San Francisco Bay University

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

Due day: 3/8/2024

Tasmita Tanjim (19723)

REPLIT LINK: https://replit.com/@TASMITA-TANJIMT/LAB3-2

1. Analyze the following program and explain each statement and commented-down statements in red. Finally, run the program and type in appropriate inputs from standard input device to show the running results

```
#include <stdio.h>
#include <iostream>

using namespace std;

class A {
public:
  A();          //
  A(int);       //
  A(const A&); //
                //
  ~A();         //
public:
  void operator=(const A& rhs); //
  void Print();         //
  void PrintC() const; //
                       //
  int x;        //
public:
  //
  int& X() { return x; }
};

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

A::A() initializes x to 0 and prints "Hello from A::A()
Default constructor". It shows how to create an object
```

of class A with no arguments, assigning a default value to x.

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

# initializes x with the value of i and prints "Hello from A::A(int) constructor". This constructor allows for the creation of A objects with a specified initial value for x.


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

initializes x with the value of a.x and prints "Hello from A::A(const A&) constructor".
It is used to create a new object as a copy of an existing object.


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

A::~A() prints "Hello from A::A destructor" when an A object is destroyed; used for cleanup before the object's memory is reclaimed

void A::operator=(const A& rhs)
{
  x = rhs.x;
  cout << "Hello from A::operator=" << endl;
}
void A::operator=(const A& rhs) assigns the value of rhs.x to x and prints "Hello from A::operator=". This
```

```
operator is used to copy the value from one A object to
another.



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

void A::Print() prints the current value of x for non-
const objects

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

void A::PrintC() const prints the current value of x
for const objects.


void PassAByValue(A a)

//PassAByValue, an object of class A is passed by value.
This means a copy of the object is made using the copy
constructor

{
  cout << "PassAByValue, a.x " << a.x << endl;
  a.x++;  // increments the x value of the copy of the
object a
  a.Print(); // prints the incremented value of x
  a.PrintC(); // prints the value of x but is marked as
const
}

void PassAByReference(A& a)
//an object of class A is passed by reference.
{
  cout << "PassAByReference, a.x " << a.x << endl;
  a.x++;  // increments the x value of the original
object a
  a.Print(); // prints the incremented value of x
  a.PrintC(); // prints the value of x, and it can be
called on both const and non-const objects
}
```

```
void PassAByConstReference(const A& a)
{
  cout << "PassAByReference, a.x " << a.x << endl;
//it should probably be "PassAByConstReference" to
match the function's purpose

  a. PrintC();  // a const member function

  //a.Print(); // Call to "non-const" print function
  fails!

  // because calling it would result in a compiler error.
  The reason is that a.Print() is a non-const member
  function which means it could potentially modify the
  object.

  // Compiler error from above line.  Why?
}

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

// function receives a pointer to an object of type A.
It directly manipulates the object a points to.

  a->x++;
  a->Print();
  a->PrintC();
}


int main()
{
  cout << "Creating a0"; getchar();
  A a0;    // creates an instance of A using the default
constructor, initializing x to 0 and printing the
default constructor message.

  cout << "Creating a1"; getchar();
  A a1(1); // uses the parameterized constructor to
create an instance of A with x initialized to 1

  cout << "Creating a2"; getchar();
```

```
  A a2(a0); // creates an instance of A by copying a0,
using the copy constructor.

  cout << "Creating a3"; getchar();
  A a3 = a0; // creates a copy of a0 and uses direct
initialization syntax.

  cout << "Assigning a3 = a1"; getchar();
  a3 = a1; // Assigns the value of a1 to a3 using the
assignment operator, changing a3's x to match a1's x


  // Call some of the "A" subroutines
  cout << "PassAByValue(a1)"; getchar();
  PassAByValue(a1); // A copy of a1 is created and
manipulated within PassAByValue.

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

  cout << "PassAByReference(a1)"; getchar();
  PassAByReference(a1); // a1 is directly passed to the
function, allowing it to modify the original object.

