UEE1303(1070) S12: Object-Oriented Programming
Operator Overloading and Function Overloading

What you will learn from Lab 7

In this laboratory, you will learn how to use operator overloading and function overloading, which are important functionality provided by C++.

TASK 7-1 FUNCTION OVERLOADING

✓ Overloaded functions are functions in the same scope that have the same name but their arguments are different.

```
// lab7-1-1.cpp
#include <iostream>

using std::cout;
using std::endl;

int sum(int *array, int len)
{
    int n = 0;
    for (int i=0;i<len;i++)
       n += array[i];
    return n;
}

double sum(double *array, int len)
{
    double n = 0.0;
    for (int i=0;i<len;i++)
       n += array[i];
    return n;
}
int main()
{
    int array1[5] = {1,2,3,4,5};
    cout << "sum(array1) = " << sum(array1,5) << endl;

    double array2[5] = {1.1,2.2,3.3,4.4,5.5};
    cout << "sum(array2) = " << sum(array2,5) << endl;

    return 0;
}
```

➢ In this function, `int sum(int *array, int len)` and `double sum(double *array, int len)` have the same name in global scope, but the types of argument lists are different.

✓ The different number of argument lists is also one kind of function overloading.

```
// lab7-1-2.cpp
#include <iostream>

using std::cout;
using std::endl;

int min(int n1, int n2)
{
    int tmp = n1 < n2 ? n1 : n2;
    return tmp;
}

int min(int n1, int n2, int n3)
{
    int tmp = min(n1,n2);
    return min(tmp,n3);
}

int main()
{
    cout << "min(4,3) = " << min(4,3) << endl;
    cout << "min(1,3,2) = " << min(1,3,2) << endl;

    return 0;
}
```

➢ Notice that, the function overloading can be achieved by different data type and different number of argument list, but it cannot be different return type. For example, `int sum(int *array, int len)` and `double sum(int *array, int len)` cannot exist at the same time.

TASK 7-2 OVERLOADED FUNCTIONS AS MEMBER FUNCTIONS

✓ In this example, there are three overloaded constructors and two overloaded member functions.

```
// lab7-2.cpp
#include <iostream>

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

public:
    Point2D();
    Point2D(int n1, int n2);
```

```
    Point2D(int n1, int n2, double v);
    void assignPoint2D(int n1, int n2);
    void assignPoint2D(int n1, int n2, double v);
    void displayPoint2D() const;
};

Point2D::Point2D()
{
    x = 0;
    y = 0;
    value = 0;
}

Point2D::Point2D(int n1, int n2)
{
    assignPoint2D(n1,n2,0.0);
}

Point2D::Point2D(int n1, int n2, double v)
{
    assignPoint2D(n1,n2,v);
}

void Point2D::assignPoint2D(int n1, int n2)
{
    assignPoint2D(n1,n2,value);
}

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

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

int main()
{
    Point2D pt1(3,4,3.9);
    Point2D pt2;

    pt1.displayPoint2D();
    pt2.displayPoint2D();

    std::cout << "after assignment " << std::endl;
```

```
    pt1.assignPoint2D(1,3);
    pt2.assignPoint2D(2,3,1.1);
    pt1.displayPoint2D();
    pt2.displayPoint2D();

    return 0;
}
```

