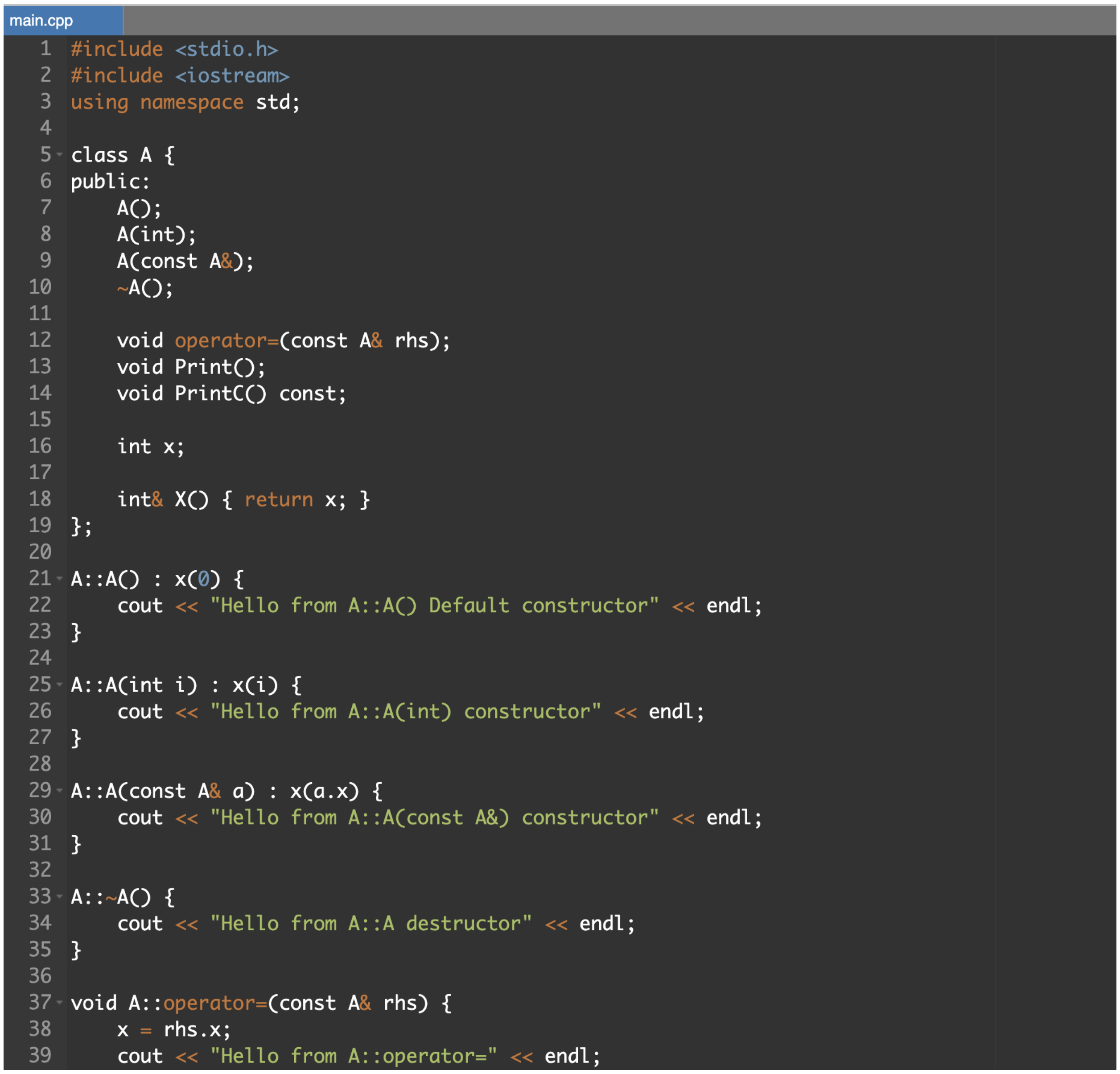
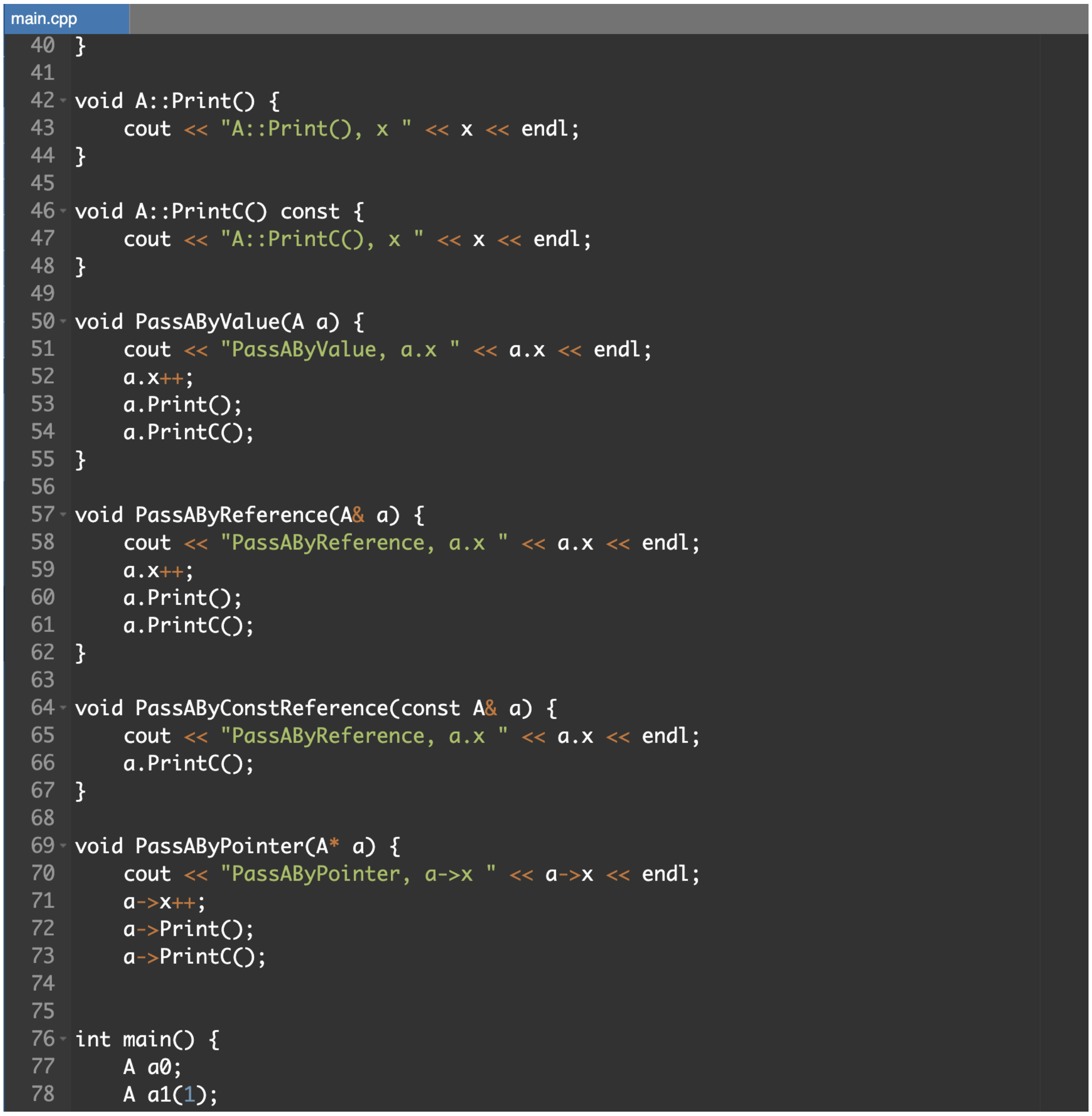
Samir Khadka
CS360L - Programming in C and C++ Lab
Lab Assignment #3

