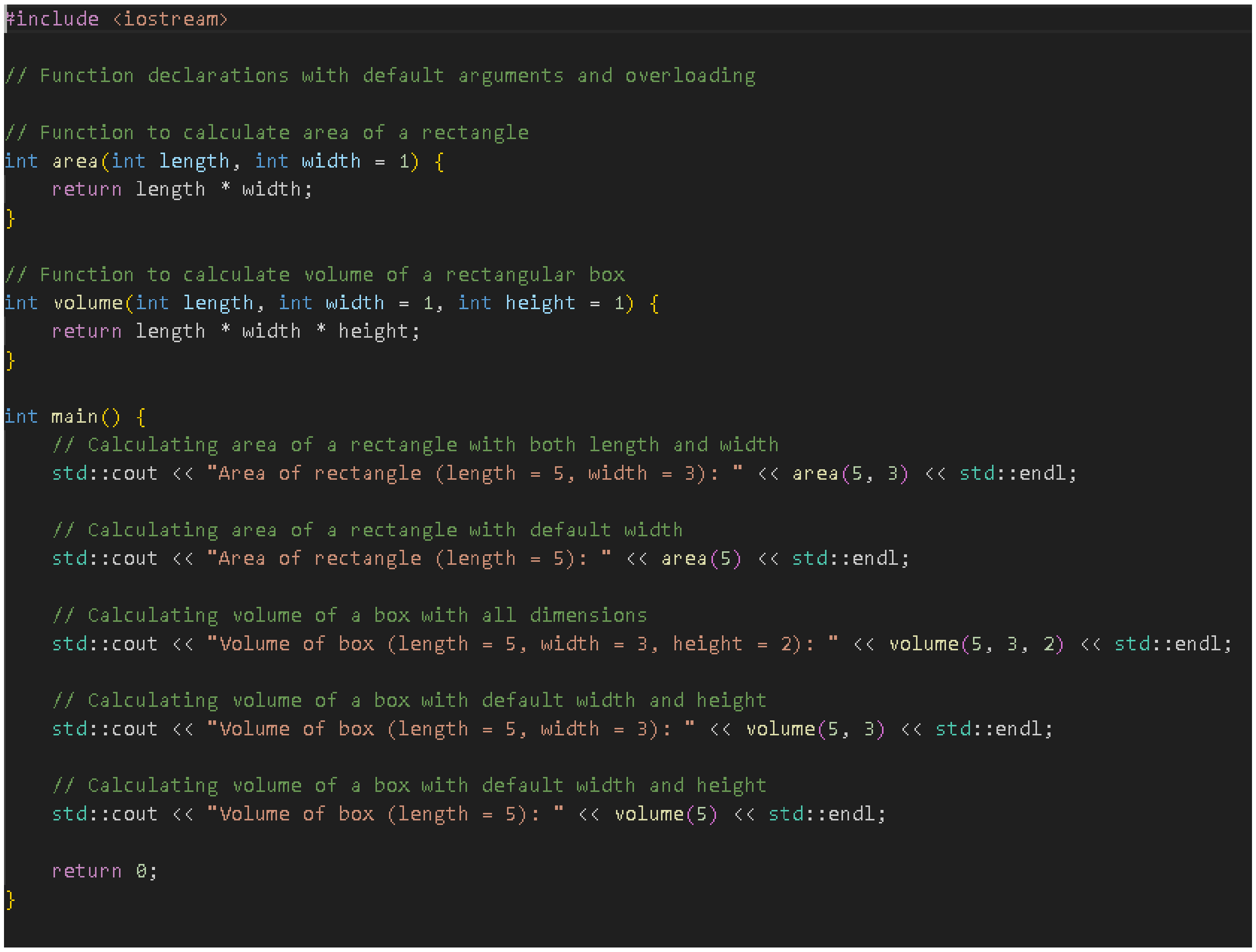
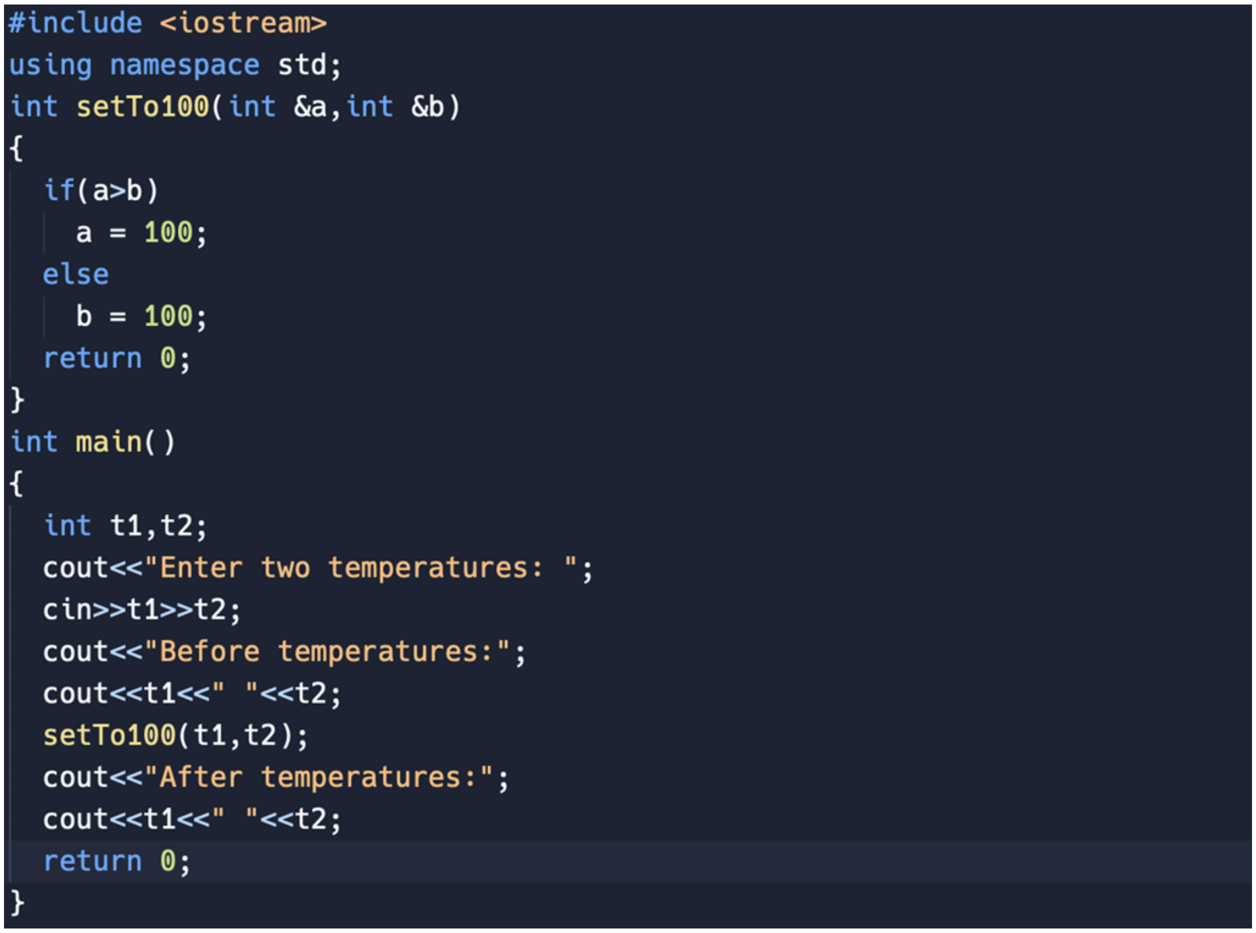
21). WAP to relate function overloading and default arguments in the same program.



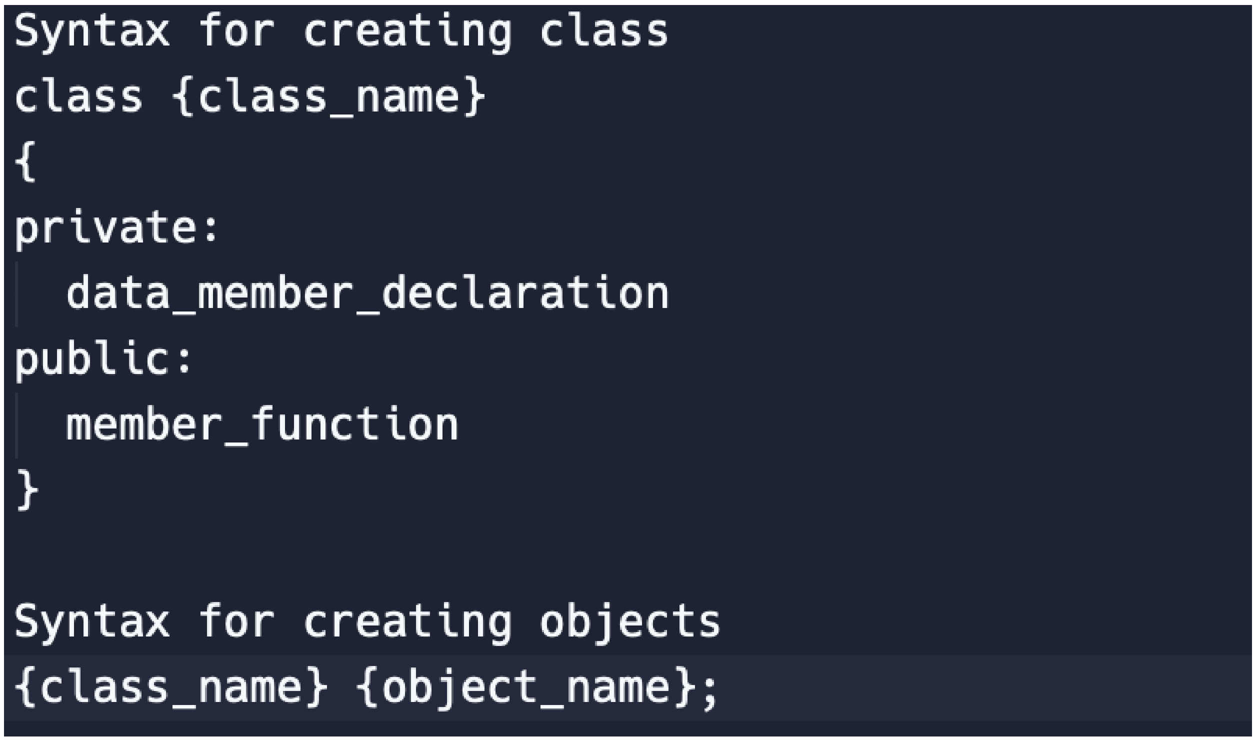
```
#include <iostream>

// Function declarations with default arguments and overloading

// Function to calculate area of a rectangle
int area(int length, int width = 1) {
    return length * width;
}

// Function to calculate volume of a rectangular box
int volume(int length, int width = 1, int height = 1) {
    return length * width * height;
}

int main() {
    // Calculating area of a rectangle with both length and width
    std::cout << "Area of rectangle (length = 5, width = 3): " << area(5, 3) << std::endl;

    // Calculating area of a rectangle with default width
    std::cout << "Area of rectangle (length = 5): " << area(5) << std::endl;

    // Calculating volume of a box with all dimensions
    std::cout << "Volume of box (length = 5, width = 3, height = 2): " << volume(5, 3, 2) << std::endl;

    // Calculating volume of a box with default width and height
    std::cout << "Volume of box (length = 5, width = 3): " << volume(5, 3) << std::endl;

    // Calculating volume of a box with default width and height
    std::cout << "Volume of box (length = 5): " << volume(5) << std::endl;

    return 0;
}
```

22). Write a program that creates a function that passes two temperatures by reference and sets the larger of the two temperatures to 100 by return by reference.



```
#include <iostream>
using namespace std;
int setTo100(int &a,int &b)
{
  if(a>b)
    a = 100;
  else
    b = 100;
  return 0;
}
int main()
{
  int t1,t2;
  cout<<"Enter two temperatures: ";
  cin>>t1>>t2;
  cout<<"Before temperatures:";
  cout<<t1<<" "<<t2;
  setTo100(t1,t2);
  cout<<"After temperatures:";
  cout<<t1<<" "<<t2;
  return 0;
}
```

23). Write syntax for class and object. Explain briefly about data specifiers.



```
Syntax for creating class
class {class_name}
{
private:
  data_member_declaration
public:
  member_function
}

Syntax for creating objects
{class_name} {object_name};
```

Data Access Specifiers

Data access specifiers control the visibility and accessibility of class members (data members and member functions) within a program. They are essential for implementing data hiding and encapsulation in object-oriented programming.