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

  cout << "PassAByConst(a1)"; getchar();
  PassAByConstReference(a1); // Shows passing an object
by const reference. This approach passes a1 directly
but prohibits modification
  cout << "After PassAByConstReference(a1)" << endl;
  a1.Print();

  cout << "PassAByPointer(&a1)"; getchar();
  PassAByPointer(&a1); // Demonstrates passing a pointer
to an object. This method allows direct manipulation of
a1 through its pointer.
  cout << "After PassAByPointer(a1)" << endl;
  a1.Print();

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

  cout << "PassAByConstReference"; getchar();
```

```
    PassAByConstReference(20);

    // Why does the above compile?  What does it do?
  // The line PassAByConstReference(20); compiles because
  C++ allows implicit construction of a temporary object
  of class A using the A(int) constructor with 20.

    return 0;
  }
```

2. Write the program based on the following requirements

a. Define a class called *student* that has the following data members:
   i. *int* student number
   ii. *string* student name
   iii. *double* student average

b. The following member functions:
   i. Constructor that initialize the data members with default values.
   ii. *set* and *get* functions for each data member
   iii. *Print* function to print the values of data members.

c. Define a class called *graduatestudent* that inherits data members and functions from the class *student*, and then declare the following data members :
   i. int level
   ii. int year

d. Member functions:
   i. constructor
   ii. *set* and *get* functions for each data member
   iii. Print function.

e. Define a class called *master* that inherits data members and functions from *graduatestudent* class, and then declare the following data member:
   i. int newid

f. Member function:
   i. constructor
   ii. *set* and *get* function for the data member
   iii. *Print* function

g. Write a driver program(i.e. main function) that:
   i. Declare object of type student with suitable values then print it
   ii. Declare object of type master with your information then print it.

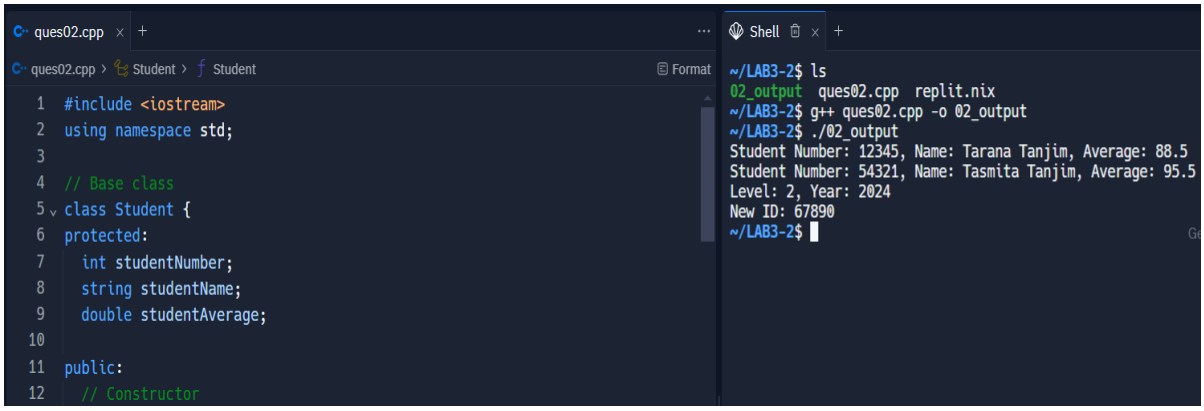
ANSWER:

CODE:

```
#include <iostream>
```

```
using namespace std;

// Base class
class Student {
protected:
  int studentNumber;
  string studentName;
  double studentAverage;

public:
  // Constructor
  Student(int number = 0, string name = "", double average = 0.0)
     : studentNumber(number), studentName(name), studentAverage(average) {}

  // Set and get functions for each data member
  void setStudentNumber(int number) { studentNumber = number; }

  int getStudentNumber() const { return studentNumber; }

  void setStudentName(string name) { studentName = name; }

  string getStudentName() const { return studentName; }

  void setStudentAverage(double average) { studentAverage = average; }

  double getStudentAverage() const { return studentAverage; }

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

// Derived class
class GraduateStudent : public Student {
protected:
  int level;
  int year;

public:
  // Constructor
  GraduateStudent(int number = 0, string name = "", double average = 0.0,
                  int lvl = 0, int yr = 0)
     : Student(number, name, average), level(lvl), year(yr) {}

  // Set and get functions for each new data member
  void setLevel(int lvl) { level = lvl; }

  int getLevel() const { return level; }

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

```
  int getYear() const { return year; }

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

// Another derived class
class Master : public GraduateStudent {
private:
  int newId;

public:
  // Constructor
  Master(int number = 0, string name = "", double average = 0.0, int lvl = 0,
         int yr = 0, int id = 0)
      : GraduateStudent(number, name, average, lvl, yr), newId(id) {}

  // Set and get function for new data member
  void setNewId(int id) { newId = id; }

  int getNewId() const { return newId; }

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

int main() {
  // Declare object of type Student
  Student student1(12345, "Tarana Tanjim", 88.5);
  student1.print();

  // Declare object of type Master
  Master master1(54321, "Tasmita Tanjim", 95.5, 2, 2024, 67890);
  master1.print();

  return 0;
}
```



3. Answer the questions after going through the following class:

```
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;
      }
};
```

a. Write statements in C++ that would execute *Function 1* and *Function 3* of class Seminar.

