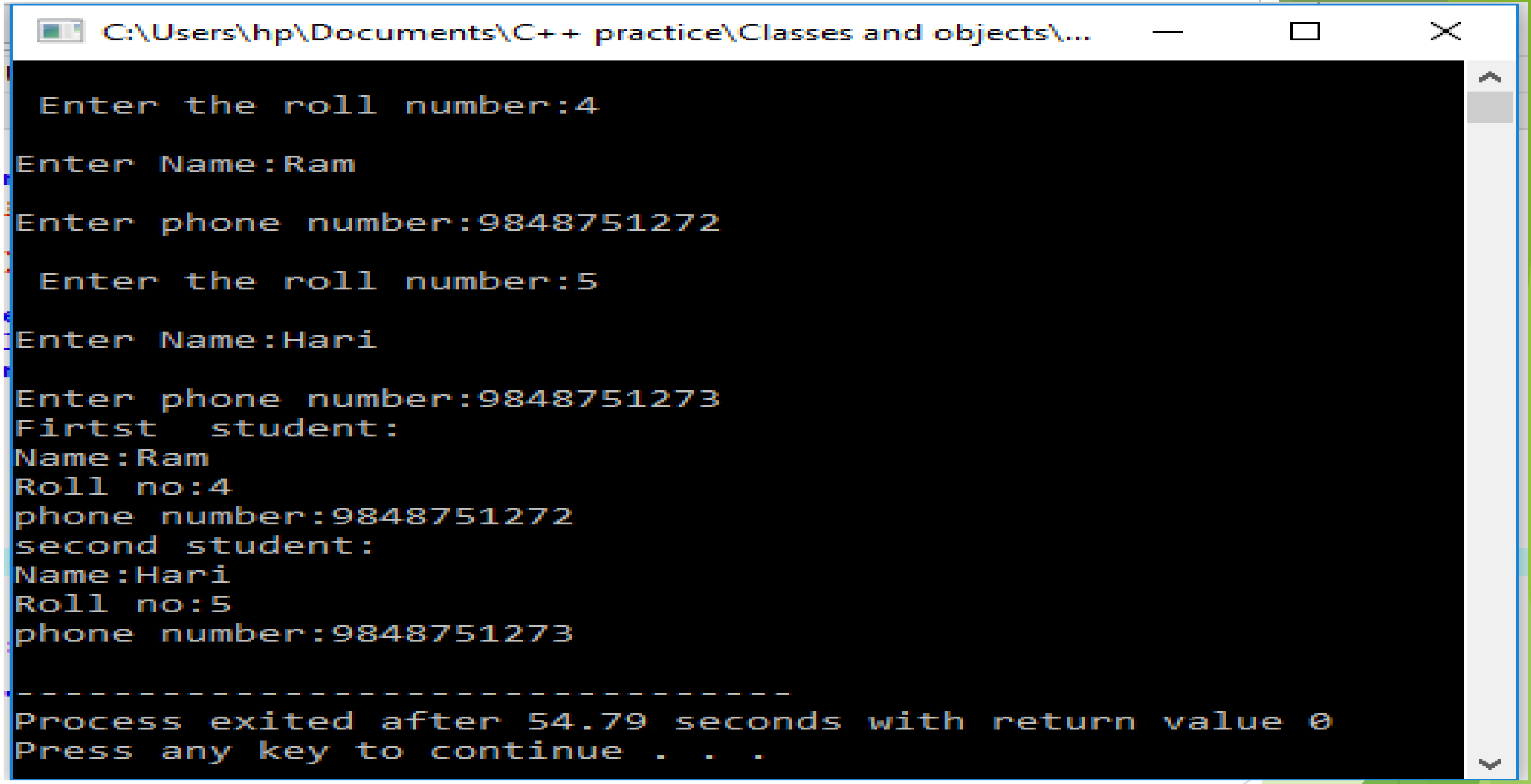
Example: Contd..



```cpp
//Defining function with in class definition
# include<iostream>
using namespace std;
class student
{
    private:
        int roll;
        char name[20];
        char phone[10];
    public:
        void getdata()
        {
            cout<<"\n Enter the roll number:";
            cin>>roll;
            cout<<"\nEnter Name:";
            cin>>name;
            cout<<"\nEnter phone number:";
            cin>>phone;
        }
        void showdata()
        {
            cout<<"Name:"<<name<<endl;
            cout<<"Roll no:"<<roll<<endl;
            cout<<"phone number:"<<phone<< endl;

        }
};
int main()
{

student s1, s2;
s1.getdata();
s2.getdata();
cout<<"Firtst  student:"<<endl;
s1.showdata();
cout<<"second student:"<<endl;
s2.showdata();
return 0;
}
```

Contd..

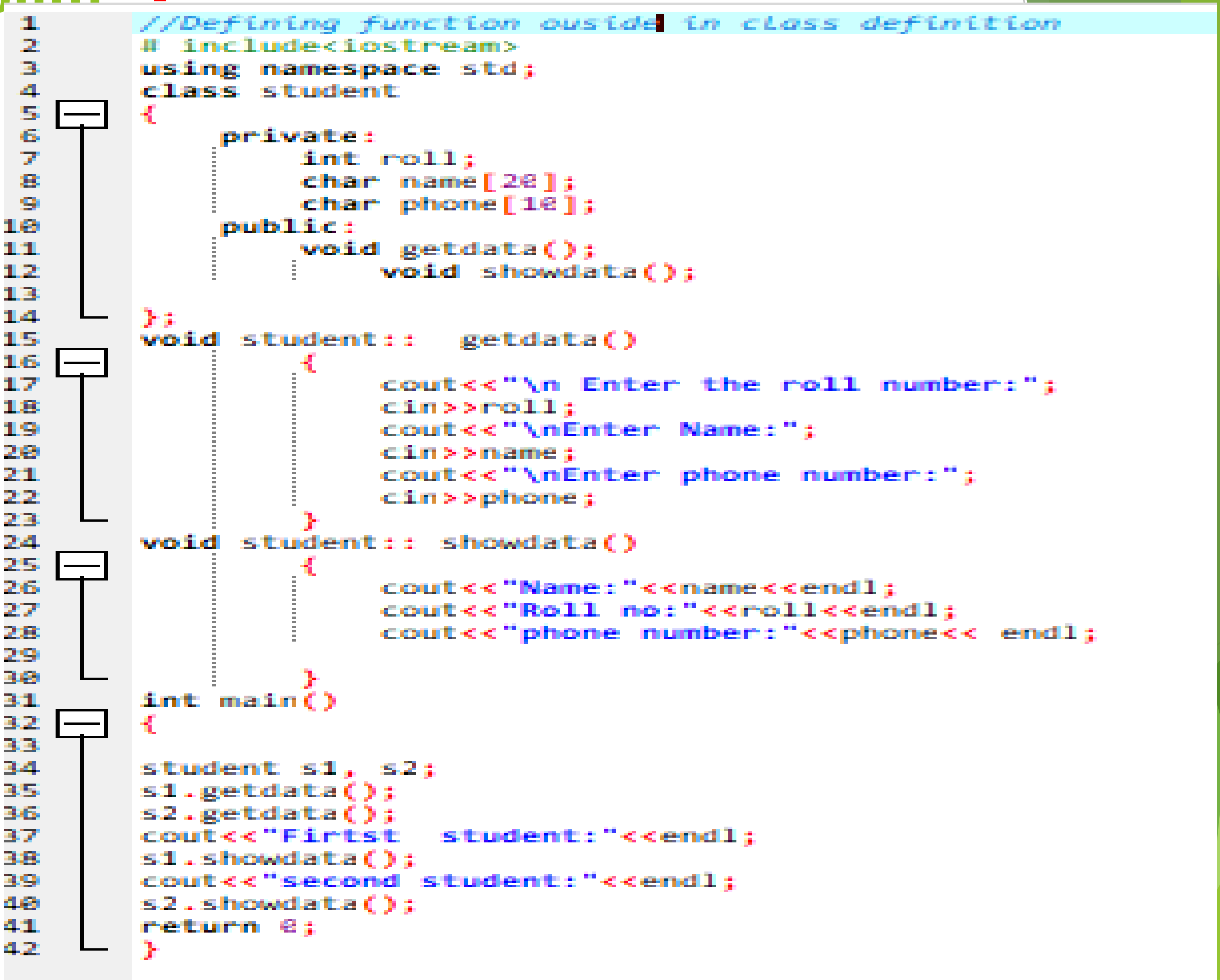
- **Output:**



```
 Enter the roll number:4

Enter Name:Ram

Enter phone number:9848751272

 Enter the roll number:5

Enter Name:Hari

Enter phone number:9848751273
Firtst  student:
Name:Ram
Roll no:4
phone number:9848751272
second student:
Name:Hari
Roll no:5
phone number:9848751273

--------------------------------
Process exited after 54.79 seconds with return value 0
Press any key to continue . . .
```

Contd

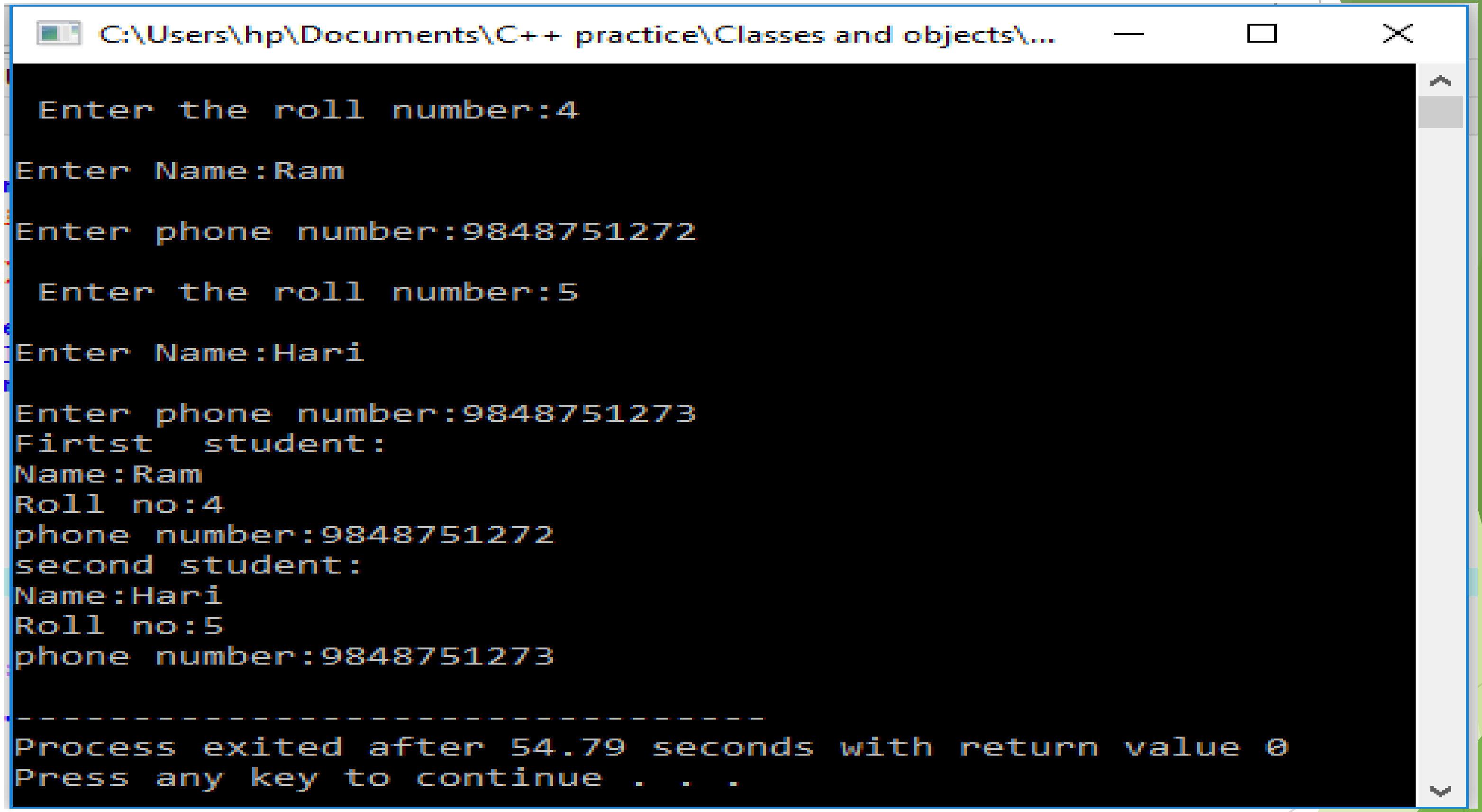
Example:



```
//Defining function ouside in class definition
# include<iostream>
using namespace std;
class student
{
    private:
        int roll;
        char name[20];
        char phone[10];
    public:
        void getdata();
            void showdata();

};
void student::  getdata()
        {
            cout<<"\n Enter the roll number:";
            cin>>roll;
            cout<<"\nEnter Name:";
            cin>>name;
            cout<<"\nEnter phone number:";
            cin>>phone;
        }
void student:: showdata()
        {
            cout<<"Name:"<<name<<endl;
            cout<<"Roll no:"<<roll<<endl;
            cout<<"phone number:"<<phone<< endl;

        }
int main()
{

student s1, s2;
s1.getdata();
s2.getdata();
cout<<"Firtst  student:"<<endl;
s1.showdata();
cout<<"second student:"<<endl;
s2.showdata();
return 0;
}
```

Contd..

- **Output**:



C:\Users\hp\Documents\C++ practice\Classes and objects\...

```
 Enter the roll number:4

Enter Name:Ram

Enter phone number:9848751272

 Enter the roll number:5

Enter Name:Hari

Enter phone number:9848751273
Firtst  student:
Name:Ram
Roll no:4
phone number:9848751272
second student:
Name:Hari
Roll no:5
phone number:9848751273

--------------------------------
Process exited after 54.79 seconds with return value 0
Press any key to continue . . .
```

Constructors and Destructors

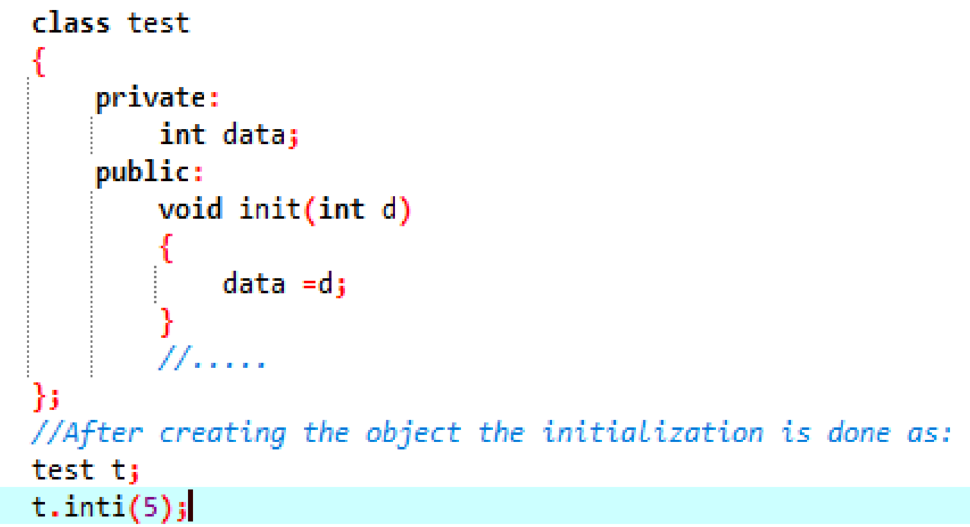
- When an object is created all the members of the object are allocated memory spaces.

- Each objects has its individual copy of member variables.

- However the data members(member variables) are not initialized automatically.

Contd..

- For initialization of data members we need to make member functions like init( ) as follows:



```
class test
{
    private:
        int data;
    public:
        void init(int d)
        {
            data =d;
        }
        //.....
};
//After creating the object the initialization is done as:
test t;
t.inti(5);
```

- Problem: If the programmer forgets to initialize the variable by calling the function init(), the program will produce unusual result.
- Solution: Constructor!

Contd..

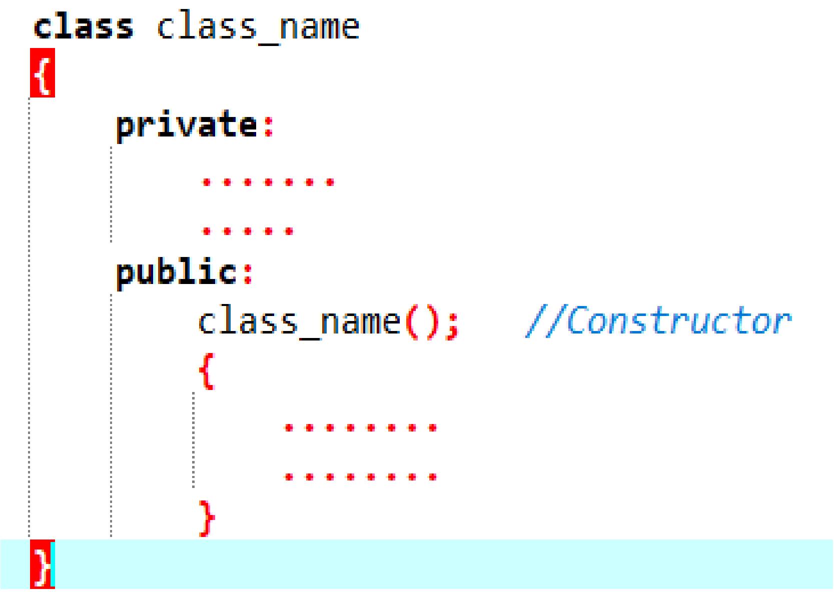
- What is constructor?
  - A constructor is a special member function that is executed automatically whenever an object is created.
  - It is used for automatic initialization.
    - Automatic initialization is the process of initializing object’s data members when it is first created, without making a separate call to a member function.

