UEE 1303(1067): Object-Oriented Programming

Lab #5: Constructors and Destructors

In this laboratory session you will:

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

Lab 5-1: Constructor

✓ Please try to compile and execute the program `lab5-1-1`, and observe the results.

```
// lab5-1-1.cpp
#include <iostream>
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()
{
   std::cout << "(" << x << "," << y << ") = ";
   std::cout << value << std::endl;
}
int main()
{
   Point2D ptArray[10];
```

```
    for (int i = 0; i < 10; i++)
        ptArray[i].displayPoint2D();
    return 0;
}
```

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

```
// lab5-1-2.cpp
……
class Point2D
{
public:
    Point2D(); // default constructor
    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 a *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.

```
// lab5-1-3.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 5-2: 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.

```
// lab5-2.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 5-3: Copy Constructor

- ✓ Copy constructor is a constructor used to create a new object as a copy of an existing object.

```
// lab5-3.cpp
#include <iostream>

/* 1. class PointND defined in lab5-2
   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 5-1 (COMPLEX NUMBER)

- ✓ Please add the constructor and copy constructor to the class `Complex` you defined in program `ex4-1`.
- ✓ 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.
```

```
// ex5-1.cpp
#include <iostream>
#include "Complex.h"
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

   return 0;
}
```

Exercise 5-2 (HUGE INTEGER)

- ✓ In C++, the largest `int` value is 2,147,483,647. So, an integer larger than this cannot be stored and processed as an integer. Similarly, if the sum or product of two positive integers is greater than 2,147,483,647, the result will be incorrect. One way to store and manipulate large integers is to store each individual digit of the number in an array.
- ✓ Create a class `HugeInteger` that uses a 40-element array of digits to store integer as large as 40 digits each. Provide member function `add`, `subtract`, `output`. For comparing `HugeInteger` objects, provide functions `isEqualTo`, `isNotEqualTo`, and `isLessThan` – each of these is a function that simply returns `true` if the relationship holds between the two `HugeInteger` and returns `false` if the relationship does not hold. Also please add the constructor and destructor to the class `HugeInteger`.

- ✓ The sample output is shown as follows

```
7654321 + 100000000000000 = 100000007654321
100000000000000 - 5 = 99999999999995
99999999999995 + 5 = 100000000000000
100000000000000 is equal to 100000000000000
7654321 is not equal to 100000000000000
5 is less than 100000000000000
```

- ✓ In this exercise, your `HugeInteger` class has defined three private data member as follow:

```
// HugeInteger.h
#ifndef HUGEINTEGER_H
#define HUGEINTEGER_H
class HugeInteger
{
public:
   // put your member function prototype
private:
   int *data;
   int length;
   bool sign;
}
#endif
```

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

✓ In this exercise, your `HugeInteger` class has defined three private data member as follow:

```
// ex5-2.cpp
#include <iostream>
#include "HugeInteger.h"
using namespace std;

int main()
{
  HugeInteger n1( 7654321 ); // HugeInteger object n1
  // HugeInteger object n2
  HugeInteger n2( "100000000000000" );
  HugeInteger n4( 5 ); // HugeInteger object n4
  HugeInteger n5; // HugeInteger object n5

  // outputs the sum of n1 and n2
  n5 = n1.add( n2 );
  n1.output();
  cout << " + ";
  n2.output();
  cout << " = ";
  n5.output();
  cout << endl;

  // assigns the difference of n2 and n4 to n5 then outputs
```

```
n5
    n5 = n2.subtract( n4 );
    n2.output();
    cout<< " - ";
    n4.output();
    cout << " = ";
    n5.output();
    cout << endl;

    HugeInteger n3(n5); // call copy constructor
    // outputs the sum of n3 and n4
    n2 = n3.add ( n4 );
    n3.output();
    cout<< " + ";
    n4.output();
    cout << " = ";
    n2.output();
    cout << endl;

    // checks for equality between n2 and n2
    if ( n2.isEqualTo( n2 ) == true )
    {
        n2.output();
        cout << " is equal to ";
        n2.output();
        cout << endl;
    } // end if

    // checks for inequality between n1 and n2
    if ( n1.isNotEqualTo( n2 ) == true )
    {
        n1.output();
        cout << " is not equal to ";
        n2.output();
        cout << endl;
    } // end if
    // tests for smaller number between n4 and n2
    if ( n4.isLessThan( n2 ) == true )
    {
       n4.output();
       cout << " is less than ";
       n2.output();
       cout << endl;
    } // end if

    return 0;
}
```