EEEC10008(515169) S23 : Object-Oriented Programming

Nested Class, Operator Overloading and Function Overloading

What you will learn from Lab 5

In this laboratory, you will learn how to use nested class, operator overloading and function overloading.

TASK 5-1 CONST AND MUTABLE MEMBERS

✔ `const` member functions are not supposed to modify objects of a class. However, if a data member is declared to be `mutable`, then it can be changed by any member function even in a `const` member function. Please identify which member function should be `const` to make the program work successfully.

```
// lab5-1.cpp
#include <iostream>
/* class Point2D declares and defines in lab4-1*/
/* add mutable (int) member named color to class Point2D */

void Point2D::displayPoint2D() const
{
    x = 5; y = 4;
    color = 10;
    std::cout << "(" << x << "," << y << ") = ";
    std::cout << value << std::endl;
}

int main()
{
    const Point2D pt1;
    Point2D pt2;

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

    return 0;
}
```

☐ Please identify the difference between `mutable` data member and non-`mutable` data member.

TASK5-2 NESTED CLASS

✔ A class can be defined in another class, so called nested class. ***Nested class*** can be taken as a (`public`, `private`, or `protected`) member in the ***enclosing class***. The name of nested class can be resolved in enclosing class scope, but it cannot be access in other class scope or

other namespace.

```
// lab5-2.cpp
#include <iostream>
#include <assert.h>

class Vec
{
public:
    Vec(){len =0;}
    Vec(int n);
    ~Vec();

    void setValue(int idx, int v);
    void printVec() const;

private:
    class Items                          // nested class Items for Vec
    {                                      // all members in Items are private
        friend class Vec;               // make Vec can access member in Items
        Items(){value = 0;}
        Items(int v){value = v;}
        int value;
    };

    int len;
    Items *vec;
};


Vec::Vec(int n)
{
    len = n;
    vec = new Items [len];
}

Vec::~Vec()
{
    if (len > 0)
       delete []vec;
}

void Vec::setValue(int idx, int v)
{
    assert(idx < len);
    vec[idx].value = v;
}

void Vec::printVec() const
{
    for (int i=0;i<len;i++)
        std::cout << vec[i].value << " ";
    std::cout << std::endl;
}
```

```
int main()
{
    Vec vector(5);
    vector.printVec();

    for (int i=0;i<5;i++)
        vector.setValue(i,i);
    vector.printVec();

    Items n;

    return 0;
}
```

☐ There is a compiler error in this example because the nested class cannot be used in global scope. Try to modify the program and make `Items` be accessed in global scope.

TASK5-3 FUNCTION OVERLOADING

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

```
// lab5-3-1.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];
lab5-3-1
```

☐ 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.

```
// lab5-3-2.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 5-4 OVERLOADED FUNCTIONS AS MEMBER FUNCTIONS

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

```
// lab5-4.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 5-5 OPERATOR OVERLOADING

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

```
// lab5-5.cpp
#include <iostream>
#include <math.h>

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;
}
```

EXERCISE 5-1: COMPLEX NUMBER

- ✔ In this exercise, you need to write a class `Complex` to make a Complex Calculator.
- ✔ In this work, you should implement some overloading operators. Please implement those overloading operators as **member functions**.
- ✔ `printComplex` could be **friend functions** of the class `Complex`.
- ✔ In class `Complex`, there are two `private` variables, `real` and `imag`. The data types of these two private variables are `double`.

■ HERE IS SOME FUNCTION YOU HAVE TO COMPLETE.

1. Basic operation '+', '-', '*'.
2. Operate '>>' to transform Complex into Polar form.
3. Operate '=' to remember last operation and do it again.
4. Note the format must meet common sense. (ex. 0+5j => 5j, 2+1j => 2+j)

```
$ ./ex5-1
input complex number: 1+j
input operator: >>
Polar form is: 1.41421(cos45+isin45)
Complex number is: 1+j
input operator: +
input complex number: 2+3j
Complex number is: 3+4j
input operator: -
input complex number: 5+7j
Complex number is: -2-3j
input operator: +
input complex number: 1+j+3+6j
Complex number is: 2+4j
input operator: -
input complex number: 2+3+5j-2-6j
Complex number is: -1+5j
input operator: +
input complex number: 5
Complex number is: 4+5j
input operator: -
input complex number: 7j-3j
Complex number is: 4+j
input operator: +
input complex number: 5-7
Complex number is: 2+j
input operator: *
input complex number: 2
Complex number is: 4+2j
input operator: -
input complex number: 4j-2j
```

```
Complex number is: 4
input operator: +
input complex number: -4
Complex number is: 0
input operator: +
input complex number: 2+j
Complex number is: 2+j
input operator: =
Complex number is: 4+2j
input operator: =
Complex number is: 6+3j
input operator: =
Complex number is: 8+4j
input operator: ^c
```

EXERCISE 5-2: SCORE RECORD

- ✔ In this exercise, you need to write a nested class Weight in class Score.
- ✔ You should follow the class Score below to finish your work.
- ✔ Change function let all subject score Multiply by 10 after opening the square root

Score.h

```
class Score
{
private:
   string name;
   double math, science, english;
   double average;

   class Weight
   {
   private:
      double weighted_avg;
      double math_weight, science_weight, english_weight;
      Score &score;

   public:
      //Weight(){}
      Weight(Score& x)
      void set_weight()
      void weight_avg()
   };

public:
   Weight w;
   Score();
   void set_score();
   void avg();
   void Change();
};
```

Result

```
$ ./ex5-2
input 1~4 to select function :
1.Set score
2.Set weight
3.Change score
4.Show average
5.Show weight average
1
name: Jim
math score: 80
science score: 90
english score: 70
input 1~4 to select function :
1.Set score
2.Set weight
3.Change score
4.Show average
5.Show weight average
4
average is: 80
input 1~4 to select function :
1.Set score
2.Set weight
3.Change score
4.Show average
5.Show weight average
2
math weight: 0.5
science weight: 0.3
english weight: 0.2
input 1~4 to select function :
1.Set score
2.Set weight
3.Change score
4.Show average
5.Show weight average
5
Weighted average is: 81
input 1~4 to select function :
1.Set score
2.Set weight
3.Change score
4.Show average
5.Show weight average
3
average is: 89.3257
input 1~4 to select function :
```

```
1.Set score
2.Set weight
3.Change score
4.Show average
5.Show weight average
5
Weighted average is: 89.9151
```