3.4重载

重载

- C++重载分为函数重载和运算符重载，其实质都是函数重载。
- 重载函数，函数名相同，函数参数的类型不同。
- 重载的必要性
  - 问题：计算绝对值

```
int my_abs(int val)
{
  return (val < 0) ? -val :val
}//不同数据类型需要分开定义，麻烦
```

例子：重载

```
#include <bits/stdc++.h>
using namespace std;

int my_abs(int val)
{
    return val < 0 ? -val : val;
}
float my_abs(float val)
{
    return val < 0 ? -val : val;
}

int main()
{
    int i = -10;
    float f = -125.8f;
    cout << my_abs(i) << endl;
    cout << my_abs(f) << endl;
    return 0;
}
```

例子：构造函数重载

```
#include <bits/stdc++.h>
using namespace std;

class Box
{
private:
    double hight, width, length;

public:
    Box();
    Box(double h);
```

```
    Box(double h, double w);
    Box(double h, double w, double l);
}
int main()
{
    Box box1;
    Box box2(10);
    Box box3(10, 20);
    Box box4(10, 20, 30);//
    return 0;
}
//超级构造函数,一个顶四个
Box(hight = 0, width = 0, length = 0);
```

重载规则

重载：函数名相同，参数类型不同。

- 以下重载非法：
  - 返回类型不同

```
long fun(int);
float fun(int);
```

  - 不能利用引用重载

```
void fun(int&);
void fun(int);
```

const可用于重载，以下合法：

```
void fun();
void fun() const;
```

运算符重载

- 运算符重载本质上也是函数重载。
- 运算符重载的关键：运算符对应的运算符函数
- 运算符函数的一般形式：

```
<函数类型>operator<运算符>(<形参表>);
{
  <函数体>
}
```

- 可采用：普通函数形式、成员函数形式

```
friend ostream& operator<<(ostream& os, const CComplex1& c1)
{
    os << "(" << c1.r << ", " << c1.i << ")";
    return os;
}
```

运算符函数原型

- 运算符函数的重载，首先要知道函数原型。
- 输出流运算符函数原型

```
ostream& operator<<(ostream& os,<操作对象>)
```

- 加法运算符函数原型

```
<返回类型> operator+(<操作对象>,<操作对象>)
```

- 自增运算符函数原型

```
前缀:<返回类型>operator++(<操作对象>)
后缀:<返回类型>operator++(<操作对象>,int)
```

- 下标运算符函数原型

```
<返回类型>&operator[](<操作对象>,int i)
```

例子：普通函数形式重载

```
class CComplex
{
private:
    double r, i;

public:
    CComplex(double r = 0, double i = 0);
    virtual ~CComplex();
    friend CComplex operator+(CComplex c1, CComplex c2);
    //输出流重载,略。
};

CComplex operator+(CComplex c1, CComplex c2);
void eg4_18()
{
    CComplex c1(1, 2), c2(3, 4);
    CComplex c3 = c1 + c2;
    cout << c1 << endl;
    cout << c2 << endl;
    cout << c3 << endl;
}
CComplex operator+(CComplex c1, CComplex c2)
{
    CComplex CTemp;
    CTemp.r = c1.r + c2.r;
    CTemp.i = c1.i + c2.i;
    return CTemp;
}
```

