week 4

运算符重载的基本概念

对抽象数据类型也能够直接使用C++提供的运算符，程序更简洁，代码更容易理解

对已有的运算符赋予多重的含义

目的：扩展C++中提供的运算符的适用范围，以用于类所表示的抽象数据类型

运算符重载的实质是函数重载

```
返回值类型 operator 运算符 （形参表）
{
    ...
}
```

运算符可以被重载成普通函数，也可以被重载成类的成员函数

```
Complex operator+(const Complex &a, const Complex &b) {
    return Complex(a.real + b.real, a.imaginary +  b.imaginary);
}
```

重载为普通函数时，参数个数为运算符目数

```
class Complex {
    Complex operator+(const Complex &);
};
```

重载为成员函数式，参数个数为运算符目数减一 ## 赋值运算符的重载 ## 赋值运算符两边的类型可以不匹配

赋值运算符=只能重载为成员函数

```
char *String::operator=(const char *s) {
    if (str)
        delete []str;
    if (s) {
        str = new char[strlen(s) + 1];
        strcpy(str, s);
    } else
        str = NULL;
    return str;
}
```

浅复制/浅拷贝，执行逐个字节的复制工作

深复制/深拷贝，将一个对象中指针变量指向的内容，复制到另一个对象中指针成员对象指向的地方

```
String &operator=(const String &s) {
    if (str) delete[] str;
    str = new char[strlen(s.str) + 1];
    strcpy(str, s.str);
    return *this;
}
```

如果 str:

如果 s=s;

```
String &String::operator=(const String &s) {
    if (str == s.str) return *this;
    if (str) delete[] str;
    if (s.str) {
        str = new char[strlen(s.str) + 1];
        strcpy(str, s.str);
    } else
        str = NULL;
    return *this;
}
```

返回值类型为void好的不好？考虑a = b = c的情况

```
a.operator= (b.operator=(c));
```

String好不好，为什么是String&

运算符重载时，好的风格——尽量保留运算符原本的特性

```
(a = b) = c;
(a.operator= (b)).operator= (c);
```

为String类编写**复制构造函数**时，会面临和=同样的问题，用同样的方法处理

```
String::String (String &s) {
    if (s.str) {
        str = new char[strlen(s.str) + 1];
        strcpy(str, s.str);
    } else
        str = NULL;
}
```

运算符重载为友元函数

重载为友元函数的情况：成员函数不能满足使用要求，普通函数又不能访问类的私有成员

```
c = c + 5;
c = 5 + c;
class Complex {
    friend Complex operator+(double r, const Complex &c);
};
Complex operator+(double r, const Complex &c); // 普通函数不能访问私有成员
```

可变长整型数组

```
class CArray {
    int size;
    int *ptr;
    public:
        CArray (int s = 0);
        CArray (CArray &a);
        ~CArray();
        void push_back(int v);
        CArray &operator=(const CArray &a);
        int length();
        int &CArray::operator[](int i) {
            return ptr[i];
        } // 返回值不能为int，否则不支持a[i] = 4;
};

CArray::CArray(CArray &a) {
    if (!a.ptr) {
        ptr = NULL;
```

```
        size = 0;
        return;
    }
    ptr = new int[a.size];
    memcpy(ptr, a.ptr, sizeof(int) * size);
    size = a.size;
}
CArray &CArray::operator=(const CArray &a){
    if (ptr == a.ptr)
        return *this;
    if (a.ptr == NULL) {
        if (ptr) delete []ptr;
        ptr = NULL;
        size = 0;
        return *this;
    }
    if (size < a.size) {
        if (ptr) delete [] ptr;
        ptr = new int[a.size];
    }
    memcpy(ptr, a.ptr, sizeof(int) * a.size);
    size = a.size;
    return *this;
}
```