Contd..

- Some characteristics of constructors are:
  - The name of the constructor is same as the class name.
  - Constructors should be defined or declared in the public section
  - They do not have return types.
  - Like functions they can have default arguments.

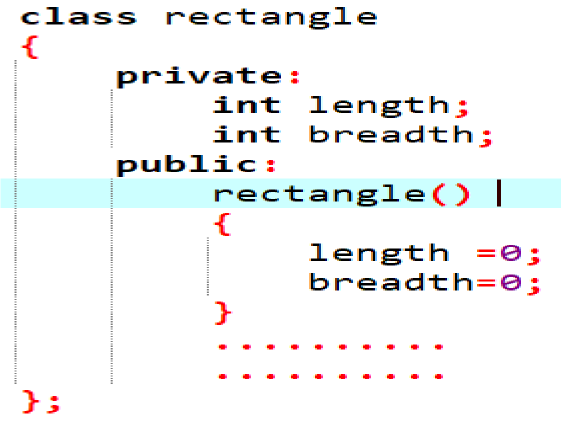
Contd..

- **Syntax:**



```
class class_name
{
    private:
        .......
        .....
    public:
        class_name();    //Constructor
        {
            ........
            ........
        }
}
```

- **Example:**



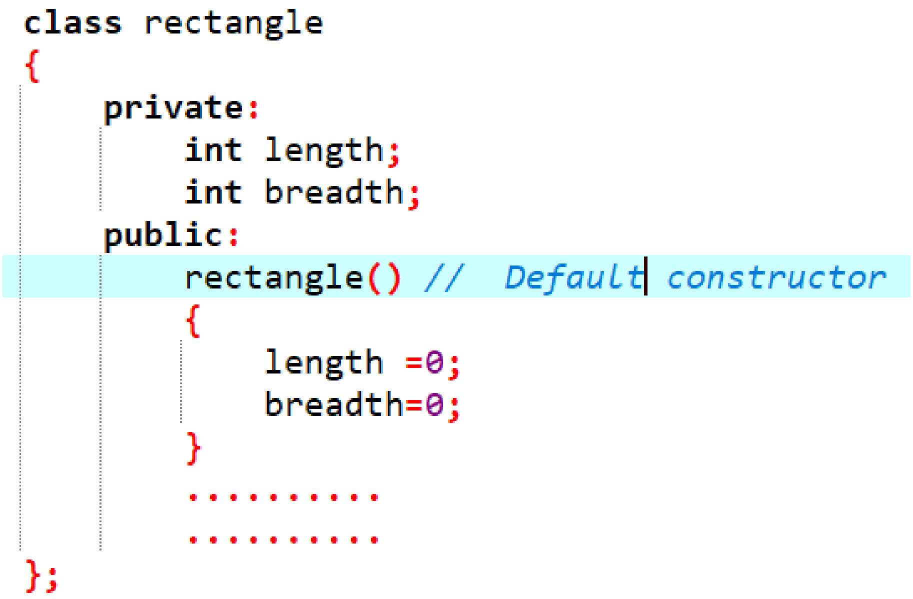
```
class rectangle
{
        private:
                int length;
                int breadth;
        public:
                rectangle() |
                {
                        length =0;
                        breadth=0;
                }
                ...........
                ...........
};
```

Types of constructor

- Default constructor
- Parameterized constructor
- Copy constructor
- Default copy constructor

Contd..

- **Default Constructor**:
  - A constructor that takes no arguments is called default constructor.
    - **E.g.,:**



```
class rectangle
{
    private:
        int length;
        int breadth;
    public:
        rectangle() //  Default constructor
        {
            length =0;
            breadth=0;
        }
        ..........
        ..........
};
```

Contd..

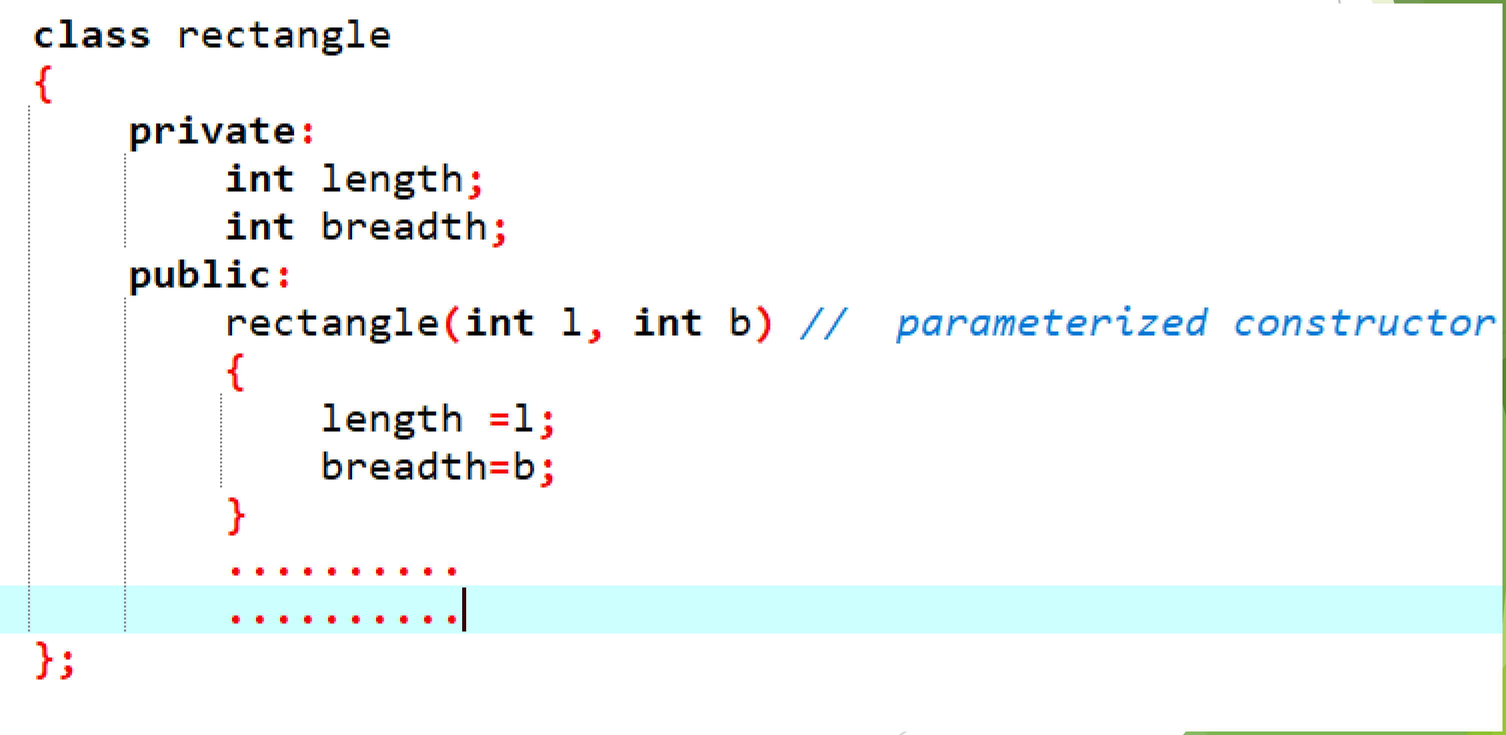
- A default constructor is called automatically at the time of object creation (if no arguments are supplied) and does nothing more than initializing the data variables of the object to valid initial values.

- Syntax: class_name object_name; // default constructor is invoked
- E.g.,: rectangle r1; // default constructor rectangle of the class rectangle is invoked.

- Note: Compiler generates the default constructor if none of the constructors are defined.

Contd..

- Parameterized constructors:
  - The constructor that can take arguments are called parameterized constructors.
  - E.g.,:



```
class rectangle
{
    private:
        int length;
        int breadth;
    public:
        rectangle(int l, int b) //  parameterized constructor
        {
            length =l;
            breadth=b;
        }
        ..........
        ..........
};
```

Contd..

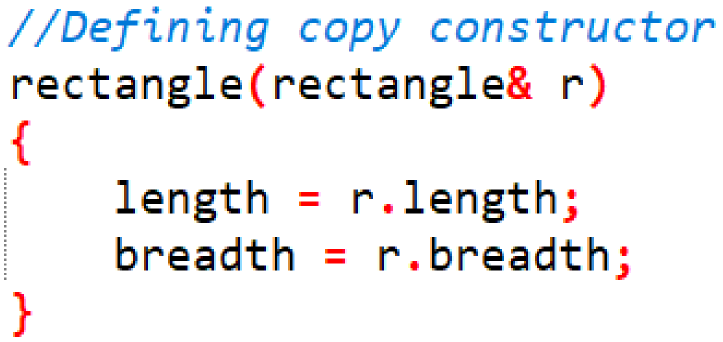
- Parameterized constructors are used if it is necessary to initialize the various data elements of different objects with different values when they are created.
- This can be achieved by passing arguments to the constructor when the objects are created.
- Two ways:
  - By calling the constructor explicitly: e.g., rectangle r1 = rectangle(5,6)
  - By calling the constructor implicitly: e.g., rectangle r1(5, 6);

Contd..

- **Copy constructors:**
  - Copy constructor allows an object to be initialized with another object of the same class.
  - It implies that the values stored in data members of an existing object can be copied into the data members of the object being constructed, provided the objects belong to the same class.
  - A copy constructor has a single parameter of reference type that refers to the class itself as shown below:

Contd..

- E.g.,:



```
//Defining copy constructor
rectangle(rectangle& r)
{
    length = r.length;
    breadth = r.breadth;
}
```

- Once a copy constructor is defined, we can use copy constructor as :

  rectangle r2(r1) ;

- It creates new object r2 and perform member-by-member copy of r1 into r2 .

Contd..

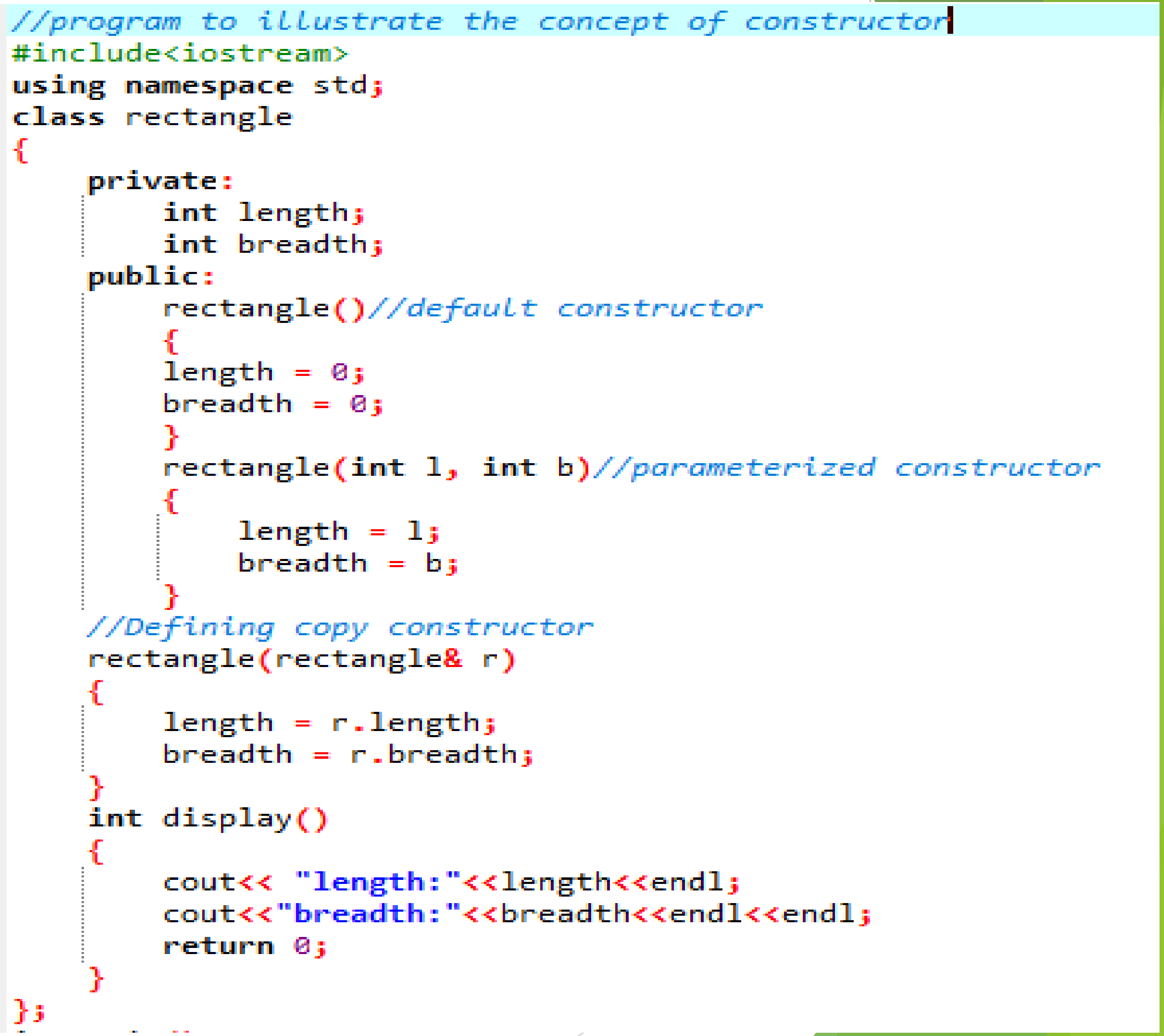
- **Default copy constructor:**
  - If you don’t define copy constructor, the compiler creates a default copy constructor for each class which does a member-wise copy between objects.
  - But it will do shallow copy, if class contains pointers than default copy constructor can not copy value pointed by pointer variable of one object into pointer variable of another object rather both of them will point to the same place.
  - To handle this situation we need to create copy constructor.
  - Default copy constructor can be invoked as below:

```
rectangle r2 = r1;
```

It creates new object r2 and perform member-by-member copy of r1 into r2.

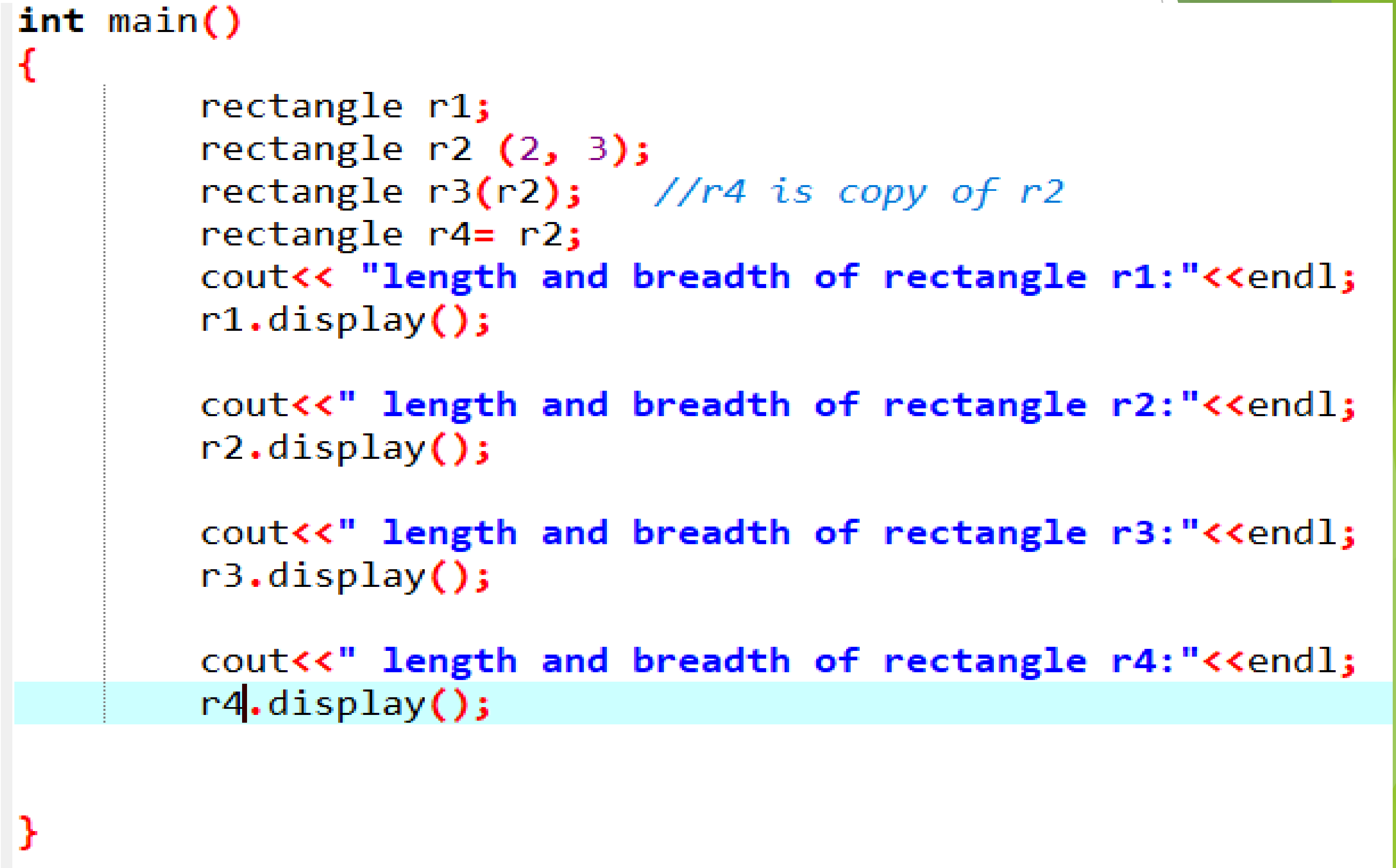
Contd..

