Himalaya College of Engineering

Advanced C++ Programming Lab Report

Lab 3: OBJECTS AND CLASSES

Prepared By: Ankit Belbase(HCE081BEI007)

Subject: Object-Oriented Programming (OOP)

Program: Bachelor of Electronics Engineering

Institution: Himalaya College of Engineering

Date: June 14,2025

OBJECTIVE

- To understand the concept of classes and objects in C++.
- To implement constructors for automatic initialization of objects.
- To use destructors for releasing resources and observing object lifecycle.
- To develop modular and reusable code using object-oriented principles.

BACKGROUND THEORY

In C++, classes are user-defined data types that encapsulate data and functions. An object is an instance of a class, allowing us to represent real-world entities in code. A constructor is a special function that is automatically invoked when an object is created, used to initialize data members. Constructors can be default, parameterized, or copy constructors. A destructor is called automatically when an object goes out of scope or is deleted; it is used to release resources or perform cleanup tasks. Together, these features form the foundation of object-oriented programming, promoting code reusability, abstraction, and maintainability.

Object:

An object is any entity, thing or organization that exists in real world that consists of two fundamentals characteristics: its attributes and behavior. In OOP, the problem is divided into a group of objects and each object consists of own properties (data) and behavior (functions).

- Objects are the basic runtime entities which may be created or destroyed at run time.
- Object can communicate with others by using message passing mechanism.
- The member function of an object can only access its data.
- For example; a dog having attributes such as color, weight, age etc. and behaviors such as barking, wagging tail etc.

Class:

A class is the collection of similar objects which is defined as the template or prototype to define the common attributes and behavior for all the objects of the class. In fact objects are variables of type class. The entire set of data and code of an object can be made a user-defined data type with the help of a class.

- Once a class has been defined, can create any number of Objects associated with that class.
- No memory is allocated when class is created.
- Class has three access specifiers: public, private and protected. So, class incorporates the concept of data hiding.
- For example, mango, apple and orange are members of class fruit.

Syntax:

```
class ClassName {
private:
   // private data members
   // private member functions
protected:
   // protected members (optional)
public:
   // public data members
   // public member functions
};
```

Example:

```
#include <iostream>
using namespace std;
class Demo {
private:
   int a;
protected:
   int b;
public:
   int c;
   void setValues() {
      a = 10;
      b = 20;
      c = 30;
   }
   void display() {
      cout << "Private a = " << a << endl;
      cout << "Protected b = " << b << endl;
      cout << "Public c = " << c << endl;
```

```
  }};
int main() {
  Demo obj;
  obj.setValues();
  obj.display();
   // obj.a = 1;  //Error: private member
  // obj.b = 2;  // Error: protected member
  obj.c = 3;     // OK: public member
  return 0;
}
```

Explanation:

- `a` is private → Only accessible inside class methods.
- `b` is protected → Also accessible inside the class and in derived classes.
- `c` is public → Can be accessed directly from outside the class (like in `main()`)

Constructors:

- In C++, a constructor is a special member function of a class that is automatically called when an object of the class is created.
- It is used to initialize the data members of the class.

Purpose:

□Automatically initializes objects when no values are passed.

□Sets default or fixed values to class members.

□Useful for creating multiple objects with the same initial state

Characteristics:

1. No parameters.

2. Automatically invoked when an object is created without arguments.

3. Provided by the compiler if no other constructor is defined.

4. Can be explicitly defined by the user.

Syntax:

```
class ClassName {
public:
ClassName(); // Default constructor
};
```

Types of constructors:

a. Default constructor:
   - If you do not define any constructor, the compiler automatically provides a default constructor that does nothing but creates the object.

```
class Test {

// No constructor defined

};

int main() {

Test t1; // Compiler provides default constructor

return 0;

}
```

b. Parameterized constructor:
   - A parameterized constructor is a constructor that takes arguments/parameters.
   - It is used to initialize objects with specific values at the time of creation.

Syntax:

```
class ClassName {

public:

ClassName(data_type parameter1, data_type parameter2, ...);

};
```

c. Copy constructor:
   - A copy constructor is a special constructor in C++ used to create a new object as an exact copy of an existing object.
   - It copies the data members of one object to another.

Syntax:

```
class ClassName {

public:

ClassName(const ClassName &obj); // Copy constructor declaration

};
```

Destructor:

- A destructor is a special member function in C++ that is automatically called when an object goes out of scope or is deleted.
- It is used to free resources allocated to the object.

Purpose:

- To perform clean-up tasks (e.g., releasing memory, closing files).
- To avoid memory leaks in programs using dynamic memory.

Characteristics:

- Name is the same as the class, prefixed with a tilde ~.
- Takes no arguments and returns nothing.
- Only one destructor is allowed per class (no overloading).
- Automatically invoked at the end of object's lifetime.

Syntax:

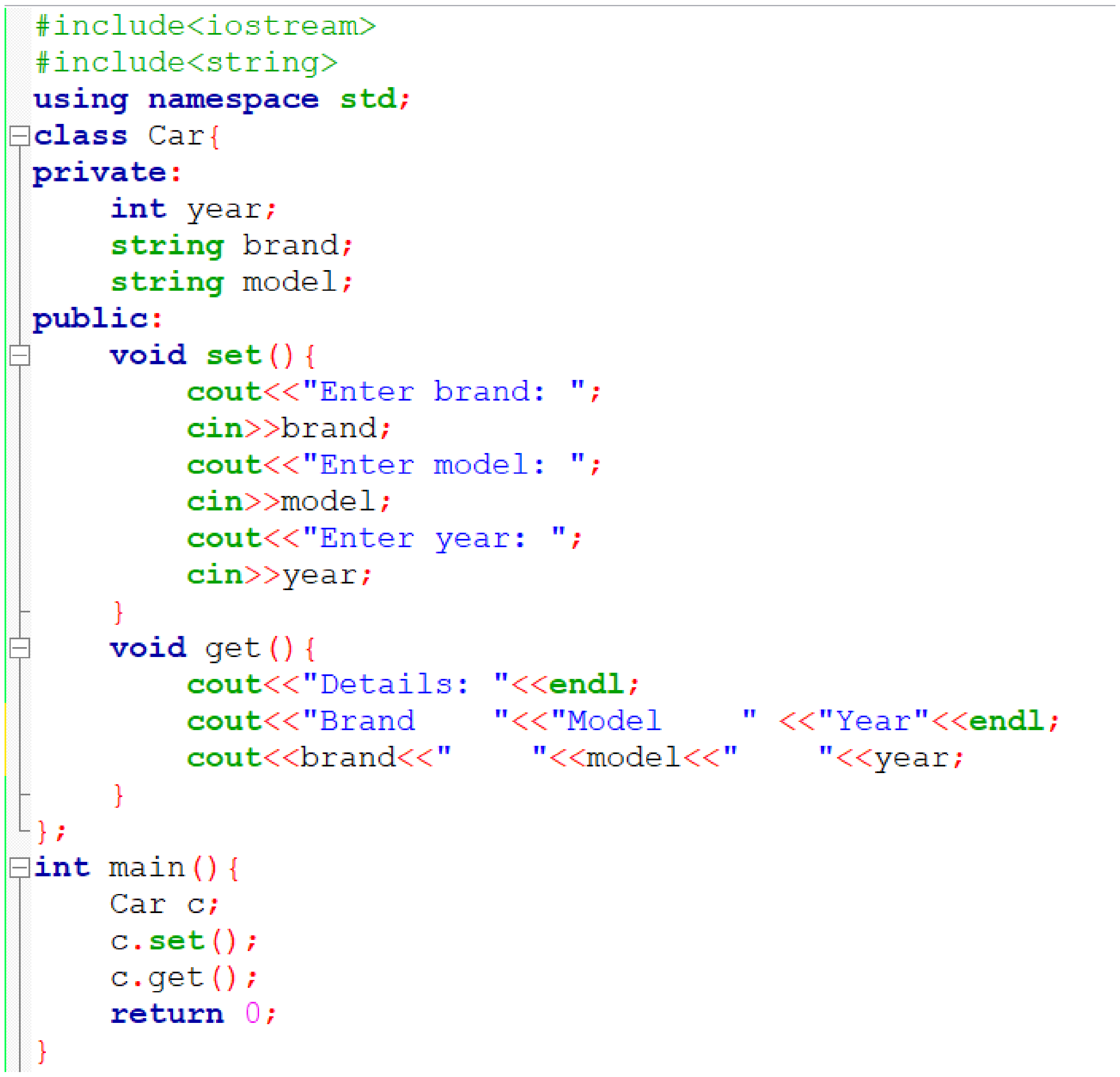
```
~ClassName() {
// code to release resources
}
```

Example of constructor and destructor:

```
#include     <iostream>     using
namespace std; class Demo {
public:
Demo() {
cout << "Constructor called." << endl;
}
~Demo() {
cout << "Destructor called." << endl;
}
void show() {
cout << "Inside show function." << endl;
}
};
int main() {
Demo obj; // Constructor is called obj.show(); //
Function call
return 0; // Destructor is called automatically
```

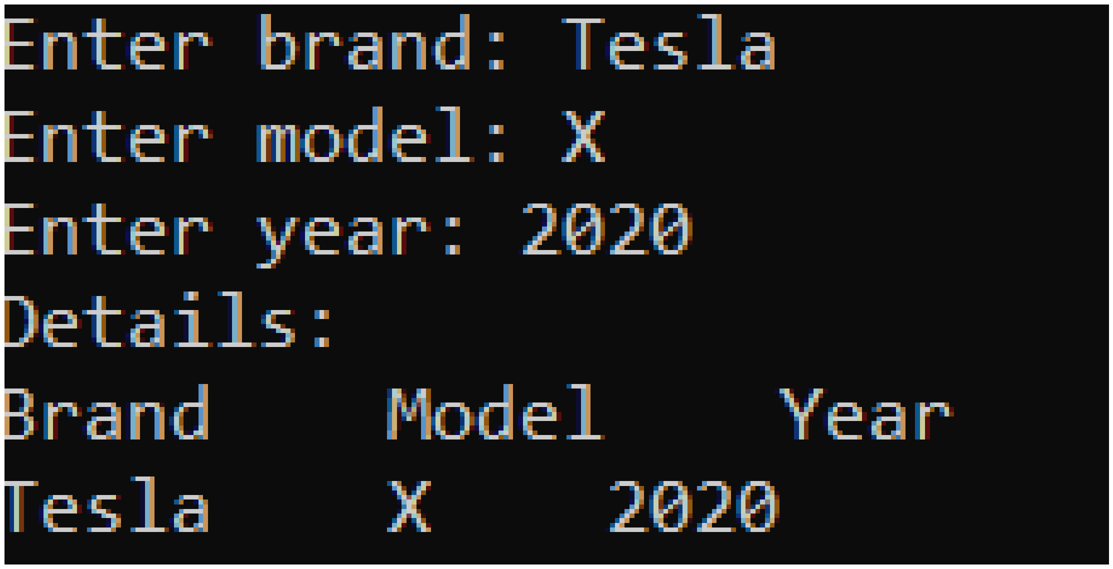
LAB QUESTIONS

Q1. Define a class Car with private members brand, model, and year. Include public member functions to set and get these private members. Ensure that only member functions can access these private members.



