C++程序设计基础 第七章

前言

本文档由 @ItsJiale 创作，作者博客：https://jiale.domcer.com/，作者依据数学与大数据学院 2024 级大数据教学班的授课重点倾向编写而成。所有内容均为手动逐字录入，其中加上了不少自己的理解与思考，耗费近一周时间精心完成。

此文档旨在助力复习 C++ 程序设计基础，为后续学习数据结构筑牢根基。信计专业的同学，也可参考本文档规划复习内容。需注意，若个人学习过程中存在不同侧重点或对重难点的理解有差异，应以教材内容为准。倘若文档内容存在任何不妥之处，恳请各位读者批评指正。

By：ItsJiale

2025.4.8

第7章 运算符重载

> Q：本章开始之前，***为什么要使用运算符重载?***

1. 增强代码可读性

在处理自定义类型时，使用重载运算符能让代码更接近自然语言表达。例如，有一个表示复数的 `Complex` 类，当实现复数相加时，若不使用运算符重载，代码可能是这样：

```cpp
Complex result = addComplex(c1, c2);
```

而使用运算符重载后：

```cpp
Complex result = c1 + c2;
```

后者明显更直观，代码的意图一目了然，就如同在操作基本数据类型一样，方便开发者阅读和理解。

2. 提高代码可维护性

使用重载运算符可以让代码结构更清晰，逻辑更连贯。当需要修改某个操作的实现时，只需要在运算符重载函数里进行修改，而不用在代码中到处查找调用该操作的地方。例如，对于 `Complex` 类的乘法操作，如果之后要改变乘法的计算规则，只需要调整重载的 `*` 运算符函数即可。

3. 实现多态性

运算符重载是实现静态多态性（编译时多态）的一种方式。同样的运算符，根据操作数的类型不同，可以执行不同的操作。比如，`+` 运算符既可以用于两个整数相加，也可以用于两个 `Complex` 对象相加，编译器会在编译时根据操作数的类型选择合适的运算符重载函数，增加了代码的灵活性和复用性。

4. 保持与内置类型的一致性

使用运算符重载能让自定义类型和内置类型在使用上保持一致。开发者在使用自定义类型时，无需额外学习新的操作方法，直接使用熟悉的运算符就能完成相应操作，降低了学习成本，使代码风格更加统一

7.1 运算符重载概述

在类中定重新定义的运算符，赋予运算符新的功能，对类对象进行相关操作。在类中重新定义运算符，赋予运算符新的功能以适应自定义数据类型的运算，就称为运算符重载

7.1.1 运算符重载的语法

使用operator关键字定义运算符重载

```
返回值 operator 运算符名称（参数列表）{
    ........//函数体
}
```

operator 运算符名称 这个整体等价于函数名

从运算符重载语法格式可以看出，运算符重载的返回值类型、参数列表可以是任意数据类型。除了函数名称中的operator关键字，运算符重载函数与普通函数没有区别

> **运算符重载中：**
>
> **①类的成员函数中**
>
> **形参个数 = 运算符操作数 – 1**
>
> **②类的友元函数中**
>
> **形参个数 = 运算符个数**

7.1.2 运算符重载的规则

上述讲解了运算符重载的语法格式，需要注意的是，有的运算符不可以重载的，并且运算符不可以改变语义

1. 只能重载C++中已有的运算符，且不能创建新的运算符
2. 重载后运算符不能改变优先级和结合性，也不能改变操作数和语法结构
3. 运算符重载的目的是针对实际运算数据类型的需要，重载要保持原有运算符的语义，且要避免没有目的地使用运算符重载
4. 并非所有C++运算符都可以重载，其中***“：：”***、***“.“***、 ”.*“、 ***”?：“***、sizeof、typeid等不能重载

7.1.3 运算符重载的形式

由于本章在授课时重点倾向于代码演示，所以以下内容***以代码演示理解为主***，理论为辅

1. 重载为类的成员函数

> **①类的成员函数中**
>
> **形参个数 = 运算符操作数 – 1**

此处所以重载运算符类的成员函数中为1 （双目运算符 – 1）

C++

```cpp
#include <iostream>
using namespace std;
class complex {

public:
    complex() {}
    complex(double r, double i) {
        real = r;
        imag = i;
    }
    void display() {
        cout << "(" << real << "," << imag << ")" << endl;
    }

    complex operator+(complex c1); //声明一个重载运算符函数
    complex operator-(complex c1); //形参个数 = 运算符操作数 - 1
private:
    double real;
    double imag;
};


complex complex::operator+(complex c1) {  //类外实现该函数
    complex c;
    c.real = real + c1.real;
    c.imag = imag + c1.imag;
    return complex(c1.real, c.imag);
}

complex complex::operator-(complex c1) {
    complex c;
    c.real = real - c1.real;
    c.imag = imag - c1.imag;
    return complex(c1.real, c.imag);
}

int main()
{
    complex c2(3, 5), c3(2, 4), c4, c5;
    c4 = c2 + c3;
    cout << "c4 = c2 + c3:";
    c4.display();
    c5 = c2 - c3;
    cout << "c5 = c2 - c3:";
    c5.display();
```

```
46          return 0;
47      }
```

2. 重载为类的友元函数

运算符重载为类的友元函数，需要在函数前加friend关键字

`friend 返回值类型 operator 运算符（参数列表）{`

`.......//函数体`

`}`

> ②类的友元函数中
>
> 形参个数 = 运算符个数

C++

```cpp
#include <iostream>
using namespace std;
class complex {
public:
    complex() {}
    complex(double r, double i) {
        real = r;
        imag = i;
    }
    void display() {
        cout << "(" << real << "," << imag << ")" << endl;
    }

     friend complex operator+(complex c1,complex c2);
     friend complex operator-(complex c1,complex c2);
private:
    double real;
    double imag;
};


 complex operator+(complex c1,complex c2) {
    complex c;
    c.real = c1.real + c2.real;
    c.imag = c1.imag + c2.imag;
    return complex(c.real, c.imag);
}

 complex operator-(complex c1, complex c2) {
     complex c;
     c.real = c1.real - c2.real;
     c.imag = c1.imag - c2.imag;
     return complex(c.real, c.imag);
 }

int main()
{
    complex c2(3, 5), c3(2, 4), c4, c5;
    c4 = c2 + c3;
    cout << "c4 = c2 + c3=";
    c4.display();
    c5 = c2 - c3;
    cout << "c5 = c2 - c3=";
    c5.display();
    return 0;
```

```
46  }
```