Khoi Duong

Prof. Yang

CS360L

6/10/2022

LAB #3

1.

Source code with explanation:

```
#include <stdio.h>
#include <iostream>
using namespace std;
class A {
public:
  A();          //Default constructor
  A(int);       //Constructor taking an integer as argument
  A(const A&); //Constructor initialize with the address of an object of class A
                //Another object of class A is assigned address to the constant
argument of the constructor
  ~A();         //Destructor
public:
  void operator=(const A& rhs); //function "operator=" takes object "rhs" of class
A and assigns to the constant argument
  void Print();        //function "Print" which print out the object, this function
can only call non-const object
  void PrintC() const; //this "Print" function is const and it doesn't allow to
modify the object it called
                       //this function can be called on both const and non-const
object
  int x;        //declare 'x' as an integer variable
public:
  //This function 'X' returns a reference of the variable 'x'
  //This doesn't return the value of 'x', but it returns the variable itself
(address)
  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;
}
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++;  //Increment of value of int variable 'x' of object 'a'
  a.Print();
  a.PrintC();
}
void PassAByReference(A& a)
{
  cout << "PassAByReference, a.x " << a.x << endl;
  a.x++;  //Increment of the variable 'x' itself (address) of object 'a'
  a.Print();
  a.PrintC();
```

```
}
void PassAByConstReference(const A& a)
{
  cout << "PassAByReference, a.x " << a.x << endl;
  a. PrintC();  // The function 'PrintC' can be called on the const object 'a'
                // Therefore, object 'a' can call function 'PrintC'
//a.Print(); // Call to "non-const" print function fails!
  // Compiler error from above line.  Why?
// Because the non-const function cannot call const objects. In this case, 'a' is
a const object
// The variable taken by function 'PassAByConstReference' is a const object of class
A
}
void PassAByPointer(A* a)
{
  cout << "PassAByPointer, a->x " << a->x << endl;
  a->x++;
  a->Print();
  a->PrintC();
}
int main()
{
  cout << "Creating a0 "; getchar();
  A a0;     // Read the next char from stdin (keyboard) and assign to object 'a0',
this call default constructor
  cout << "Creating a1 "; getchar();
  A a1(1); // Skip getchar(), variable is int i = 1 and assign to object 'a1', this
call A(int) constructor
  cout << "Creating a2 "; getchar();
  A a2(a0); // Read the next char from 'a0' since it points to object 'a0'
            // The variable 'x' of object 'a0' is also assigned to 'x' of object
'a2'
            // This call A(const A&) constructor
  cout << "Creating a3 "; getchar();
  A a3 = a0; // Same as A a3(a0); => This also call A(const A&) constructor
             // Skip getchar(), read 'x' from object 'a0', and assign 'x' of 'a0'
to 'x' of 'a3'
  cout << "Assigning a3 = a1 "; getchar();
  a3 = a1; // This call 'operator=' to assign variable 'x' of object 'a1' to 'x' of
object 'a3'
  // Call some of the "A" subroutines
  cout << "PassAByValue(a1) "; getchar();
  PassAByValue(a1); // Pass 'x' of object 'a1', increment of the value 'x' (x = 2),
and print out the non-const object
  // The value of a function parameter is copied to another location of the memory.
```

```
  // When accessing or modifying the variable within the function, it accesses only the copy.
  cout << "After PassAByValue(a1) " << endl;
  // There will be a copy when we pass object 'a1' to the function
  // However, this copy will have a copy of the pointer, that is, the two objects will point to the same 'a1'.
  // When exiting the function, the copy will be destroyed and will delete that pointer
  // Therefore, there exists a notification line: "Hello from A::A destructor".
  // Changes made inside the function are not reflected in the original value.
  // And the object 'a1' is still unchanged => Thus, the value 'x' of 'a1' is x = 1
  a1.Print();

  cout << "PassAByReference(a1) "; getchar();
  PassAByReference(a1); // PassAByReference is a mechanism of passing the actual parameters to the function.
  // Modify a variable without having to create a copy of it
  // The function can access the original object's content, which is the value 'x' of object 'a1'.
  // Changes made inside the function will be reflected in the original value
  // Therefore, after function executes, x = 2
  cout << "After PassAByReference(a1) " << endl;
  a1.Print();

  cout << "PassAByConst(a1) "; getchar();
  PassAByConstReference(a1); // same use as PassAByReference function
  // However, object 'a1' becomes const, which means that the content/value of 'a1' cannot be changed.
  cout << "After PassAByConstReference(a1) " << endl;
  a1.Print();

  cout << "PassAByPointer(&a1) "; getchar();
  PassAByPointer(&a1); // In PassAByPointer, the memory location of the variables is passed to the
                       // parameters of the function, and the operations are performed
  // Thus, it only points to the parameters, not performs any direct changes to the original object
  cout << "After PassAByPointer(a1) " << endl;
  a1.Print();
  // Since the value of 'x' in 'a1' is already changed to 2 (from function PassAByReference),
    // with a->x++, x = 3 and the result after the function is 3.
  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 object of class A can be constructed with the constructor A(int)
  // Thus, when the parameter of the function takes reference to an object of class
A,
    // we can set an integer and let the parameter points to the value of '20'.
  // This performs the same as PassAByConstReference(A(20));
  return 0;
}
```