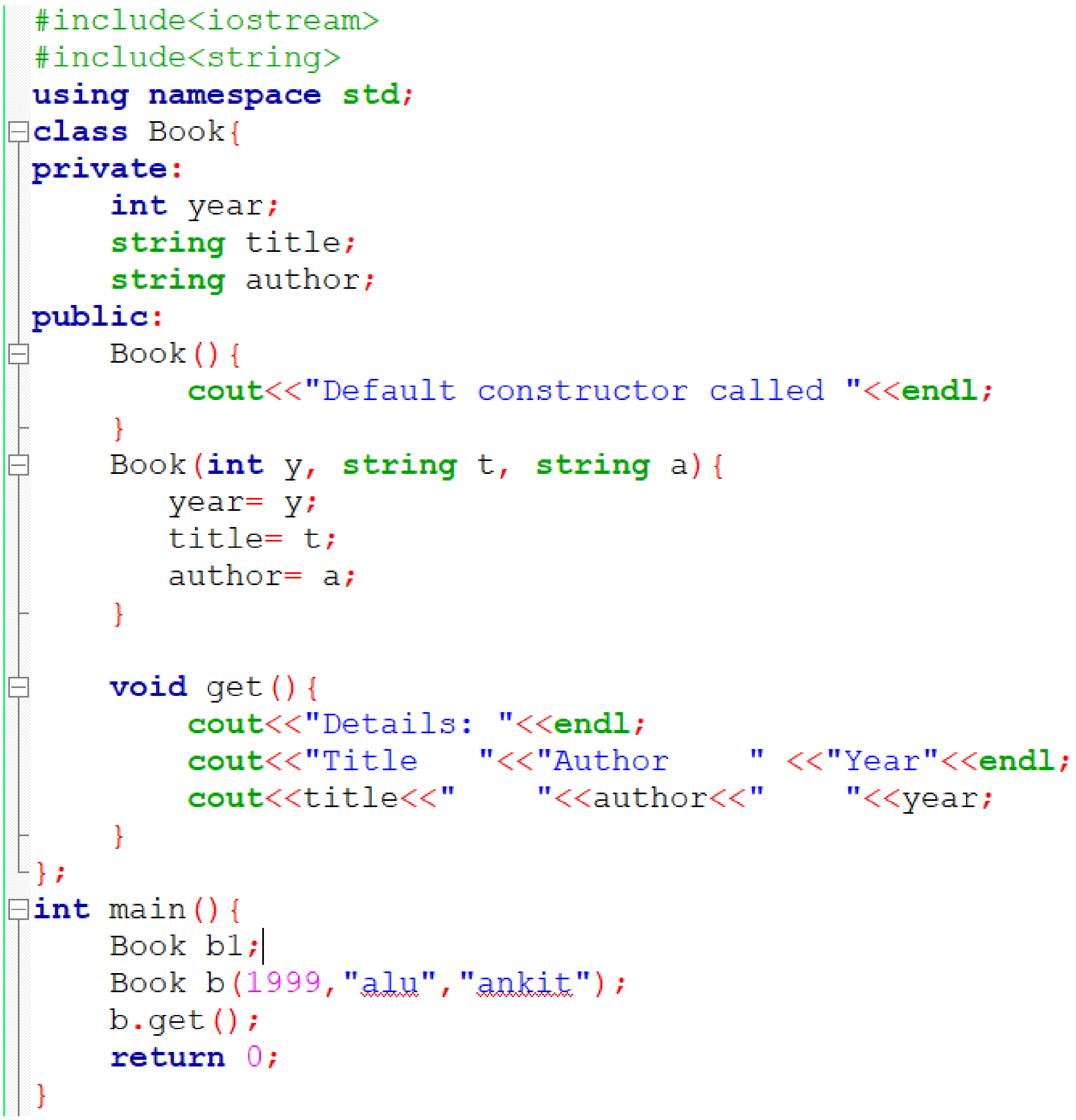
```
#include<iostream>
#include<string>
using namespace std;
class Car{
private:
    int year;
    string brand;
    string model;
public:
    void set(){
        cout<<"Enter brand: ";
        cin>>brand;
        cout<<"Enter model: ";
        cin>>model;
        cout<<"Enter year: ";
        cin>>year;
    }
    void get(){
        cout<<"Details: "<<endl;
        cout<<"Brand    "<<"Model    " <<"Year"<<endl;
        cout<<brand<<"    "<<model<<"    "<<year;
    }
};
int main(){
    Car c;
    c.set();
    c.get();
    return 0;
}
```

Output:



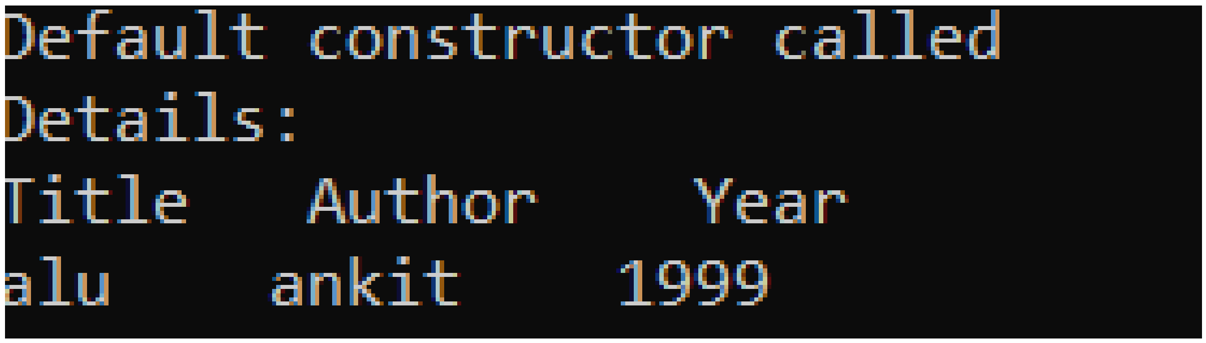
```
Enter brand: Tesla
Enter model: X
Enter year: 2020
Details:
Brand    Model    Year
Tesla    X    2020
```

Q2. Define a class Book with private members title, author, and year. Implement both default and parameterized constructors. The default constructor should initialize the members with default values, and the parameterized constructor should set these values based on user input. Provide a method to display the details of a book.