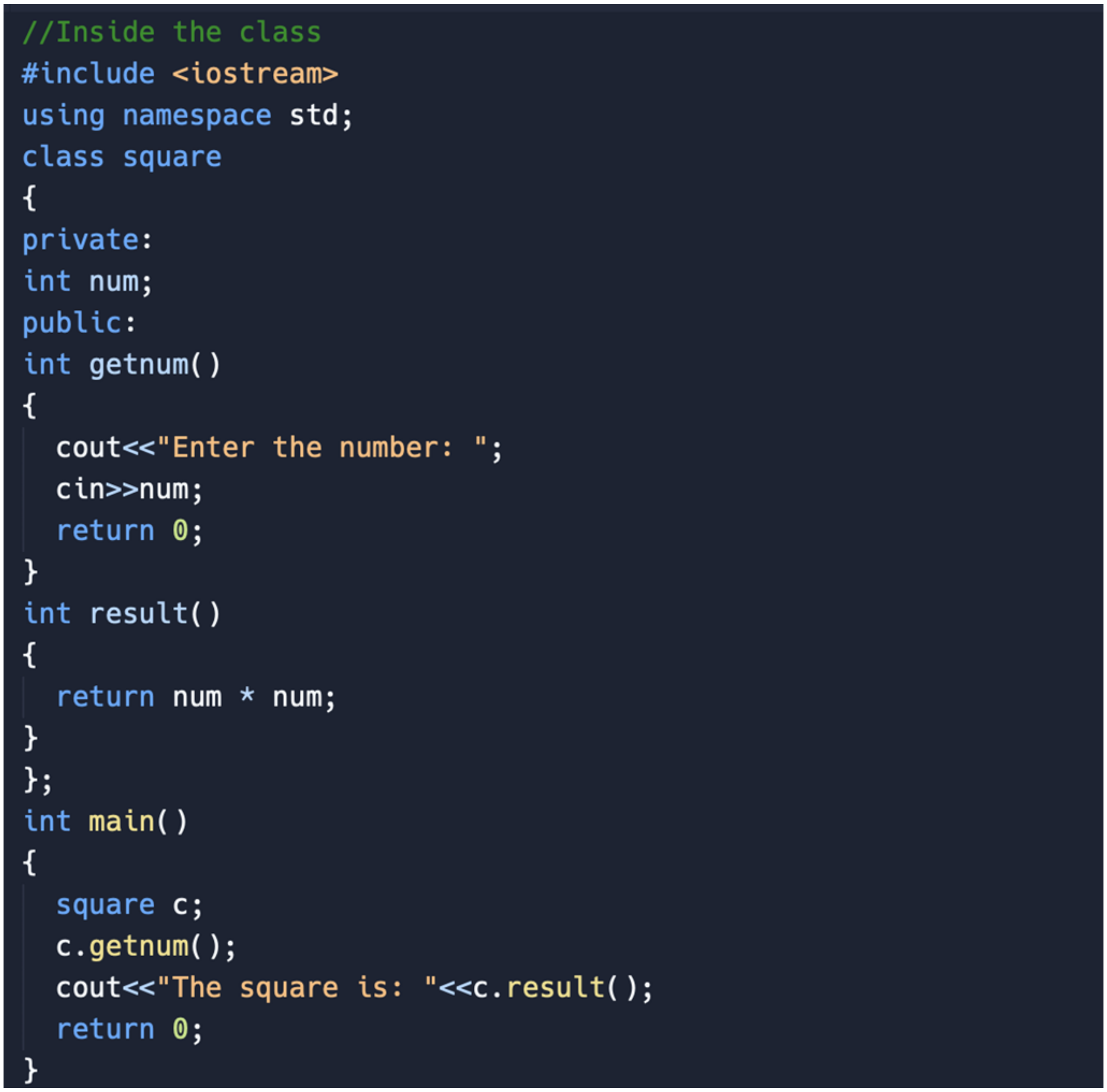
Types of Access Specifiers

- **Public:** Members declared as public are accessible from anywhere in the program.
- **Private:** Members declared as private are only accessible within the class itself.
- **Protected:** Members declared as protected are accessible within the class and its derived classes.

Importance of Access Specifiers

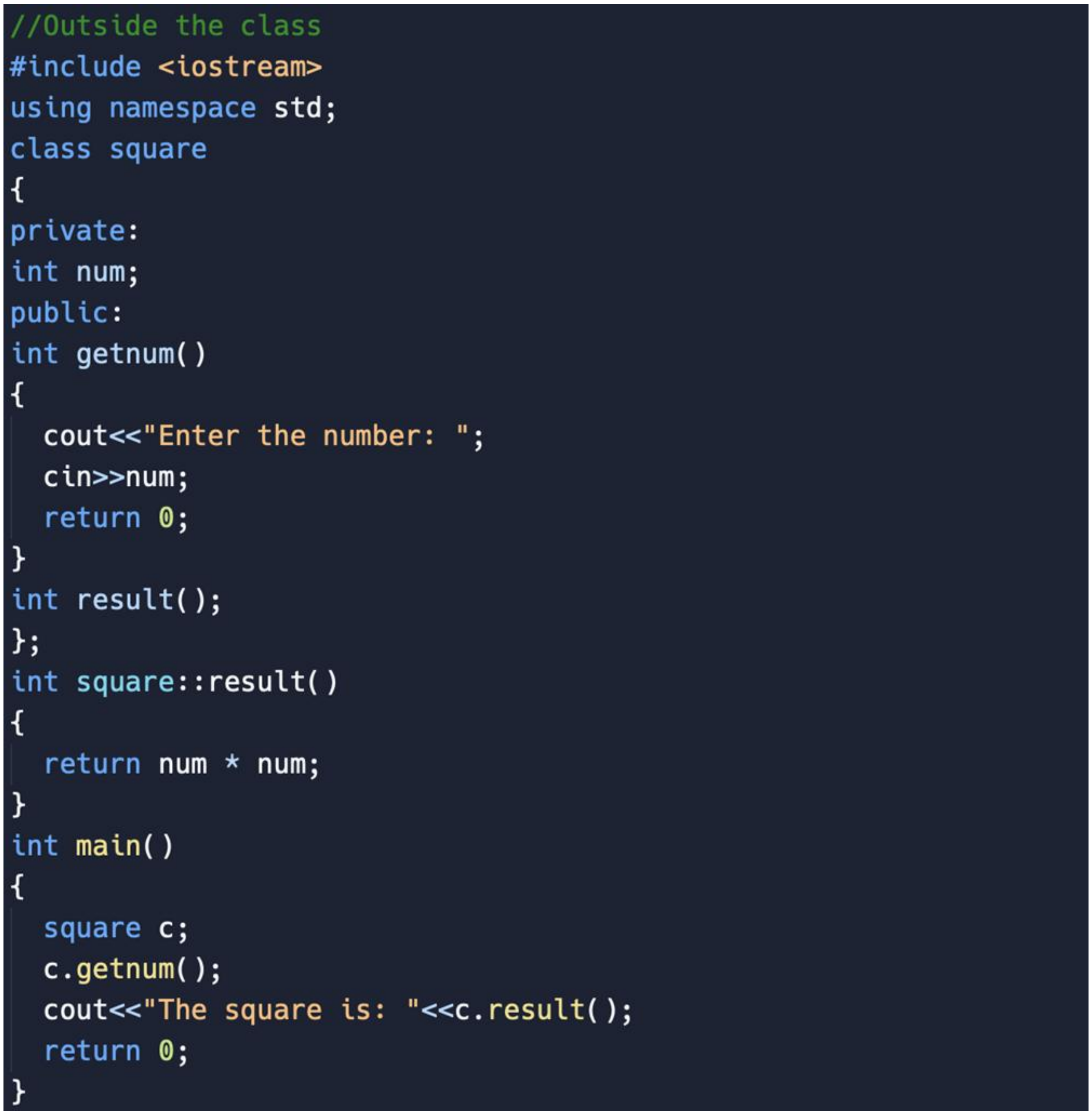
- **Data Hiding:** Protect sensitive data by making it private.
- **Encapsulation:** Combine data and functions into a single unit, controlling access to data.
- **Code Reusability:** Create reusable code by using inheritance and protected members.
- **Security:** Prevent unauthorized access to data.