Question 1:

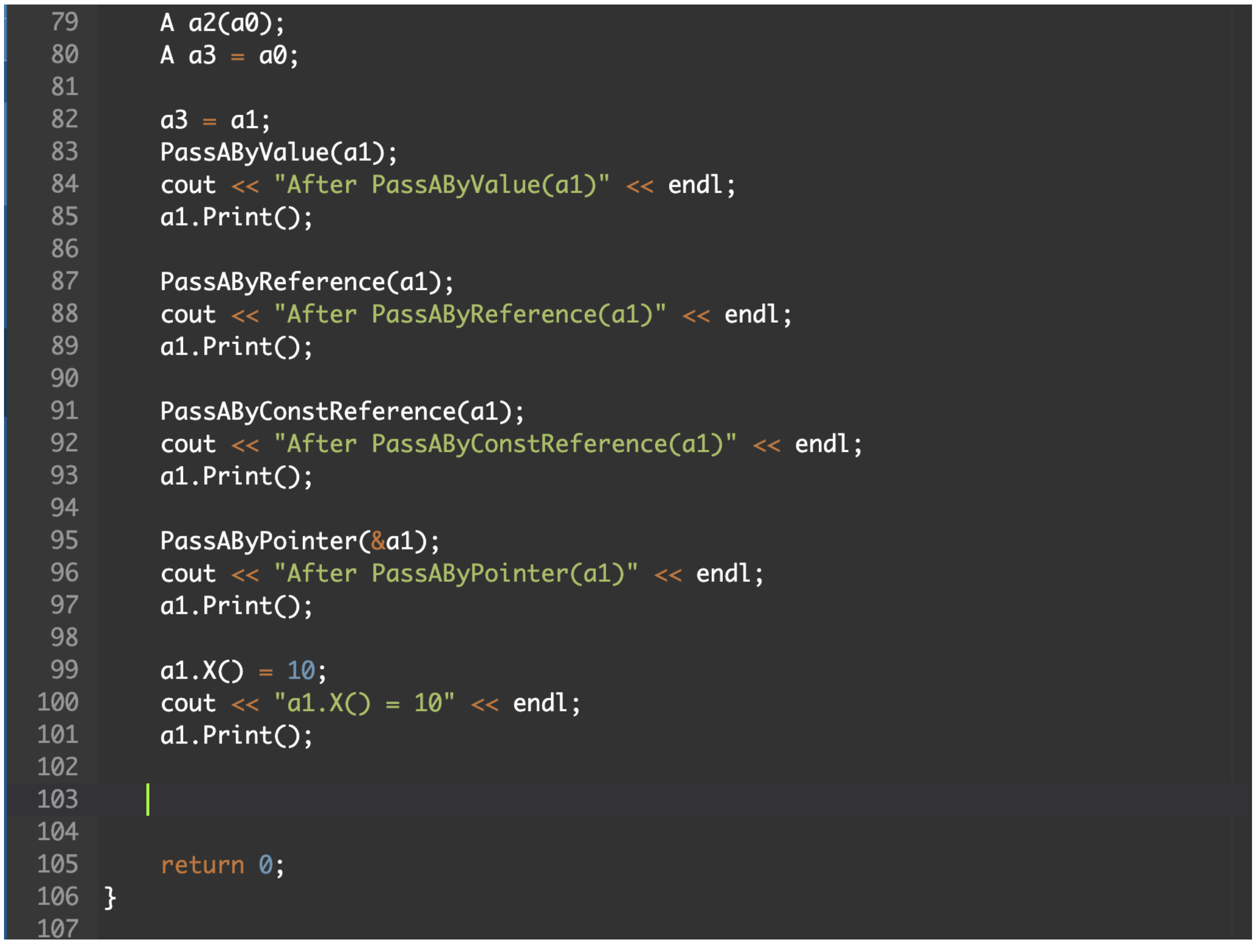

main.cpp

```cpp
#include <stdio.h>
#include <iostream>
using namespace std;

class A {
public:
    A();
    A(int);
    A(const A&);
    ~A();

    void operator=(const A& rhs);
    void Print();
    void PrintC() const;

    int x;

    int& X() { return x; }
};

A::A() : x(0) {
    cout << "Hello from A::A() Default constructor" << endl;
}

A::A(int i) : x(i) {
    cout << "Hello from A::A(int) constructor" << endl;
}

A::A(const A& a) : x(a.x) {
    cout << "Hello from A::A(const A&) constructor" << endl;
}

A::~A() {
    cout << "Hello from A::A destructor" << endl;
}

void A::operator=(const A& rhs) {
    x = rhs.x;
    cout << "Hello from A::operator=" << endl;
```