Example:



```
//program to illustrate the concept of constructor
#include<iostream>
using namespace std;
class rectangle
{
    private:
        int length;
        int breadth;
    public:
        rectangle()//default constructor
        {
        length = 0;
        breadth = 0;
        }
        rectangle(int l, int b)//parameterized constructor
        {
            length = l;
            breadth = b;
        }
    //Defining copy constructor
    rectangle(rectangle& r)
    {
        length = r.length;
        breadth = r.breadth;
    }
    int display()
    {
        cout<< "length:"<<length<<endl;
        cout<<"breadth:"<<breadth<<endl<<endl;
        return 0;
    }
};
```

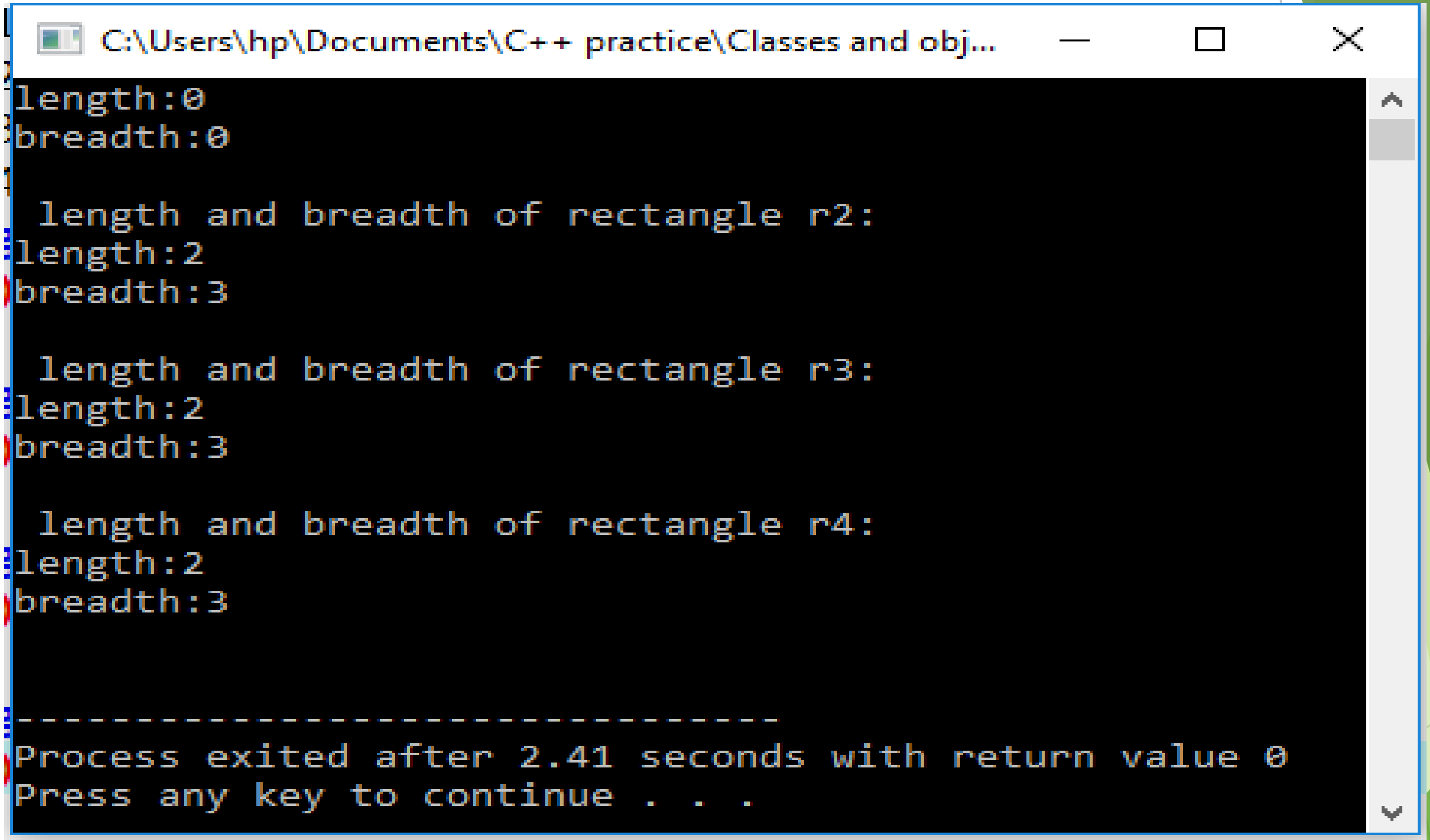
Contd..



```
int main()
{
        rectangle r1;
        rectangle r2 (2, 3);
        rectangle r3(r2);   //r4 is copy of r2
        rectangle r4= r2;
        cout<< "length and breadth of rectangle r1:"<<endl;
        r1.display();

        cout<<" length and breadth of rectangle r2:"<<endl;
        r2.display();

        cout<<" length and breadth of rectangle r3:"<<endl;
        r3.display();

        cout<<" length and breadth of rectangle r4:"<<endl;
        r4.display();


}
```

Contd..

- **Output:**



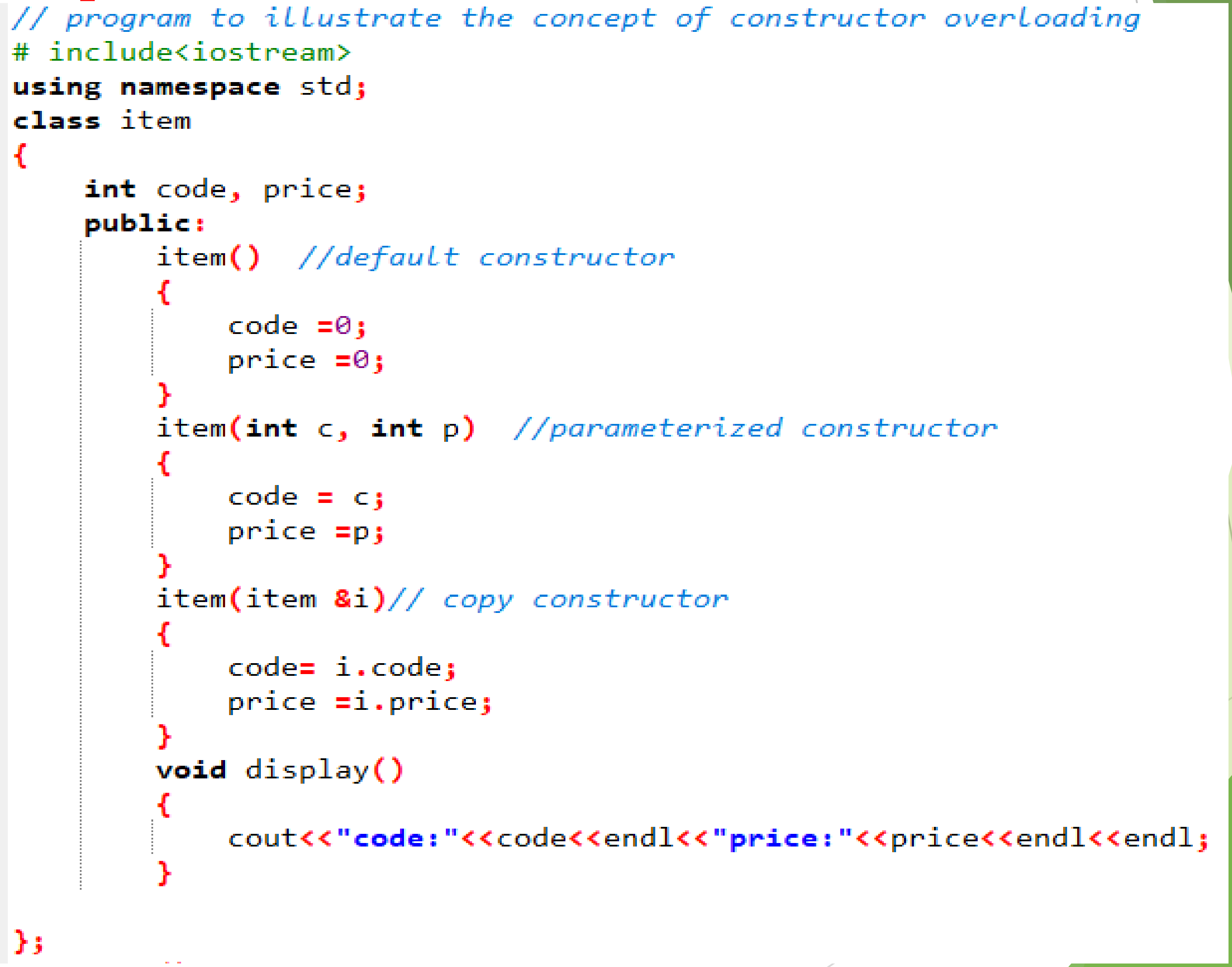
```
length:0
breadth:0

 length and breadth of rectangle r2:
length:2
breadth:3

 length and breadth of rectangle r3:
length:2
breadth:3

 length and breadth of rectangle r4:
length:2
breadth:3



--------------------------------
Process exited after 2.41 seconds with return value 0
Press any key to continue . . .
```

Contd..

- **Constructor overloading:**
  - A class can have multiple constructors.
  - If more than one constructor is used in a class, it is known as constructor overloading.
  - Constructors can be overloaded either by:
    - different number of argument or
    - with different type of argument

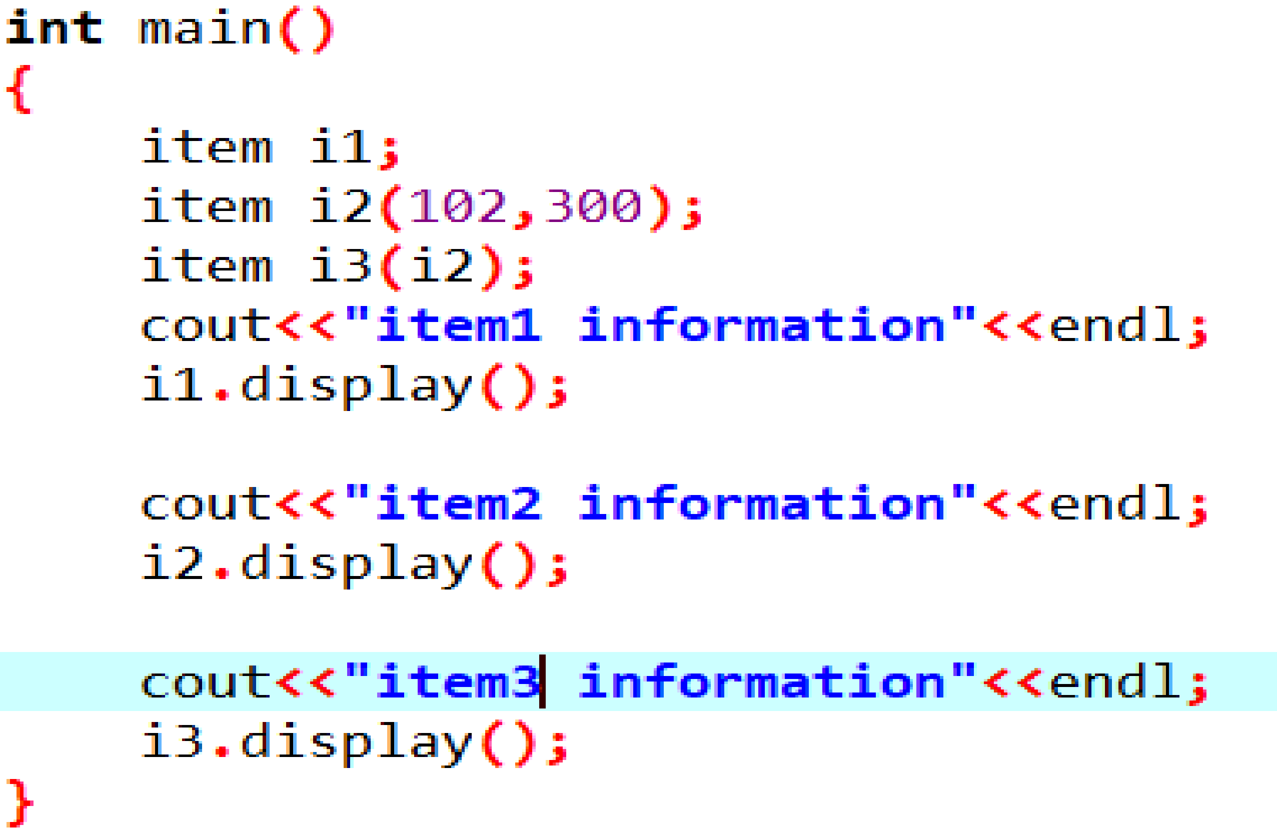
Contd..

- **Example:**



```
// program to illustrate the concept of constructor overloading
# include<iostream>
using namespace std;
class item
{
    int code, price;
    public:
        item()  //default constructor
        {
            code =0;
            price =0;
        }
        item(int c, int p)  //parameterized constructor
        {
            code = c;
            price =p;
        }
        item(item &i)// copy constructor
        {
            code= i.code;
            price =i.price;
        }
        void display()
        {
            cout<<"code:"<<code<<endl<<"price:"<<price<<endl<<endl;
        }

};
```

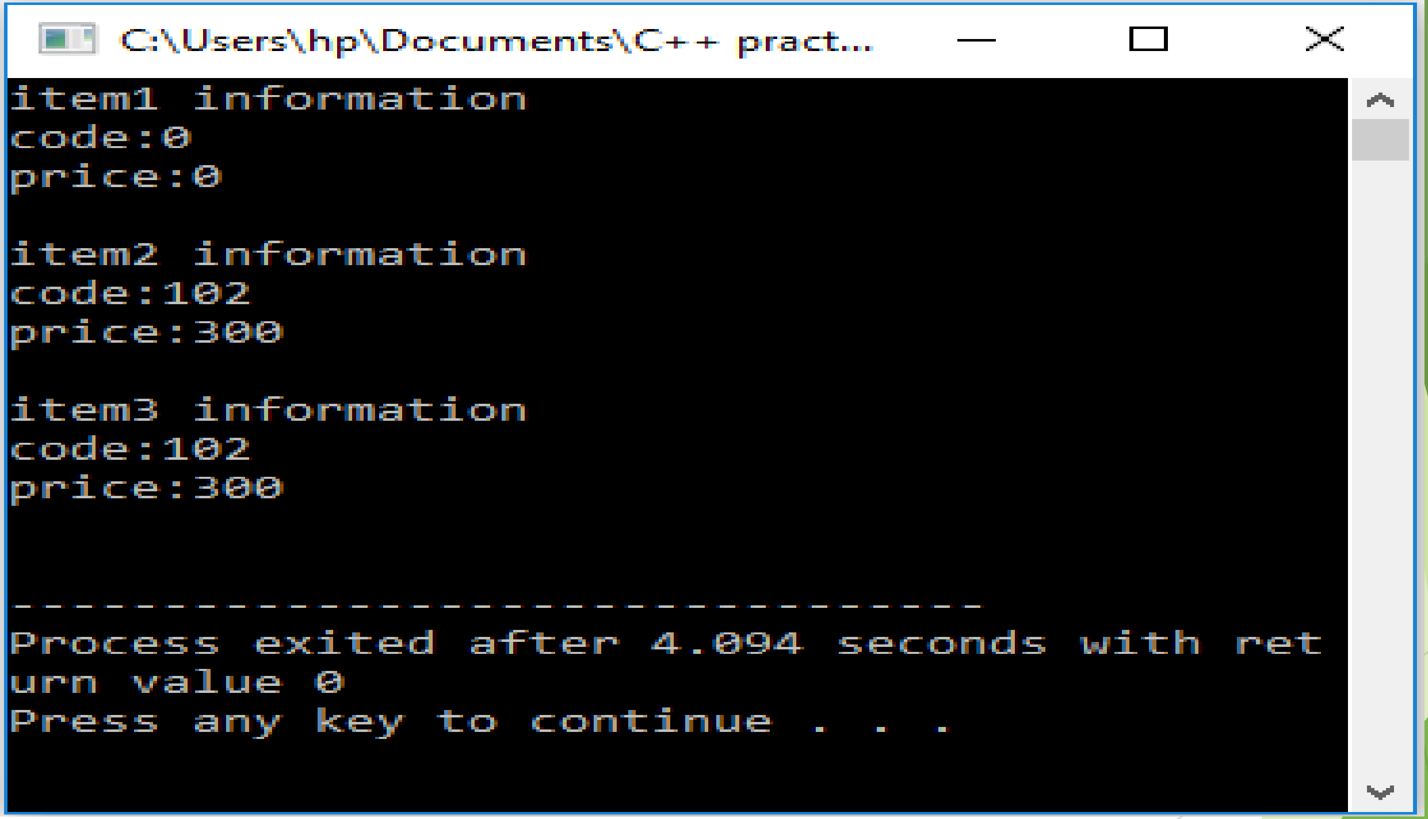
Contd..



```
int main()
{
    item i1;
    item i2(102,300);
    item i3(i2);
    cout<<"item1 information"<<endl;
    i1.display();

    cout<<"item2 information"<<endl;
    i2.display();

    cout<<"item3 information"<<endl;
    i3.display();
}
```

Contd..

