**Question no.1**

→

#include <iostream> // Include the input/output stream library

#include <stdio.h> // Include the standard input/output library

using namespace std; // Use the standard namespace

class A { // Define a class named A

public:

A(); // Declare a default constructor

A(int); // Declare a constructor that takes an integer

A(const A &); // Declare a copy constructor that takes a constant reference to

// an object of class A

~A(); // Declare a destructor

public:

void

operator=(const A &rhs); // Declare an overloaded assignment operator that

// takes a constant reference to an object of class A

void Print(); // Declare a function that prints the value of x

void PrintC() const; // Declare a const function that prints the value of x

int x; // Declare an integer member variable

public:

int &X() {

return x;

} // Declare a function that returns a reference to x. This allows you to

// modify x directly.

};

A::A() // Define the default constructor

: x(0) // Initialize x to 0

{

cout << "Hello from A::A() Default constructor" << endl; // Print a message

}

A::A(int i) // Define the constructor that takes an integer

: x(i) // Initialize x to the passed integer

{

cout << "Hello from A::A(int) constructor" << endl; // Print a message

}

A::A(const A &a) // Define the copy constructor

: x(a.x) // Initialize x to the value of x in the passed object

{

cout << "Hello from A::A(const A&) constructor" << endl; // Print a message

}

A::~A() // Define the destructor

{

cout << "Hello from A::A destructor" << endl; // Print a message

}

void A::operator=(const A &rhs) // Define the overloaded assignment operator

{

x = rhs.x; // Copy the value of x from the right-hand side object to the

// current object

cout << "Hello from A::operator=" << endl; // Print a message

}

void A::Print() // Define the function that prints the value of x

{

cout << "A::Print(), x " << x << endl; // Print a message

}

void A::PrintC() const // Define the const function that prints the value of x

{

cout << "A::PrintC(), x " << x << endl; // Print a message

}

void PassAByValue(

A a) // Define a function that takes an object of class A by value

{

cout << "PassAByValue, a.x " << a.x << endl; // Print the value of x

a.x++; // Increment the value of x

a.Print(); // Call the Print function

a.PrintC(); // Call the PrintC function

}

void PassAByReference(

A &a) // Define a function that takes an object of class A by reference

{

cout << "PassAByReference, a.x " << a.x << endl; // Print the value of x

a.x++; // Increment the value of x

a.Print(); // Call the Print function

a.PrintC(); // Call the PrintC function

}

void PassAByConstReference(

const A &a) // Define a function that takes a constant reference to an

// object of class A

{

cout << "PassAByReference, a.x " << a.x << endl; // Print the value of x

a.PrintC(); // Call the PrintC function

// a.Print(); // This line would cause a compiler error because Print is not a

// const function

}

void PassAByPointer(

A \*a) // Define a function that takes a pointer to an object of class A

{

cout << "PassAByPointer, a->x " << a->x << endl; // Print the value of x

a->x++; // Increment the value of x

a->Print(); // Call the Print function

a->PrintC(); // Call the PrintC function

}

int main() // Define the main function

{

cout << "Creating a0";

getchar(); // Print a message and wait for an input from the user

A a0; // Create an object of class A using the default constructor

cout << "Creating a1";

getchar(); // Print a message and wait for an input from the user

A a1(1); // Create an object of class A using the constructor that takes an

// integer

cout << "Creating a2";

getchar(); // Print a message and wait for an input from the user

A a2(a0); // Create an object of class A using the copy constructor

cout << "Creating a3";

getchar(); // Print a message and wait for an input from the user

A a3 = a0; // Create an object of class A and assign it the value of a0

cout << "Assigning a3 = a1";

getchar(); // Print a message and wait for an input from the user

a3 = a1; // Assign the value of a1 to a3

cout << "PassAByValue(a1)";

getchar(); // Print a message and wait for an input from the user

PassAByValue(

a1); // Call the function that takes an object of class A by value

cout << "After PassAByValue(a1)" << endl; // Print a message

a1.Print(); // Call the Print function

cout << "PassAByReference(a1)";

getchar(); // Print a message and wait for an input from the user

PassAByReference(

a1); // Call the function that takes an object of class A by reference

cout << "After PassAByReference(a1)" << endl; // Print a message

a1.Print(); // Call the Print function

cout << "PassAByConst(a1)";