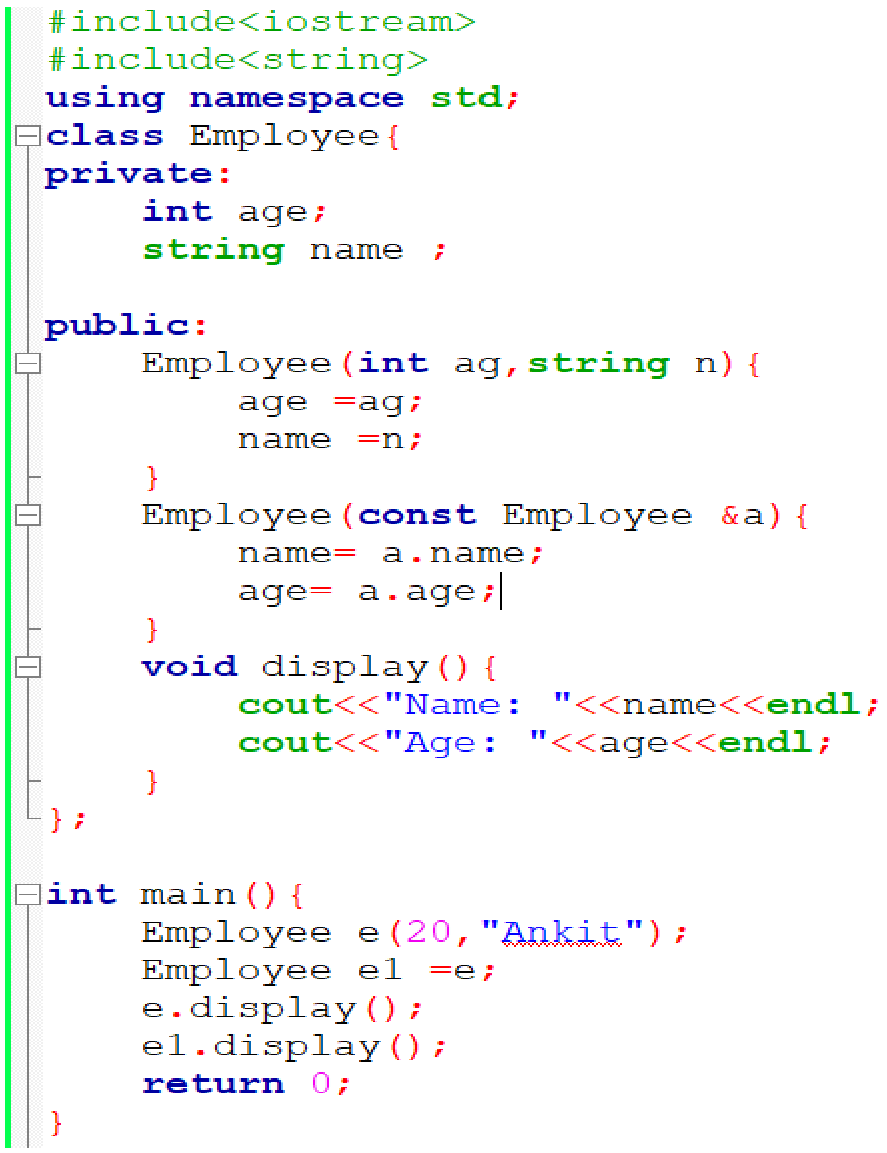
```cpp
#include<iostream>
#include<string>
using namespace std;
class Book{
private:
    int year;
    string title;
    string author;
public:
    Book(){
        cout<<"Default constructor called "<<endl;
    }
    Book(int y, string t, string a){
       year= y;
       title= t;
       author= a;
    }

    void get(){
        cout<<"Details: "<<endl;
        cout<<"Title   "<<"Author    " <<"Year"<<endl;
        cout<<title<<"    "<<author<<"    "<<year;
    }
};
int main(){
    Book b1;
    Book b(1999,"alu","ankit");
    b.get();
    return 0;
}
```

Output:



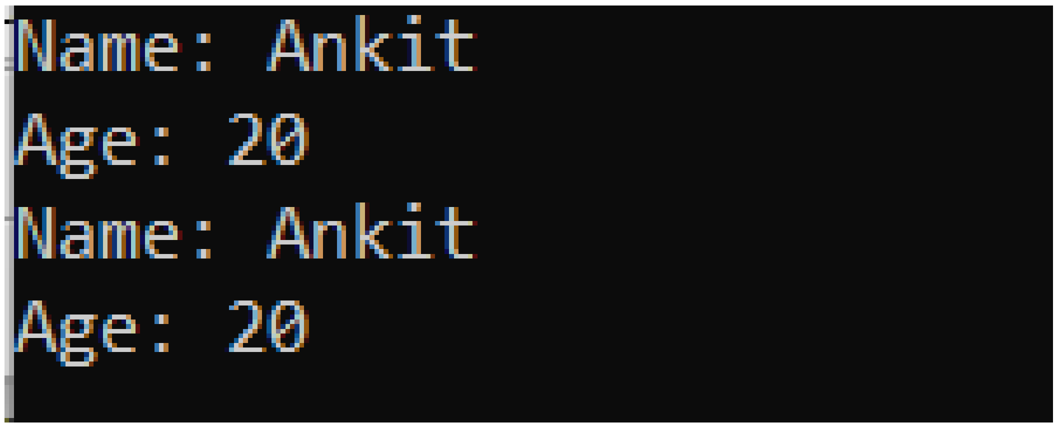
```
Default constructor called
Details:
Title    Author     Year
alu     ankit     1999
```

- Q3. Create a class Employee with private members name and age. Implement a copy constructor to create a deep copy of an existing Employee object. Provide a method to display the details of an employee.



