UEE1303(1070) S12: Object-Oriented Programming

Constant Pointer and Class

What you will learn from Lab 4

In this laboratory, you will learn how to use `const` to identify constant pointer and the basic of class.

TASK 4-1: POINTER TO CONSTANT

✓ Please differentiate the following three examples (***a-point-to-a-constant***, ***a-constant-pointer*** and ***a-constant-pointer-to-a-constant***) and fix the compiler errors with improper usage.

```
// lab4-1-1.cpp
#include <iostream>
using namespace std;

int main()
{
    double a = 1.34;
    const double *pa = &a; // a pointer to a constant

    cout << "*pa = " << *pa << endl;

    double b = 6.5;
    pa = &b;            // a pointer to a constant can change the pointer
    cout << "*pa = " << *pa << endl;

    *pa = 7.6;           // cannot modify a pointer to a constant
    cout << "*pa = " << *pa << endl;        [Error] assignment of read-only location '* pa'

    return 0;
}
```

```
// lab4-1-2.cpp
#include <iostream>
using namespace std;

int main()
{
    double a = 1.34;
    double *const pa = &a; // a const pointer to a double

    cout << "*pa = " << *pa << endl;

    double b = 6.5;
```

```
    pa = &b;            // a constant pointer cannot be changed
    cout << "*pa = " << *pa << endl;
    *pa = 7.6;           // a constant pointer can be modified
    cout << "*pa = " << *pa << endl;

    return 0;
}
```

[Error] assignment of read-only variable 'pa'

```
// lab4-1-3.cpp
#include <iostream>
using namespace std;

int main()
{
    double a = 1.34;
    const double *const pa = &a; // a const pointer to a constant

    cout << "*pa = " << *pa << endl;

    double b = 6.5;
    pa = &b;            // a constant pointer to constant cannot be changed
    cout << "*pa = " << *pa << endl;

    *pa = 7.6;           // a constant pointer to constant cannot be modified
    cout << "*pa = " << *pa << endl;

    return 0;
}
```

[Error] assignment of read-only variable 'pa'

[Error] assignment of read-only location '*(const double*)pa'

TASK 4-2: BASIC CLASS

✓ We rewrite the structure `Point2D`, defined in program `lab3-1-3`, as a class object.

```
// lab4-2-1.cpp
#include <iostream>

class Point2D
{
    int x;
    int y;
    double value;
    public:
    void assignPoint2D(int n1, int n2, double v);
    void displayPoint2D();
};

void Point2D::assignPoint2D(int n1, int n2, double v)
{
```

```
    x = n1;
    y = n2;
    value = v;
}

void Point2D::displayPoint2D()
{
    std::cout << "(" << x << "," << y << ") = ";
    std::cout << value << std::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;
}
```

- Please fix the compiler error in this example.
- If you do not specific the member access modifiers, the compiler will take as private member.

✓ We rewrite the above program and modify the class `Point2D` with member access modifiers.

```
// lab4-2-2.cpp
#include <iostream>

class Point2D
{
private:
    int x;
    int y;
    double value;

public:
    void assignPoint2D(int n1, int n2, double v);
    void displayPoint2D();
};

void Point2D::assignPoint2D(int n1, int n2, double v)
{
    x = n1;
    y = n2;
    value = v;
```

```
}

void Point2D::displayPoint2D()
{
    std::cout << "(" << x << "," << y << ") = ";
    std::cout << value << std::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;
}
```

✓ To access private data members, we can add accessor and mutator as public member functions.

```
// lab4-2-3.cpp
…
class Point2D
{
private:
    int x;
    int y;
    double value;

public:
    void setCoord(int n1, int n2);
    void setValue(double v);
    int getCoordX();
    int getCoordY();
    double getValue();

    void assignPoint2D(int n1, int n2, double v);
    void displayPoint2D();
};

// Please implement the definitions of five additional member functions.

int main()
{
    Point2D a;
    a.setCoord(1,3);
    cout << “a(x,y) = ” << a.getCoordX() << “ “ << a.getCoordY() << endl;
```

```
    Point2D *b = new Point2D;
    b->setValue(5);
    cout << "value of b is " << b->getValue() << endl;

    return 0;
}
```

TASK 4-3: EXERCISE

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)
```

✓ The main structure of the program is like as,

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

/* Write class definition for Complex */

#endif
```

```
// Complex.cpp
#include <iostream>
using std::cout;

#include "Complex.h"

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

```
// ex4-1.cpp
#include <iostream>
using std::cout;
using std::endl;

#include "Complex.h"

int main()
{
   Complex a, b, c;  // create three Complex objects
   a.assign(1.0,7.0);
   b.assign(9.0,2.0);

   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;

   a.printComplex(); // output object a
   cout << " - ";
   b.printComplex(); // output object b
   cout << " = ";
   c = a.subtract(b); // invoke subtract function and assign to object c
   c.printComplex(); // output object c
   cout << endl;

   a.printComplex(); // output object a
   cout << " * ";
   b.printComplex(); // output object b
   cout << " = ";
   c = a.multiply(b); // invoke multiply function and assign to object c
   c.printComplex(); // output object c
   cout << endl;

   a.printComplex(); // output object a
```

```
    cout << " / ";
    b.printComplex(); // output object b
    cout << " = ";
    c = a.division(b); // invoke division function and assign to object c
    c.printComplex(); // output object c
    cout << endl;

    a.assignReal(10.0);  // reset object a
    b.assignImage(-1.0); // reset object b
    a.printComplex(); // output object a
    cout << " - ";
    b.printComplex(); // output object b
    cout << " = ";
    c = a.subtract( b ); // invoke subtract function and assign to object c
    c.printComplex(); // output object c
    cout << endl;

    return 0;
}
```

2. MATRIX OPERATION

✓ 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 = [10 2 8; 1 5 8; 1 4 8];
B = [7 4 7; 3 2 6; 6 9 10];
A' = [10 1 1; 2 5 4; 8 8 8];
B' = [7 3 6; 4 2 9; 7 6 10];
A + B = [17 6 15; 4 7 14; 7 13 18];
A – B = [3 -2 1; -2 3 2; -5 -5 -2];
A * B = [124 116 162; 70 86 117; 67 84 111];
```

➢ 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

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

✓ The main structure of the program is like as,

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

/* Write class definition for 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;

#include "Matrix.h"

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

   Matrix A, B, C; // create three Matrix objects
   A.assignDimension(n);
   A.assignElements(); // assign elements in Matrix A randomly
```

```
cout << "A = ";
A.printMatrix(); // output object A
cout << endl;

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



Matrix tA;
tA.assignMatrix(A);   // copy elements and dimension from A
tA.transposeMatrix(); // transpose Matrix tA
cout << "A' = ";
tA.printMatrix(); // output object tA
cout << endl;

Matrix tB;
tB.assignMatrix(B);   // copy elements and dimension from B
tB.transposeMatrix(); // transpose Matrix tB
cout << "B' = ";
tB.printMatrix(); // output object tB
cout << endl;

C = A.addMatrix(B); // C = A + B
cout << "A+B = ";
C.printMatrix(); // output object C
cout << endl;

C = A.subtractMatrix(B); // C = A - B
cout << "A-B = ";
C.printMatrix(); // output object C
cout << endl;

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

```
    cout << endl;


    return 0;
}
```