计算机学院 高级程序语言设计 课程实验报告

实验题目：多态（一） 运算符重载（静态_编译时多态）		学号：202300130150
日期：2024.4.18	班级： 4	姓名： 王成意

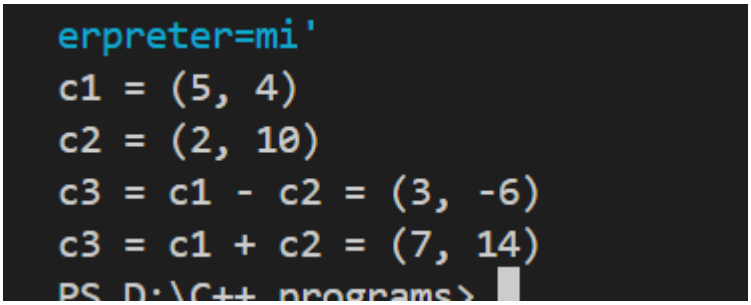
实验目的：

1. 掌握双目、单目（前、后）运算符重载
2. 练习运算符重载的两种方式：类成员和非成员函数实现

实验步骤与内容：

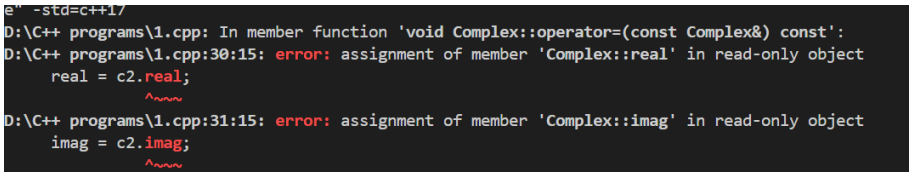
1. 练习第 8 章 PPT 例 8-1，复数类加减法用成员函数实现运算符重载，形参是引用。截图并回答一下问题

   原代码截图：



   1）void operator = (const Complex &c2);能否变成 void operator = (const Complex &c2) const;？为什么

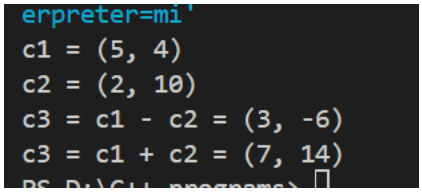
   答：不能。



   也就是说，重载运算符“=”改变了左值，不可能是常函数。

   2）Complex operator + (const Complex &c2) const;中的两个 const 能去掉吗？为什么

   可以。



   原因是在重载运算“+”时，并没有对*this 或者 c2 做出任何更改。

   （3）通过添加一个 Complex operator + (const double &db) const;重载函数实现负数类 complex 与普通 double 类型数据的相加。在 main 中添加

c3 = c1 + 0.5; c3.display();截图并执行

实现：

```
Complex Complex::operator + (const double &db)
const
{
    return Complex(this->real + db, this->imag);
}
```

结果：

```
erpreter=m1'
c1 = (5, 4)
c2 = (2, 10)
c3 = c1 - c2 = (3, -6)
c3 = c1 + c2 = (7, 14)
(5.5, 4)
```

2. 练习第 8 章 PPT 例 8-2，单目运算符重载，返回类型是引用。在 main()中试用++++myclock 和后++++，看能否实现连续加 1？ main 改为如下：（略）

原：

```
First time output: 23:59:59
Show myClock++:    23:59:59
Show ++myClock:    0:0:1
```

更改后：

```
First time output: 23:59:59
Show myClock++++:    23:59:59
Show ++++myClock:    0:0:2
```

预期为 0：0：3，不符。

再次更改后发现：

```
First time output: 23:59:59
Show myClock++:    23:59:59
Show ++++myClock:    0:0:2
```

可见后++不能连续。

由此尝试：

```
int main()
{
    Clock myClock(23, 59, 59);
```

```
    cout << "First time output: ";
    myClock.showTime();
    cout << "Show myClock++:    ";
    (myClock++).showTime();
    cout << "Show ++++++myClock:    ";
    (++++++myClock).showTime();
    return 0;
}
```

发现：

```
First time output: 23:59:59
Show myClock++:    23:59:59
Show ++++++myClock:    0:0:3
```

综上所述，前++可连续而后++不行。

3. 实践第 8 章 PPT 例 8-3，以非成员函数形式重载 Complex 的加减法运算和“<<”运算符，两个形参都为引用。

原函数结果：

```
c1 = (5, 4)
c2 = (2, 10)
c3 = c1 - c2 = (3, -6)
c3 = c1 + c2 = (7, 14)
```