24). Write a program showing member function definition (i) inside the class



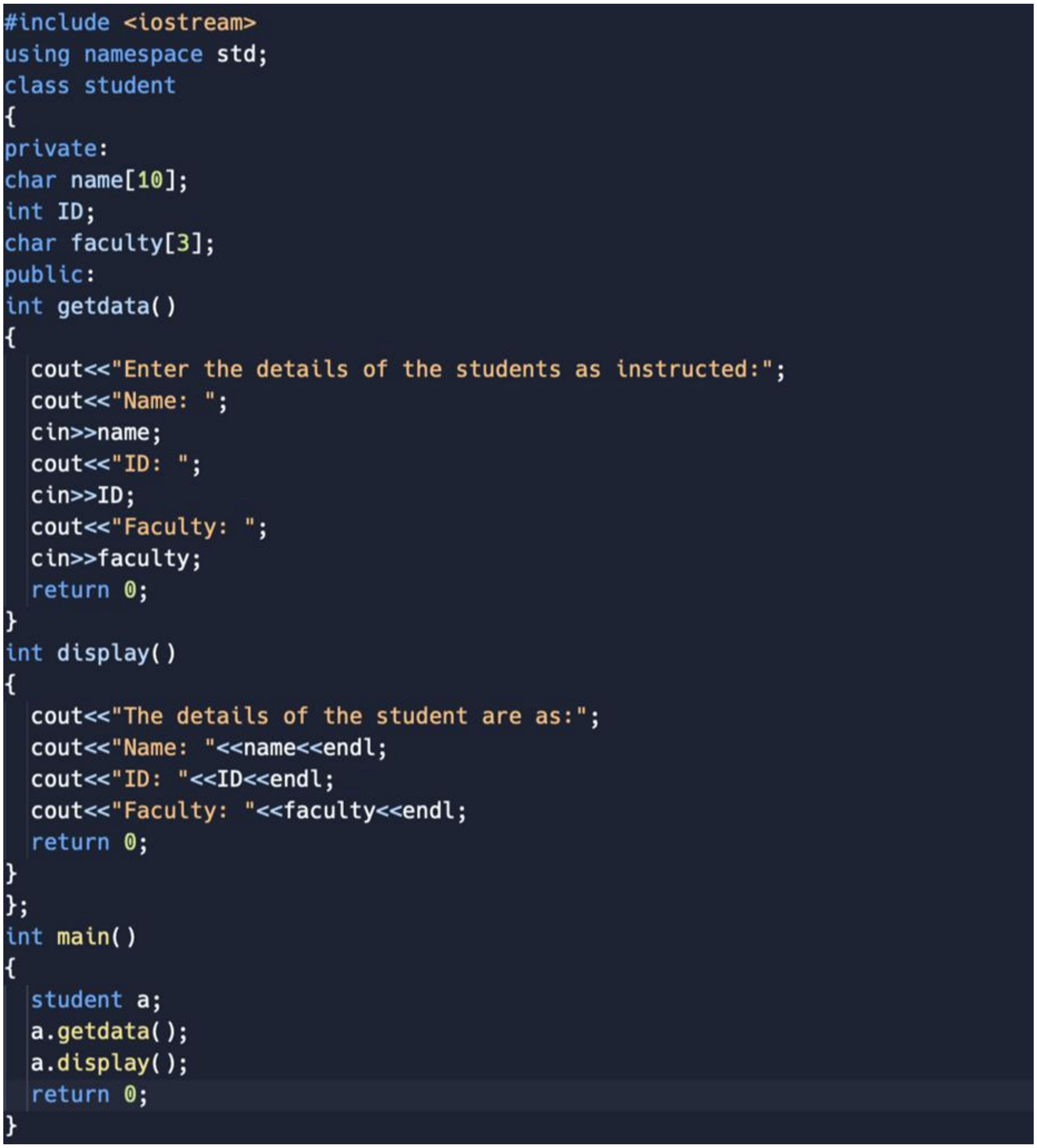
```cpp
//Inside the class
#include <iostream>
using namespace std;
class square
{
private:
int num;
public:
int getnum()
{
   cout<<"Enter the number: ";
   cin>>num;
   return 0;
}
int result()
{
   return num * num;
}
};
int main()
{
   square c;
   c.getnum();
   cout<<"The square is: "<<c.result();
   return 0;
}
```

(ii) outside the class



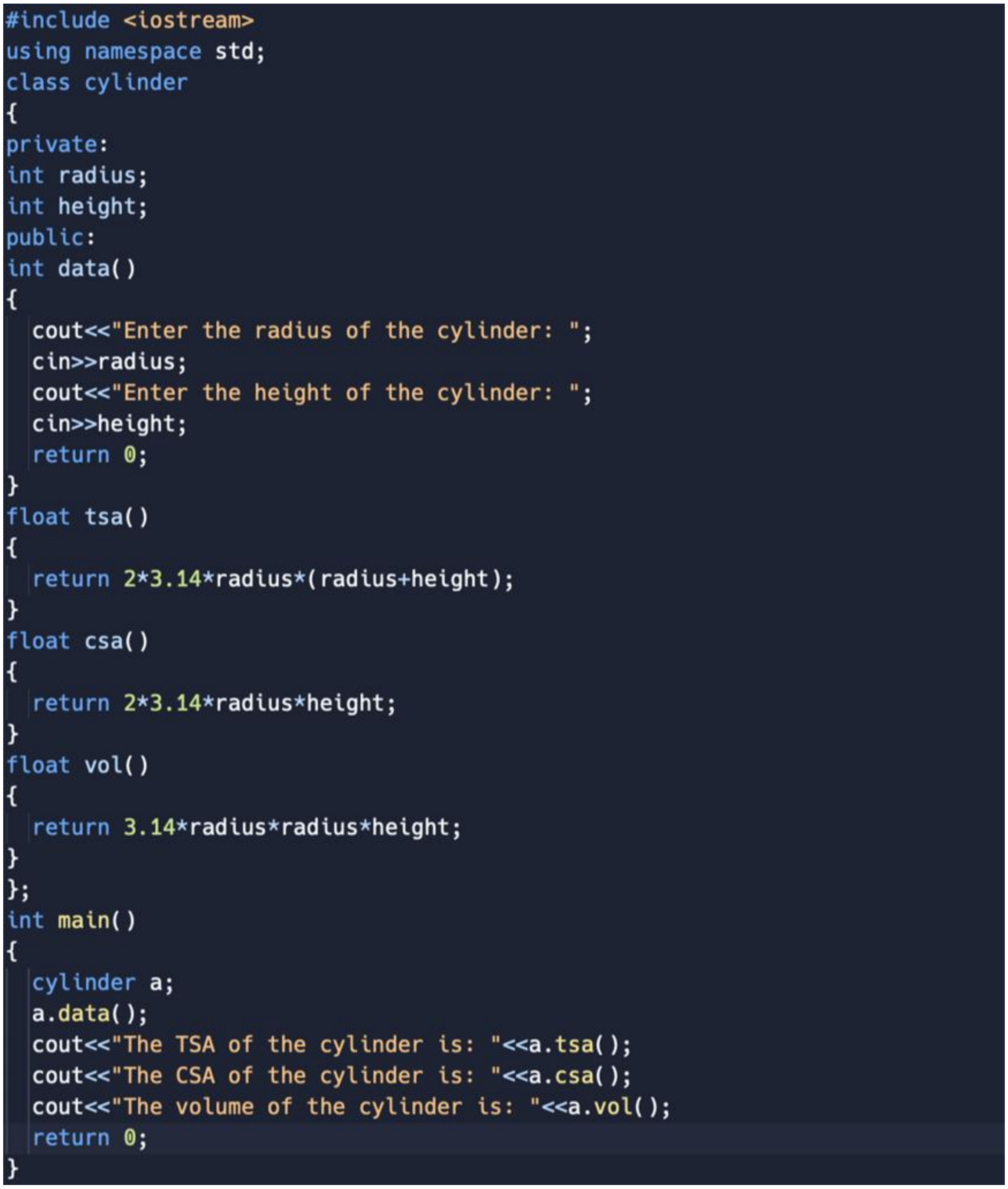
```
//Outside the class
#include <iostream>
using namespace std;
class square
{
private:
int num;
public:
int getnum()
{
  cout<<"Enter the number: ";
  cin>>num;
  return 0;
}
int result();
};
int square::result()
{
  return num * num;
}
int main()
{
  square c;
  c.getnum();
  cout<<"The square is: "<<c.result();
  return 0;
}
```

25). Create a class called student with data member name, ID and faculty. Use the necessary member function to read the data member from the user and display the details.



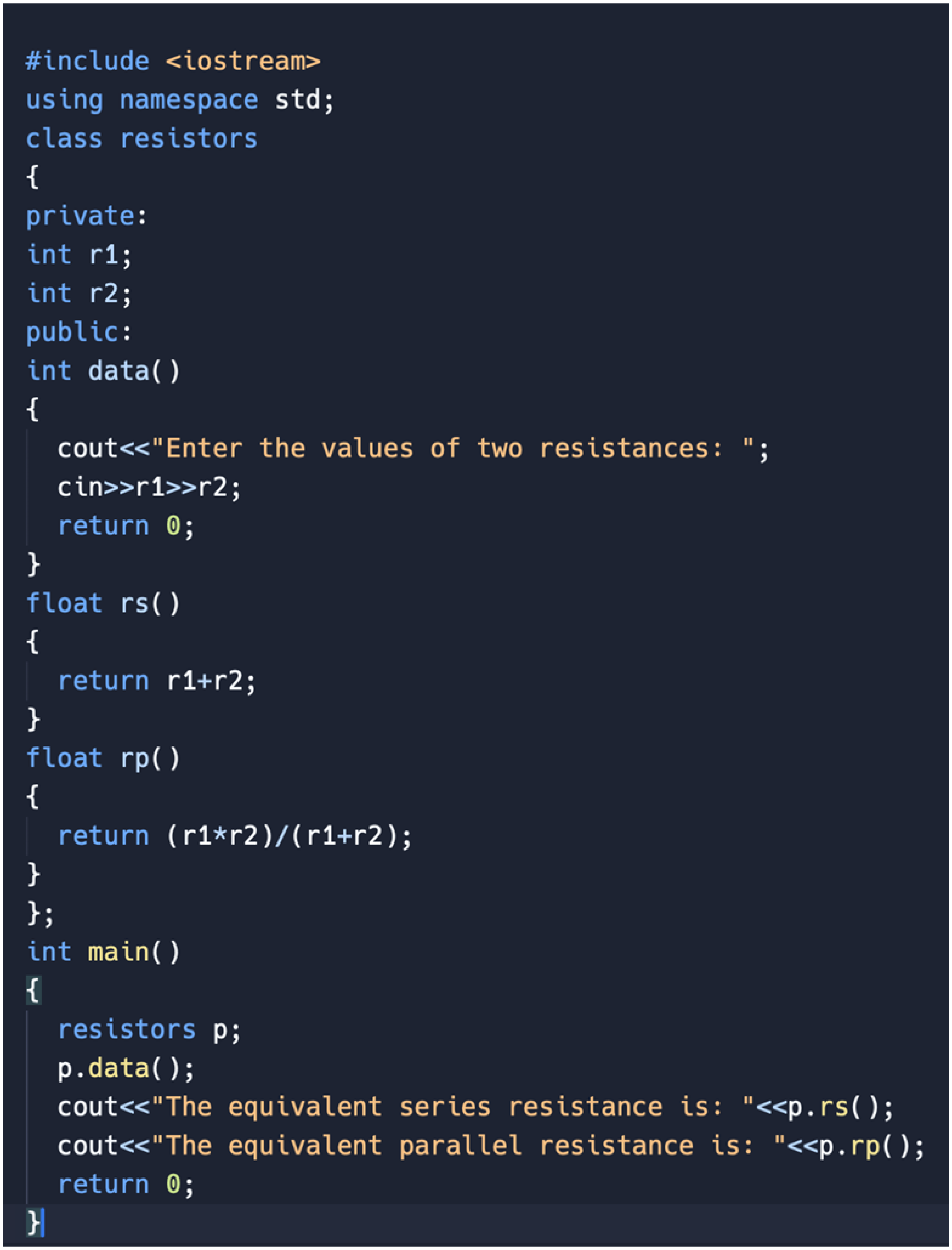
```
#include <iostream>
using namespace std;
class student
{
private:
char name[10];
int ID;
char faculty[3];
public:
int getdata()
{
  cout<<"Enter the details of the students as instructed:";
  cout<<"Name: ";
  cin>>name;
  cout<<"ID: ";
  cin>>ID;
  cout<<"Faculty: ";
  cin>>faculty;
  return 0;
}
int display()
{
  cout<<"The details of the student are as:";
  cout<<"Name: "<<name<<endl;
  cout<<"ID: "<<ID<<endl;
  cout<<"Faculty: "<<faculty<<endl;
  return 0;
}
};
int main()
{
  student a;
  a.getdata();
  a.display();
  return 0;
}
```