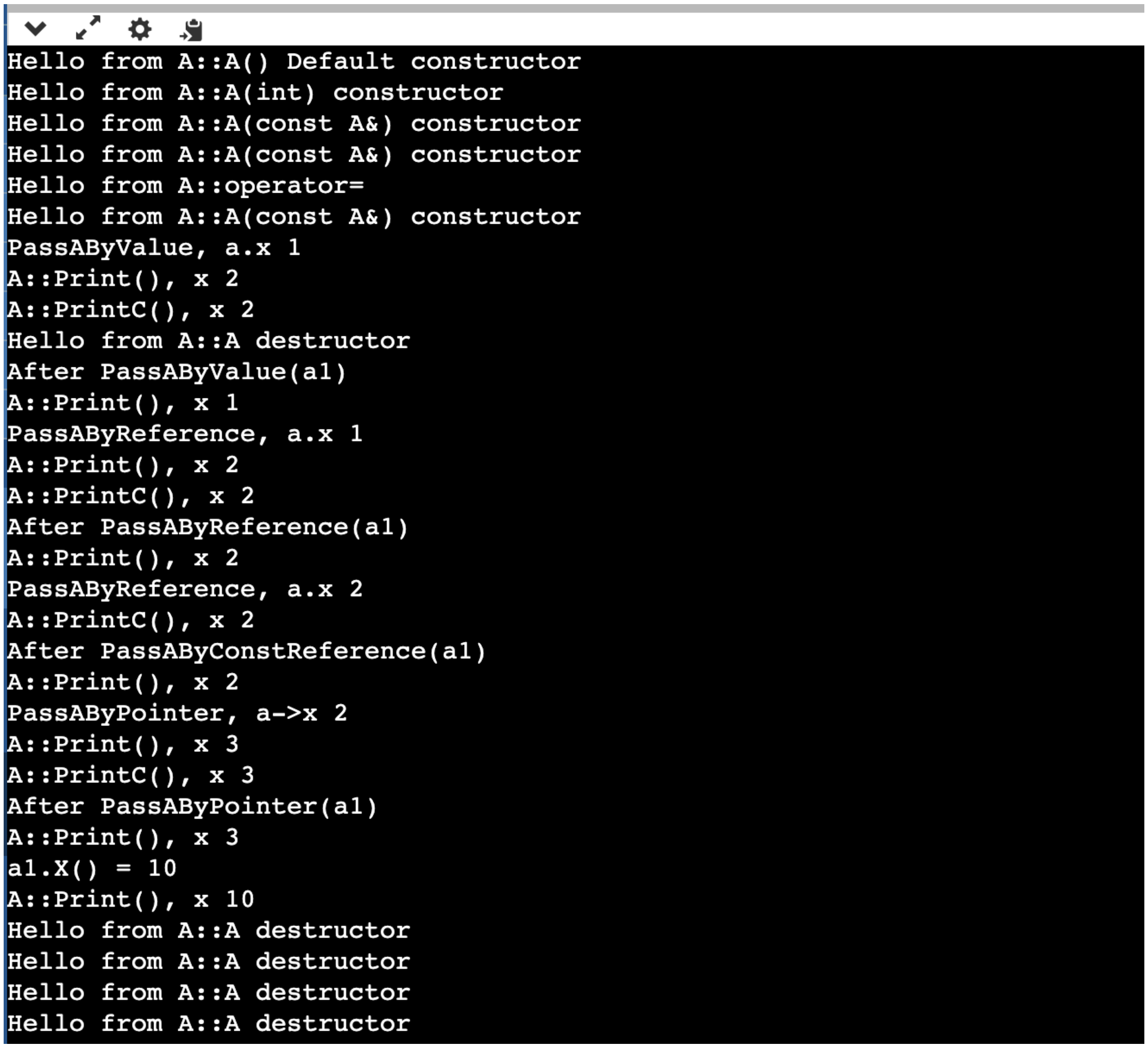


main.cpp

```cpp
}

void A::Print() {
    cout << "A::Print(), x " << x << endl;
}

void A::PrintC() const {
    cout << "A::PrintC(), x " << x << endl;
}

void PassAByValue(A a) {
    cout << "PassAByValue, a.x " << a.x << endl;
    a.x++;
    a.Print();
    a.PrintC();
}

void PassAByReference(A& a) {
    cout << "PassAByReference, a.x " << a.x << endl;
    a.x++;
    a.Print();
    a.PrintC();
}

void PassAByConstReference(const A& a) {
    cout << "PassAByReference, a.x " << a.x << endl;
    a.PrintC();
}

void PassAByPointer(A* a) {
    cout << "PassAByPointer, a->x " << a->x << endl;