getchar(); // Print a message and wait for an input from the user

PassAByConstReference(a1); // Call the function that takes a constant

// reference to an object of class A

cout << "After PassAByConstReference(a1)" << endl; // Print a message

a1.Print(); // Call the Print function

cout << "PassAByPointer(&a1)";

getchar(); // Print a message and wait for an input from the user

PassAByPointer(

&a1); // Call the function that takes a pointer to an object of class A

cout << "After PassAByPointer(a1)" << endl; // Print a message

a1.Print(); // Call the Print function

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

getchar(); // Print a message and wait for an input from the user

a1.X() = 10; // Assign the value 10 to x in a1

a1.Print(); // Call the Print function

cout << "PassAByConstReference";

getchar(); // Print a message and wait for an input from the user

PassAByConstReference(

20); // Call the function that takes a constant reference to an object of

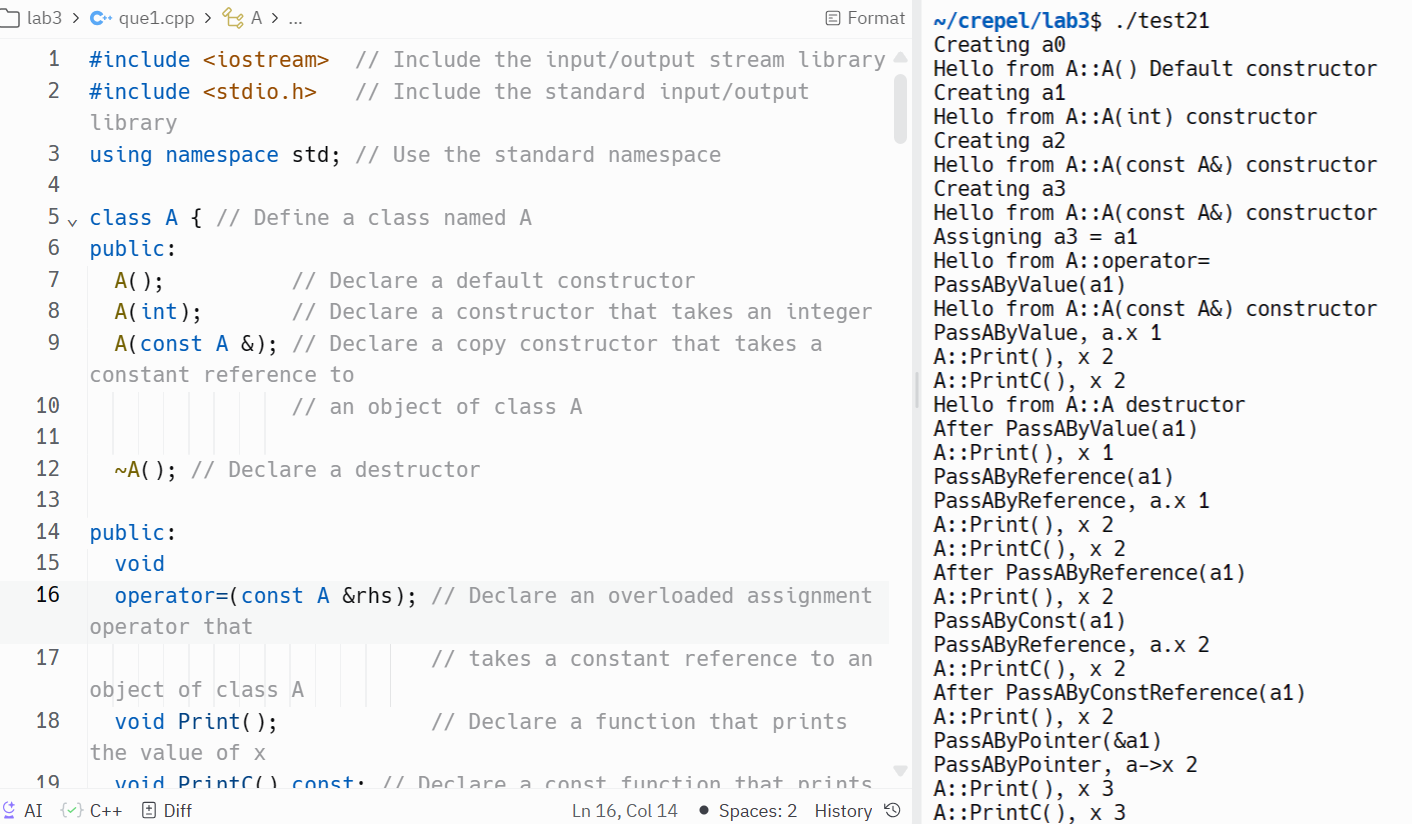
// class A with an integer. This will implicitly convert the integer