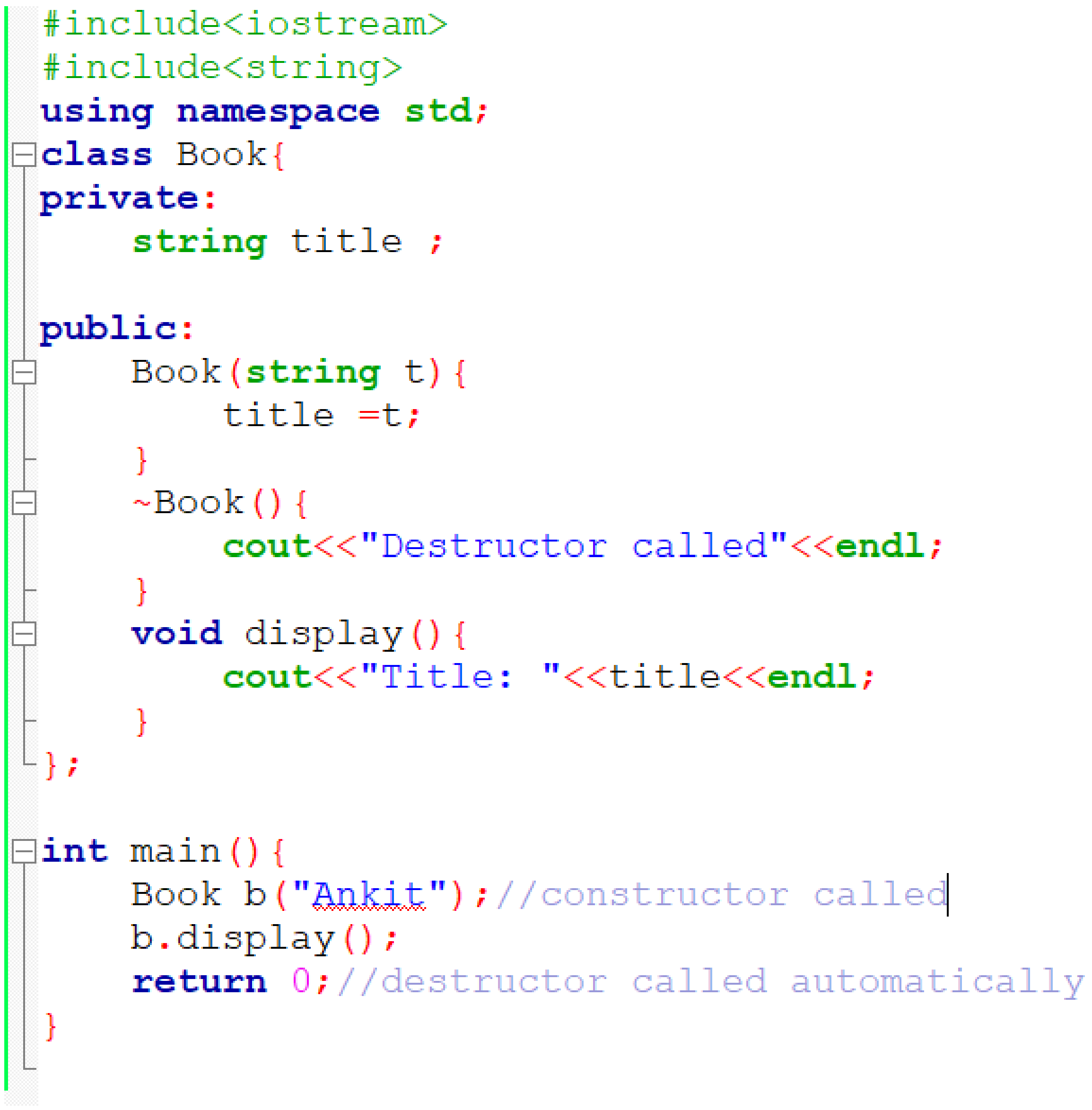
```cpp
#include<iostream>
#include<string>
using namespace std;
class Employee{
private:
    int age;
    string name ;

public:
    Employee(int ag,string n){
        age =ag;
        name =n;
    }
    Employee(const Employee &a){
        name= a.name;
        age= a.age;
    }
    void display(){
        cout<<"Name: "<<name<<endl;
        cout<<"Age: "<<age<<endl;
    }
};

int main(){
    Employee e(20,"Ankit");
    Employee e1 =e;
    e.display();
    e1.display();
    return 0;
}
```

Output:



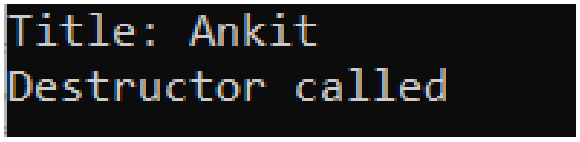
```
Name: Ankit
Age: 20
Name: Ankit
Age: 20
```

Q4. Define a class Book with a private member title. Implement a constructor that initializes title and a destructor that prints a message when the Book object is destroyed. Create instances of Book objects in main() to demonstrate the usage of the destructor.



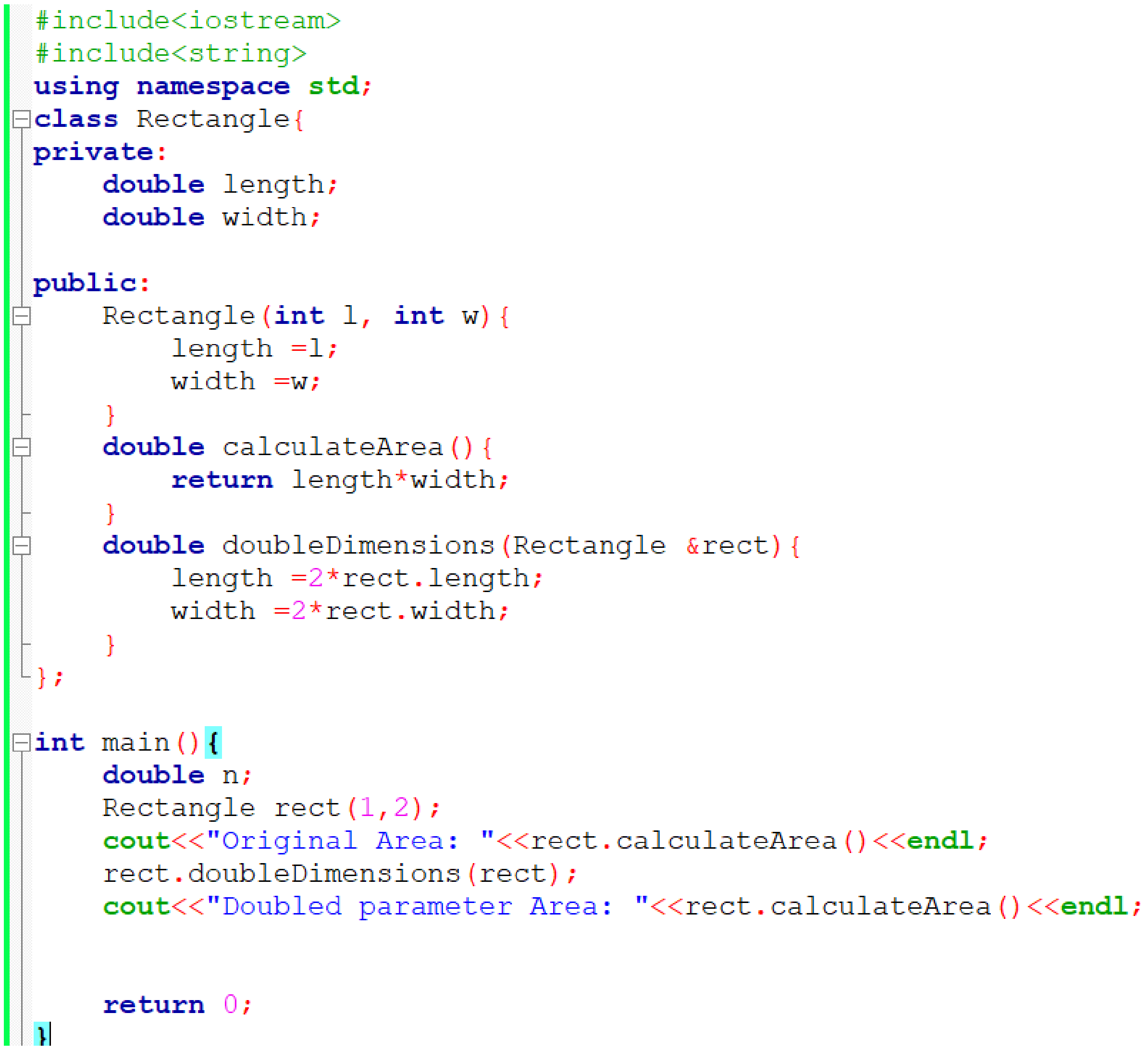
```cpp
#include<iostream>
#include<string>
using namespace std;
class Book{
private:
    string title ;

public:
    Book(string t){
        title =t;
    }
    ~Book(){
        cout<<"Destructor called"<<endl;
    }
    void display(){
        cout<<"Title: "<<title<<endl;
    }
};

int main(){
    Book b("Ankit");//constructor called
    b.display();
    return 0;//destructor called automatically
}
```