Run program & result:

```
PS D:\VS CODE> cd "d:\VS CODE\C C++\CS360L\Lab3\" ; if ($?) { g++ 1.
cpp -o 1 } ; if ($?) { .\1 }
Creating a0
Hello from A::A() Default constructor
Creating a1
Hello from A::A(int) constructor
Creating a2
Hello from A::A(const A&) constructor
Creating a3
Hello from A::A(const A&) constructor
Assigning a3 = a1
Hello from A::operator=
PassAByValue(a1)
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(a1)
PassAByReference, a.x 1
A::Print(), x 2
A::PrintC(), x 2
After PassAByReference(a1)
A::Print(), x 2
PassAByConst(a1)
PassAByReference, a.x 2
A::PrintC(), x 2
After PassAByConstReference(a1)
A::Print(), x 2
PassAByPointer(&a1)
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
PassAByConstReference
Hello from A::A(int) constructor
PassAByReference, a.x 20
A::PrintC(), x 20
Hello from A::A destructor
Hello from A::A destructor
Hello from A::A destructor
Hello from A::A destructor
Hello from A::A destructor
PS D:\VS CODE\C C++\CS360L\Lab3>
```

2.

Source code:

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

class Student {
    public:
    int student_number;
    string student_name;
    double student_average;

    //Constructor
    Student(int number, string name, double average){
        student_number = number;
        student_name = name;
        student_average = average;
    }

    //Get functions
    void getNumber(){
        cout << "Student Number: " << student_number << endl;
    }
    void getName(){
        cout << "Student Name: " << student_name << endl;
    }
    void getAverage(){
        cout << "Student Average: " << student_average << endl;
```

```
    }

    //Set functions
    void setNumber(int number) {
        student_number = number;
    }
    void setName(string name) {
        student_name = name;
    }
    void setAverage(double average) {
        student_average = average;
    }

    void print(){
        cout << "Student details: " << endl;
        cout << "Number: " << student_number << endl;
        cout << "Name: " << student_name << endl;
        cout << "Average: " << student_average << endl;
    }
};

class Graduate_student: public Student {
    public:
    int level, year;

    //Constructor
    Graduate_student(int number, string name, double average, int lvl, int y) :
Student(number, name, average) {
        level = lvl;
        year = y;
    }

    //Get functions
    void getLevel() {
        cout << "Student Level: " << level << endl;
    }
    void getYear() {
        cout << "Student Year: " << year << endl;
    }

    //Set functions
    void setLevel(int lvl) {
        level = lvl;
    }
    void setYear(int y) {
```

```
        year = y;
    }

    void print(){
        Student::print();
        cout << "Level: " << level << endl;
        cout << "Year: " << year << endl;
    }
};

class Master: public Graduate_student {
    public:
    int newid;

    //Constructor
    Master(int number, string name, double average, int lvl, int y, int new_id) :
Graduate_student(number, name, average, lvl, y) {
        newid = new_id;
    }

    //Get functions
    void getNewId() {
        cout << "New Id: " << newid << endl;
    }

    //Set functions
    void setNewId(int new_id) {
        newid = new_id;
    }

    void print() {
        Graduate_student::print();
        cout << "New Id: " << newid << endl;
    }
};

int main() {
    Student student1(291, "Charles", 3.97);
    student1.print();
    cout << endl;
    Master master1(602, "Anna", 3.89, 2, 5, 1020);
    master1.print();
}
```

Run program & result:

```
                                > cd "d:\VS CODE\C C++\CS360L\Lab3\"
; if ($?) { g++ 2.cpp -o 2 } ; if ($?) { .\2 }
Student details:
Number: 291
Name: Charles
Average: 3.97

Student details:
Number: 602
Name: Anna
Average: 3.89
Level: 2
Year: 5
New Id: 1020
PS D:\VS CODE\C C++\CS360L\Lab3>
```

3.

a.

Execute *Function 1*: Seminar();

Execute *Function 3*: Seminar(30);

b.

Function 4 is a destructor. A destructor is called for a class object when that object passes out of scope or is explicitly deleted.

c.

Function 1 illustrates the default constructor. And *Function 3* presents a constructor that takes an integer as the function's parameter They both represent the concept of the polymorphism in OOP, which means that the function can go with no, one, or more parameters

4.

a.

Execute *Function 1*: Test();

Execute *Function 2*: Test("Computer");

Execute *Function 1*: Test(0);

Execute *Function 3*: Test("Computer", 0);

b.

All 4 functions demonstrate the feature of "Message Passing" in OOP. The objects communicate with one another by sending and receiving information from each other.

5.

a.

Constructor 1:

```
Sample::Sample(){
    x = 0;
    y = 0;
}
```

Default constructor that sets the private member variables to 0.

b.

Constructor 2:

```
Sample::Sample(int a){
    x = a;
    y = 0;
}
```

Constructor that initializes x according to the value of the parameter and the private member y is initialized to 0.

c.

Constructor 3:

```
Sample::Sample(int a, int b){
    x = a;
    y = b;
}
```

Constructor 4:

```
Sample::Sample(int a, double b){
    x = a;
    y = b;
}
```

Source code of the whole exercise 5:

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

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
};
```

```
Sample::Sample(){
    x = 0;
    y = 0;
}

Sample::Sample(int a){
    x = a;
    y = 0;
}

Sample::Sample(int a, int b){
    x = a;
    y = b;
}

Sample::Sample(int a, double b){
    x = a;
    y = b;
}
```