    a->x++;
    a->Print();
    a->PrintC();


int main() {
    A a0;
    A a1(1);
```

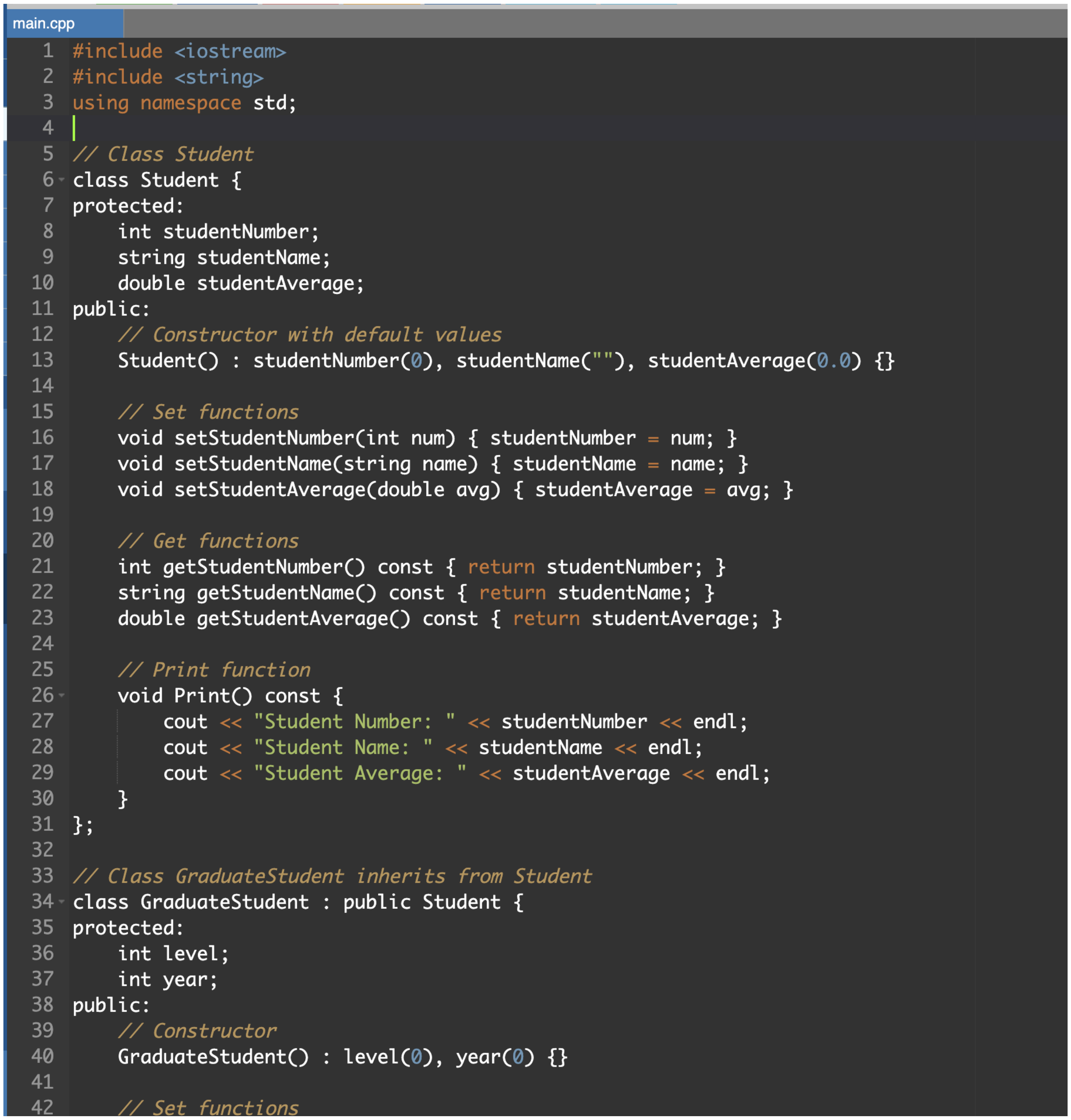


```
    A a2(a0);
    A a3 = a0;

    a3 = a1;
    PassAByValue(a1);
    cout << "After PassAByValue(a1)" << endl;
    a1.Print();

    PassAByReference(a1);
    cout << "After PassAByReference(a1)" << endl;
    a1.Print();

    PassAByConstReference(a1);
    cout << "After PassAByConstReference(a1)" << endl;
    a1.Print();

    PassAByPointer(&a1);
    cout << "After PassAByPointer(a1)" << endl;
    a1.Print();

    a1.X() = 10;
    cout << "a1.X() = 10" << endl;
    a1.Print();



    return 0;
}
```