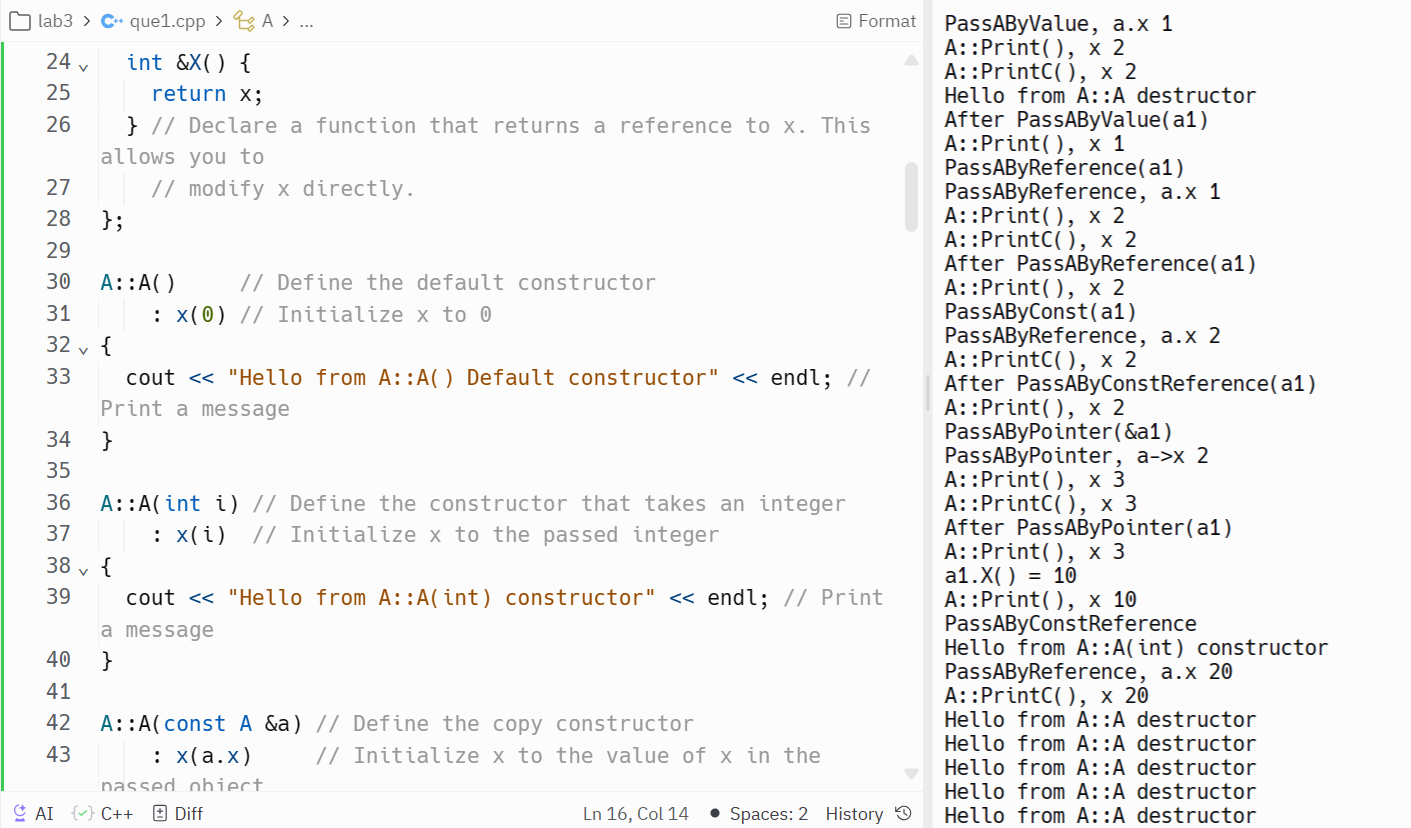
// to an object of class A using the constructor that takes an