- **Output:**



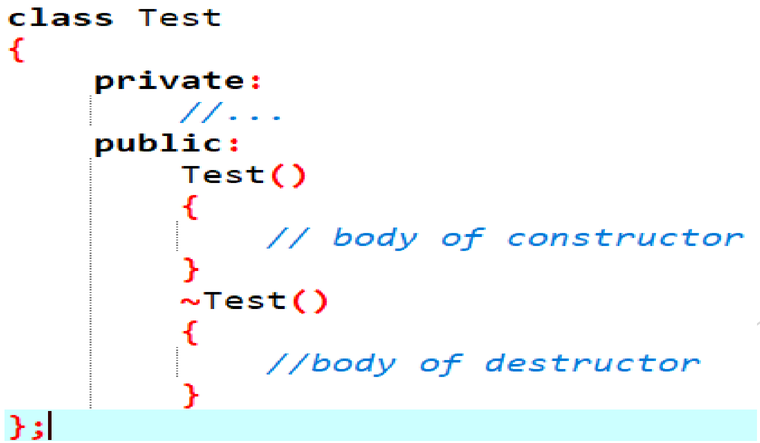
```
item1 information
code:0
price:0

item2 information
code:102
price:300

item3 information
code:102
price:300


--------------------------------
Process exited after 4.094 seconds with ret
urn value 0
Press any key to continue . . .
```

Destructors

- A destructor is a special member function that is executed automatically just before lifetime of an object is finished.
- The most common use of destructor is to destroy the memory that was allocated for the object by the constructor.
- In many cases program do not need user supplied destructors. In that case compiler supplies a default destructor.
- However, if program creates dynamic objects, then destructors should be defined explicitly in the program by the programmer

Contd..

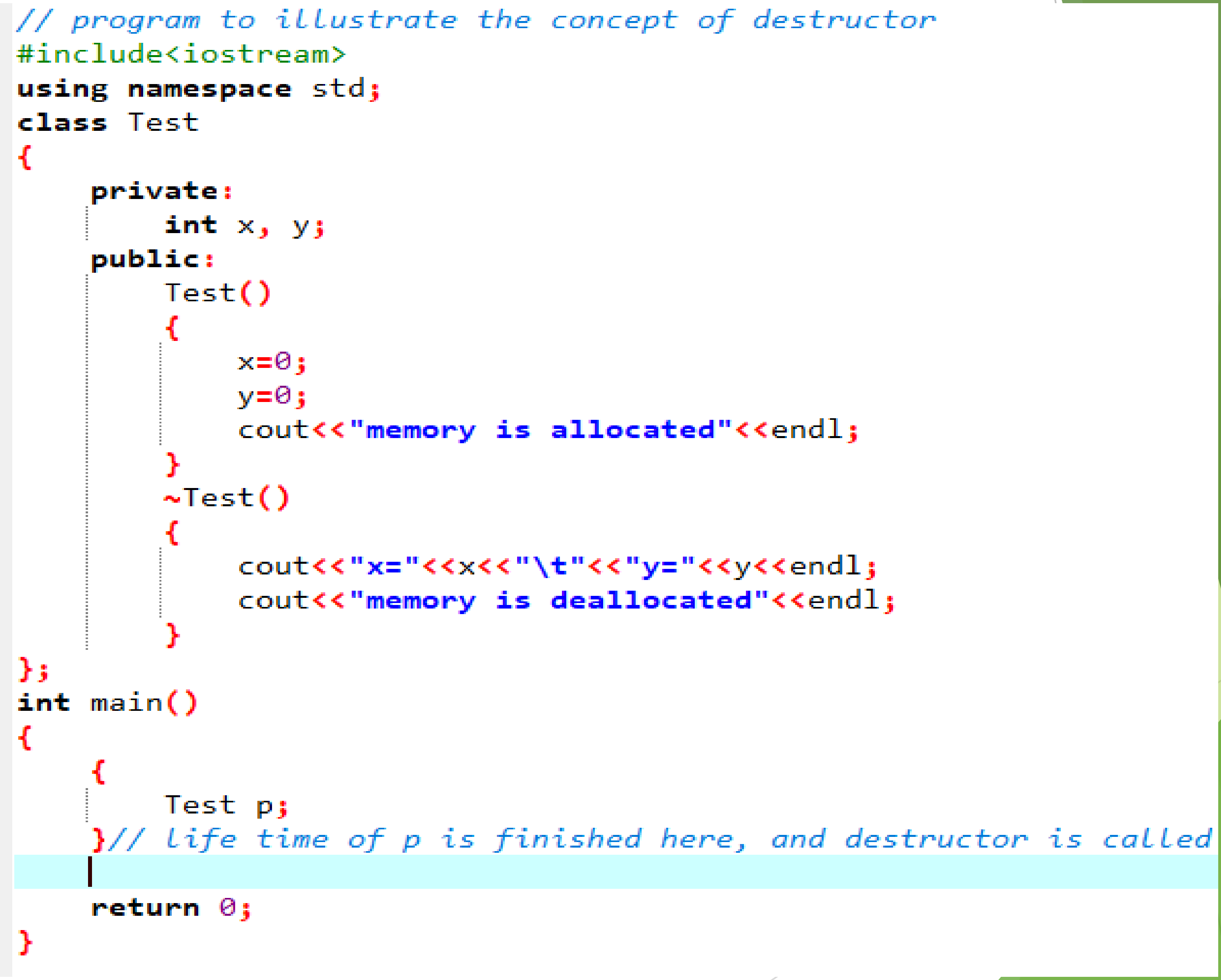
- Destructors have the following characteristics:
  - A Destructor has the same name as the constructor(which is the same as the class name) but is preceded by a tilde(~).
  - Like constructors, destructors do not have a return type.
  - Destructors takes no arguments. Hence we can use only one destructor in a class(i.e., we can not overload destructors).
  - Objects are destroyed in the reverse order of their creation.
  - **Syntax:**



```
class Test
{
    private:
        //...
    public:
        Test()
        {
            // body of constructor
        }
        ~Test()
        {
            //body of destructor
        }
};
```

Contd.

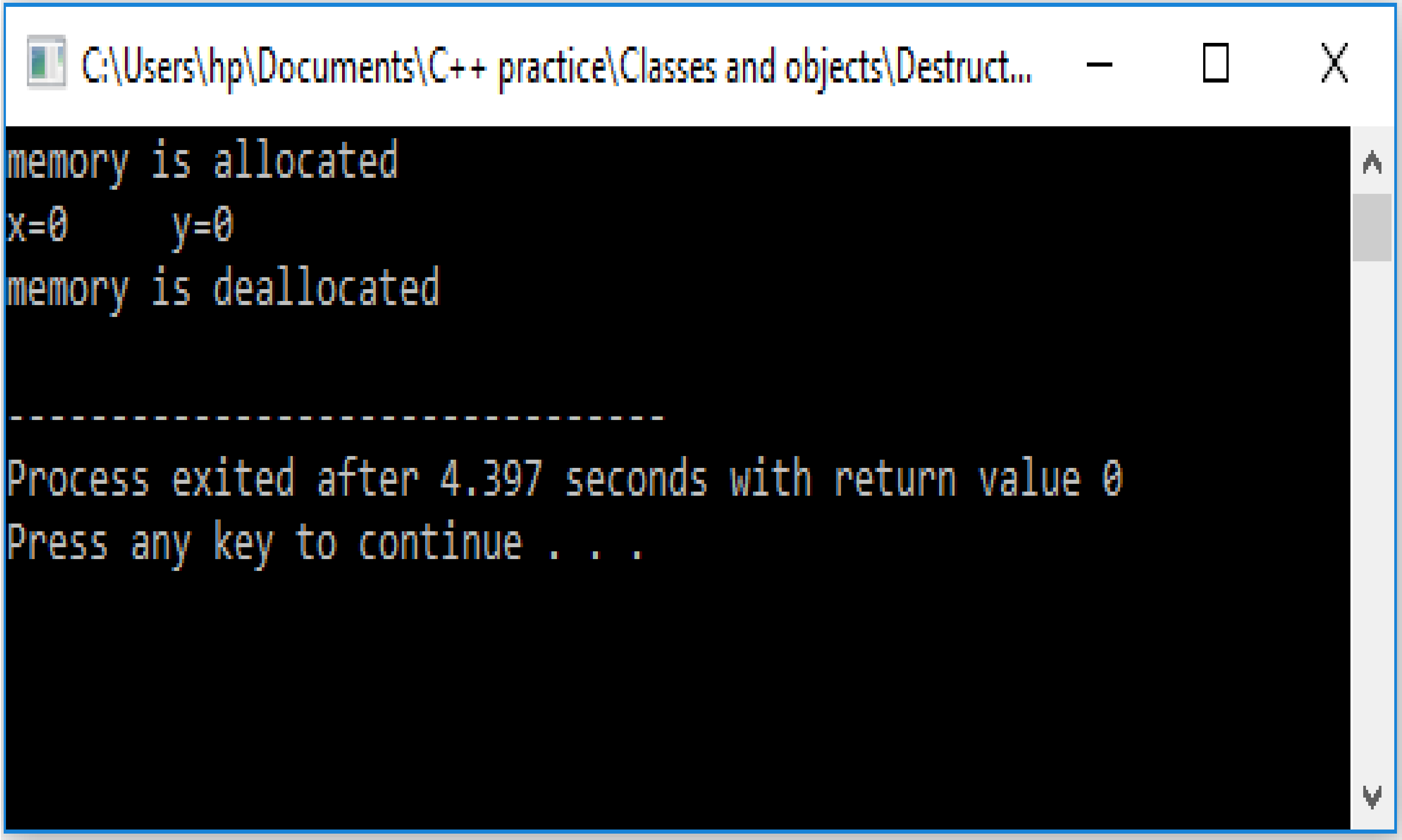
Example:



```
// program to illustrate the concept of destructor
#include<iostream>
using namespace std;
class Test
{
    private:
        int x, y;
    public:
        Test()
        {
            x=0;
            y=0;
            cout<<"memory is allocated"<<endl;
        }
        ~Test()
        {
            cout<<"x="<<x<<"\t"<<"y="<<y<<endl;
            cout<<"memory is deallocated"<<endl;
        }
};
int main()
{
    {
        Test p;
    }// life time of p is finished here, and destructor is called

    return 0;
}
```

Contd..

- **Output:**

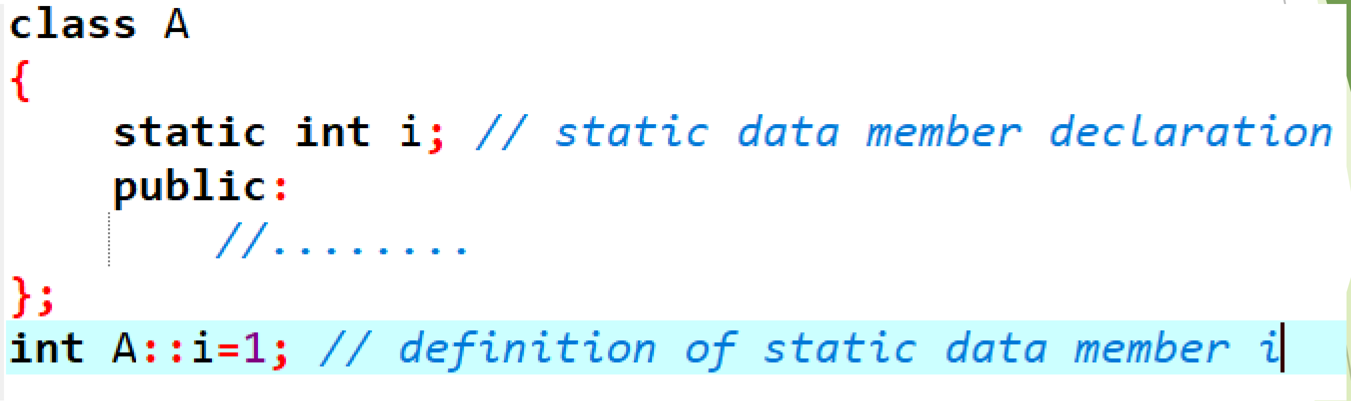


C:\Users\hp\Documents\C++ practice\Classes and objects\Destruct...

```
memory is allocated
x=0      y=0
memory is deallocated


--------------------------------
Process exited after 4.397 seconds with return value 0
Press any key to continue . . .
```

Static data members

- Static data members declared inside class by using static keyword and must be defined outside the class using scope resolution operator.
- Syntax:

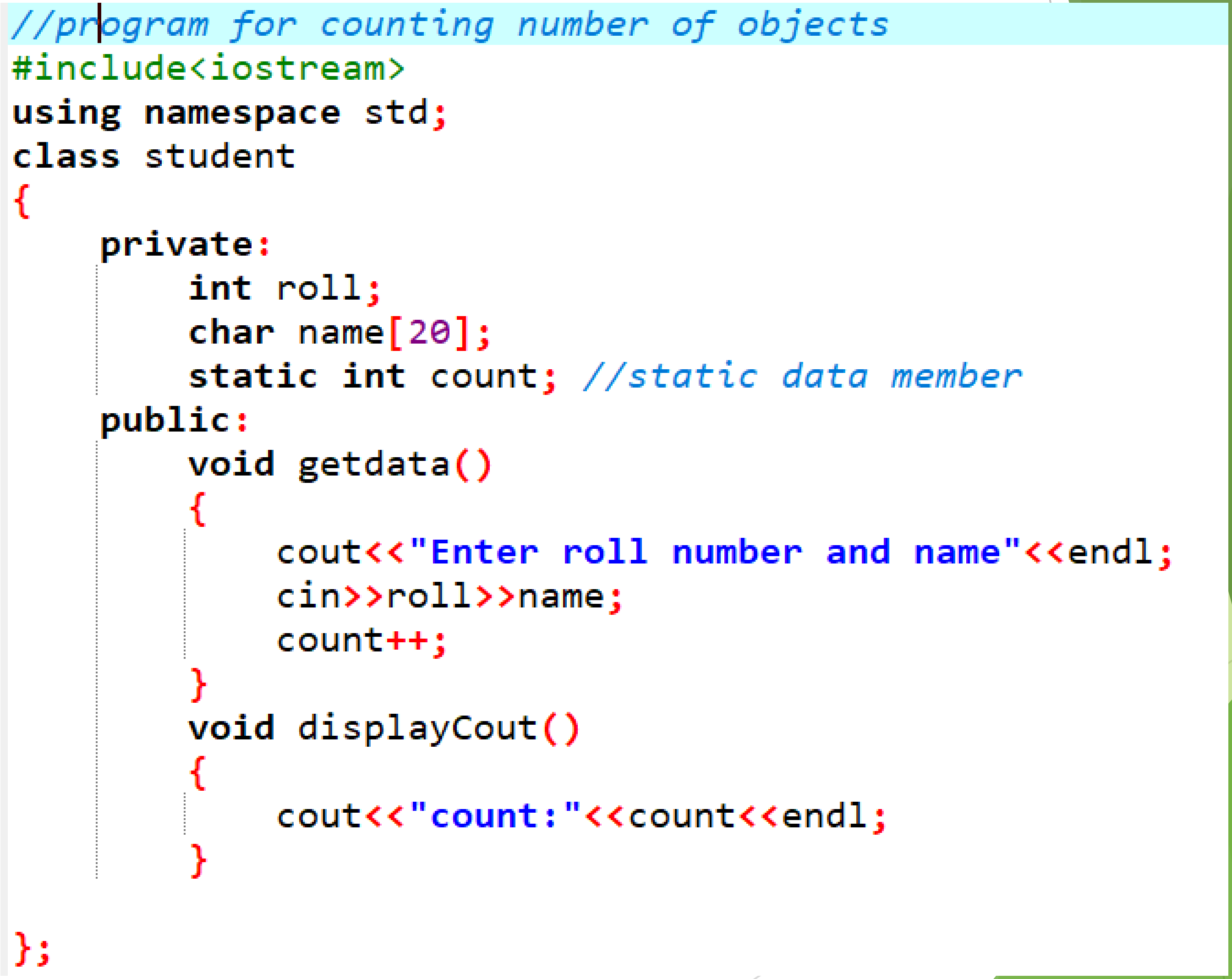


```
class A
{
    static int i; // static data member declaration
    public:
        //........
};
int A::i=1; // definition of static data member i
```

- Only one copy of static member is created for the entire class and is shared by all the objects of that class.
- Used to maintain values common to entire class .

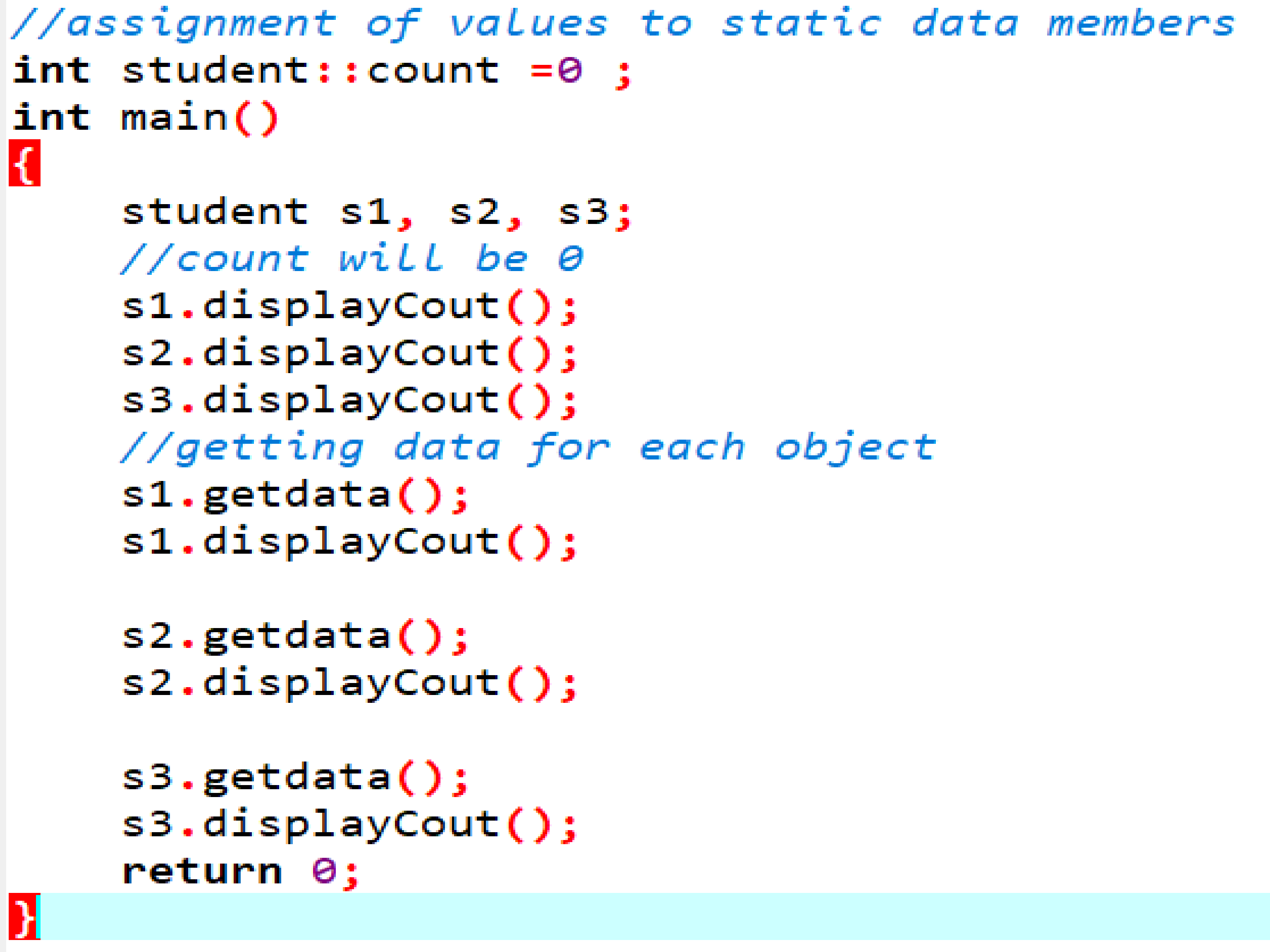
Contd..