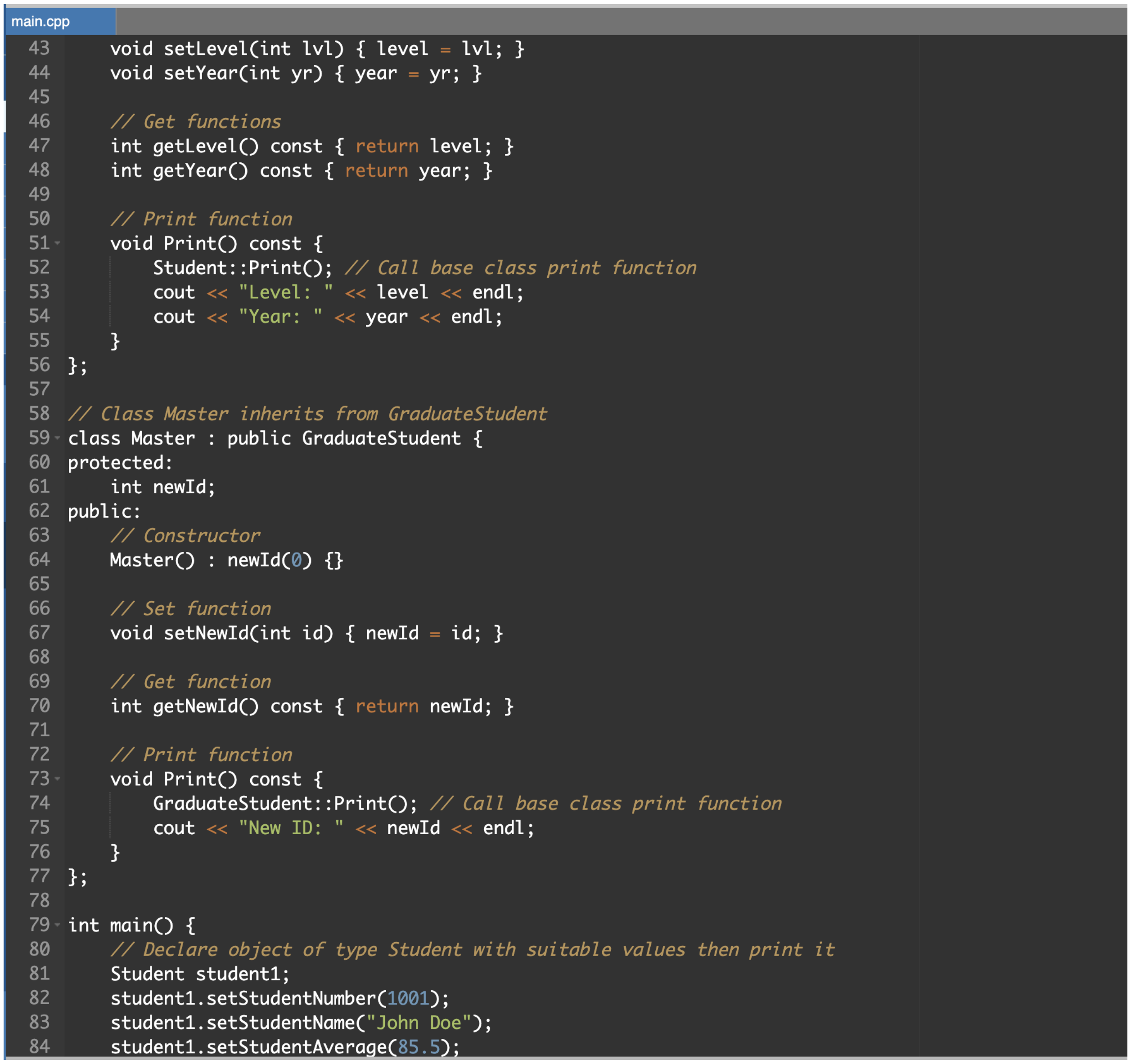


```
Hello from A::A() Default constructor
Hello from A::A(int) constructor
Hello from A::A(const A&) constructor
Hello from A::A(const A&) constructor
Hello from A::operator=
Hello from A::A(const A&) constructor
PassAByValue, a.x 1
A::Print(), x 2
A::PrintC(), x 2
Hello from A::A destructor
After PassAByValue(a1)
A::Print(), x 1
PassAByReference, a.x 1
A::Print(), x 2
A::PrintC(), x 2
After PassAByReference(a1)
A::Print(), x 2
PassAByReference, a.x 2
A::PrintC(), x 2
After PassAByConstReference(a1)
A::Print(), x 2
PassAByPointer, a->x 2
A::Print(), x 3
A::PrintC(), x 3
After PassAByPointer(a1)
A::Print(), x 3
a1.X() = 10
A::Print(), x 10
Hello from A::A destructor
Hello from A::A destructor
Hello from A::A destructor
Hello from A::A destructor
```