不用友元函数能不能实现对各操作符的非成员函数的重载？如果可以的话可以通过什么样的方法？

可以。

```
Complex operator+(const Complex &c1); // 运算符+重载
Complex operator-(const Complex &c2); // 运算符-重载
Complex Complex::operator+(const Complex &c1)
{ // 重载运算符函数实现
    return Complex(c1.real + real, c1.imag + imag);
```

```
}

Complex Complex::operator-(const Complex &c2)

{ // 重载运算符函数实现

    return Complex(real - c2.real, imag - c2.imag);

}
```

对于<<无法不用 friend 重载，因为输出的内容在右边，而不用 friend 重载的话默认 complex 型的*this 在左边，使得 “<< c1”语句报错。

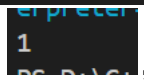
4. 将课上常用的 Point 类（有数据成员 x 和 y）用友元函数或成员函数的方式实现双目操作符+，-，=，==的重载。

代码：

```
#include <cstdio>

using namespace std;

class point

{

private:

    int x, y;


public:

    point(int x, int y)

    {

        this->x = x;

        this->y = y;

    }

    point operator+(const point &x) const;
```

```
    point operator-(const point &x) const;
    point operator=(const point &x);
    bool operator==(const point &x) const;
    void display();
};
void point::display()
{
    printf("x=%d,y=%d\n", this->x, this->y);
}
point point::operator+(const point &x) const
{
    return point(this->x + x.x, this->y + x.y);
}
point point::operator-(const point &x) const
{
    return point(this->x - x.x, this->y - x.y);
}
point point::operator=(const point &x)
{
    return point(this->x, this->y);
}
bool point::operator==(const point &x) const
```

```
{
    if(this->x==x.x && this->y==x.y)
        return true;
    else
        return false;
}
int main()
{
    point x(1, 2), y(3, 4);
    point z = x + y, a(4, 6);
    if(a==z)
    {
        printf("1");
    }
    else
        printf("0");
    return 0;
}
```

结果： 

，符合预期。

5. 通过重载赋值操作符实现深拷贝。假设有如下类

1）请重新设计针对 MyArray 类的=运算符实现深拷贝操作。当有两个 MyArray 对象 arr1 和 arr2 执行 arr1=arr2 操作时，不是简单的将对象内容进行拷贝，而是执行如下操作。首先判断两个对象的 array_size 是否相等，不相等的话

打印错误信息，之后停止操作，将 arr2 的 arr 数组中的 array_size 个值一个个的赋值给 arr1 的 arr 数组相应的位置中。

```
void MyArray::operator=(const MyArray &kkk)

{

    assert(this->getArraySize() ==

kkk.getArraySize());

    for (int i = 0; i < kkk.getArraySize();++i)

    {

        this->arr[i] = kkk.arr[i];

    }

}
```

结果：

```
erpreter=mi'
addr: 0x10217d0 contents:  0 1 2 3 4 5 6 7 8 9
addr: 0x1021830 contents:  0 2 4 6 8 10 12 14 16 18
addr: 0x10217d0 contents:  0 2 4 6 8 10 12 14 16 18
PS D:\C++ programs>
```

或者：

```
MyArray arr1(15), arr2(10);
for (int i = 0; i < 10; i++)
```

之后的结果：

```
addr: 0xe017d0 contents:  0 1 2 3 4 5 6 7 8 9 0 0 0 0 0
addr: 0xe01840 contents:  0 2 4 6 8 10 12 14 16 18
Assertion failed!

Program: D:\C++ programs\1.exe
File: D:\C++ programs\1.cpp, Line 59

Expression: this->getArraySize() == kkk.getArraySize()
PS D:\C++ programs>
```

（2）去掉 getArraySize 和 getArr 函数定义中的 const 会产生什么样的编译错误？结合之前我们学习的常量定义的知识解释一下为什么。

```
D:\C++ programs\1.cpp: In member function 'void MyArray::operator=(const MyArray&)':
D:\C++ programs\1.cpp:59:47: error: passing 'const MyArray' as 'this' argument discards qualifiers [-fpermissive]
     assert(getArraySize() == kkk.getArraySize());
                                               ^
D:\C++ programs\1.cpp:23:9: note:   in call to 'int MyArray::getArraySize()'
     int getArraySize()
         ^~~~~~~~~~~~
D:\C++ programs\1.cpp:60:42: error: passing 'const MyArray' as 'this' argument discards qualifiers [-fpermissive]
     for (int i = 0; i < kkk.getArraySize();++i)
                                          ^
D:\C++ programs\1.cpp:23:9: note:   in call to 'int MyArray::getArraySize()'
     int getArraySize()
         ^~~~~~~~~~~~
```

```
void MyArray::operator=(const MyArray &kkk)
{
    assert(getArraySize() == kkk.getArraySize());
    for (int i = 0; i < kkk.getArraySize();++i)
    {
        arr[i] = kkk.arr[i];
    }
```