- **Example:**



```
//program for counting number of objects
#include<iostream>
using namespace std;
class student
{
    private:
        int roll;
        char name[20];
        static int count; //static data member
    public:
        void getdata()
        {
            cout<<"Enter roll number and name"<<endl;
            cin>>roll>>name;
            count++;
        }
        void displayCout()
        {
            cout<<"count:"<<count<<endl;
        }

};
```

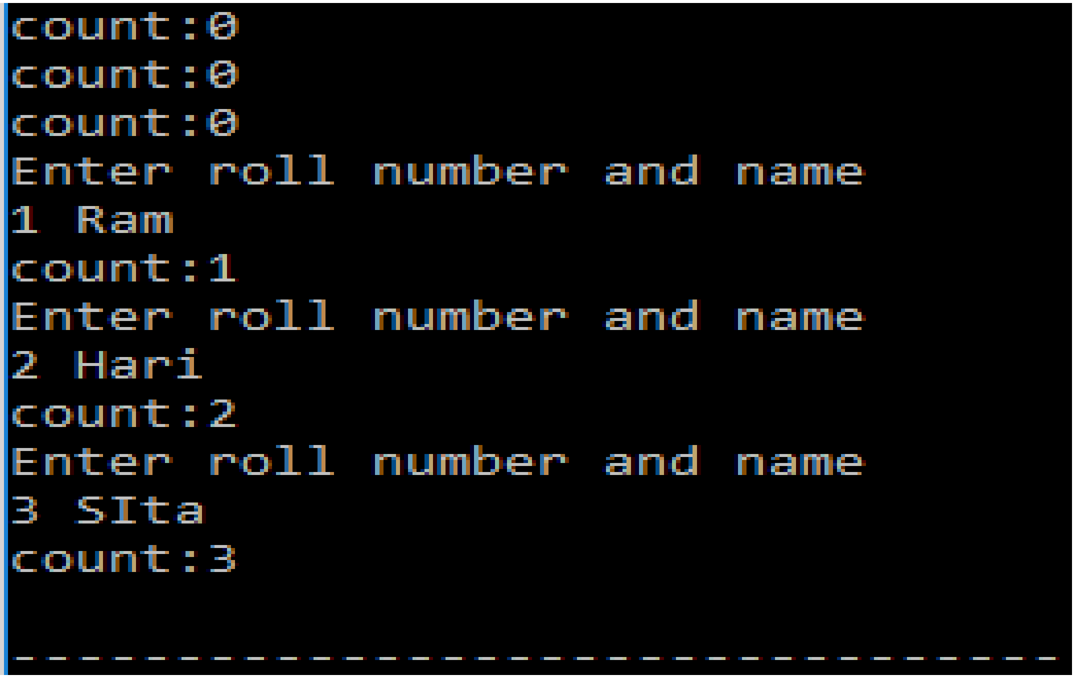
Contd..



```
//assignment of values to static data members
int student::count =0 ;
int main()
{
    student s1, s2, s3;
    //count will be 0
    s1.displayCout();
    s2.displayCout();
    s3.displayCout();
    //getting data for each object
    s1.getdata();
    s1.displayCout();

    s2.getdata();
    s2.displayCout();

    s3.getdata();
    s3.displayCout();
    return 0;
}
```

Contd..

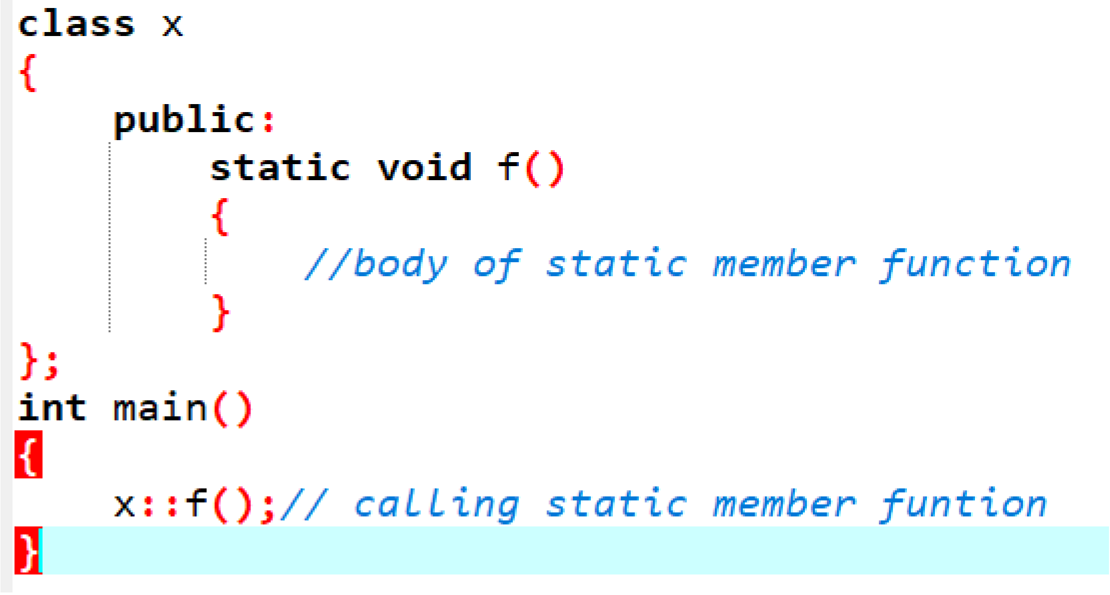
- **Output:**



```
count:0
count:0
count:0
Enter roll number and name
1 Ram
count:1
Enter roll number and name
2 Hari
count:2
Enter roll number and name
3 SIta
count:3


--------------------------------
```

Static member functions

- Static member function can be defined by using static keyword and called by using scope resolution operator.
- Syntax:



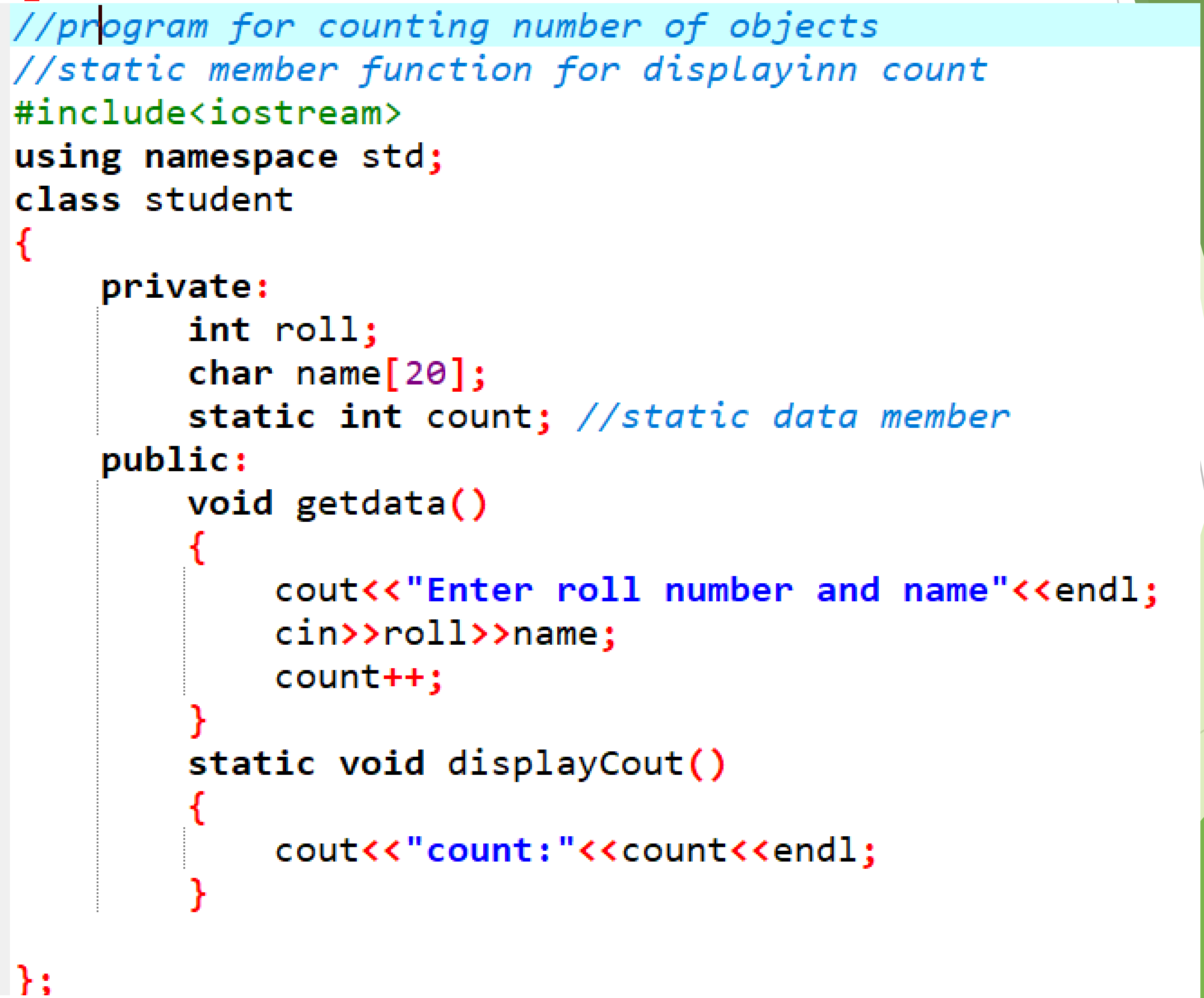
```
class x
{
    public:
        static void f()
        {
            //body of static member function
        }
};
int main()
{
    x::f();// calling static member funtion
}
```

Contd..

- Features:
  - A static member function can be called even if no objects of the class exist .
  - The **static** functions are accessed using the class name and the scope resolution operator **::**.
  - A static member function can only access static data member, and other static member functions.
  - A static member function cannot access ordinary data members and member functions

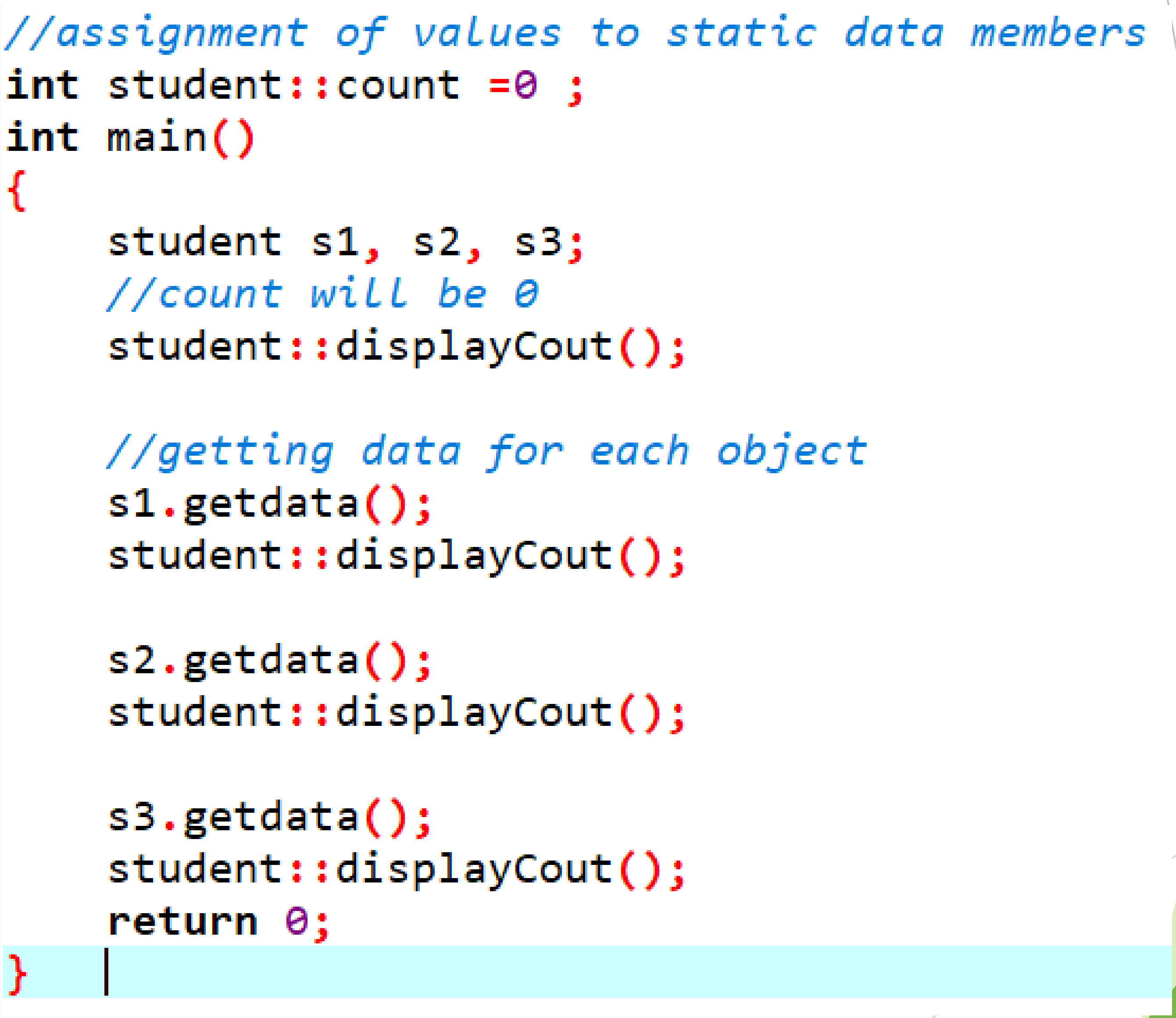
Contd..

- **Example:**



```
//program for counting number of objects
//static member function for displayinn count
#include<iostream>
using namespace std;
class student
{
    private:
        int roll;
        char name[20];
        static int count; //static data member
    public:
        void getdata()
        {
            cout<<"Enter roll number and name"<<endl;
            cin>>roll>>name;
            count++;
        }
        static void displayCout()
        {
            cout<<"count:"<<count<<endl;
        }

};
```

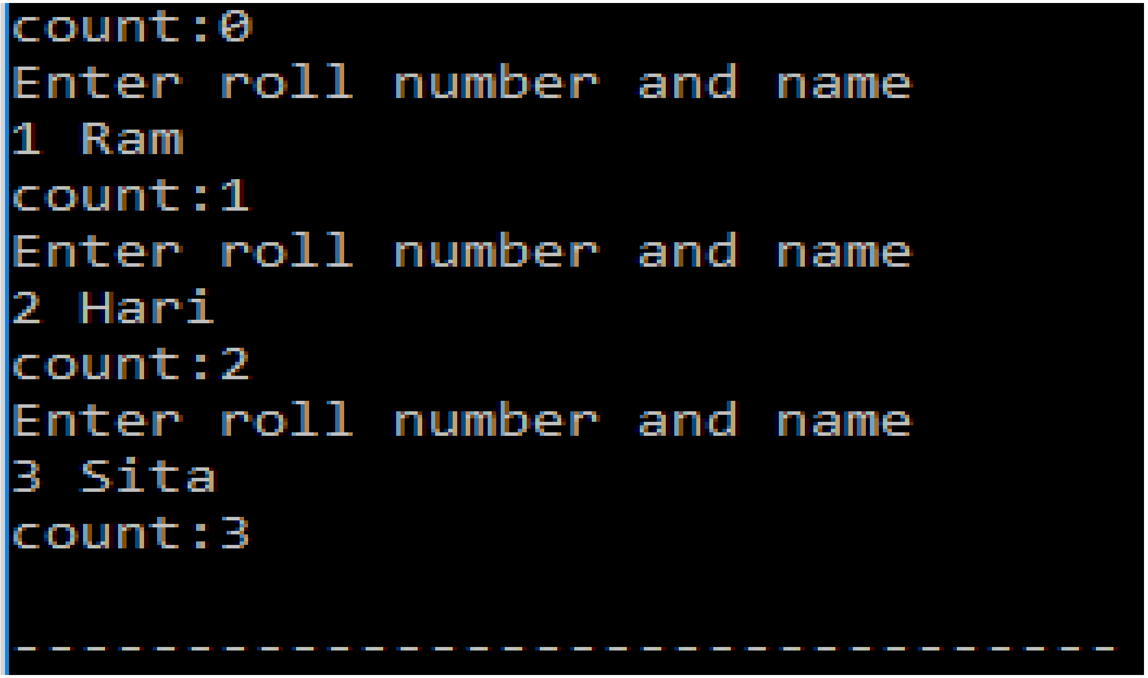
Contd..



```
//assignment of values to static data members
int student::count =0 ;
int main()
{
    student s1, s2, s3;
    //count will be 0
    student::displayCout();

    //getting data for each object
    s1.getdata();
    student::displayCout();

    s2.getdata();
    student::displayCout();

    s3.getdata();
    student::displayCout();
    return 0;
}
```

Contd..