Output:



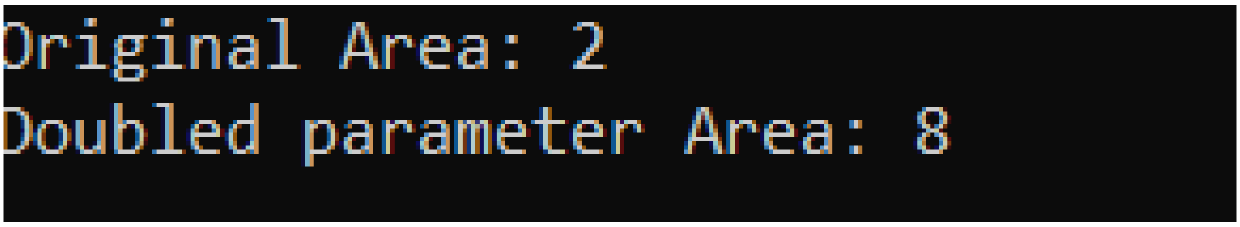
```
Title: Ankit
Destructor called
```

Q5. Define a class Rectangle with private members length and width. Implement a constructor to initialize these members. Write a function calculateArea() that calculates and returns the area of the rectangle. Define another function doubleDimensions(Rectangle rect) that takes a Rectangle object as an argument and doubles its length and width. Demonstrate the usage of both functions in main().



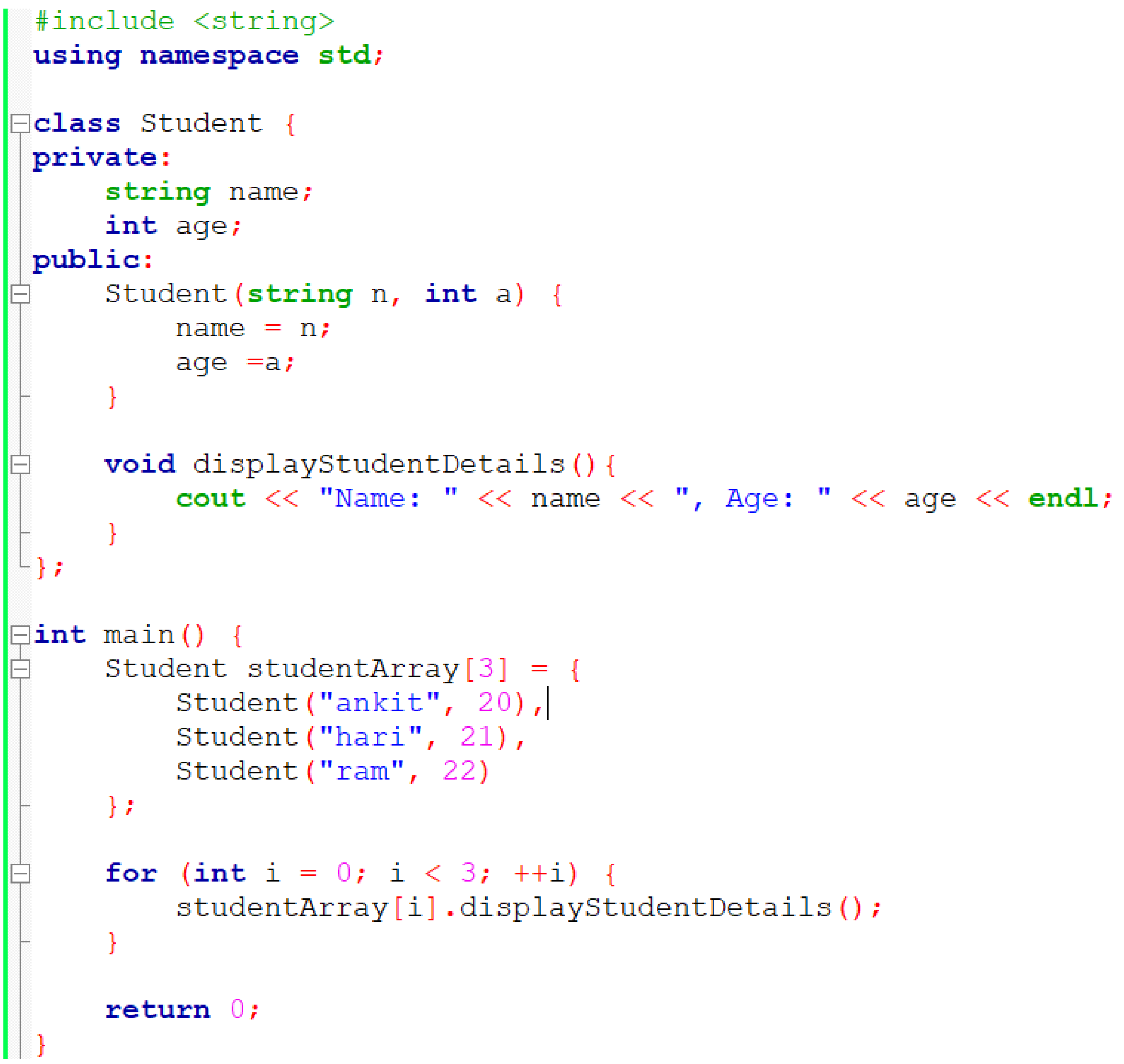
```
#include<iostream>
#include<string>
using namespace std;
class Rectangle{
private:
    double length;
    double width;

public:
    Rectangle(int l, int w){
        length =l;
        width =w;
    }
    double calculateArea(){
        return length*width;
    }
    double doubleDimensions(Rectangle &rect){
        length =2*rect.length;
        width =2*rect.width;
    }
};

int main(){
    double n;
    Rectangle rect(1,2);
    cout<<"Original Area: "<<rect.calculateArea()<<endl;
    rect.doubleDimensions(rect);
    cout<<"Doubled parameter Area: "<<rect.calculateArea()<<endl;


    return 0;
}
```