26). WAP to find TSA, CSA and volume of cylinder using the concept of class and objects.



```
#include <iostream>
using namespace std;
class cylinder
{
private:
int radius;
int height;
public:
int data()
{
   cout<<"Enter the radius of the cylinder: ";
   cin>>radius;
   cout<<"Enter the height of the cylinder: ";
   cin>>height;
   return 0;
}
float tsa()
{
   return 2*3.14*radius*(radius+height);
}
float csa()
{
   return 2*3.14*radius*height;
}
float vol()
{
   return 3.14*radius*radius*height;
}
};
int main()
{
   cylinder a;
   a.data();
   cout<<"The TSA of the cylinder is: "<<a.tsa();
   cout<<"The CSA of the cylinder is: "<<a.csa();
   cout<<"The volume of the cylinder is: "<<a.vol();
   return 0;
}
```

27). WAP to find the equivalent resistance in parallel and series for two resistances.



```
#include <iostream>
using namespace std;
class resistors
{
private:
int r1;
int r2;
public:
int data()
{
  cout<<"Enter the values of two resistances: ";
  cin>>r1>>r2;
  return 0;
}
float rs()
{
  return r1+r2;
}
float rp()
{
  return (r1*r2)/(r1+r2);
}
};
int main()
{
  resistors p;
  p.data();
  cout<<"The equivalent series resistance is: "<<p.rs();
  cout<<"The equivalent parallel resistance is: "<<p.rp();
  return 0;
}
```

28). Describe briefly about array of objects. Create a class student with

data member name, id and faculty. Use necessary member functions and write a program to scan and display the details of 5 students

-> An **array of objects** in C++ is a collection of objects of the same class type stored in contiguous memory locations. This allows you to manage multiple objects using a single array structure, facilitating operations on collections of objects in a structured manner.

- Example: `ClassName objectsArray[size];`

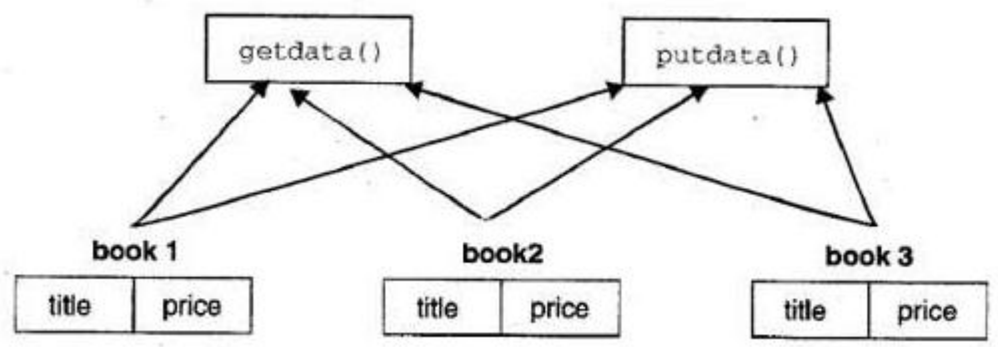
- **Initialization**:

- Objects in the array can be initialized using constructor calls or by providing default values in the class definition.

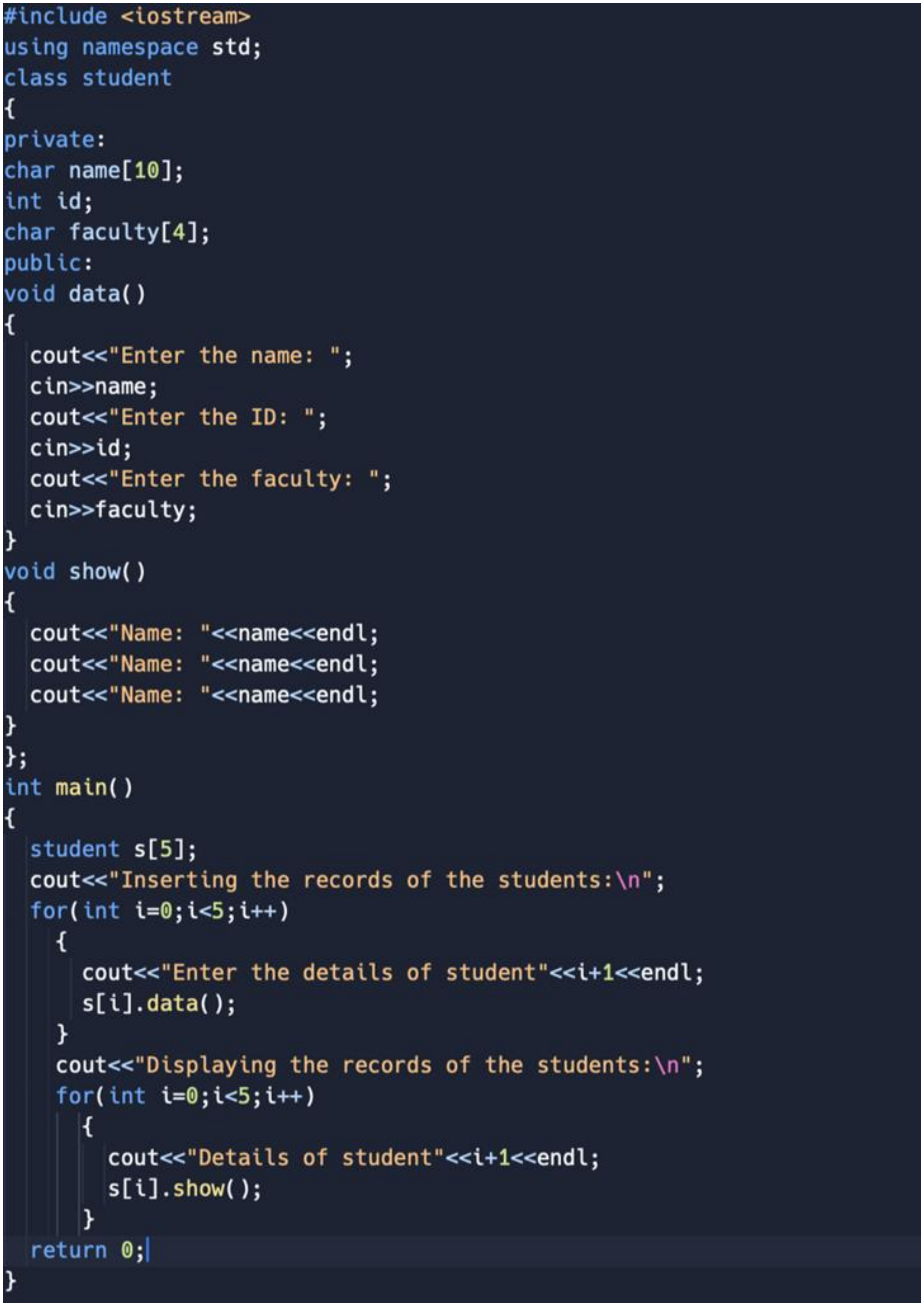
- Example: `ClassName objectsArray[3] = {ClassName(arg1, arg2), ClassName(arg3, arg4), ClassName(arg5, arg6)};`

- **Accessing Elements**:

  - Elements of the array are accessed using the array subscript notation, similar to accessing elements in an array of basic data types.
  - Example: `objectsArray[0].memberFunction();`



Memory Allocation for the Objects of the Class book



```cpp
#include <iostream>
using namespace std;
class student
{
private:
char name[10];
int id;
char faculty[4];
public:
void data()
{
  cout<<"Enter the name: ";
  cin>>name;
  cout<<"Enter the ID: ";
  cin>>id;
  cout<<"Enter the faculty: ";
  cin>>faculty;
}
void show()
{
  cout<<"Name: "<<name<<endl;
  cout<<"Name: "<<name<<endl;
  cout<<"Name: "<<name<<endl;
}
};
int main()
{
  student s[5];
  cout<<"Inserting the records of the students:\n";
  for(int i=0;i<5;i++)
    {
      cout<<"Enter the details of student"<<i+1<<endl;
      s[i].data();
    }
    cout<<"Displaying the records of the students:\n";
    for(int i=0;i<5;i++)
      {
        cout<<"Details of student"<<i+1<<endl;
        s[i].show();
      }
  return 0;
}
```

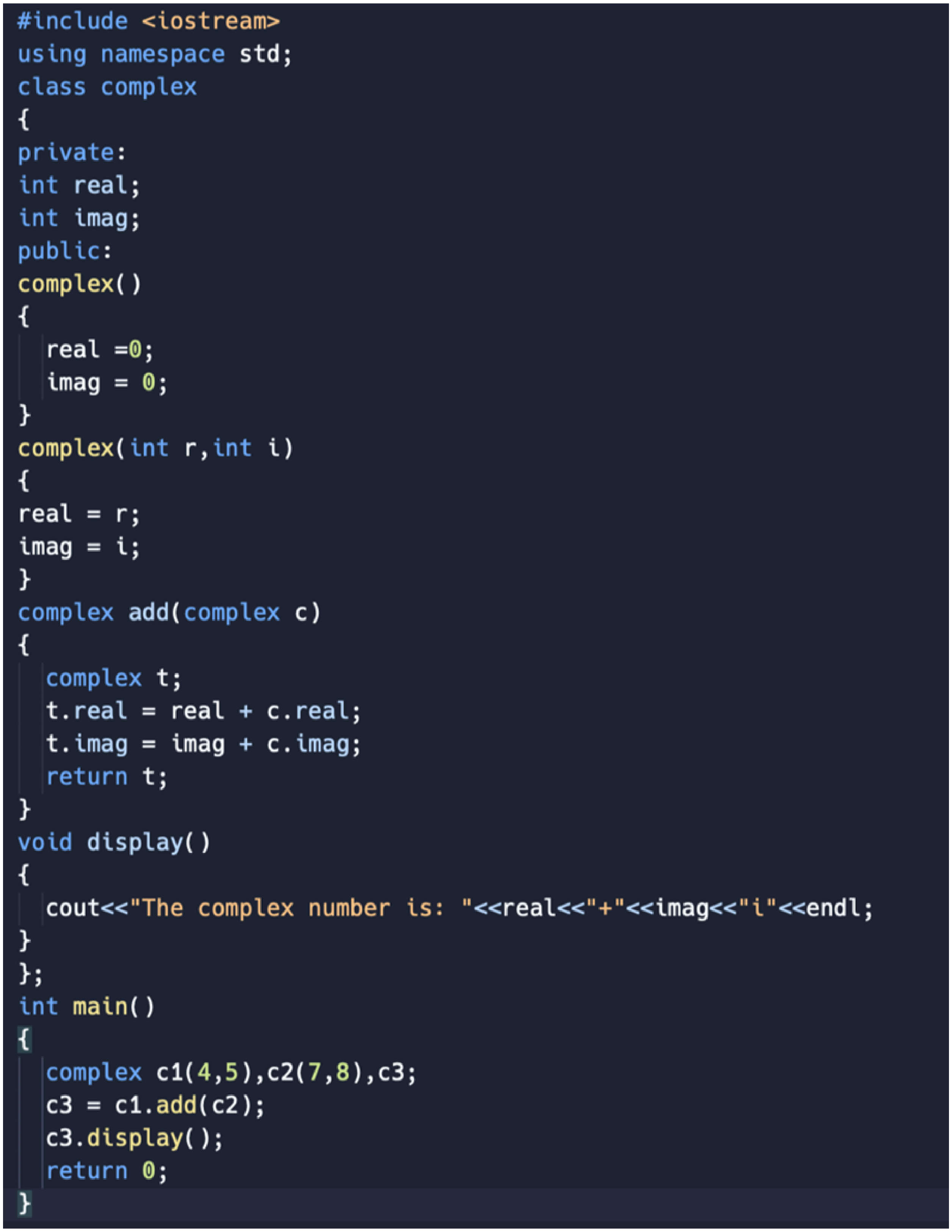
29). How can you pass an object as an argument? WAP to add two complex numbers passing objects as arguments.