- **Output:**



```
count:0
Enter roll number and name
1 Ram
count:1
Enter roll number and name
2 Hari
count:2
Enter roll number and name
3 Sita
count:3


--------------------------------
```

Objects as a function arguments

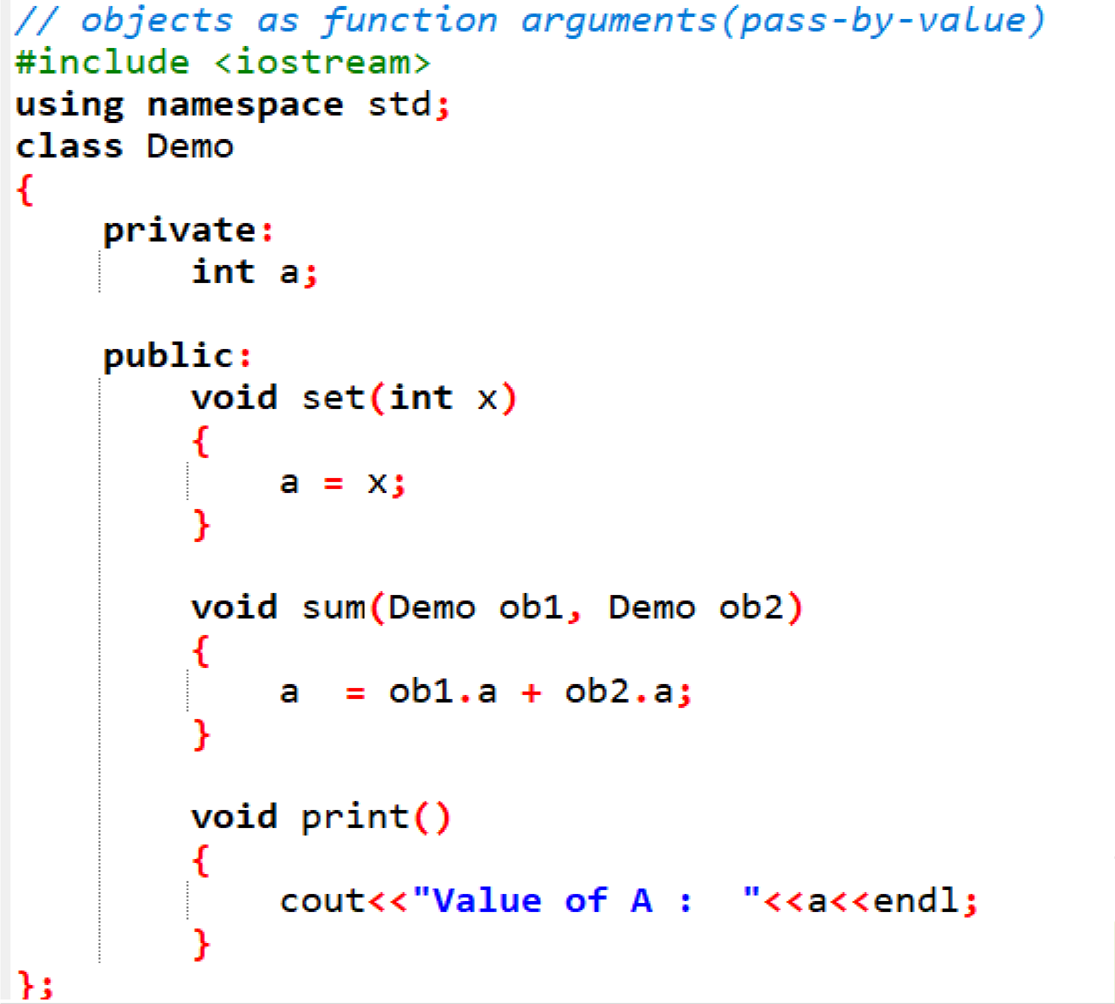
- Like any other variables, an object may be used as a function argument in three ways:
  - Pass- by-value
  - Pass-by-reference, and
  - Pass-by-pointer

Contd..

- **Pass by value:**
  - In pass by value a copy of the object is passed to the function. so, the changes made to the object inside the function do not affect the actual object.
  - since all values of objects need to be copied into arguments of method invoked, it makes program slower when larger objects are used.
  - An object can be passed by value to the function as below:

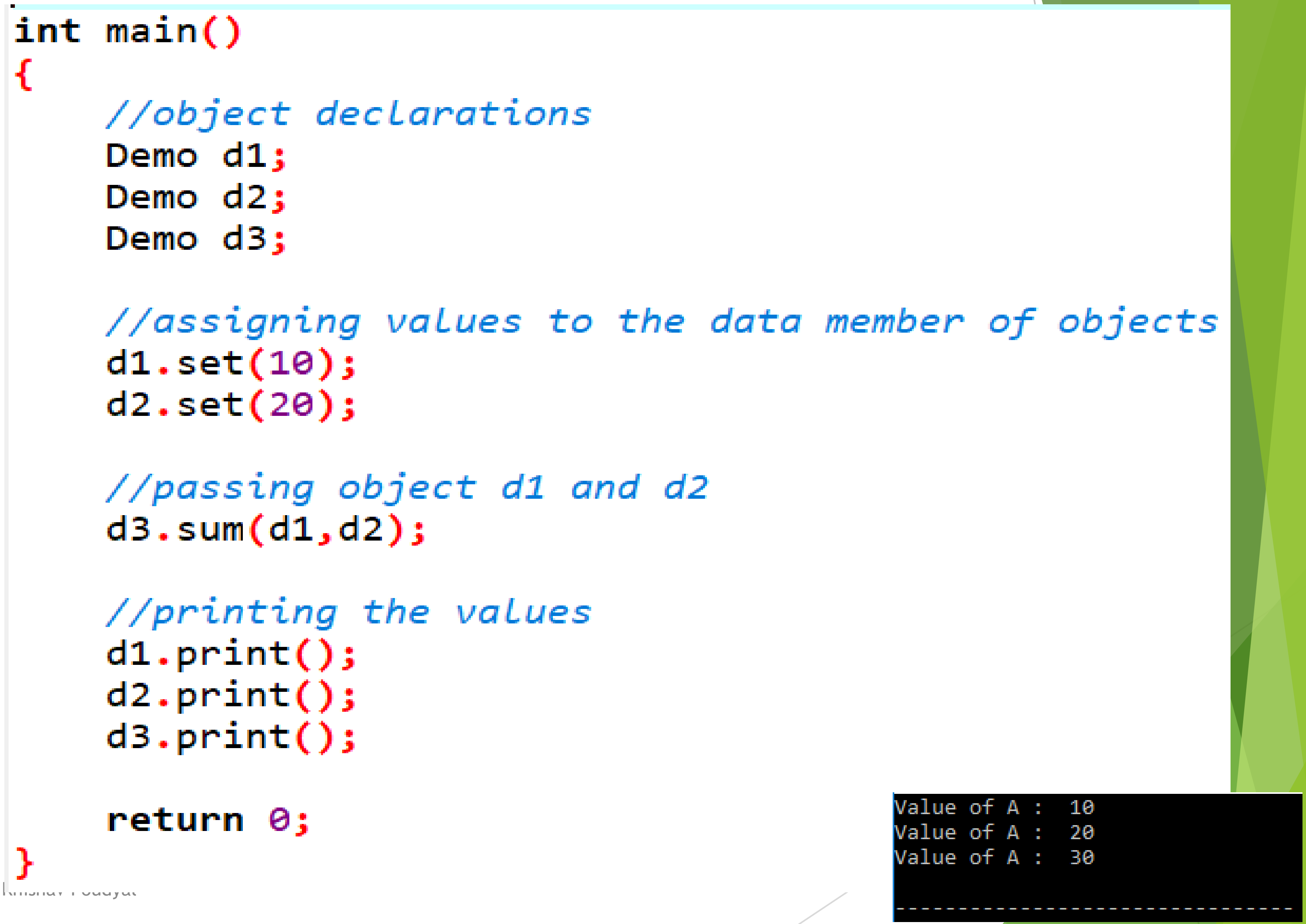
Contd..

- Example:



```
// objects as function arguments(pass-by-value)
#include <iostream>
using namespace std;
class Demo
{
    private:
        int a;

    public:
        void set(int x)
        {
            a = x;
        }

        void sum(Demo ob1, Demo ob2)
        {
            a  = ob1.a + ob2.a;
        }

        void print()
        {
            cout<<"Value of A :  "<<a<<endl;
        }
};
```

Contd..



```
int main()
{
    //object declarations
    Demo d1;
    Demo d2;
    Demo d3;

    //assigning values to the data member of objects
    d1.set(10);
    d2.set(20);

    //passing object d1 and d2
    d3.sum(d1,d2);

    //printing the values
    d1.print();
    d2.print();
    d3.print();

    return 0;
}
```

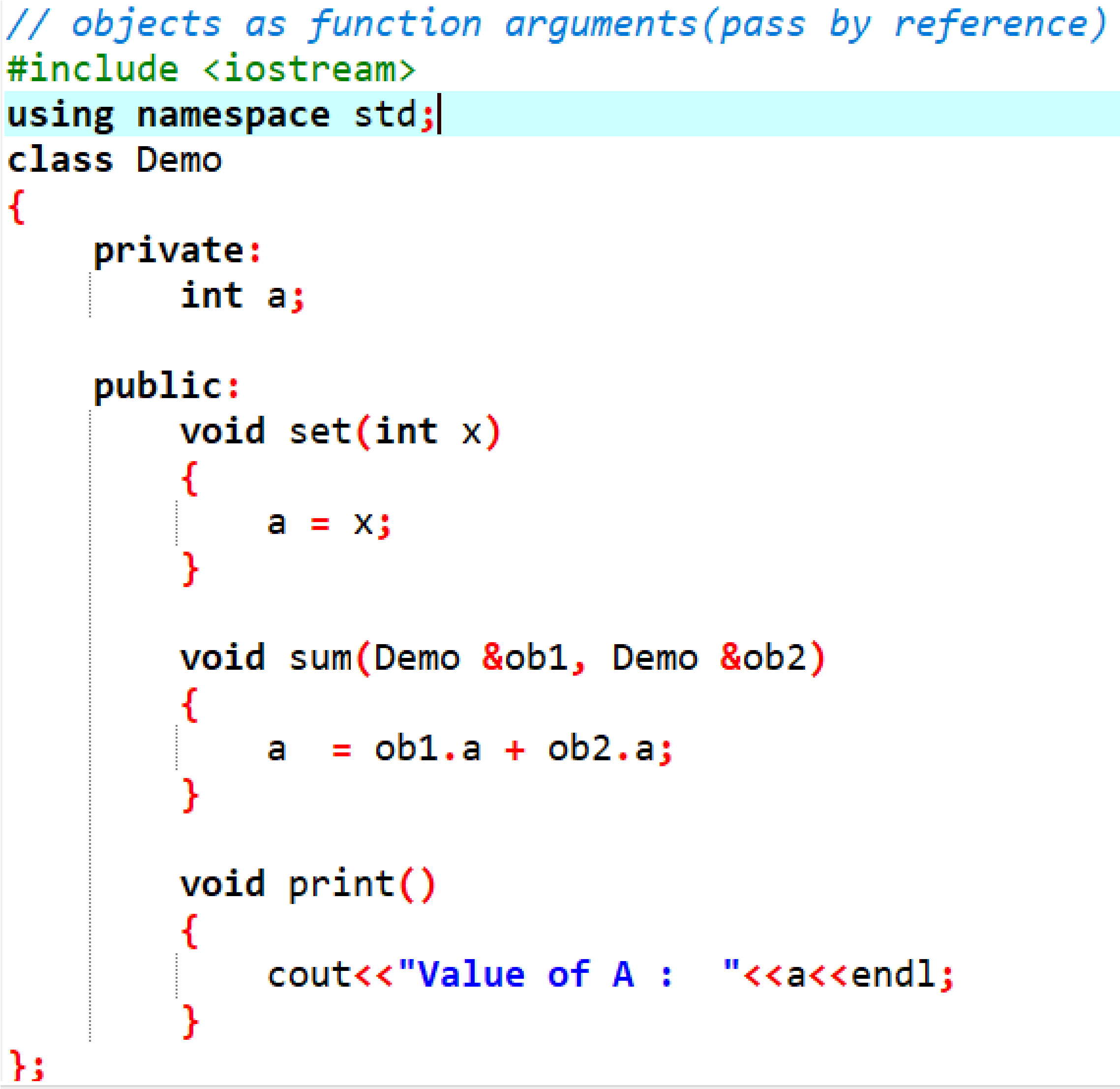
```
Value of A :   10
Value of A :   20
Value of A :   30

--------------------------------
```

Contd..

- **Pass-by-reference:**
  - In pass by reference, an address of the object is passed to the function, it makes program faster when using larger objects.
  - The function works directly on the actual object used in the function call.
  - So any changes made to object inside the function will reflect in the actual object.

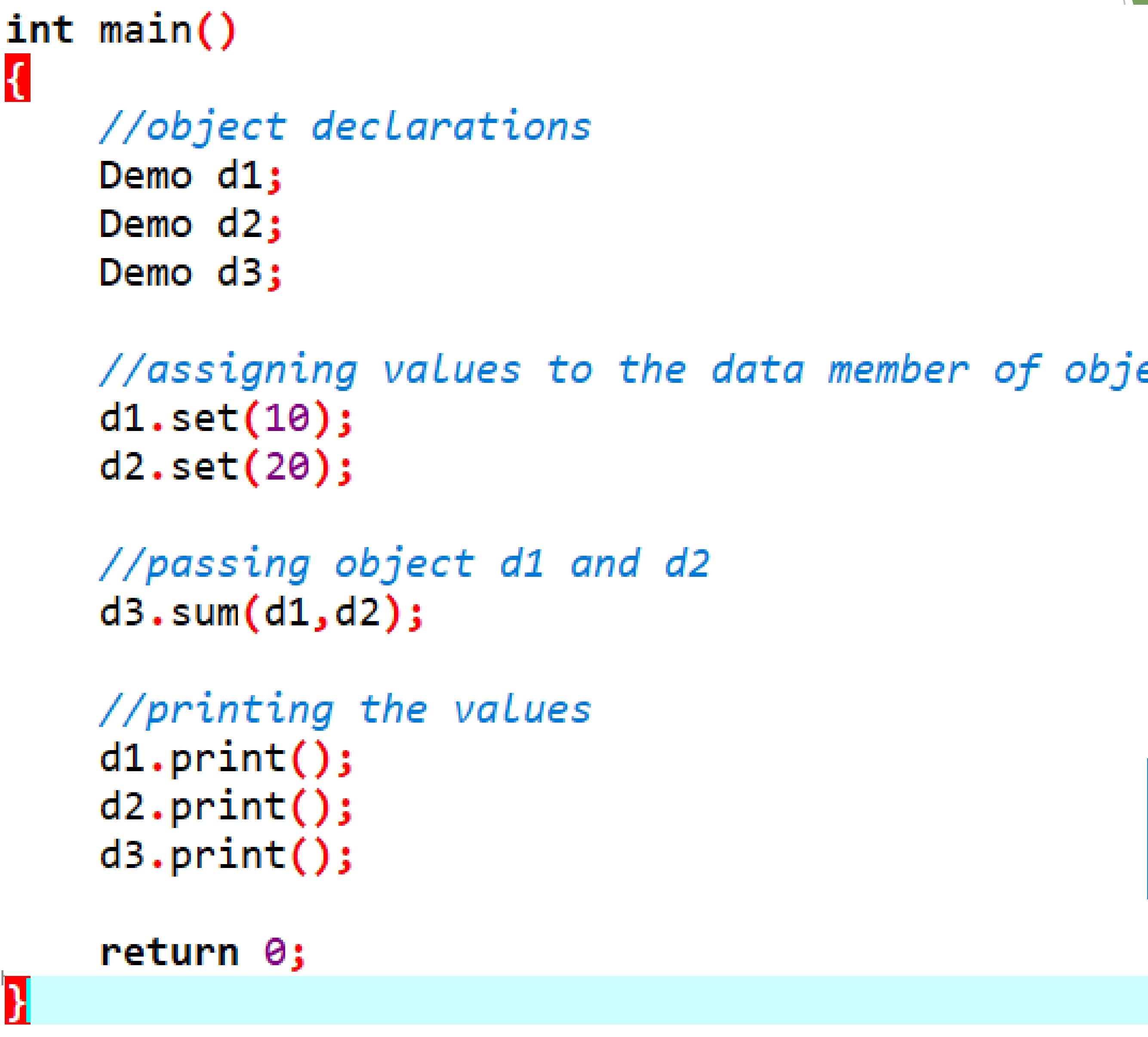
Contd..