Output:



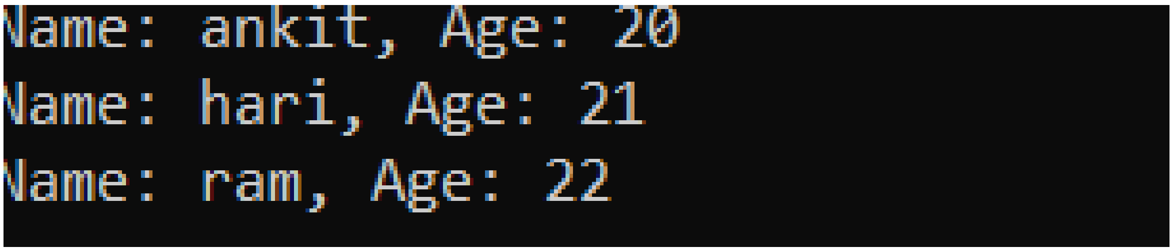
```
Original Area: 2
Doubled parameter Area: 8
```

- Q6. Define a class Student with private members name and age. Implement a constructor to initialize these members. Create an array studentArray of size 3 to store objects of type Student. Populate the array with student details and display the details using a method displayStudentDetails().



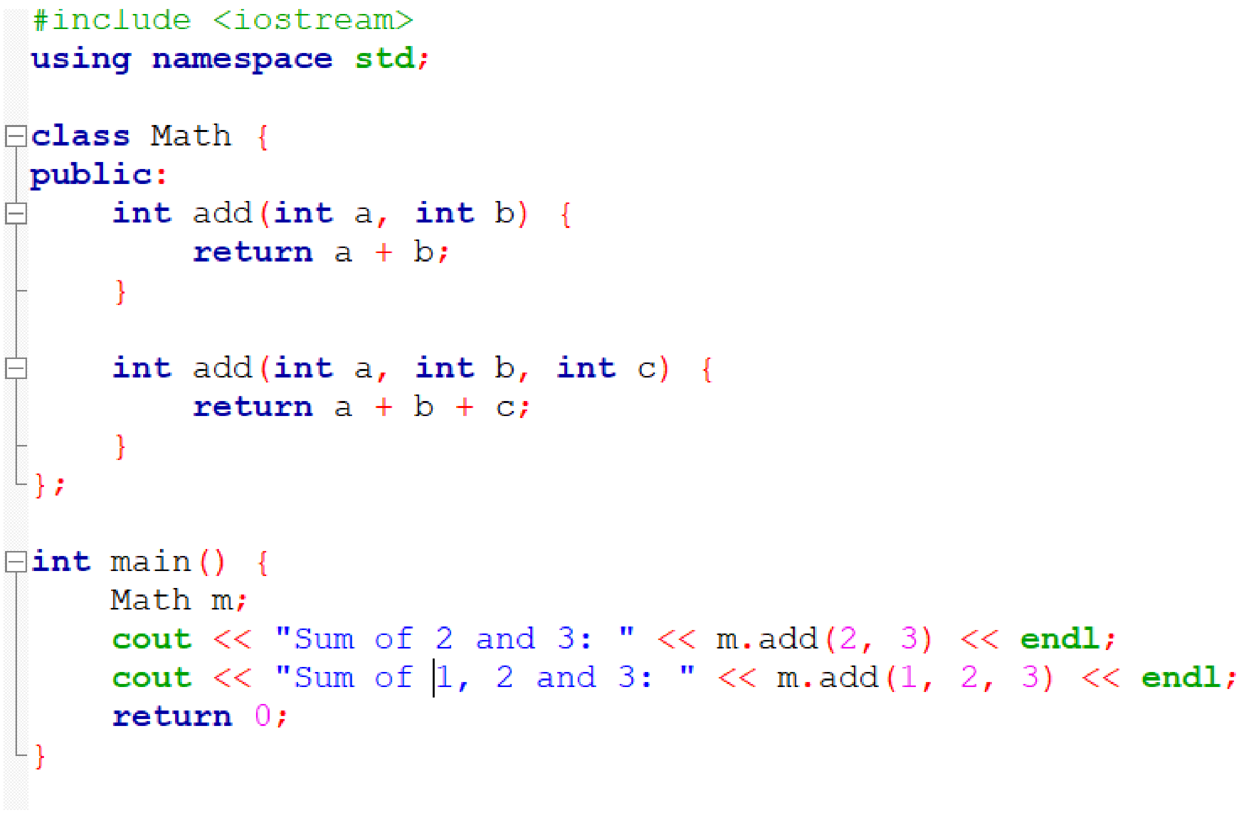
```cpp
#include <string>
using namespace std;

class Student {
private:
    string name;
    int age;
public:
    Student(string n, int a) {
        name = n;
        age =a;
    }

    void displayStudentDetails(){
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    Student studentArray[3] = {
        Student("ankit", 20),
        Student("hari", 21),
        Student("ram", 22)
    };

    for (int i = 0; i < 3; ++i) {
        studentArray[i].displayStudentDetails();
    }

    return 0;
}
```

Output.



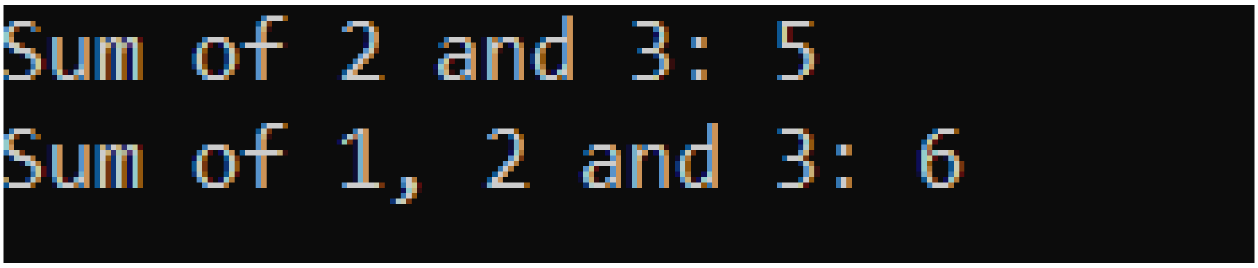
```
Name: ankit, Age: 20
Name: hari, Age: 21
Name: ram, Age: 22
```

- Q7. Define a class Math with two overloaded methods add(). The first method should take two integers as parameters and return their sum. The second method should take three integers as parameters and return their sum. Demonstrate the usage of both methods in main().



