

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


Question 1:

```
main.cpp
1  #include <stdio.h>
2  #include <iostream>
3  using namespace std;
4
5  class A {
6  public:
7      A();
8      A(int);
9      A(const A&);
10     ~A();
11
12     void operator=(const A& rhs);
13     void Print();
14     void PrintC() const;
15
16     int x;
17
18     int& X() { return x; }
19 };
20
21 A::A() : x(0) {
22     cout << "Hello from A::A() Default constructor" << endl;
23 }
24
25 A::A(int i) : x(i) {
26     cout << "Hello from A::A(int) constructor" << endl;
27 }
28
29 A::A(const A& a) : x(a.x) {
30     cout << "Hello from A::A(const A&) constructor" << endl;
31 }
32
33 A::~~A() {
34     cout << "Hello from A::A destructor" << endl;
35 }
36
37 void A::operator=(const A& rhs) {
38     x = rhs.x;
39     cout << "Hello from A::operator=" << endl;
```

main.cpp

```
40 }
41
42 void A::Print() {
43     cout << "A::Print(), x " << x << endl;
44 }
45
46 void A::PrintC() const {
47     cout << "A::PrintC(), x " << x << endl;
48 }
49
50 void PassAByValue(A a) {
51     cout << "PassAByValue, a.x " << a.x << endl;
52     a.x++;
53     a.Print();
54     a.PrintC();
55 }
56
57 void PassAByReference(A& a) {
58     cout << "PassAByReference, a.x " << a.x << endl;
59     a.x++;
60     a.Print();
61     a.PrintC();
62 }
63
64 void PassAByConstReference(const A& a) {
65     cout << "PassAByReference, a.x " << a.x << endl;
66     a.PrintC();
67 }
68
69 void PassAByPointer(A* a) {
70     cout << "PassAByPointer, a->x " << a->x << endl;
71     a->x++;
72     a->Print();
73     a->PrintC();
74
75
76 int main() {
77     A a0;
78     A a1(1);
```

```
79     A a2(a0);
80     A a3 = a0;
81
82     a3 = a1;
83     PassAByValue(a1);
84     cout << "After PassAByValue(a1)" << endl;
85     a1.Print();
86
87     PassAByReference(a1);
88     cout << "After PassAByReference(a1)" << endl;
89     a1.Print();
90
91     PassAByConstReference(a1);
92     cout << "After PassAByConstReference(a1)" << endl;
93     a1.Print();
94
95     PassAByPointer(&a1);
96     cout << "After PassAByPointer(a1)" << endl;
97     a1.Print();
98
99     a1.X() = 10;
100    cout << "a1.X() = 10" << endl;
101    a1.Print();
102
103    |
104
105    return 0;
106 }
107
```



```
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

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  // Class Student
6  class Student {
7  protected:
8      int studentNumber;
9      string studentName;
10     double studentAverage;
11 public:
12     // Constructor with default values
13     Student() : studentNumber(0), studentName(""), studentAverage(0.0) {}
14
15     // Set functions
16     void setStudentNumber(int num) { studentNumber = num; }
17     void setStudentName(string name) { studentName = name; }
18     void setStudentAverage(double avg) { studentAverage = avg; }
19
20     // Get functions
21     int getStudentNumber() const { return studentNumber; }
22     string getStudentName() const { return studentName; }
23     double getStudentAverage() const { return studentAverage; }
24
25     // Print function
26     void Print() const {
27         cout << "Student Number: " << studentNumber << endl;
28         cout << "Student Name: " << studentName << endl;
29         cout << "Student Average: " << studentAverage << endl;
30     }
31 };
32
33 // Class GraduateStudent inherits from Student
34 class GraduateStudent : public Student {
35 protected:
36     int level;
37     int year;
38 public:
39     // Constructor
40     GraduateStudent() : level(0), year(0) {}
41
42     // Set functions
```