// integer.

return 0; // Return 0 to indicate that the program has executed successfully

}





**Question no. 2**

#include <iostream>

#include <string>

using namespace std;

class Student {

public:

Student() : student\_number(0), student\_name(""), student\_average(0.0) {}

void setStudentNumber(int number) { student\_number = number; }

int getStudentNumber() { return student\_number; }

void setStudentName(string name) { student\_name = name; }

string getStudentName() { return student\_name; }

void setStudentAverage(double average) { student\_average = average; }

double getStudentAverage() { return student\_average; }

void print() {

cout << "Student Number: " << student\_number << endl;

cout << "Student Name: " << student\_name << endl;

cout << "Student Average: " << student\_average << endl;

}

private:

int student\_number;

string student\_name;

double student\_average;

};

class GraduateStudent : public Student {

public:

GraduateStudent() : level(0), year(0) {}

void setLevel(int lvl) { level = lvl; }

int getLevel() { return level; }

void setYear(int yr) { year = yr; }

int getYear() { return year; }

void print() {

Student::print();

cout << "Level: " << level << endl;

cout << "Year: " << year << endl;

}

private:

int level;

int year;

};

class Master : public GraduateStudent {

public:

Master() : newid(0) {}

void setNewId(int id) { newid = id; }

int getNewId() { return newid; }

void print() {

GraduateStudent::print();

cout << "New ID: " << newid << endl;

}

private:

int newid;

};

int main() {

Student s;

s.setStudentNumber(123);

s.setStudentName("John Doe");

s.setStudentAverage(85.5);

s.print();

cout << "\n-------------------------\n" << endl; // Separator

Master m;

m.setStudentNumber(456);

m.setStudentName("Jane Smith");

m.setStudentAverage(90.0);

m.setLevel(2);

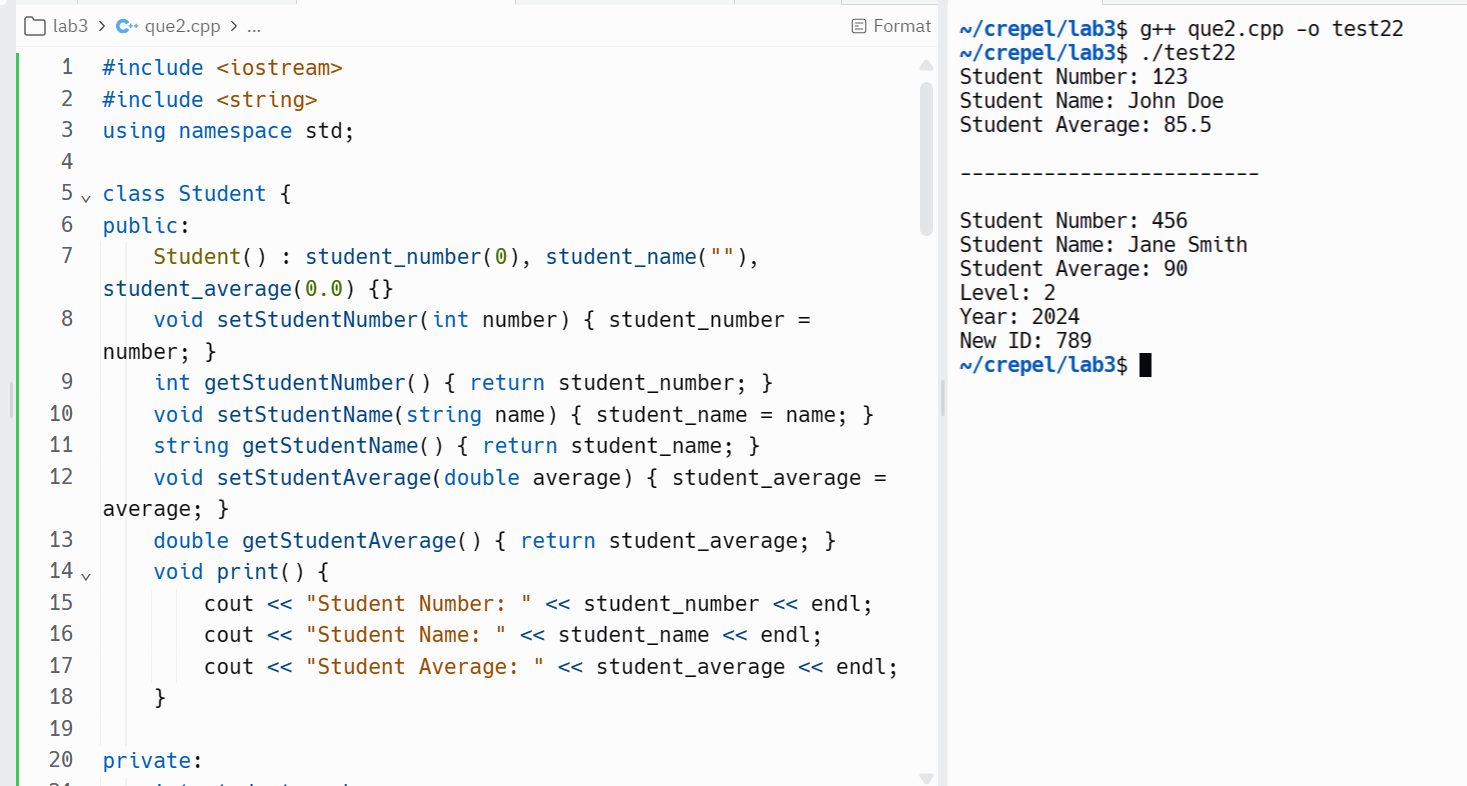
m.setYear(2024);