原因：

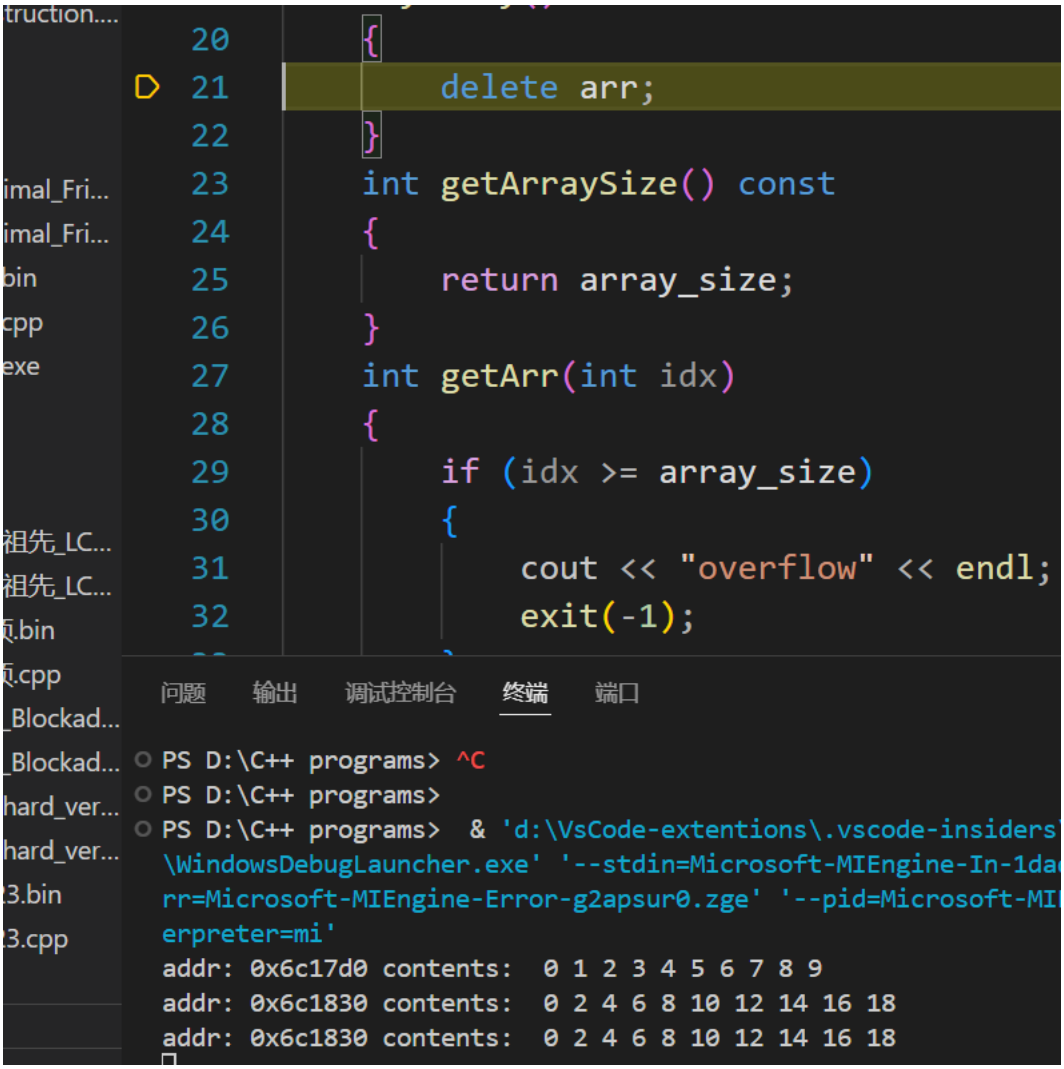
将“const xxx”作为“this”参数传递会丢弃限定词。

discards qualifiers：丢弃限定符，当尝试修改 const 类型时，会报 discards qualifiers；

也就是说，在这个函数里没有把 this 当成 const，降级处理报错。

（3）分析一下重载=之前和之后 main 函数中打印的结果的区别，截图并解释。

重载之前：



之后：

```
addr: 0xfd17d0 contents:  0 1 2 3 4 5 6 7 8 9
addr: 0xfd1830 contents:  0 2 4 6 8 10 12 14 16 18
addr: 0xfd17d0 contents:  0 2 4 6 8 10 12 14 16 18
```

区别在于其地址不同。

重载之前的“=”相当于引用，直接传递了地址，这样导致删除的时候重复；而重载之后是在当前的地址，不会引发问题。

结论分析与体会：

非常好的实验，使我对重载运算符有了崭新的认知。