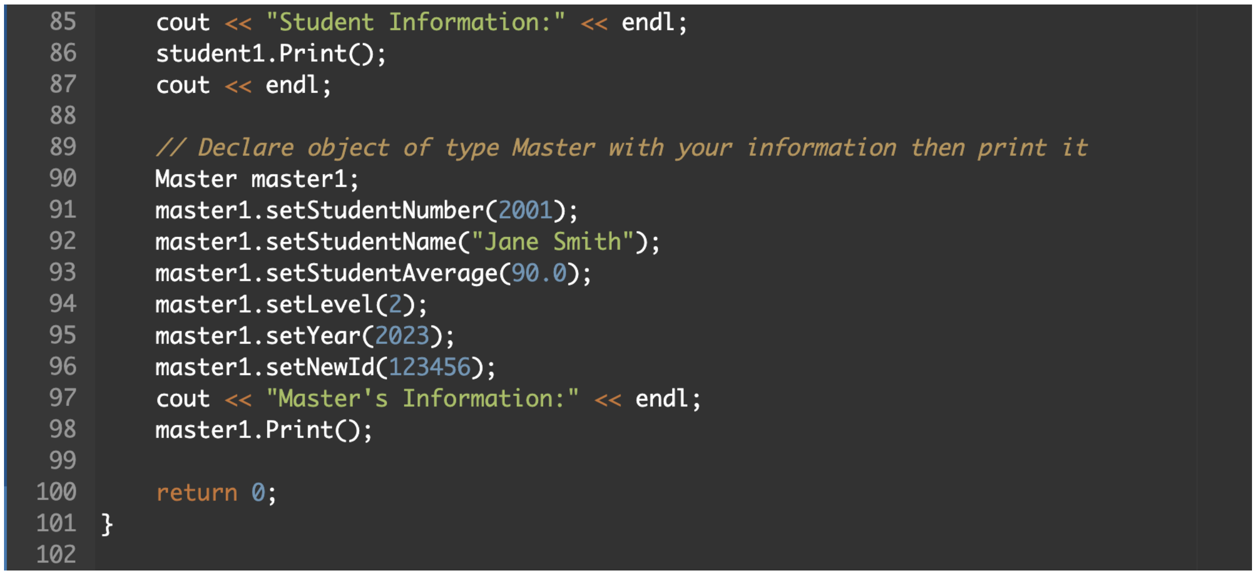
Question 2:


main.cpp

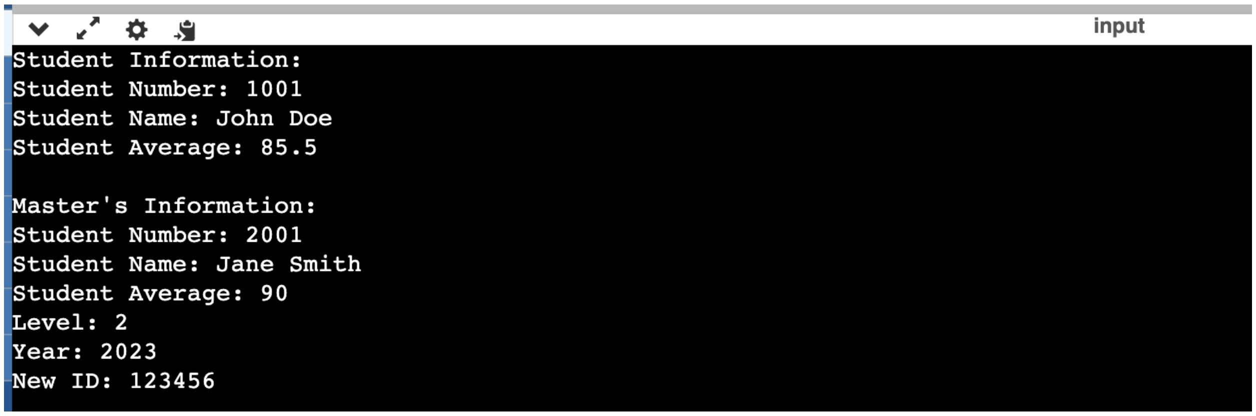
```cpp
#include <iostream>
#include <string>
using namespace std;

// Class Student
class Student {
protected:
    int studentNumber;
    string studentName;
    double studentAverage;
public:
    // Constructor with default values
    Student() : studentNumber(0), studentName(""), studentAverage(0.0) {}

    // Set functions
    void setStudentNumber(int num) { studentNumber = num; }
    void setStudentName(string name) { studentName = name; }
    void setStudentAverage(double avg) { studentAverage = avg; }

    // Get functions
    int getStudentNumber() const { return studentNumber; }
    string getStudentName() const { return studentName; }
    double getStudentAverage() const { return studentAverage; }

    // Print function
    void Print() const {
        cout << "Student Number: " << studentNumber << endl;
        cout << "Student Name: " << studentName << endl;
        cout << "Student Average: " << studentAverage << endl;
    }
};

// Class GraduateStudent inherits from Student
class GraduateStudent : public Student {
protected:
    int level;
    int year;
public:
    // Constructor
    GraduateStudent() : level(0), year(0) {}

    // Set functions
```


main.cpp

```cpp
    void setLevel(int lvl) { level = lvl; }
    void setYear(int yr) { year = yr; }

    // Get functions
    int getLevel() const { return level; }
    int getYear() const { return year; }

    // Print function
    void Print() const {
        Student::Print(); // Call base class print function
        cout << "Level: " << level << endl;
        cout << "Year: " << year << endl;
    }
};

// Class Master inherits from GraduateStudent
class Master : public GraduateStudent {
protected:
    int newId;
public:
    // Constructor
    Master() : newId(0) {}

    // Set function
    void setNewId(int id) { newId = id; }

    // Get function
    int getNewId() const { return newId; }

    // Print function
    void Print() const {
        GraduateStudent::Print(); // Call base class print function
        cout << "New ID: " << newId << endl;
    }
};

int main() {
    // Declare object of type Student with suitable values then print it
    Student student1;
    student1.setStudentNumber(1001);
    student1.setStudentName("John Doe");
    student1.setStudentAverage(85.5);
```