m.setNewId(789);

m.print();

return 0;

}



Question no. 3

#include <iostream>

using namespace std;

class Seminar {

int time;

public:

Seminar() // Function 1

{

time = 30;

cout << "Seminar starts now" << endl;

}

void lecture() // Function 2

{

cout << "Lectures in the seminar on" << endl;

}

Seminar(int duration) // Function 3

{

time = duration;

cout << "Seminar starts now" << endl;

}

~Seminar() // Function 4

{

cout << "Thanks" << endl;

}

};

int main() {

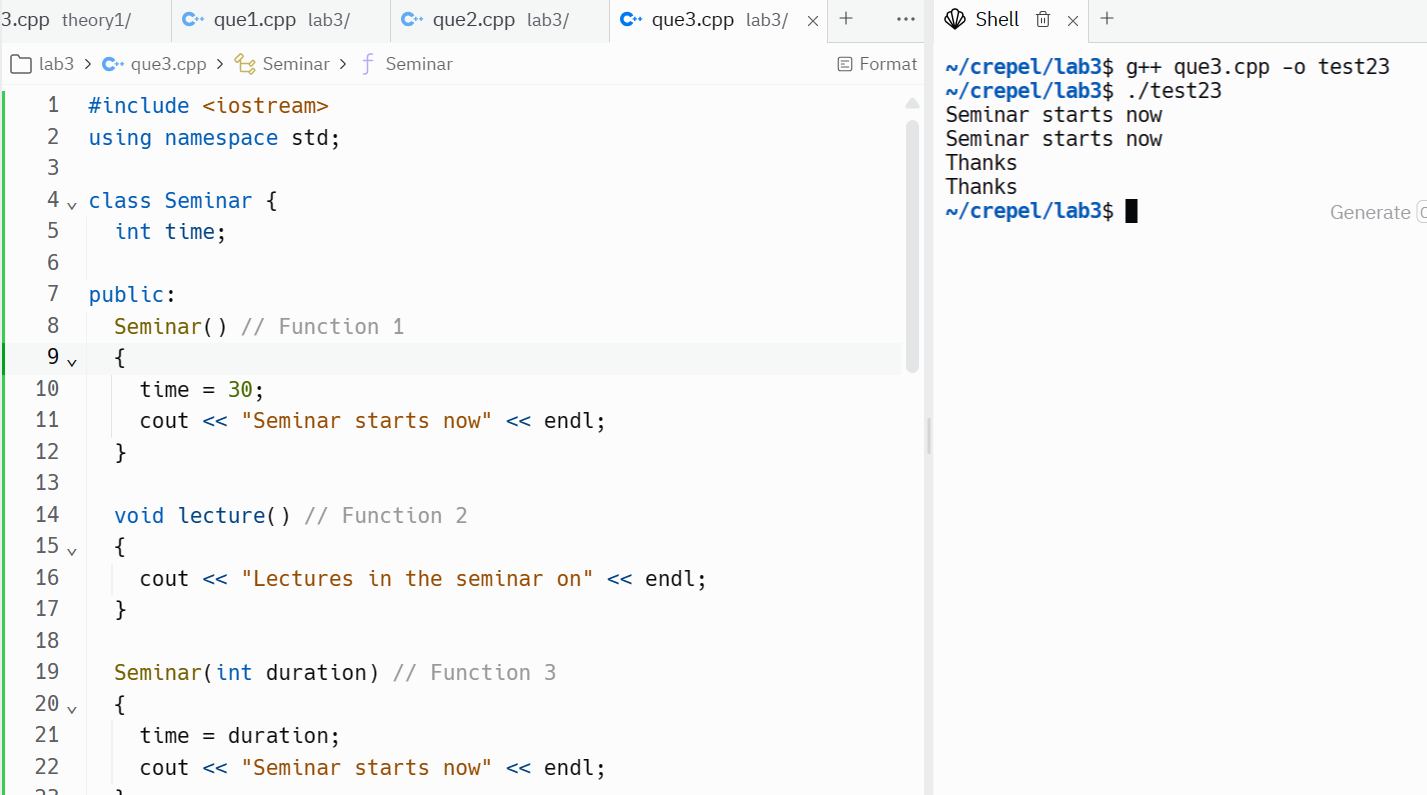
// Creating an instance of the Seminar class

Seminar seminar1; // Calls Function 1 (default constructor)

Seminar seminar2(45); // Calls Function 3 (parameterized constructor)

return 0;

}



1. in Object-Oriented Programming (OOP), Function 4 (the destructor) is referred to as the destructor. It gets invoked automatically when an object goes out of scope (e.g., when the program exits or when the object is explicitly destroyed). The destructor is responsible for cleaning up any resources associated with the object (e.g., releasing memory, closing files, etc.).
2. Function 1 (default constructor) and Function 3 (parameterized constructor) together illustrate the concept of constructor overloading. Constructor overloading allows a class to have multiple constructors with different parameter lists. Depending on how an object is created (with or without arguments), the appropriate constructor is called.

**Question no. 4**

#include <iostream>

#include <string>

#include <cstring>

using namespace std;

class Test {

char paper[20];

int marks;

public:

Test() {

strcpy(paper, "Computer");

marks = 0;

}

Test(char p[]) {

strcpy(paper, p);

marks = 0;

}

Test(int m) {

strcpy(paper, "Computer");

marks = m;

}

Test(char p[], int m) {

strcpy(paper, p);

marks = m;

}

void print() {

cout << "Paper: " << paper << ", Marks: " << marks << endl;

}

};

int main() {

// Create an object using the default constructor (Function 1)

Test t1;

t1.print();

// Create an object using the constructor that takes a char array (Function 2)

char paper[] = "Math";

Test t2(paper);

t2.print();

// Create an object using the constructor that takes an int (Function 3)

Test t3(90);

t3.print();

// Create an object using the constructor that takes a char array and an int (Function 4)