main.cpp

```
43 void setLevel(int lvl) { level = lvl; }
44 void setYear(int yr) { year = yr; }
45
46 // Get functions
47 int getLevel() const { return level; }
48 int getYear() const { return year; }
49
50 // Print function
51 void Print() const {
52     Student::Print(); // Call base class print function
53     cout << "Level: " << level << endl;
54     cout << "Year: " << year << endl;
55 }
56 };
57
58 // Class Master inherits from GraduateStudent
59 class Master : public GraduateStudent {
60 protected:
61     int newId;
62 public:
63     // Constructor
64     Master() : newId(0) {}
65
66     // Set function
67     void setNewId(int id) { newId = id; }
68
69     // Get function
70     int getNewId() const { return newId; }
71
72     // Print function
73     void Print() const {
74         GraduateStudent::Print(); // Call base class print function
75         cout << "New ID: " << newId << endl;
76     }
77 };
78
79 int main() {
80     // Declare object of type Student with suitable values then print it
81     Student student1;
82     student1.setStudentNumber(1001);
83     student1.setStudentName("John Doe");
84     student1.setStudentAverage(85.5);
```

```

85     cout << "Student Information:" << endl;
86     student1.Print();
87     cout << endl;
88
89     // Declare object of type Master with your information then print it
90     Master master1;
91     master1.setStudentNumber(2001);
92     master1.setStudentName("Jane Smith");
93     master1.setStudentAverage(90.0);
94     master1.setLevel(2);
95     master1.setYear(2023);
96     master1.setNewId(123456);
97     cout << "Master's Information:" << endl;
98     master1.Print();
99
100    return 0;
101 }
102

```

input

```

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
1 #include <iostream>
2 using namespace std;
3
4 class Seminar {
5     int time;
6
7 public:
8     // Function 1: Default Constructor
9     Seminar() {
10         time = 30;
11         cout << "Seminar starts now" << endl;
12     }
13
14     // Function 2: Member Function lecture
15     void lecture() {
16         cout << "Lectures in the seminar on" << endl;
17     }
18
19     // Function 3: Parameterized Constructor
20     Seminar(int duration) {
21         time = duration;
22         cout << "Seminar starts now" << endl;
23     }
24
25     // Function 4: Destructor
26     ~Seminar() {
27         cout << "Thanks" << endl;
28     }
29 };
30
31 int main() {
32     // Part a: Executing Function 1 and Function 3
33     Seminar seminar1;           // Function 1 will be executed (Default Constructor)
34     Seminar seminar2(60);       // Function 3 will be executed (Parameterized Constructor)
35
36     // Part b: Destructor explanation
37     // Destructor is automatically called when an object goes out of scope
38     // or when the delete operator is used on a dynamically allocated object.
39     // It is responsible for releasing resources held by the object.
40
41     // Part c: Constructor Overloading
42     // Function 1 and Function 3 illustrate constructor overloading.
43
44     // Constructor overloading allows the class to have multiple constructors
45     // with different sets of parameters. Depending on how an object is instantiated,
46     // the appropriate constructor will be called.
47
48     return 0;
49 }

```

input

```
Seminar starts now
Seminar starts now
Thanks
Thanks

```

Question 4:

a.