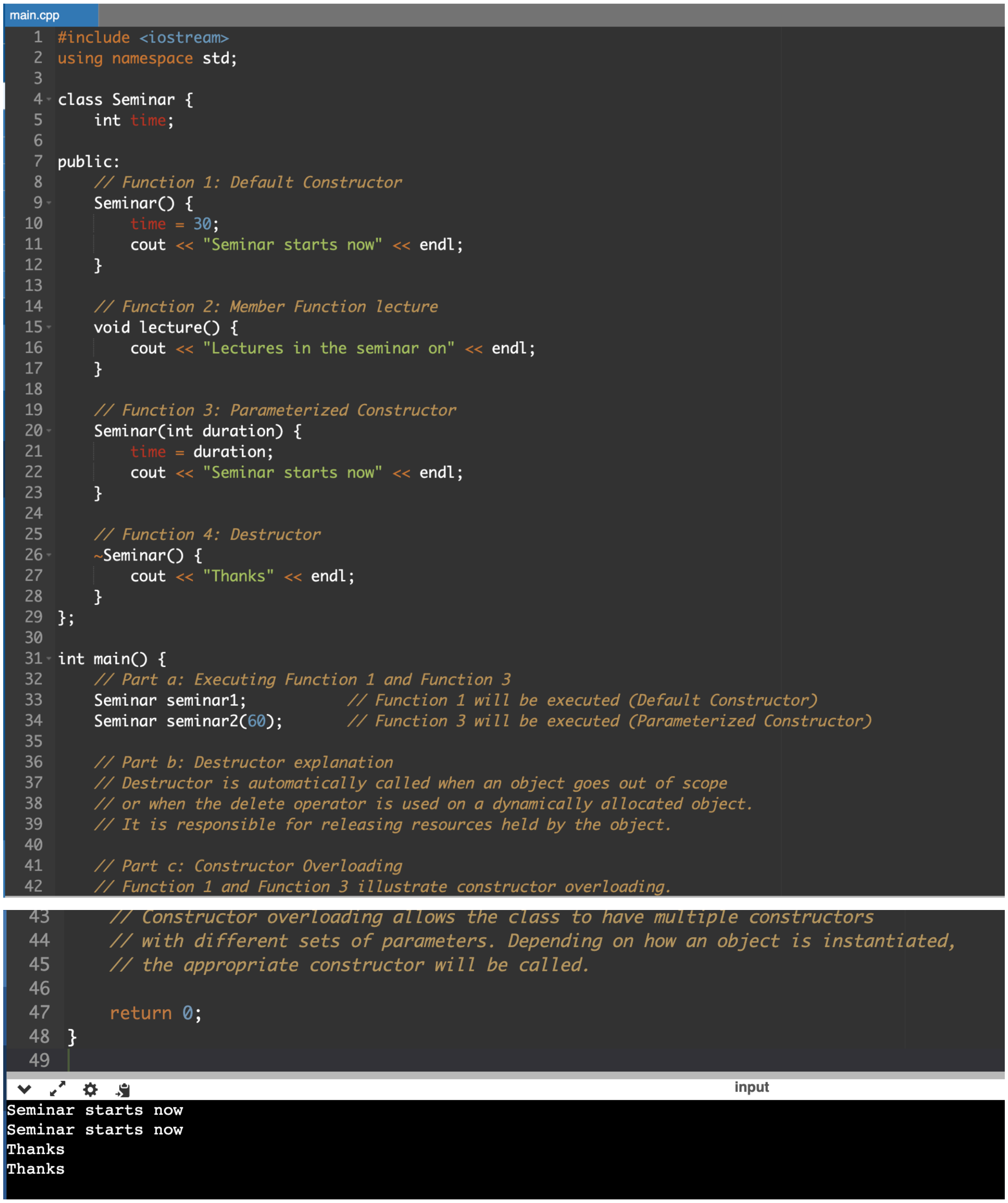


```cpp
    cout << "Student Information:" << endl;
    student1.Print();
    cout << endl;

    // Declare object of type Master with your information then print it
    Master master1;
    master1.setStudentNumber(2001);
    master1.setStudentName("Jane Smith");
    master1.setStudentAverage(90.0);
    master1.setLevel(2);
    master1.setYear(2023);
    master1.setNewId(123456);
    cout << "Master's Information:" << endl;
    master1.Print();

    return 0;
}
```



```
Student Information:
Student Number: 1001
Student Name: John Doe
Student Average: 85.5

Master's Information:
Student Number: 2001
Student Name: Jane Smith
Student Average: 90
Level: 2
Year: 2023
New ID: 123456
```

Question 3:



main.cpp

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

class Seminar {
    int time;

public:
    // Function 1: Default Constructor
    Seminar() {
        time = 30;
        cout << "Seminar starts now" << endl;
    }

    // Function 2: Member Function lecture
    void lecture() {
        cout << "Lectures in the seminar on" << endl;
    }

    // Function 3: Parameterized Constructor
    Seminar(int duration) {
        time = duration;
        cout << "Seminar starts now" << endl;
    }

    // Function 4: Destructor
    ~Seminar() {
        cout << "Thanks" << endl;
    }
};

int main() {
    // Part a: Executing Function 1 and Function 3
    Seminar seminar1;            // Function 1 will be executed (Default Constructor)
    Seminar seminar2(60);        // Function 3 will be executed (Parameterized Constructor)

    // Part b: Destructor explanation
    // Destructor is automatically called when an object goes out of scope
    // or when the delete operator is used on a dynamically allocated object.
    // It is responsible for releasing resources held by the object.