- **Example:**

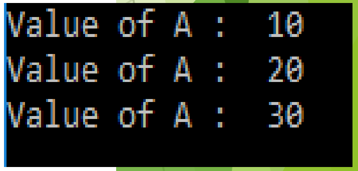


```
// objects as function arguments(pass by reference)
#include <iostream>
using namespace std;
class Demo
{
    private:
        int a;

    public:
        void set(int x)
        {
            a = x;
        }

        void sum(Demo &ob1, Demo &ob2)
        {
            a  = ob1.a + ob2.a;
        }

        void print()
        {
            cout<<"Value of A :  "<<a<<endl;
        }
};
```

Contd..



```
int main()
{
    //object declarations
    Demo d1;
    Demo d2;
    Demo d3;

    //assigning values to the data member of objects
    d1.set(10);
    d2.set(20);

    //passing object d1 and d2
    d3.sum(d1,d2);

    //printing the values
    d1.print();
    d2.print();
    d3.print();

    return 0;
}
```

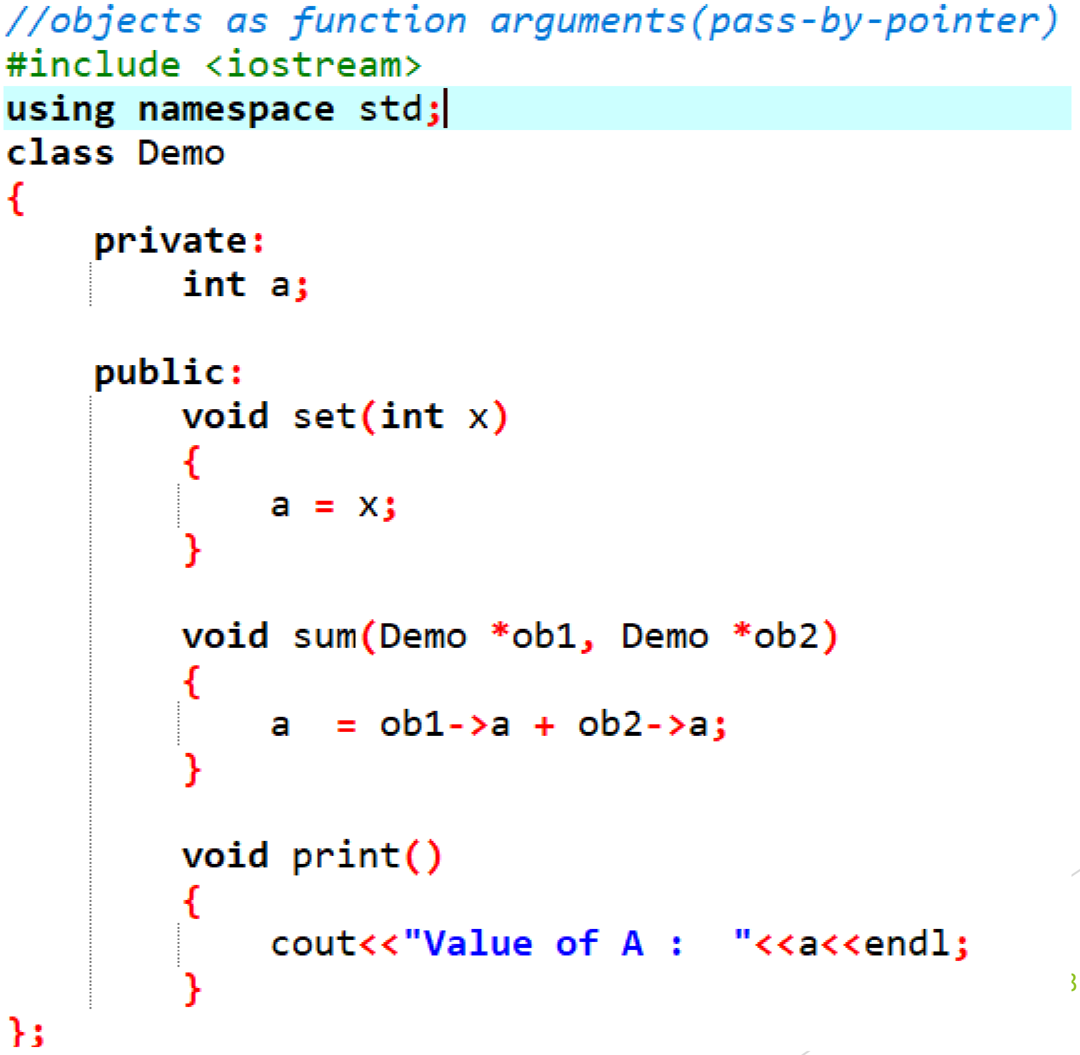


```
Value of A :  10
Value of A :  20
Value of A :  30
```

Contd..

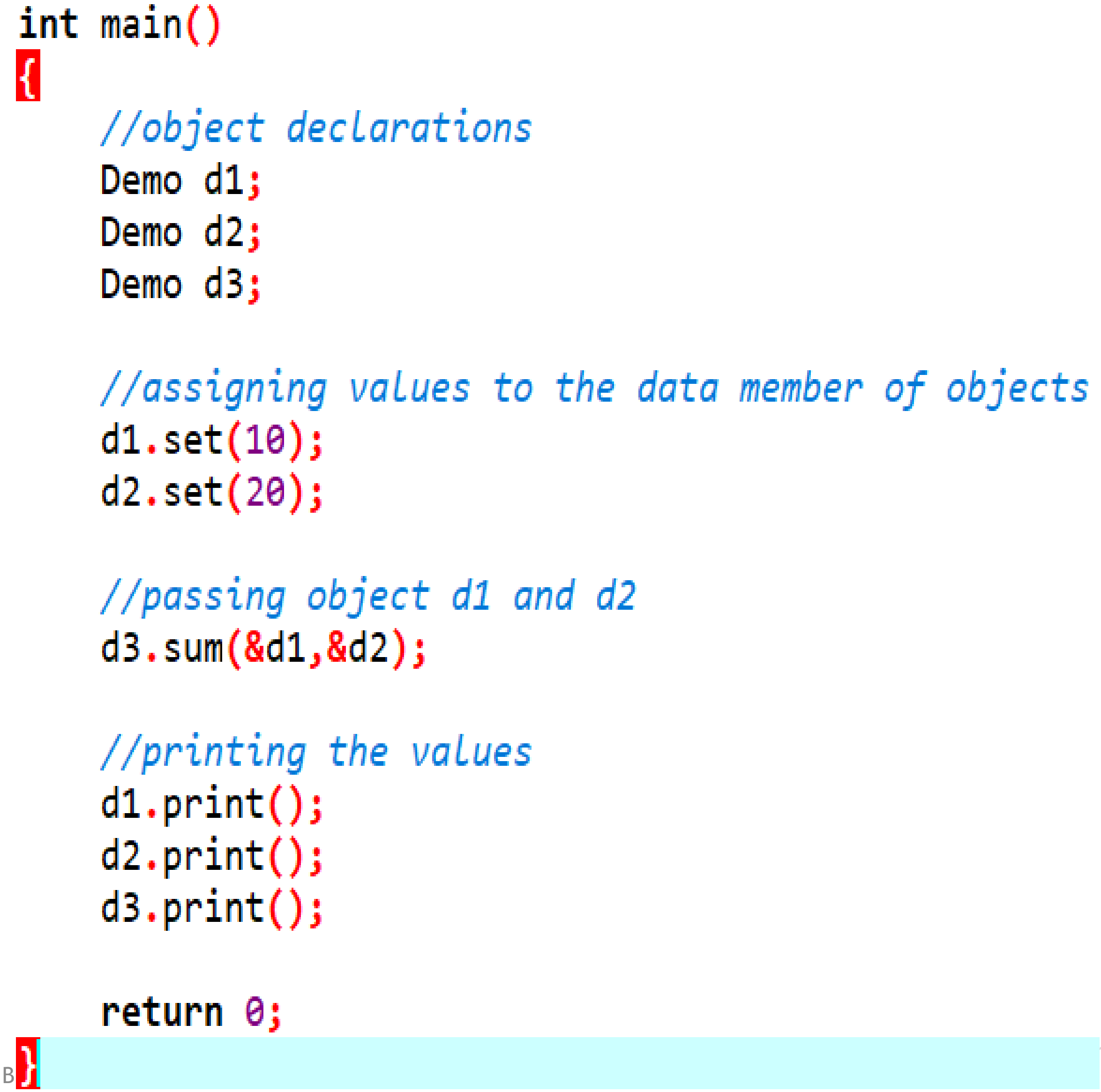
Pass-by-pointer:

- Also works directlv on the actual obiect used in the function call.

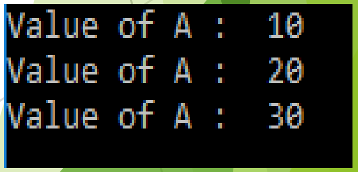


```
//objects as function arguments(pass-by-pointer)
#include <iostream>
using namespace std;
class Demo
{
    private:
        int a;

    public:
        void set(int x)
        {
            a = x;
        }

        void sum(Demo *ob1, Demo *ob2)
        {
            a  = ob1->a + ob2->a;
        }

        void print()
        {
            cout<<"Value of A :  "<<a<<endl;
        }
};
```

Contd..



```
int main()
{
    //object declarations
    Demo d1;
    Demo d2;
    Demo d3;

    //assigning values to the data member of objects
    d1.set(10);
    d2.set(20);

    //passing object d1 and d2
    d3.sum(&d1,&d2);

    //printing the values
    d1.print();
    d2.print();
    d3.print();

    return 0;
}
```



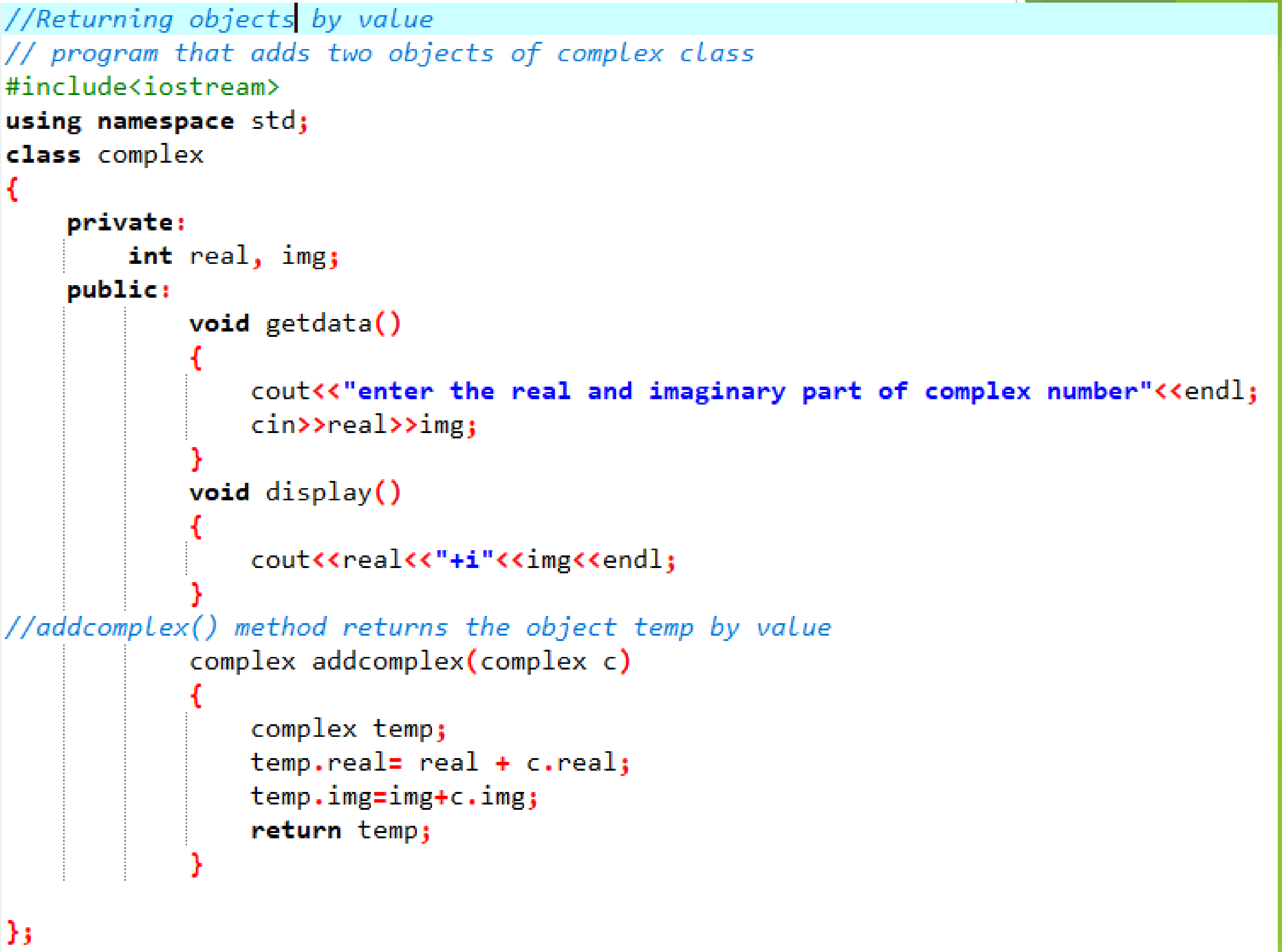
```
Value of A :  10
Value of A :  20
Value of A :  30
```

Returning objects from the functions

- A function not only receives objects as arguments but also can return them.
- A function can return objects also in three ways:
  - Return-by- value
  - Return-by- reference, and
  - Return-by-pointer.

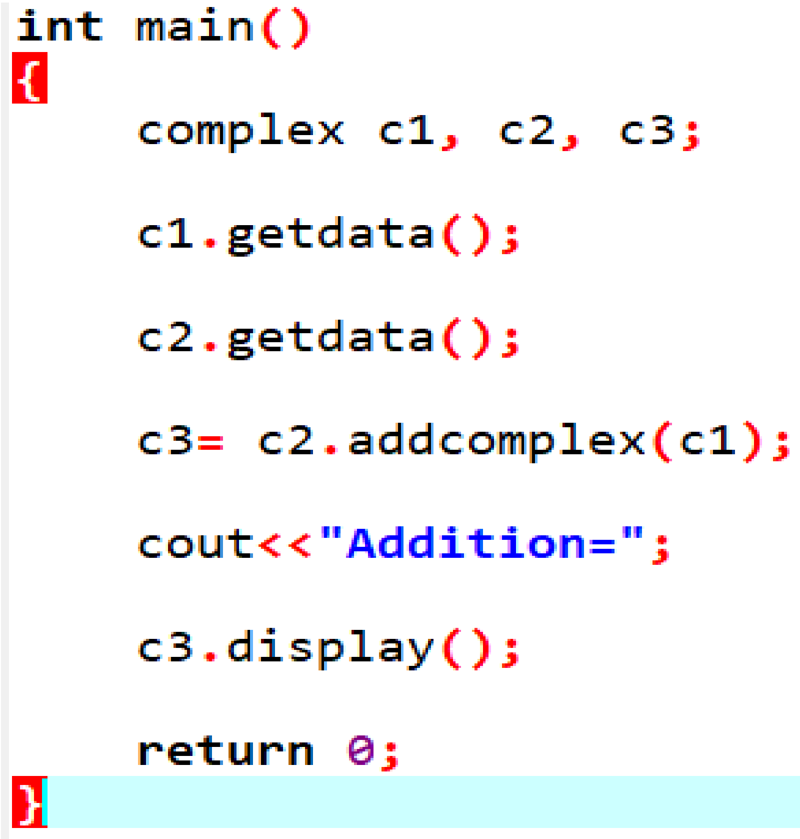
Contd..

- **Return-by-value:** A copy of the object is returned to the function call.

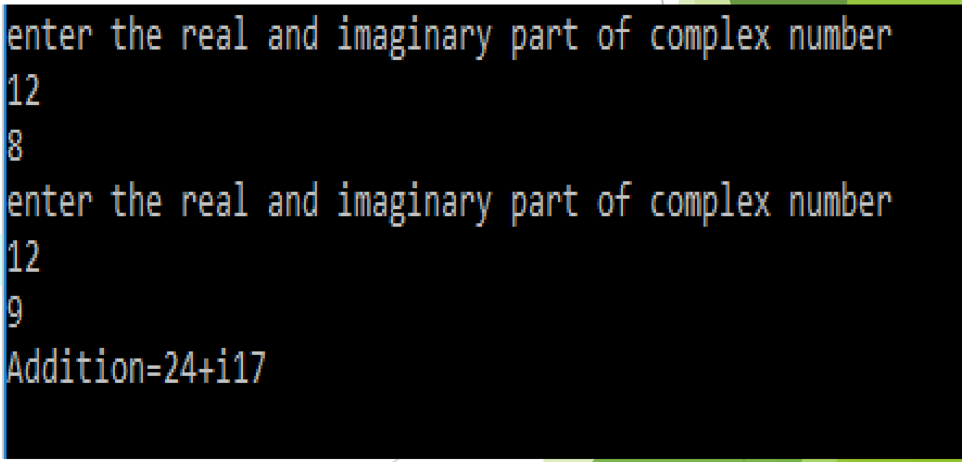


```
//Returning objects by value
// program that adds two objects of complex class
#include<iostream>
using namespace std;
class complex
{
    private:
        int real, img;
    public:
            void getdata()
            {
                cout<<"enter the real and imaginary part of complex number"<<endl;
                cin>>real>>img;
            }
            void display()
            {
                cout<<real<<"+i"<<img<<endl;
            }
//addcomplex() method returns the object temp by value
            complex addcomplex(complex c)
            {
                complex temp;
                temp.real= real + c.real;
                temp.img=img+c.img;
                return temp;
            }

};
```

Contd..



