UEE1303 S18: Object-Oriented Programming

Constructors and Destructors

What you will learn from Lab 4

In this laboratory session you will:

1. learn how to use constructor and copy constructor to create an object and use destructor to delete it.

LAB 4-1: CONSTRUCTOR

✓ Please try to compile and execute the program lab4-1-1, and observe the results.

```
// lab4-1-1.cpp
#include <iostream>
using namespace std;
class Point2D
{
    public:
    void assignPoint2D(int n1, int n2, double v);
    void displayPoint2D();
    private:
    int x;
    int y;
    double value;
};
void Point2D::assignPoint2D(int n1, int n2, double v)
{
    x = n1;
    y = n2;
    value = v;
}
void Point2D::displayPoint2D()
{
    cout << "(" << x << "," << y << ") = ";
    cout << value << endl;
}
int main()
{
    Point2D ptArray[10];
    for (int i = 0; i < 10; i++)
    {
        ptArray[i].assignPoint2D(i, i+2, i*10);
        ptArray[i].displayPoint2D();
    }
    return 0;
}
```

➢

- ✓ Add constructor to class Point2D and make program lab5-1-2 work as expect.

```
//lab4-1-2.cpp
…
class Point2D
{
        public:
        Point2D();
        void assignPoint2D(int n1, int n2, double v);
        void displayPoint2D();
        private:
        int x;
        int y;
        double value;
};
Point2D::Point2D() {
        x = 0;
        y = 0;
        value = 0.0;
}
…
```

  - ➢ You can also use an initialization list to build your default constructor. In the above example, you can replace the declaration and definition of default constructor as Point2D():x(0), y(0), value(0.0) {}.

- ✓ Please modify your class Point2D to make the lab5-1-3 work.

```
// lab4-1-3.cpp
…
int main()
{
        Point2D pt1;
        Point2D pt2(1,2);
        Point2D pt3(3,2,1.9);
        pt1.displayPoint2D();
        pt2.displayPoint2D();
        pt3.displayPoint2D();
        pt3.assignPoint2D(2,1,0.0);
        pt3.displayPoint2D();
        return 0;
}
```

  - ➢ The line Point2D pt3(3,2,1.9) and pt3.assignPoint2D(2,1,0.0) can both assign the value to pt3's private member. Can you explain their difference?

LAB 4-2: DESTRUCTOR

- ✓ In class Point2D, we do not specific the destructor ~Point2D() since the compiler will generate one. However, if you use new or delete memory in the object, you need constructor to allocate memory and destructor to release it.
- ✓ Please modify the class Point2D as PointND, which is used to record the N-dimensional coordinate using an integer array.

```
// lab4-2.cpp
#include <iostream>
const int num = 10;
class PointND
{
public:
        PointND();
      // ~PointND();
        void assignValue(double v);
        void assignCoord(int *vec, int len);
        void displayPointND();
private:
        int *coord;
        double value;
};
PointND::PointND()
{
        value = 0.0;
        coord = new int [num];
        for (int i = 0; i < num; i++) coord[i] = 0;
}
/*PointND::~PointND()
{
        delete [] coord;
}*/
void PointND::assignValue(double v)
{
        value = v;
}
void PointND::assignCoord(int *vec, int len)
{
        for (int i = 0; i < len; i++)
                coord[i] = vec[i];
}
void PointND::displayPointND()
{
        std::cout << "(";
        for (int i = 0; i < num; i++)
        {
                std::cout << coord[i];
                if (i != num - 1)
                        std::cout << ", ";
        }
        std::cout << ") = " << value << std::endl;
}
int main()
{
        PointND pt1;
        pt1.displayPointND();
        PointND pt2;
        pt2.assignValue(1.0);
        pt2.displayPointND();
        int *vec = new int [num];
        for (int i = 0; i < num; i++) vec[i] = i;

        PointND pt3;
        pt3.assignValue(4.3);
        pt3.assignCoord(vec, num);
        pt3.displayPointND();
        delete []vec;
        return 0;
}
```

LAB 4-3: COPY CONSTRUCTOR

✓ Please write a "PointND.h" file for this program and finish "Copy constructor" that is a constructor used to create a new object as a copy of an existing object.

```
// lab4-3.cpp
#include "PointND.h"
/* 1. class PointND defined in lab4-2
    2. add the definition of copy constructor to the class*/
PointND::PointND(const PointND &pt)
{
    value = pt.value;
    coord = new int [num];
    for (int i = 0; i < num; i++)
        coord[i] = pt.coord[i];
}
int main()
{
    int *vec = new int [num];
    for (int i = 0; i < num; i++) vec[i] = i;
    PointND pt1;
    pt1.assignValue(4.3);
    pt1.assignCoord(vec,num);
    pt1.displayPointND();
    PointND pt2(pt1); //call copy constructor
    pt2.displayPointND();
    delete []vec;
    return 0;
}
```