    // Part c: Constructor Overloading
    // Function 1 and Function 3 illustrate constructor overloading.
    // Constructor overloading allows the class to have multiple constructors
    // with different sets of parameters. Depending on how an object is instantiated,
    // the appropriate constructor will be called.

    return 0;
}

```

```
Seminar starts now
Seminar starts now
Thanks
Thanks
```

Question 4:

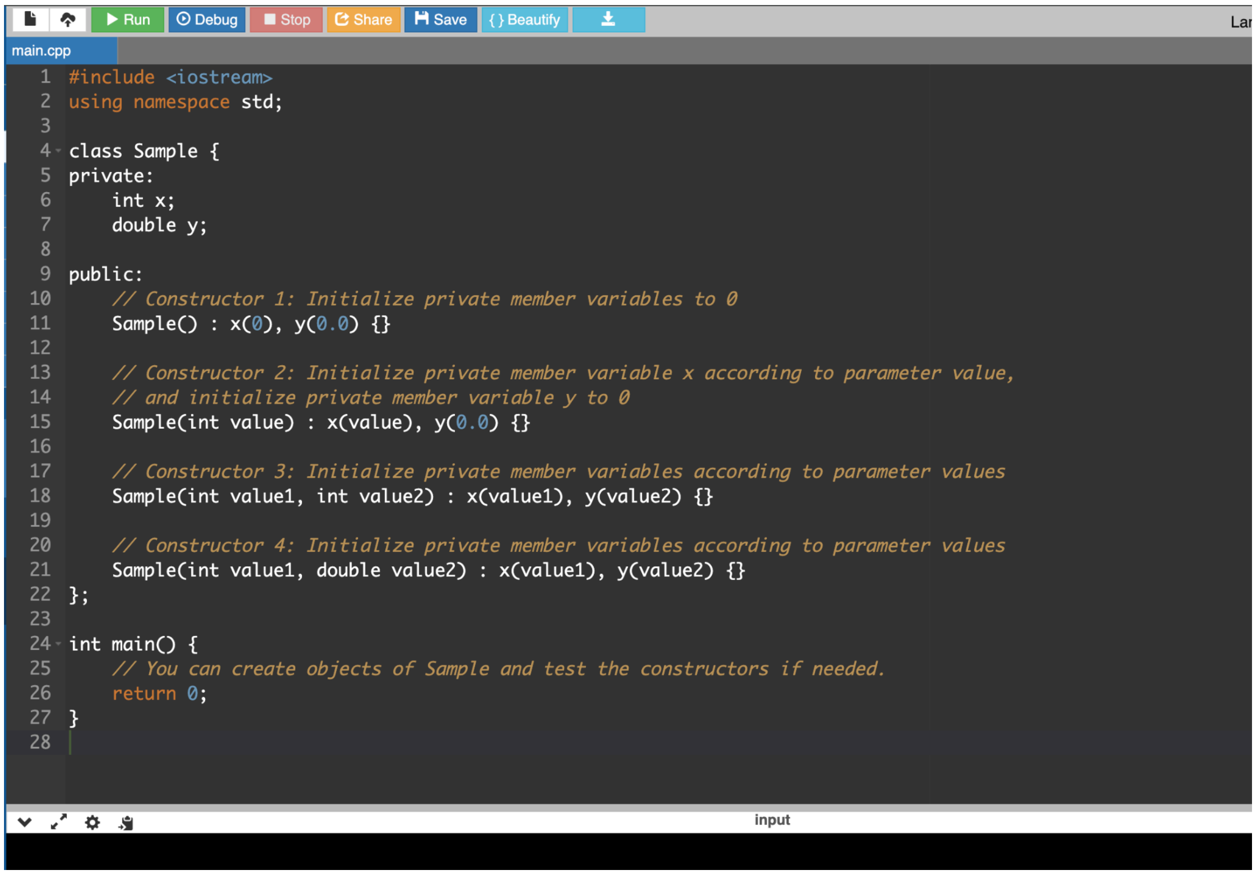
a.

```
Test test1;                // Function 1: Default constructor
Test test2("Maths");       // Function 2: Constructor with char array parameter
Test test3(90);            // Function 3: Constructor with integer parameter
```

Test test4("Physics", 85); // Function 4: Constructor with char array and integer parameters

b. The feature of Object-Oriented Programming demonstrated by Function 1, Function 2, Function 3, and Function 4 together in the above class Test is constructor overloading.

Question 5:



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

class Sample {
private:
    int x;
    double y;

public:
    // Constructor 1: Initialize private member variables to 0
    Sample() : x(0), y(0.0) {}

    // Constructor 2: Initialize private member variable x according to parameter value,
    // and initialize private member variable y to 0
    Sample(int value) : x(value), y(0.0) {}

    // Constructor 3: Initialize private member variables according to parameter values
    Sample(int value1, int value2) : x(value1), y(value2) {}

    // Constructor 4: Initialize private member variables according to parameter values
    Sample(int value1, double value2) : x(value1), y(value2) {}
};

int main() {
    // You can create objects of Sample and test the constructors if needed.
    return 0;
}
```