TASK 7-3 OPERATOR OVERLOADING

✓ Operator can be overloaded to define the operator on the object.

```
// lab7-3.cpp
#include <iostream>

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

public:
    Point2D();
    Point2D(int n1, int n2);
    Point2D(int n1, int n2, double v);

    Point2D operator + (const Point2D &);
    Point2D operator - ();

    void assignPoint2D(int n1, int n2);
    void assignPoint2D(int n1, int n2, double v);
    void displayPoint2D() const;
    friend double distPoint2D(const Point2D &, const Point2D &);
    friend double distPoint2D(const Point2D &, const Point2D &, const Point2D
&);

    friend bool operator == (const Point2D &, const Point2D &);
    friend bool operator != (const Point2D &, const Point2D &);
};

Point2D Point2D::operator + (const Point2D &pt)
{
    return Point2D(x+pt.x, y+pt.y,value+pt.value);
}
Point2D Point2D::operator - ()
{
    return Point2D(-x, -y, -value);
}
```

```
bool operator == (const Point2D &pt1, const Point2D &pt2)
{
    if (pt1.x != pt2.x || pt1.y != pt2.y || pt1.value != pt2.value)
       return false;
    return true;
}

bool operator != (const Point2D &pt1, const Point2D &pt2)
{
    return !(pt1 == pt2);
}

double distPoint2D(const Point2D &pt1, const Point2D &pt2)
{
    return sqrt((pt1.x - pt2.x)*(pt1.x - pt2.x) + (pt1.y - pt2.y)*(pt1.y -
pt2.y));
}

double distPoint2D(const Point2D &pt1, const Point2D &pt2, const Point2D &pt3)
{
    double n1 = distPoint2D(pt1, pt2);
    double n2 = distPoint2D(pt1, pt3);
    double n3 = distPoint2D(pt2, pt3);

    return (n1 + n2 + n3);
}

Point2D::Point2D()
{
    x = 0;
    y = 0;
    value = 0;
}

Point2D::Point2D(int n1, int n2)
{
    assignPoint2D(n1,n2,0.0);
}

Point2D::Point2D(int n1, int n2, double v)
{
    assignPoint2D(n1,n2,v);
}

void Point2D::assignPoint2D(int n1, int n2)
{
    assignPoint2D(n1,n2,value);
}

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

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

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

int main()
{
    Point2D pt1(3,4,4.1);
    Point2D pt2(3,2,4.5);

    if (pt1 == pt2) std::cout << "pt1 is equal to pt2 " << std::endl;
    else std::cout << "pt1 is not equal to pt2 " << std::endl;

    pt1.displayPoint2D();
    pt2.displayPoint2D();

    Point2D pt3;
    pt3 = pt1 + pt2;
    pt3.displayPoint2D();

    Point2D pt4 = -pt1;
    pt4.displayPoint2D();

    return 0;
}
```

TASK 7-4 EXERCISE

1. COMPLEX NUMBER

✓ Please modify the class `Complex` you defined in `ex5-1` which make the file `ex7-1` work.

```
// ex7-1.cpp
#include <iostream>
using std::cout;
using std::endl;

#include "Complex.h"

int main()
```

```
{
   Complex a(1.0, 7.0), b(9.0, 2.0), c; // create three Complex objects
   printMeg(a,b,'+');  // output (1.0, 7.0) + (9.0, 2.0) =
   c = a + b;          // invoke operator + and assign to object c
   printComplex(c);    // output object c
   cout << endl;

   printMeg(a,b,'-');  // output (1.0, 7.0) - (9.0, 2.0) =
   c = a - b;          // invoke operator - function and assign to object c
   printComplex(c);    // output object c
   cout << endl;

   printMeg(a,b,'*');  // output (1.0, 7.0) * (9.0, 2.0) =
   c = a * b;          // invoke operator * function and assign to object c
   printComplex(c);    // output object c
   cout << endl;

   printMeg(a,b,'-');  // output (1.0, 7.0) / (9.0, 2.0) =
   c = a / b;          // invoke operator / function and assign to object c
   printComplex(c);    // output object c
   cout << endl;

   a.setComplexNumber(10.0, 1.0); // reset object a
   b = -a;
   printMeg(a,b,'-');
   c = a - b;         // invoke operator - function and assign to object c
   printComplex(c);   // output object c
   cout << endl;

   return 0;
}
```

2. *STRING OPERATION

✓ Please implement the string class defined as follows,

```
// my_string.h
#include <iostream>
class string
{
private:
   char *p;
public:
   string (const char *s);
   string (const string &s);
   ~string() {if(p) { delete [] p; p = NULL; } }
   friend std::ostream& operator << (std::ostream&, const string &s);
   friend string operator + (const string &s, const string &t);
   bool operator <= (const string &s);
};

/*
    definition of class string.
*/
```

✓ Main function

```
int main()
{
    string t1("New");
    cout << "t1=" << t1 << endl;
    string t2 = "York";
    cout << "t2=" << t2 << endl;
    string t3 = t1 + " " + t2;
    cout << "t3=" << t3 << endl;

    if ( t1 <= t2 )
        cout << t1 << " is smaller than " << t2 << endl;
    else
        cout << t1 << " is bigger than " << t2 << endl;
    return 0;
}
```

✓ Output

```
t1 = New
t2 = York
t3 = New York;
New is smaller than New York
```