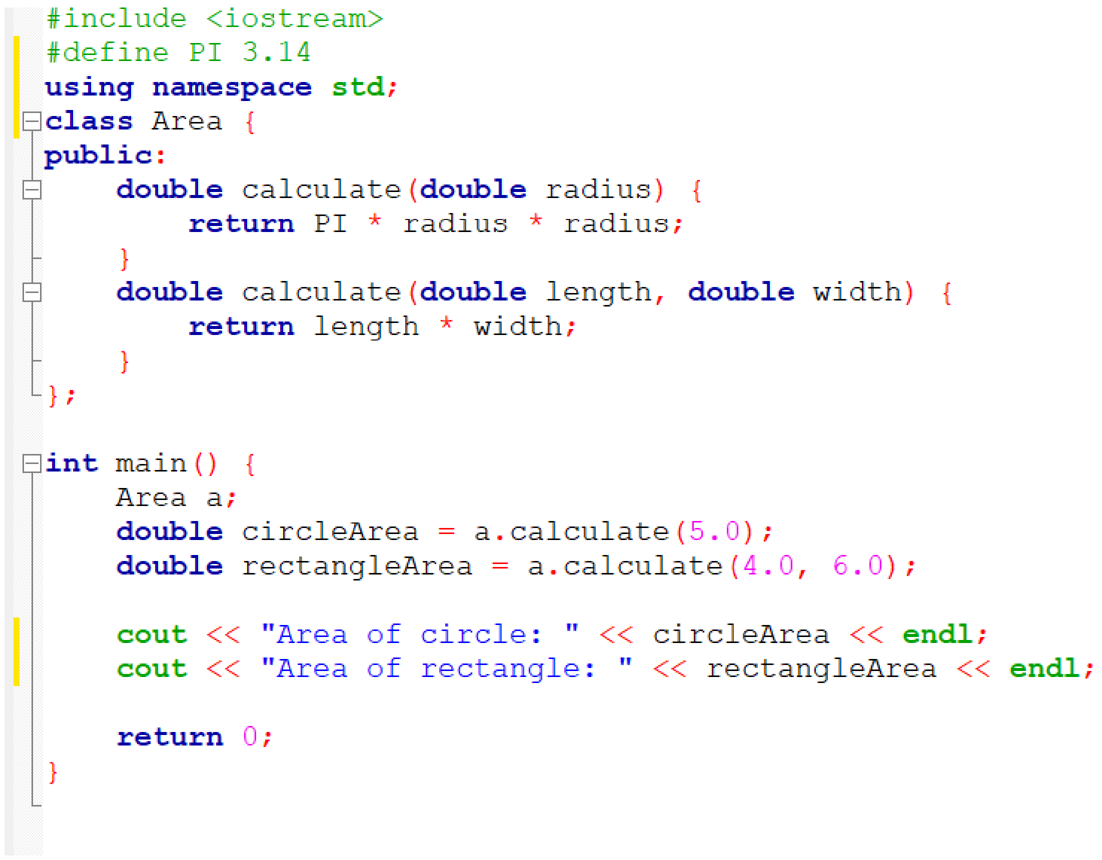
```cpp
#include <iostream>
using namespace std;

class Math {
public:
    int add(int a, int b) {
        return a + b;
    }

    int add(int a, int b, int c) {
        return a + b + c;
    }
};

int main() {
    Math m;
    cout << "Sum of 2 and 3: " << m.add(2, 3) << endl;
    cout << "Sum of 1, 2 and 3: " << m.add(1, 2, 3) << endl;
    return 0;
}
```

Output.



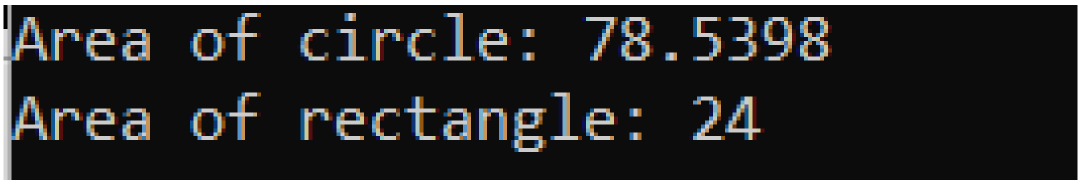
```
Sum of 2 and 3: 5
Sum of 1, 2 and 3: 6
```

- Q8. Implement a class Area with overloaded methods calculate(). The first method should take the radius of a circle as a parameter and return the area of the circle. The second method should take the length and width of a rectangle as parameters and return the area of the rectangle. Demonstrate the usage of both methods in main().



```cpp
#include <iostream>
#define PI 3.14
using namespace std;
class Area {
public:
    double calculate(double radius) {
        return PI * radius * radius;
    }
    double calculate(double length, double width) {
        return length * width;
    }
};

int main() {
    Area a;
    double circleArea = a.calculate(5.0);
    double rectangleArea = a.calculate(4.0, 6.0);

    cout << "Area of circle: " << circleArea << endl;
    cout << "Area of rectangle: " << rectangleArea << endl;

    return 0;
}
```

Output:



```
Area of circle: 78.5398
Area of rectangle: 24
```

DISCUSSION

In this lab session, we focused on classes and objects in C++, key concepts of object-oriented programming. We learned how constructors initialize objects and how destructors clean up resources when objects are destroyed. We explored three types of constructors: Default Constructor, Parameterized Constructor and Copy Constructor. Through examples like the LandMeasure class, we practiced using constructors to set initial values and destructors to release resources. This lab helped us understand how to manage object lifecycles efficiently in C++.Overall, we learned the significance of constructors and destructors for proper object initialization and memory management.

CONCLUSION

This lab session helped us solidify our understanding of class, object, constructor, and destructor concepts in C++. We learned how to effectively use these features to manage the initialization and cleanup of objects, ensuring that our programs are both functional and efficient. Understanding constructors and destructors is crucial for managing resources in C++, especially in complex programs where memory management is key to maintaining performance. Through hands-on practice, we gained experience in working with different types of constructors and the importance of using destructors for resource management.