ANSWER:

To execute Function 1 (the default constructor) and Function 3 (the parameterized constructor) of the class Seminar:

Seminar seminar1; // Executes the default constructor (Function 1)
Seminar seminar2(60); // Executes the parameterized constructor (Function 3

In this instance, seminar1 is created with the standard seminar duration, and seminar2 is initialized with a 60-minute duration.

b. In Object Oriented Programming, what is *Function 4* referred as and when does it get invoked/called?

ANSWER:

The destructor, Function 4 (abbreviated ~Seminar()) in Object Oriented Programming, is invoked automatically when an object's lifetime expires, for example, when it is deleted explicitly or goes out of scope.

```
~Seminar()
{
   cout << "Thanks" << endl;
}
```

c. In Object Oriented Programming, which concept is illustrated by *Function 1* and *Function 3* together?

ANSWER:

The concept illustrated by Function 1 (the default constructor) and Function 3 (the parameterized constructor) in conjunction is known as Constructor Overloading. In polymorphism, multiple constructors within a class share the same name but differ in their parameters, enabling multiple methods to initialize class objects.

4. Answer the questions after going through the following class:

```
class Test{
    char paper[20];
    int marks;
    public:
      Test ()    // Function 1
      {
        strcpy (paper, "Computer");
        marks = 0;
      }
      Test (char p[])    // Function 2
      {
        strcpy(paper, p);
        marks = 0;
      }
      Test (int m)    // Function 3
      {
        strcpy(paper,"Computer");
        marks = m;
```

```
        }
        Test (char p[], int m)    // Function 4
        {
          strcpy (paper, p);
          marks = m;
        }
};
```

a. Write statements in C++ that would execute Fu*nction 1, Function 2, Function 3* and *Function 4* of class *Test*.

ANSWER:

The text specifies how to instantiate objects of class Test to execute its various constructors:

- Test test1; executes the default constructor (Function 1).
- Test test2("Physics"); executes a constructor that takes a string (Function 2).
- Test test3(80); executes a constructor that takes an integer (Function 3).
- Test test4("Chemistry", 75); executes a constructor that takes a string and an integer (Function 4).

```
32  Test test1;                  // Execute Function 1 (default constructor)
33  Test test2("Physics");       // Execute Function 2
34  Test test3(80);              // Execute Function 3
35  Test test4("Chemistry", 75); // Execute Function 4
```

b. Which feature of Object Oriented Programming is demonstrated using *Function 1, Function 2, Function 3* and *Function 4* together in the above class *Test*?

ANSWER:

Constructor overloading is the aspect of object-oriented programming that is being shown. The idea of constructor overloading allows a class to have several constructors with various parameter lists. Because of this, objects of the class can be created in various states based on the arguments provided to the constructors.

5. Consider the definition of the following class:

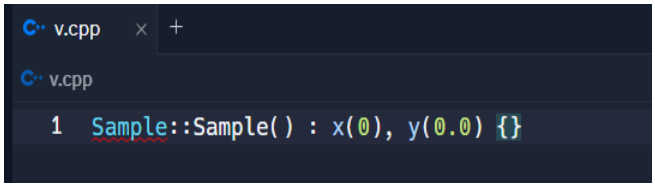
```
class Sample{
  private:
    int x;
    double y;
  public :
    Sample(); //Constructor 1
    Sample(int); //Constructor 2
```

```
    Sample(int, int); //Constructor 3
    Sample(int, double); //Constructor 4
};
```

a. **Write the definition of the *constructor 1* so that the private member variables are initialized to *0***

ANSWER:

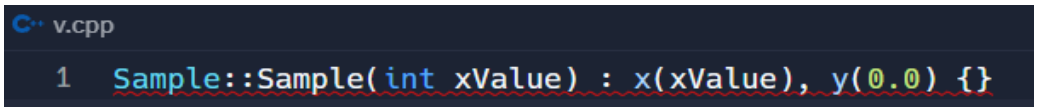
For constructor 1, which is the default constructor of the Sample class, to initialize both private member variables x and y to 0.



b. Write the definition of the *constructor 2* so that the private member variable *x* is initialized according to the value of the parameter, and the private member variable *y* is initialized to *0*

ANSWER:

For constructor 2, which takes an integer parameter to initialize x and sets y to 0 by default, here's the definition:



c. Write the definition of the *constructors 3* and *4* so that the private member variables are initialized according to the values of the parameters.

ANSWER: