实验八 ——运算符重载及应用

张林鹏_2021032449

一、实验目的

1. 理解多态性概念及分类, 熟悉C++中静态多态性的实现方法;
2. 理解运算符重载的意义, 掌握在C++中用成员函数及友元函数实现运算符重载的方法;
3. 熟悉运算符重载的应用.

二、实验内容

程序1: exp_801.cpp

1. 程序编译运行的结果是:

```
x=10  y=20
x=11  y=21
```

2. 程序中前置运算符 `++` 采用的重载方法是 成员函数重载 .

3. 将重载函数 `coord coord::operator ++()` 中的 `++x; ++y;` 改为: `++this->x; ++this->y ;` , 重新运行程序:

```
x=10  y=20
x=11  y=21
```

4. 将 `main()` 函数中的 `++ob;` 改为 `“ob.operator++( );` , 重新运算程序:

```
x=10  y=20
x=11  y=21
```

程序2: exp_802.cpp

5. 编译程序时, 将会出现编译错误, 其原因是 重载运算符**+**应该是一个类的成员函数或者一个友元函数, 而在该程序中, 是一个普通函数 .

6. 将类中的 `complex operator+(complex c1,complex c2);` 的前面加上 `friend` , 重新编译调试程序, 输出结果为:

```
2.3+4.6i
3.6+2.8i
6+7.4i
```

7. 将 `main()` 中的 `A3=A1+A2;` 改为 `A3=operator(A1,A2);` , 重新运行程序, 输出为:

```
2.3+4.6i
3.6+2.8i
6+7.4i
```

程序3: exp_803.cpp

8.
```
#include <iostream>
using namespace std;
class complex
{
private:
    double real;
    double imag;

public:
    complex(double r = 0, double i = 0)
    {
        real = r;
        imag = i;
    }
    void print();
    complex operator+(const complex &c);
    complex operator-(complex c);
};
complex complex ::operator+(const complex &c) // 重载“+”
{
    complex temp;
    temp.real = this->real + c.real;
    temp.imag = this->imag + c.imag;
    return temp;
}
complex complex ::operator-(complex c) // 重载“-”
{
    complex temp;
```

```
    temp.real = this->real - c.real;
    temp.imag = this->imag - c.imag;
    return temp;
}
void complex::print()
{
    cout << real;
    if (imag > 0)
        cout << "+";
    if (imag != 0)
        cout << imag << "i" << endl;
}
int main()
{
    complex A1(2.3, 4.6), A2(3.6, 2.8), A3, A4;
    A3 = A1 + A2;
    A4 = A1 - A2;
    A1.print();
    A2.print();
    A3.print();
    A4.print();
    return 0;
}
```

程序4: exp_804.cpp

9.
```
#include <iostream>
using namespace std;

class coord
{
    int x, y;

public:
    coord(int i = 0, int j = 0);
    void print();
    coord operator++();
    coord operator++(int);
    coord operator-();
};
coord::coord(int i, int j)
{
    x = i;
    y = j;
}
void coord::print()
{
    cout << "x=" << x << "  y=" << y << endl;
}
```

```
coord coord::operator++() // 重载前置运算符++
{
    ++x;
    ++y;
    return *this;
}
coord coord::operator++(int) // 重载后置运算符++
{
    coord temp(*this);
    ++x;
    ++y;
    return temp;
}

coord coord::operator-() // 重载负号“-”
{
    coord temp(-x, -y);
    return temp;
}
int main()
{
    coord ob1(10, 20), ob2(20, 40), ob;
    ob1.print();
    ob2.print();
    ++ob1;
    ob2.operator++(0); // 显式调后后置运算符
    ob1.print();
    ob2.print();
    ob = -ob1;
    ob.print();
    return 0;
}
```

程序5: exp_805.cpp

```
#include <string.h>
#include <iostream>
using namespace std;

class MyString
{
    char *ptr;
public :
    MyString(char *s)
    {
        ptr = new char[strlen(s) + 1];
        strcpy(ptr, s);
    }
    ~MyString()
```

```
    {
        delete [] ptr;
    }
    void print()
    {
        cout << ptr << endl;
    }
    MyString &operator=(const MyString &s);
};

MyString &MyString::operator=(const MyString &s) //重载“=”运算符
{
    if (this == &s)
        return *this;                    //当用“ob1=ob1;”时，直接返回
    delete[] ptr;                        //释放被赋值对象的空间
    ptr = new char[strlen(s.ptr) + 1];   //重新为被赋值对象分配空间
    strcpy(ptr, s.ptr);
    return *this;
}

int main()
{
    MyString p1("chen");
    {
        MyString p2("    ");
        p2 = p1;
        p2.print();
    }
    p1.print();
}
```

程序设计实验 (user_date.h & exp_806.cpp)

- user_date.h:

```
#ifndef USER_DATE_H
#define USER_DATE_H

#include "date.h"

class user_date : date
{
public:
    user_date(int y, int m, int d) : date()
    {
        set_date(y, m, d);
    }
    bool operator==(user_date &d);
    bool operator!=(user_date &d);
```

```
    bool operator>(user_date &d);
    bool operator<(user_date &d);
    bool operator>=(user_date &d);
    bool operator<=(user_date &d);
};

bool user_date::operator==(user_date &d)
{
    if (get_year() == d.get_year() && get_month() == d.get_month() &&
get_day() == d.get_day())
        return true;
    else
        return false;
}

bool user_date::operator!=(user_date &d)
{
    if (get_year() != d.get_year() || get_month() != d.get_month() ||
get_day() != d.get_day())
        return true;
    else
        return false;
}

bool user_date::operator>(user_date &d)
{
    if (get_year() > d.get_year())
        return true;
    else if (get_year() == d.get_year() && get_month() > d.get_month())
        return true;
    else if (get_year() == d.get_year() && get_month() == d.get_month() &&
get_day() > d.get_day())
        return true;
    else
        return false;
}

bool user_date::operator<(user_date &d)
{
    if (get_year() < d.get_year())
        return true;
    else if (get_year() == d.get_year() && get_month() < d.get_month())
        return true;
    else if (get_year() == d.get_year() && get_month() == d.get_month() &&
get_day() < d.get_day())
        return true;
    else
        return false;
}

bool user_date::operator>=(user_date &d)
{
    if (get_year() > d.get_year())
```

```
        return true;
    else if (get_year() == d.get_year() && get_month() > d.get_month())
        return true;
    else if (get_year() == d.get_year() && get_month() == d.get_month() &&
get_day() > d.get_day())
        return true;
    else if (get_year() == d.get_year() && get_month() == d.get_month() &&
get_day() == d.get_day())
        return true;
    else
        return false;
}

bool user_date::operator<=(user_date &d)
{
    if (get_year() < d.get_year())
        return true;
    else if (get_year() == d.get_year() && get_month() < d.get_month())
        return true;
    else if (get_year() == d.get_year() && get_month() == d.get_month() &&
get_day() < d.get_day())
        return true;
    else if (get_year() == d.get_year() && get_month() == d.get_month() &&
get_day() == d.get_day())
        return true;
    else
        return false;
}
#endif
```

- exp_806.cpp:

```
#include "date.h"
#include "user_date.h"
using namespace std;

int main()
{
    user_date u0(2020, 4, 1);
    user_date u1(2021, 3, 1);
    user_date u2(2021, 3, 1);
    user_date u3(2022, 3, 1);
    user_date u4(2022, 3, 6);

    cout << "u0 == u1: " << ((u0 == u1) ? "true" : "false") << endl;
    cout << "u1 == u2: " << ((u1 == u2) ? "true" : "false") << endl;
    cout << "u1 != u2: " << ((u1 != u2) ? "true" : "false")<< endl;
    cout << "u2 < u3:  " << ((u2 < u3) ? "true" : "false") << endl;
    cout << "u3 < u4:  " << ((u3 < u4) ? "true" : "false") << endl;
}
```