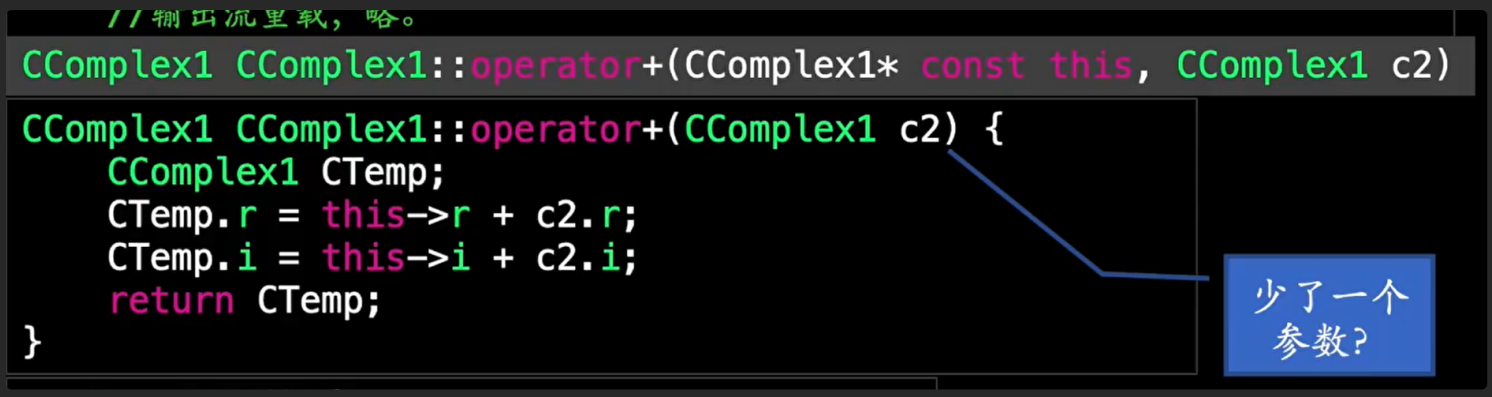
例子：成员函数形式重载

```
class CComplex1
```

```
{
private:
    double r, i;

public:
    CComplex1(double r = 0, double i = 0);
    virtual ~CComplex1();
    CComplex1 operator+(CComplex1 c2);
    //输出流重载,略。
};

CComplex1 CComplex1::operator+(CComplex1 c2)
{
    CComplex1 CTemp;
    CTemp.r = this->r + c2.r;
    CTemp.i = this->i + c2.i;
    return CTemp;
}
void eg4_19()
{
    CComplex1 c1(1, 2), c2(0.1, 0.1), c3;
    c3 = c1 + c2;
    cout << c3 << endl;
}
```



运算符重载

- 普通函数形式、成员函数形式，选择哪一种?
- 运算符重载与函数重载的 **区别** :
  - 同一个重载运算符的参数个数是相同的。
  - 不能定义新的运算符，只能重载现有的运算符。
  - 运算符重载后仍然保持原来的优先级和结合性。

例子：自增运算符重载

```
#include <bits/stdc++.h>
using namespace std;

class Counter
{
private:
    int value;
```

```
public:
    Counter(){value = 0;}
    Counter operator++();     //前缀运算符
    Counter operator++(int); //后缀运算符
    void display()
    {
        cout << "value: " << value << endl;
    }
};
Counter Counter::operator++()
{
    value++;
    return *this;
}
Counter Counter::operator++(int)
{
    Counter temp;
    temp.value = this->value++;
    return temp;
}

int main()
{
    Counter c1, c2;

    c2 = c1++;
    c1.display();
    c2.display();

    c2 = ++c1;
    c1.display();
    c2.display();

    return 0;
}
```

例子：下标运算符重载

```
#include <bits/stdc++.h>
using namespace std;

class Integer
{
private:
    int *array;
    int len;

public:
    Integer(int len)
    {
        this->len = len;
        array = new int[len];
    }
    int &operator[](int i); //重载运算符"[]"
```

```
    ~Integer() { delete[] array; }
};

int& Integer::operator[](int i)
{
    if (i < 0 || i >= len)
    {
        cout << "数组下标越界" << endl;
        exit(1);
    }
    return array[i];
}
int main()
{
    Integer array(10);
    int i = 10;
    for (int i = 0; i < 10; i++)
    {
        array[i] = i + 1;
        cout << array[i] << " ";
    }
    cout << ": i=" << i << endl;
    cout << array[i] << endl;
    return 0;
}
```

重载的问题

- 大部分代码都是重复的，只有**数据类型**不同
- 能否进一步简化？简化的思路是什么？