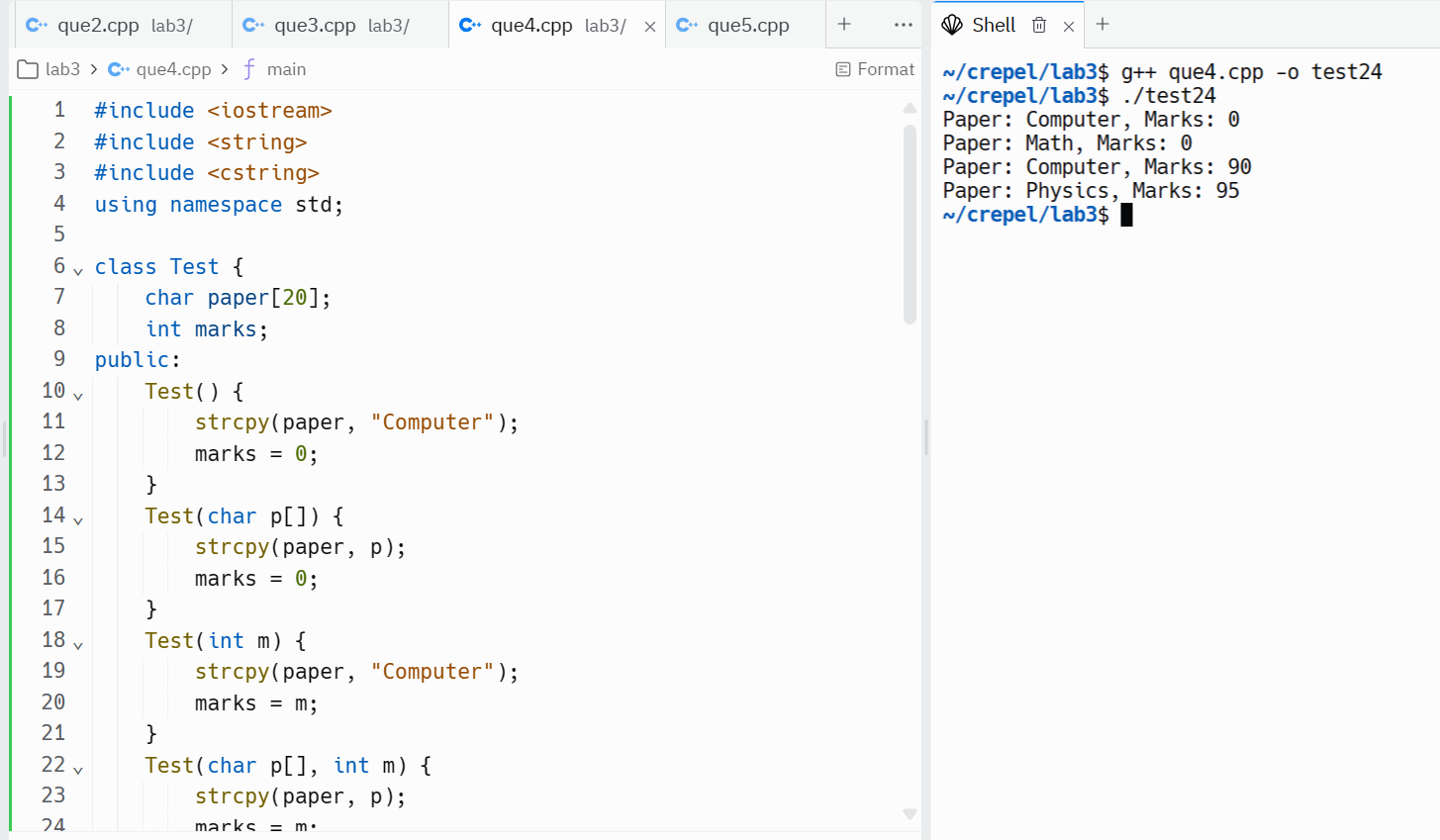
char paper2[] = "Physics";

Test t4(paper2, 95);

t4.print();

return 0;

}



**Question 5**

#include <iostream>

#include <cstring>

using namespace std;

class Sample {

private:

int x;

double y;

public:

Sample() : x(0), y(0.0) {} // Constructor 1

Sample(int x) : x(x), y(0.0) {} // Constructor 2

Sample(int x, int y) : x(x), y(static\_cast<double>(y)) {} // Constructor 3

Sample(int x, double y) : x(x), y(y) {} // Constructor 4

void print() {

cout << "x: " << x << ", y: " << y << endl;

}

};

int main() {

Sample s1; // Calls Constructor 1

s1.print();

Sample s2(10); // Calls Constructor 2

s2.print();

Sample s3(20, 30); // Calls Constructor 3

s3.print();

Sample s4(40, 50.5); // Calls Constructor 4

s4.print();

return 0;

}