Exercise 4-1 (COMPLEX NUMBER)

- ✓ Create a Complex class to perform complex number arithmetic and write a program to test your class. The class provides four complex operations: addition, subtraction, multiplication and division. The sample output is shown as follows

```
(1.0, 7.0) + (9.0, 2.0) = (10.0, 9.0)
(1.0, 7.0) - (9.0, 2.0) = (-8.0, 5.0)
(1.0, 7.0) * (9.0, 2.0) = (-5.0, 65.0)
(1.0, 7.0) / (9.0, 2.0) = (0.3, 0.7)
(10.0, 7.0) - (9.0, -1.0) = (1.0, 8.0)
```

- ✓ Please add the constructor and copy constructor to the class Complex you defined in program ex3-1.
- ✓ The main structure of the program becomes

```
// Complex.h
#ifndef COMPLEX_H
#define COMPLEX_H

// Write class definition for Complex


#endif
```

```
// Complex.cpp
#include <iostream>
#include "Complex.h"
using namespace std;
// Member-function definitions for class Complex.
```

```
// ex4-1.cpp
#include "Complex.h"
#include <iostream>
using namespace std;
int main()
{
    Complex a(1.0, 7.0), b(9.0, 2.0), c; // create three Complex objects
    a.printComplex(); // output object a
    cout << " + ";
    b.printComplex(); // output object b
    cout << " = ";
    c = a.add(b); // invoke add function and assign to object c
    c.printComplex(); // output object c
    cout << endl;
    // use copy constructor to create object d
    Complex d(c);
    d.printComplex(); // output object d
    cout << endl;
    return 0;
}
```

Exercise 4-2: MATRIX

✓ Write a class called Matrix to perform matrix arithmetic. The sample output is shown as follows.

```
Enter n for n x n matrix: 3
A = [4 7 8; 6 4 6; 7 3 8];
B = [2 3 8; 1 10 4; 7 1 7];
A' = [4 6 7; 7 4 3; 8 6 10];
B' = [2 1 7; 3 10 1; 8 4 7];
A + B = [6 10 16; 7 14 10; 14 4 17];
A – B = [2 4 0; 5 -6 2; 0 2 3];
A * B = [71 90 116; 64 106 34; 138 28 563];
```

➢ The elements (integer) in matrix is randomly generated in range [1,10]. The representation of matrix is row major. For example, A = [10 2 8; 1 5 8; 1 4 8] indicates that

$$A = \begin{bmatrix} 10 & 2 & 8 \\ 1 & 5 & 8 \\ 1 & 4 & 8 \end{bmatrix}$$

and A + B means

$$A + B = \begin{bmatrix} 10 & 2 & 8 \\ 1 & 5 & 8 \\ 1 & 4 & 8 \end{bmatrix} + \begin{bmatrix} 7 & 4 & 7 \\ 3 & 2 & 6 \\ 6 & 9 & 10 \end{bmatrix} = \begin{bmatrix} 17 & 6 & 15 \\ 4 & 7 & 14 \\ 7 & 13 & 18 \end{bmatrix}$$

✓ Please use the ***constructor***, ***copy constructor*** and ***destructor*** to the class Matrix you defined. Note that you must use dynamic memory allocation to create the object.

✓ The main structure of the program becomes,

```
// Matrix.h
#ifndef MATRIX_H
#define MATRIX_H

/* Write class definition for Matrix and add constructors and destructor*/
class Matrix{
private:
        int dim;//n*n size matrix
        int *matrix;
….
}
#endif
```

```
// Matrix.cpp
```

```
#include <iostream>
using std::cout;

#include "Matrix.h"

// Member-function definitions for class Matrix.
```

```
// ex4-2.cpp
#include <iostream>
using std::cout;
using std::endl;
using std::cin;
#include "Matrix.h"

int main()
{
        int n;
        cout << “Enter n for n x n matrix: ” << endl;
        cin >> n;

        Matrix A(n), B(n);          // create two Matrix objects
        A. assignElements(); // assign elements in Matrix A randomly
        cout << "A = ";
        A.printMatrix(); // output object A
        cout << endl;

        B. assignElements(); // assign elements in Matrix B randomly
        cout << "B = ";
        B.printMatrix(); // output object B
        cout << endl;

        Matrix tA(A);                   // use copy constructor to build tA
        cout << “tA= “;
        tA.transposeMatrix(); // transpose Matrix tA
        cout << endl;

        Matrix tB(B);                   // use copy constructor to build tB
        cout << “tB= “;
```

```
    tB.transposeMatrix(); // transpose Matrix tB
    cout << endl;

    Matrix C;
    C.multiplyMatrix(A,B); // C = A * B
    cout << "A*B = ";
    C.printMatrix(); // output object C
    cout << endl;

    return 0;
}
```

✓