In C++, objects can be passed to functions as arguments just like any other data type. This is a fundamental concept in object-oriented programming.

Working steps:

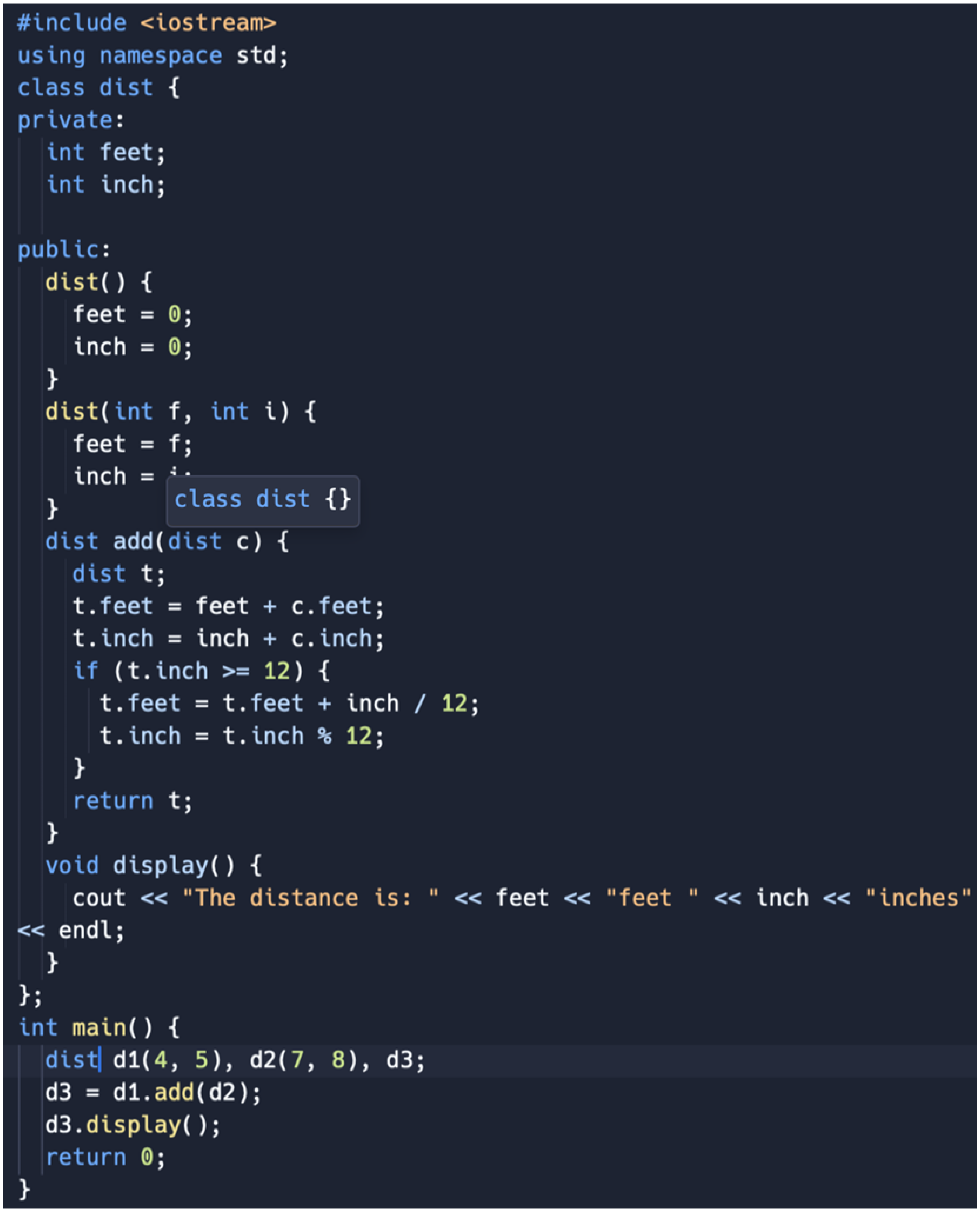
1. **Create an Object:** Instantiate an object of a class.
2. **Pass the Object:** When calling a function, pass the object as an argument.
3. **Access Object Members:** Within the function, access the object's members using the dot operator.

Program to add two complex numbers passing objects as arguments.



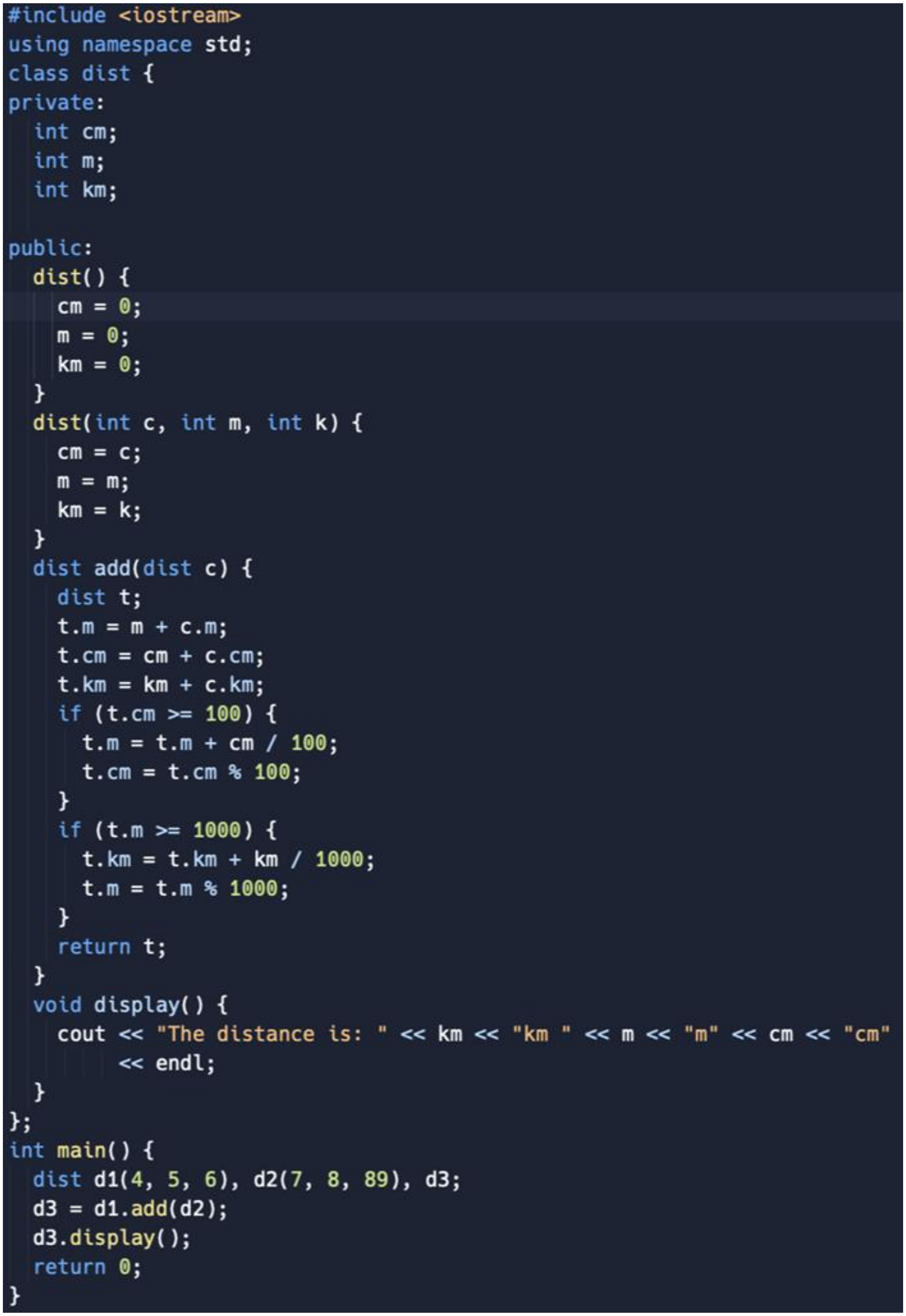
```cpp
#include <iostream>
using namespace std;
class complex
{
private:
int real;
int imag;
public:
complex()
{
   real =0;
   imag = 0;
}
complex(int r,int i)
{
real = r;
imag = i;
}
complex add(complex c)
{
   complex t;
   t.real = real + c.real;
   t.imag = imag + c.imag;
   return t;
}
void display()
{
   cout<<"The complex number is: "<<real<<"+"<<imag<<"i"<<endl;
}
};
int main()
{
   complex c1(4,5),c2(7,8),c3;
   c3 = c1.add(c2);
   c3.display();
   return 0;
}
```

30). Create a class distance with data member feet and inches. Use the necessary member function to add two distances and display the result.