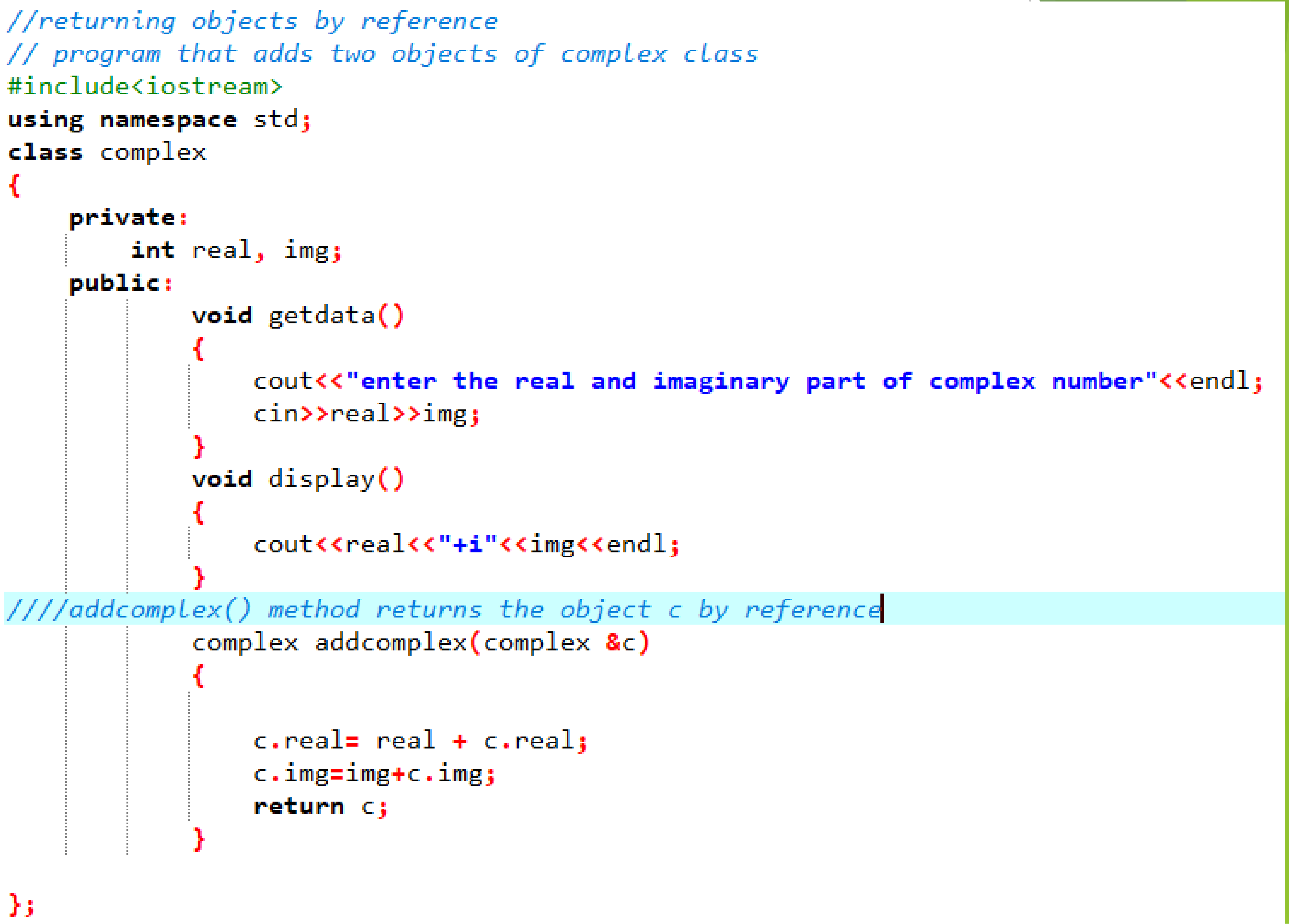
```
int main()
{
    complex c1, c2, c3;
    c1.getdata();
    c2.getdata();
    c3= c2.addcomplex(c1);
    cout<<"Addition=";
    c3.display();
    return 0;
}
```



```
enter the real and imaginary part of complex number
12
8
enter the real and imaginary part of complex number
12
9
Addition=24+i17
```

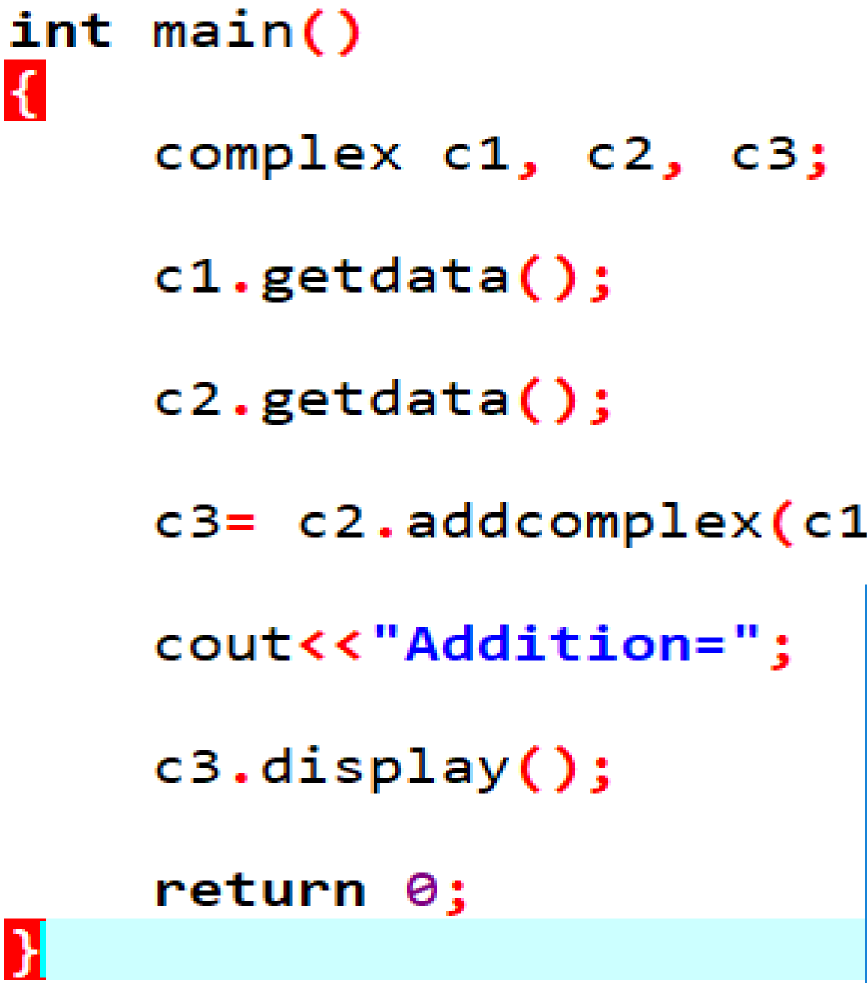
Contd..

- **Return-by-reference:** An address of the object is returned to the function call.

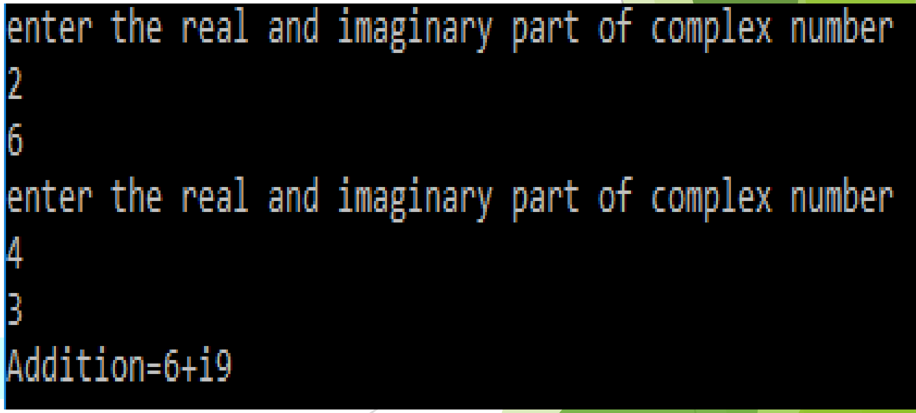


```
//returning objects by reference
// program that adds two objects of complex class
#include<iostream>
using namespace std;
class complex
{
    private:
        int real, img;
    public:
            void getdata()
            {
                cout<<"enter the real and imaginary part of complex number"<<endl;
                cin>>real>>img;
            }
            void display()
            {
                cout<<real<<"+i"<<img<<endl;
            }
////addcomplex() method returns the object c by reference
            complex addcomplex(complex &c)
            {

                c.real= real + c.real;
                c.img=img+c.img;
                return c;
            }

};
```

Contd..



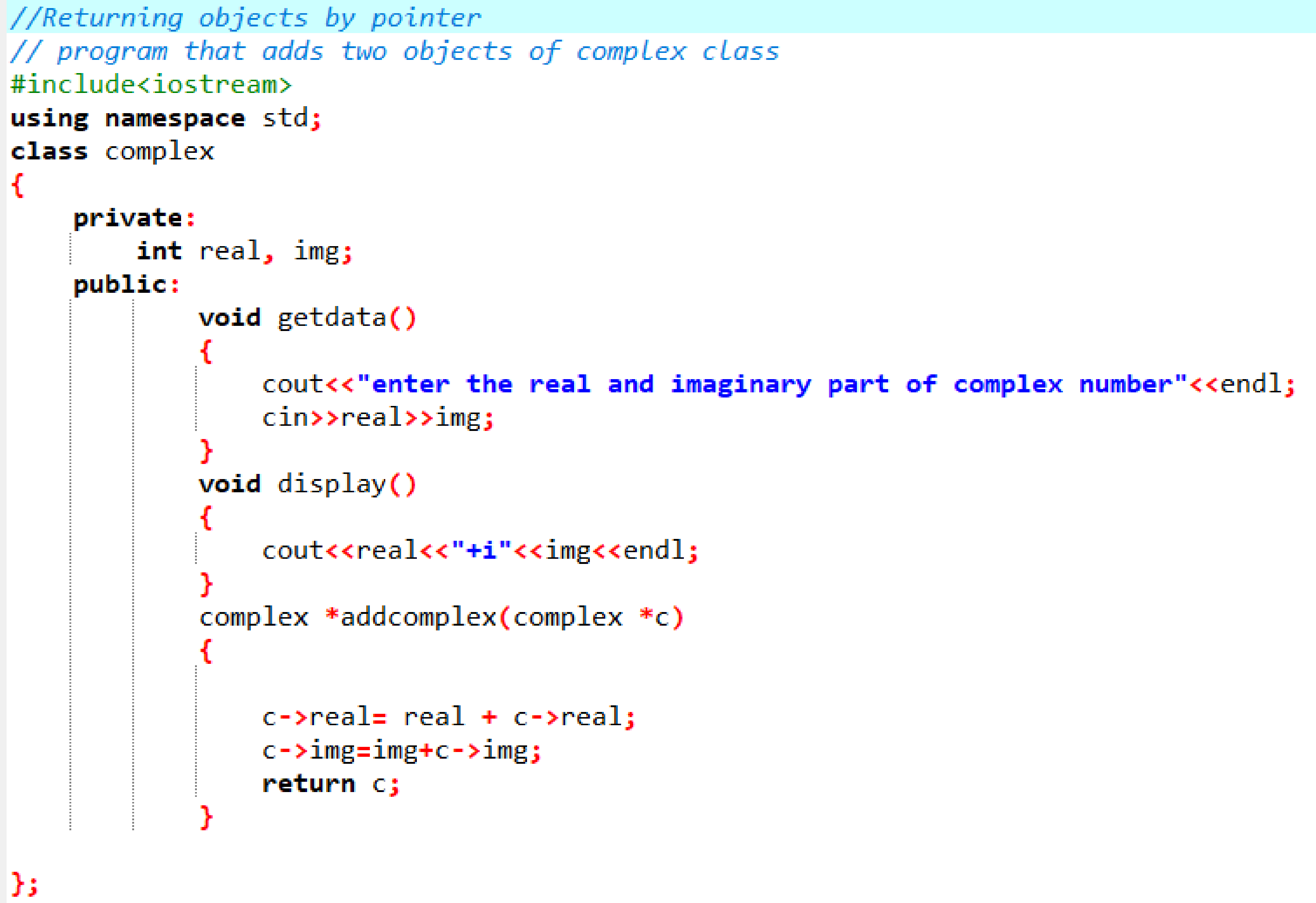
```
int main()
{
	complex c1, c2, c3;
	c1.getdata();
	c2.getdata();
	c3= c2.addcomplex(c1);
	cout<<"Addition=";
	c3.display();
	return 0;
}
```



```
enter the real and imaginary part of complex number
2
6
enter the real and imaginary part of complex number
4
3
Addition=6+i9
```

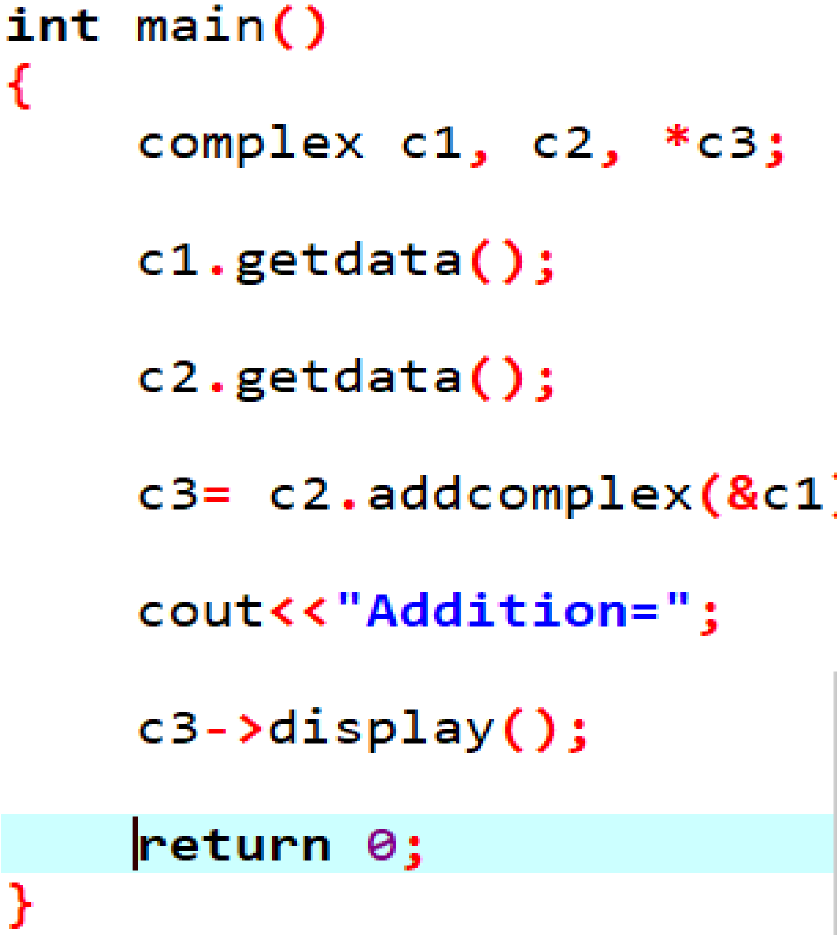
Contd.

Return-by-pointer: It also returns address of the object to the function call.

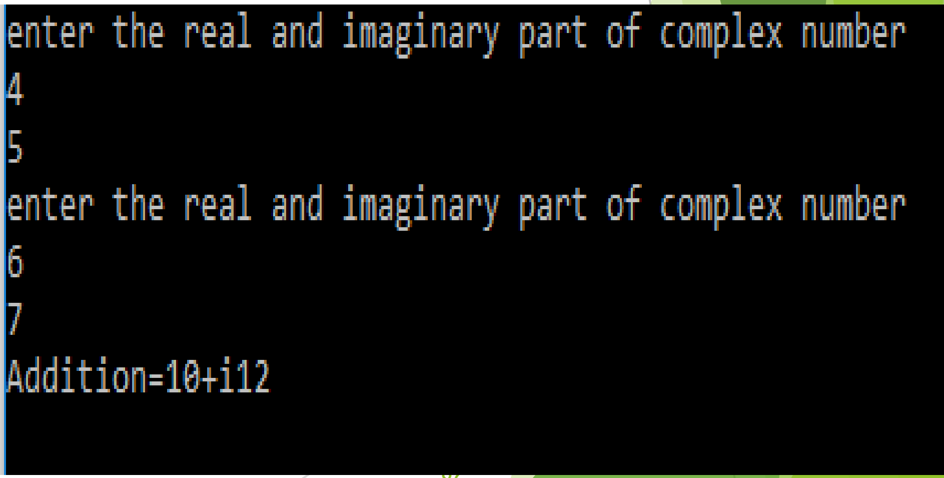


```
//Returning objects by pointer
// program that adds two objects of complex class
#include<iostream>
using namespace std;
class complex
{
    private:
        int real, img;
    public:
            void getdata()
            {
                cout<<"enter the real and imaginary part of complex number"<<endl;
                cin>>real>>img;
            }
            void display()
            {
                cout<<real<<"+i"<<img<<endl;
            }
            complex *addcomplex(complex *c)
            {
                
                c->real= real + c->real;
                c->img=img+c->img;
                return c;
            }

};
```

Contd..



```
int main()
{
    complex c1, c2, *c3;

    c1.getdata();

    c2.getdata();

    c3= c2.addcomplex(&c1);

    cout<<"Addition=";

    c3->display();

    return 0;
}
```



```
enter the real and imaginary part of complex number
4
5
enter the real and imaginary part of complex number
6
7
Addition=10+i12
```

Home Work

- What do you mean by object and class? List the differences between structure and class.
- How data hiding can be achieved in class?
- What are the different ways of defining member functions? Which one of them is better way and why?
- Write a program to read and display 10 objects of item class containing data member item, name ,code and price.
- Describe the different methods of returning objects from function with their merits and demerits.
- Write a program designing a class to represent a bank account. Include the following members
  - Data members: name of depositor, account number, type of account
  - Member functions: To assign initial values, To display name and account type

Home Work

- Write a meaningful program to illustrate the use of static data member and static function members.
- Construct a class named intamount with the following members to calculate interest and amount separately on the basis of given principle , rate and time.
  - Data members: principle, time, rate
  - Function members: getdata, interest, amount.
- Define a class circle with the following data members and member functions:
  - Data members: radius and area
  - Member functions: getdata, calcArea and display.

Thank You !