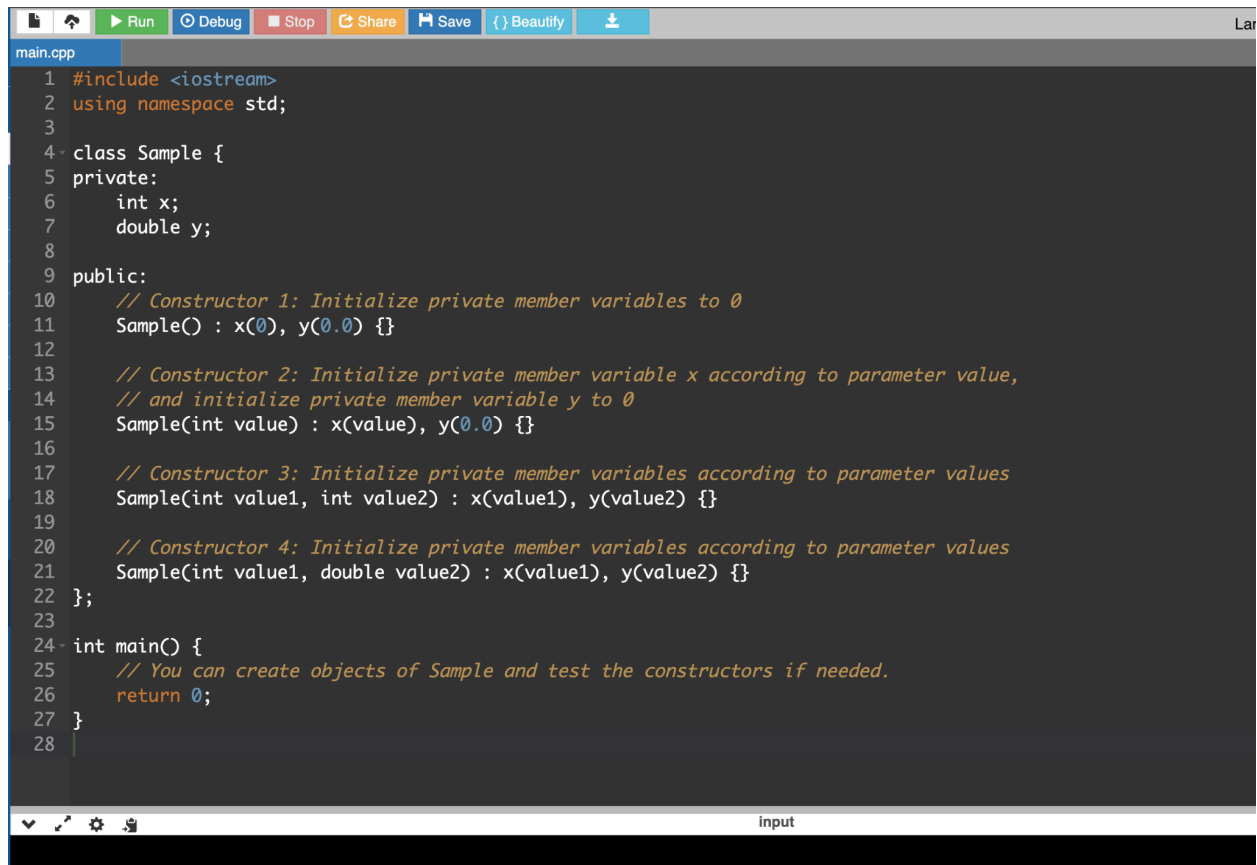
```
Test test1;           // Function 1: Default constructor
Test test2("Maths");  // Function 2: Constructor with char array parameter
Test test3(90);        // Function 3: Constructor with integer parameter

```


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:



The screenshot shows a C++ IDE with a toolbar at the top containing icons for Run, Debug, Stop, Share, Save, Beautify, and a download icon. The file name 'main.cpp' is visible in the top left. The code defines a 'Sample' class with four overloaded constructors and a 'main' function. The constructors are: a default constructor, a single integer parameter constructor, a two integer parameter constructor, and a single integer and one double parameter constructor. The 'main' function is currently empty, with a comment indicating where to test the constructors.

```
1 #include <iostream>
2 using namespace std;
3
4 class Sample {
5 private:
6     int x;
7     double y;
8
9 public:
10     // Constructor 1: Initialize private member variables to 0
11     Sample() : x(0), y(0.0) {}
12
13     // Constructor 2: Initialize private member variable x according to parameter value,
14     // and initialize private member variable y to 0
15     Sample(int value) : x(value), y(0.0) {}
16
17     // Constructor 3: Initialize private member variables according to parameter values
18     Sample(int value1, int value2) : x(value1), y(value2) {}
19
20     // Constructor 4: Initialize private member variables according to parameter values
21     Sample(int value1, double value2) : x(value1), y(value2) {}
22 };
23
24 int main() {
25     // You can create objects of Sample and test the constructors if needed.
26     return 0;
27 }
28
```

At the bottom of the IDE, there is a status bar with icons for a dropdown menu, zoom in, zoom out, settings, and a keyboard icon, followed by the text 'input'.