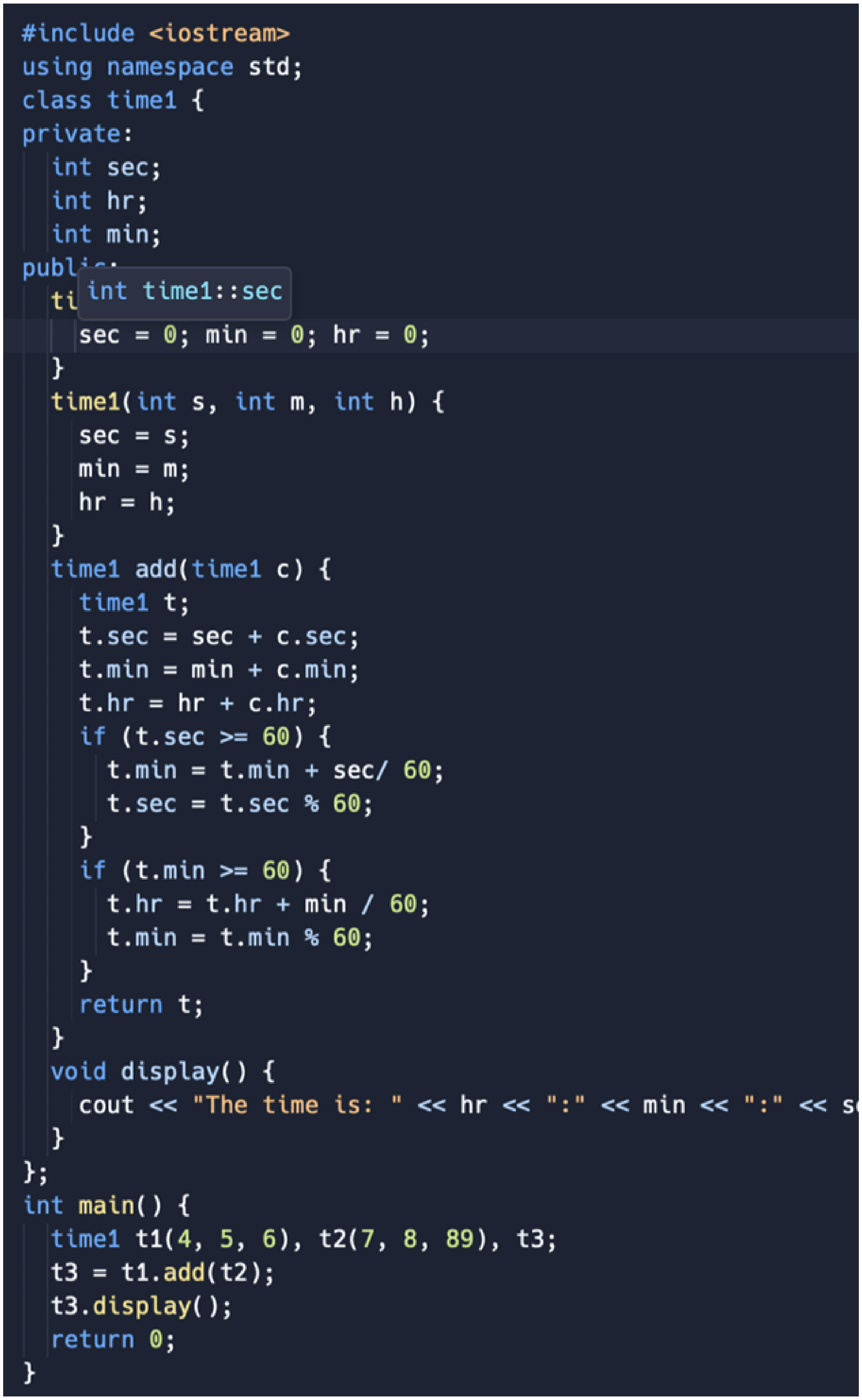
```
#include <iostream>
using namespace std;
class dist {
private:
    int feet;
    int inch;

public:
    dist() {
        feet = 0;
        inch = 0;
    }
    dist(int f, int i) {
        feet = f;
        inch = i;
    }
    dist add(dist c) {
        dist t;
        t.feet = feet + c.feet;
        t.inch = inch + c.inch;
        if (t.inch >= 12) {
            t.feet = t.feet + inch / 12;
            t.inch = t.inch % 12;
        }
        return t;
    }
    void display() {
        cout << "The distance is: " << feet << "feet " << inch << "inches"
<< endl;
    }
};
int main() {
    dist d1(4, 5), d2(7, 8), d3;
    d3 = d1.add(d2);
    d3.display();
    return 0;
}
```

31). Create a class called distance with data members cm,m and km. Use member functions to add and display the result.



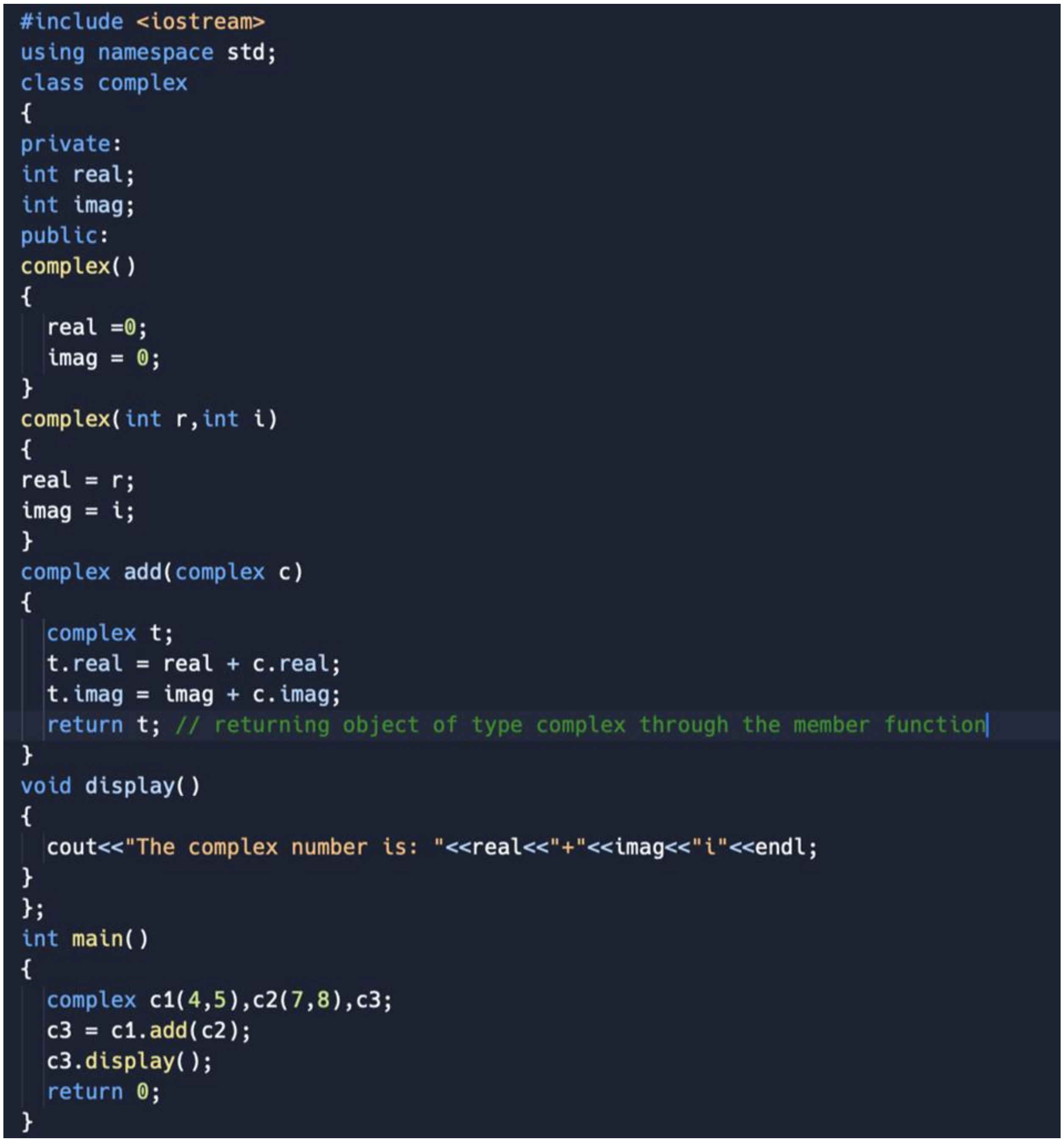
```cpp
#include <iostream>
using namespace std;
class dist {
private:
  int cm;
  int m;
  int km;

public:
  dist() {
    cm = 0;
    m = 0;
    km = 0;
  }
  dist(int c, int m, int k) {
    cm = c;
    m = m;
    km = k;
  }
  dist add(dist c) {
    dist t;
    t.m = m + c.m;
    t.cm = cm + c.cm;
    t.km = km + c.km;
    if (t.cm >= 100) {
      t.m = t.m + cm / 100;
      t.cm = t.cm % 100;
    }
    if (t.m >= 1000) {
      t.km = t.km + km / 1000;
      t.m = t.m % 1000;
    }
    return t;
  }
  void display() {
    cout << "The distance is: " << km << "km " << m << "m" << cm << "cm"
         << endl;
  }
};
int main() {
  dist d1(4, 5, 6), d2(7, 8, 89), d3;
  d3 = d1.add(d2);
  d3.display();
  return 0;
}
```

32). Create a class called time with data members second, minute and hour. Use the necessary member function to add two times.



```cpp
#include <iostream>
using namespace std;
class time1 {
private:
  int sec;
  int hr;
  int min;
public:
  ti
    sec = 0; min = 0; hr = 0;
  }
  time1(int s, int m, int h) {
    sec = s;
    min = m;
    hr = h;
  }
  time1 add(time1 c) {
    time1 t;
    t.sec = sec + c.sec;
    t.min = min + c.min;
    t.hr = hr + c.hr;
    if (t.sec >= 60) {
      t.min = t.min + sec/ 60;
      t.sec = t.sec % 60;
    }
    if (t.min >= 60) {
      t.hr = t.hr + min / 60;
      t.min = t.min % 60;
    }
    return t;
  }
  void display() {
    cout << "The time is: " << hr << ":" << min << ":" << s
  }
};
int main() {
  time1 t1(4, 5, 6), t2(7, 8, 89), t3;
  t3 = t1.add(t2);
  t3.display();
  return 0;
}
```

33). WAP to show returning objects from a function.



```cpp
#include <iostream>
using namespace std;
class complex
{
private:
int real;
int imag;
public:
complex()
{
  real =0;
  imag = 0;
}
complex(int r,int i)
{
real = r;
imag = i;
}
complex add(complex c)
{
  complex t;
  t.real = real + c.real;
  t.imag = imag + c.imag;
  return t; // returning object of type complex through the member function
}
void display()
{
  cout<<"The complex number is: "<<real<<"+"<<imag<<"i"<<endl;
}
};
int main()
{
  complex c1(4,5),c2(7,8),c3;
  c3 = c1.add(c2);
  c3.display();
  return 0;
}
```

34). What are static data members? List out its properties.

-> Each object contain its own separate data.But in a data item in a class in, defined as ‘static’ then only one item Is created for entire class no matter how many objects there are.

A static data member is usefull when all objects of the same class must share a common item of information.

Key Characteristics and Properties:

1) All static variables are initialized to zero when the first object is created.
2) The type and the scope of each static member variable must be defined outside class.
3) Unlike regular data members individual copies of a static member variables are not made for each object no matter how many objects of a class are created only one copy of a static data member exists,
   Thus,all objects of that class use same variable.

35). What are static data members? List out its properties

-> A class may also have static methods as its members. Static function is defined by using the keyword ‘static’ before the members that is to be declared as static function

Syntax

```
      Static return_type function_name(argument_list)

{
  body

}
```

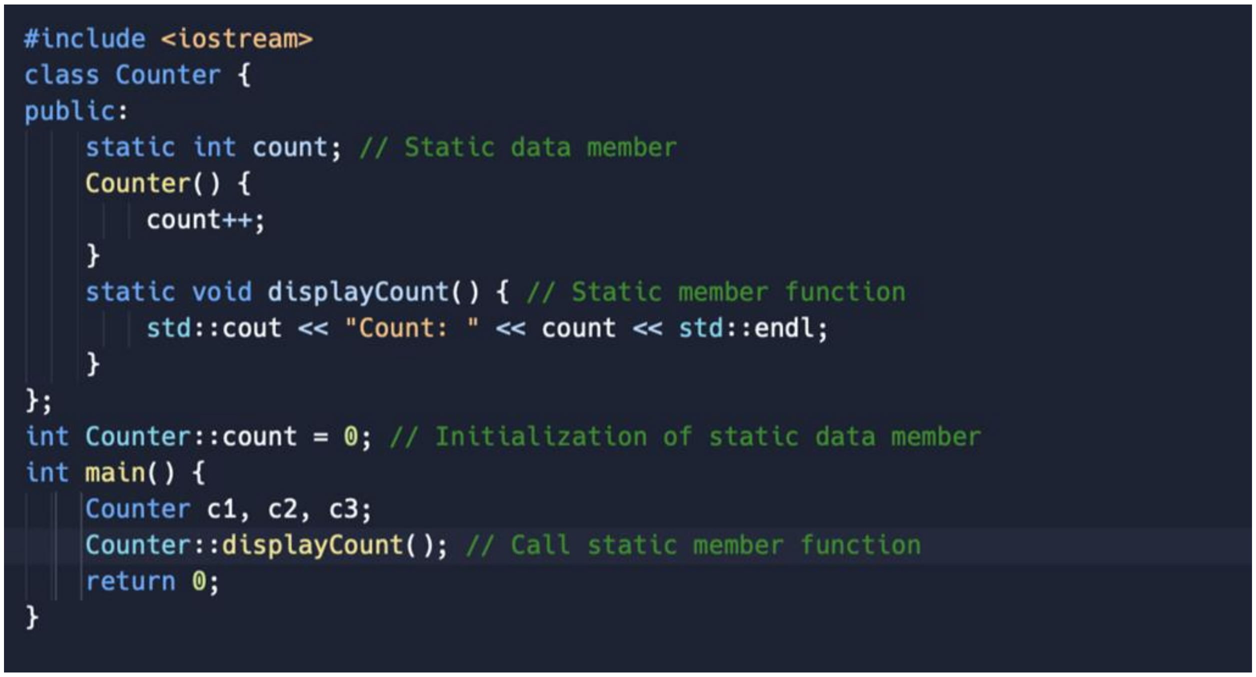
Key Characteristics and Properties:

1) A static member function cannot be declared as virtual.

2) A static member function do not have access to the ‘this’ pointer of the class.

3) A static member function can be called using the class name(instead of its objects).

36). WAP to demonstrate static data members and static data functions.



```cpp
#include <iostream>
class Counter {
public:
    static int count; // Static data member
    Counter() {
        count++;
    }
    static void displayCount() { // Static member function
        std::cout << "Count: " << count << std::endl;
    }
};
int Counter::count = 0; // Initialization of static data member
int main() {
    Counter c1, c2, c3;
    Counter::displayCount(); // Call static member function
    return 0;
}
```

37). Why do we need a friend function? List its properties. WAP to add one data member of two different classes using the concept of friend function.

Friend functions are a mechanism in C++ that allow functions to access the private and protected members of a class.

We need a friend function because of following reasons:

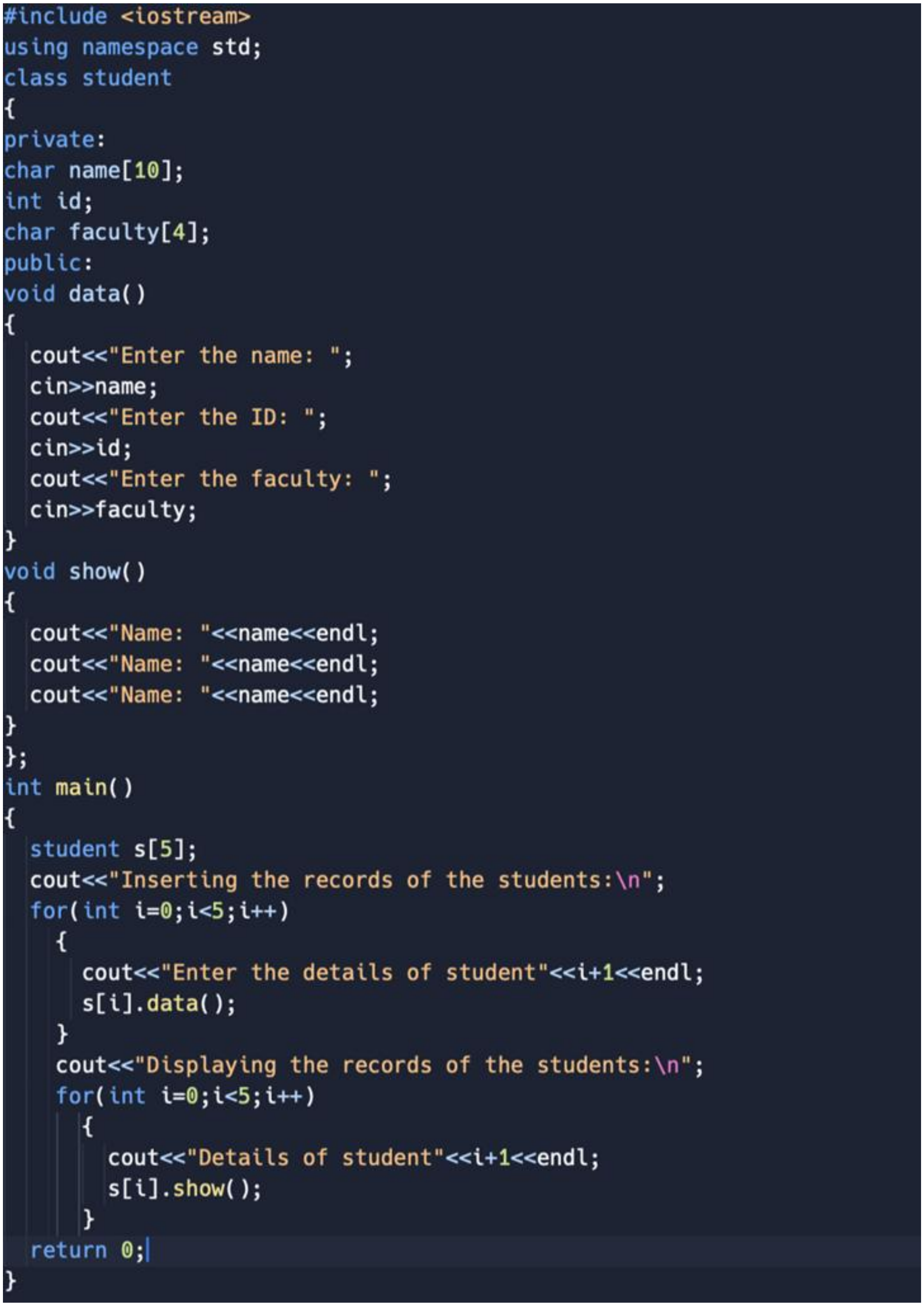
- **Access to Private and Protected Members**:
    - Friend functions can access private and protected members of a class, which allows for more flexible interaction between classes while still encapsulating the internal details.
- **Non-Member Functions**:
    - Sometimes, it is useful to have non-member functions that need to access the internals of a class. For example, operator overloading often involves friend functions, as operators might need to work with private data of the class.
- **Class Relationships**:
    - Friend functions can help manage class relationships where two or more classes need to interact closely, but do not logically belong to the same class hierarchy.
- **Operator Overloading**:

- Friend functions are commonly used for operator overloading. For example, you might want to overload operators like << or >> to handle input/output for custom classes.

Properties of Friend Functions:

- **Not a member of the class:** While declared within a class, a friend function is not considered a member. It doesn't have access to the this pointer.
- **Access to private and protected members:** A friend function can access all members of the class, including private and protected ones.
- **Declared using the friend keyword:** Within the class, you declare a function as a friend using the friend keyword.
- **Defined outside the class:** The function's definition is typically outside the class.
- **Not inherited:** Friend relationships are not inherited by derived classes.

Program to add one data member of two different classes using the concept of friend function.

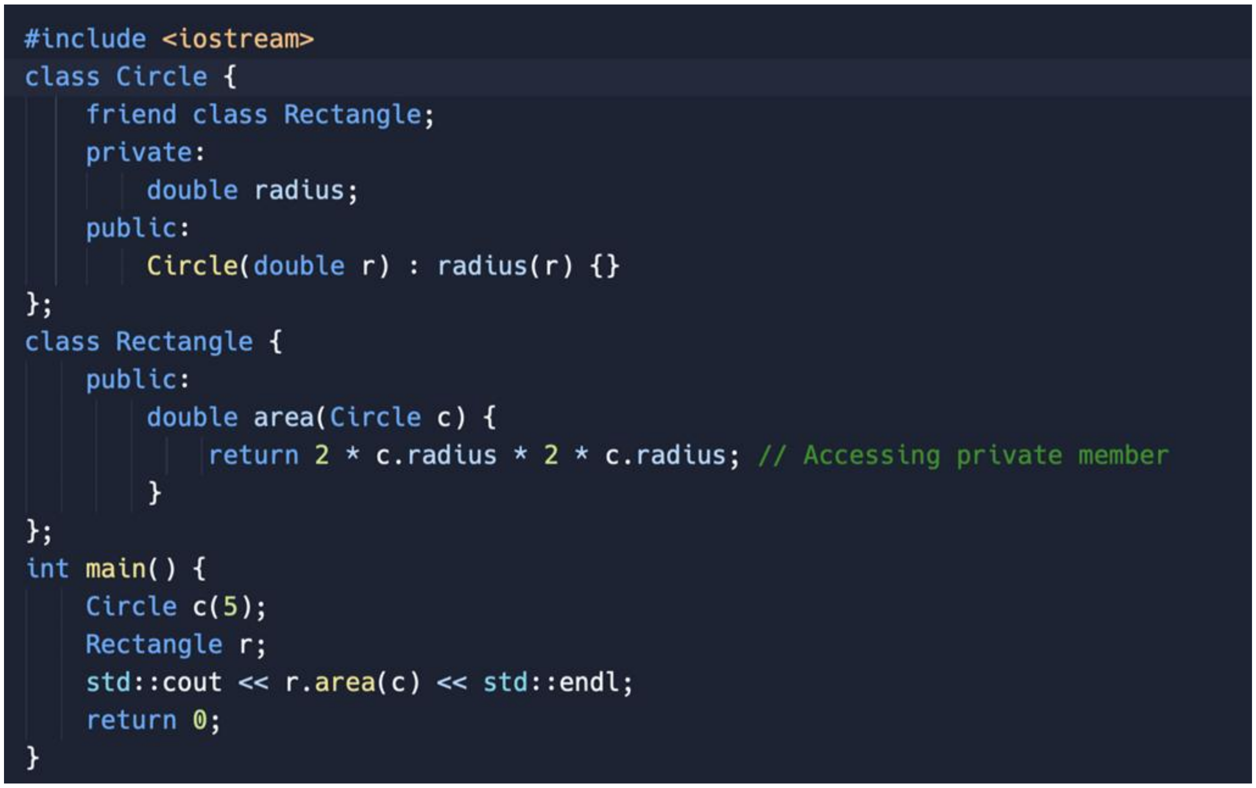


```cpp
#include <iostream>
using namespace std;
class student
{
private:
char name[10];
int id;
char faculty[4];
public:
void data()
{
  cout<<"Enter the name: ";
  cin>>name;
  cout<<"Enter the ID: ";
  cin>>id;
  cout<<"Enter the faculty: ";
  cin>>faculty;
}
void show()
{
  cout<<"Name: "<<name<<endl;
  cout<<"Name: "<<name<<endl;
  cout<<"Name: "<<name<<endl;
}
};
int main()
{
  student s[5];
  cout<<"Inserting the records of the students:\n";
  for(int i=0;i<5;i++)
    {
      cout<<"Enter the details of student"<<i+1<<endl;
      s[i].data();
    }
    cout<<"Displaying the records of the students:\n";
    for(int i=0;i<5;i++)
      {
        cout<<"Details of student"<<i+1<<endl;
        s[i].show();
      }
  return 0;
}
```

38). What are friend classes? WAP to show the use of friend classes.

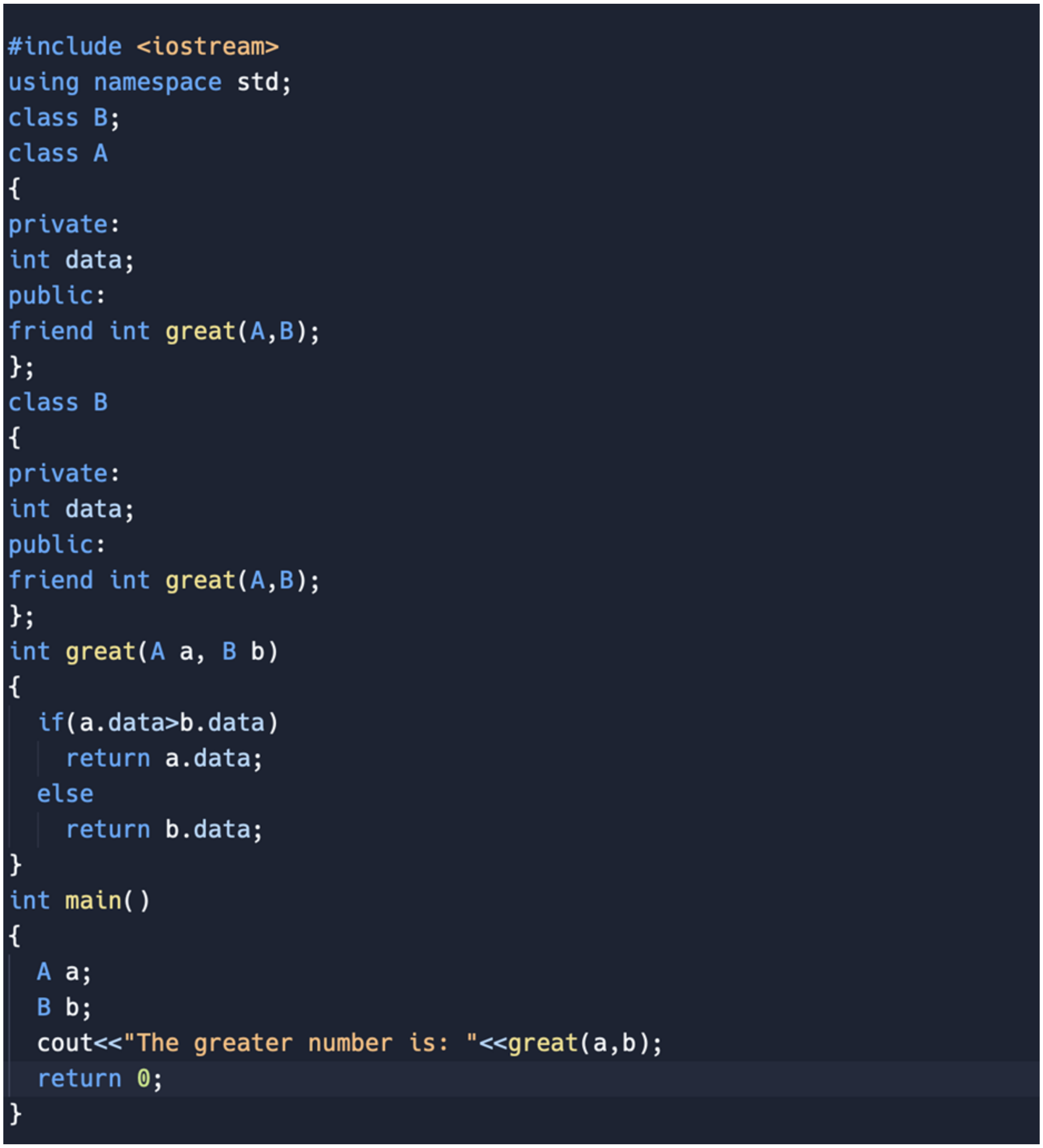
A friend class in C++ is a class that has access to the private and protected members of another class. This means that functions within a friend class can directly access the private and protected data and methods of the class it is befriended with, bypassing the usual encapsulation rules.

Program to show the use of friend classes.



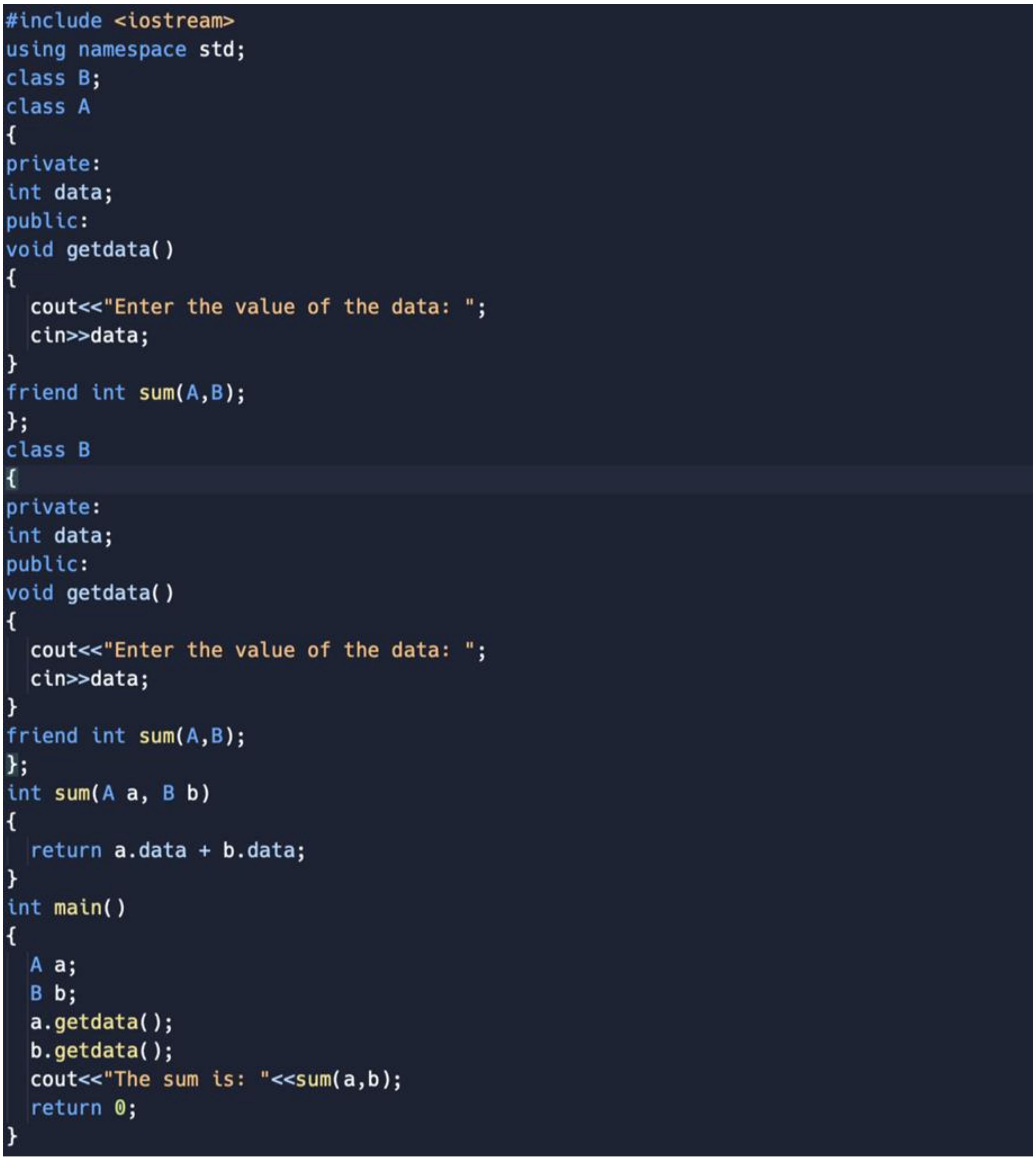
```
#include <iostream>
class Circle {
    friend class Rectangle;
    private:
        double radius;
    public:
        Circle(double r) : radius(r) {}
};
class Rectangle {
    public:
        double area(Circle c) {
            return 2 * c.radius * 2 * c.radius; // Accessing private member
        }
};
int main() {
    Circle c(5);
    Rectangle r;
    std::cout << r.area(c) << std::endl;
    return 0;
}
```

39). WAP to find the greatest among two data members using the concept of friend function.



```
#include <iostream>
using namespace std;
class B;
class A
{
private:
int data;
public:
friend int great(A,B);
};
class B
{
private:
int data;
public:
friend int great(A,B);
};
int great(A a, B b)
{
  if(a.data>b.data)
    return a.data;
  else
    return b.data;
}
int main()
{
  A a;
  B b;
  cout<<"The greater number is: "<<great(a,b);
  return 0;
}
```

40). WAP to add data members of two classes using the concept of friend function (bridging classes).



```
#include <iostream>
using namespace std;
class B;
class A
{
private:
int data;
public:
void getdata()
{
  cout<<"Enter the value of the data: ";
  cin>>data;
}
friend int sum(A,B);
};
class B
{
private:
int data;
public:
void getdata()
{
  cout<<"Enter the value of the data: ";
  cin>>data;
}
friend int sum(A,B);
};
int sum(A a, B b)
{
  return a.data + b.data;
}
int main()
{
  A a;
  B b;
  a.getdata();
  b.getdata();
  cout<<"The sum is: "<<sum(a,b);
  return 0;
}
```