实验十一 运算符重载

一、 问题描述

1. 实验目的：

掌握运算符重载的若干基本概念和特性，并能够应用于程序编写。

掌握验证性实验的基本方法和过程(认知、实验、总结）。

2. 实验内容：

分别编写一段测试代码来回答任务书中的相关问题（每一个问题，用一个 工程文件，同时需要记录相应的调试过程），具体问题请参考“实验任务 说明11.doc”；

调试的过程；（动态调试的相关截图，比如 设置断点、查看当前变量值等）；

编译出来的可执行程序单独放在一个目录下（bin/exe/debug目录下，同时 附上输入数据说明和输出结果）

二、 实验过程

一、简答题

1、标准C++中，运算符重载的功能。

答：运算符重载，即对已有的运算符赋予多重含义，同一个运算符作用于不同类型的数据导致不同类型的行为。

2、请简要比较运算符重载的两种实现形式（成员函数、一般全局函数）异同点。

答：在C++中,可以把运算符函数定义成某个类的成员函数, 称为成员运算符函数。本质是个成员函数。有访问属性的限定，有类域的限定，有this指针，调用形式：对象.成员函数。友元运算符函数本质是个一般全局函数，无访问属性的限定，通过友元与类关联，无this指针。函数定义时，根据运算符所需要的运算对象来考虑参数格式。

3、哪些运算符不能重载？

答：C++不能重载的运算符有：成员访问运算符（.），间接成员选择符（.*），域运算符（::），长度运算符（sizeof），条件运算符（?:）。重载运算符的函数不能有默认的参数；重载的运算符必须和用户定义的自定义类型的对象一起使用，其参

数至少应有一个是类对象；

4、在运算符重载中，如何区分++、--的前缀和后缀两种情况？

答：++和--只有一个操作数,是单目运算符；对于++和--而言： ++和--运算符有两种使用方式，前置自增运算符和后置自增运算符，作用是不一样的。在自增（自减）运算符重载函数中，增加一个int型虚拟形参，就是后置自增运算符函数。前缀形式的返回类型一般是当前对象递增后的引用；后缀形式，先在修改之前创建原对象的副本，再将执行递增后的原对象返回。后缀运算符的返回值常声明为const。

5、C++中，如何分别实现：基本类型向类类型转换、类类型向基本类型转换？

答：标准类型转换为类类型借助：特殊形式的构造函数。用于类型转换的构造函数，即：具有一个标准类型参数的构造函数说明了一种从参数类型到该类类型的转换。类类型转换为基本类型：需要引入一种特殊的成员函数：类型转换函数，它在类对象之间提供一种类似显式类型转换的机制。类型转换函数的功能是将class_Name类型的对象转换为类型为type的实例。type可以是一个预定义类型，也可以是一个用户定义的类型。类型转换函数没有参数，没有返回类型，但这个函数体内必须有一条返回语句，返回一个type类型的实例。类型转换函数不能被重载,因为它没有参数。类型转换函数只能定义为一个类的成员函数，而不能定义为类的友元函数。

6、运算符“=”在不同情况下功能不同。结合复制构造函数和运算符重载函数进行分析。

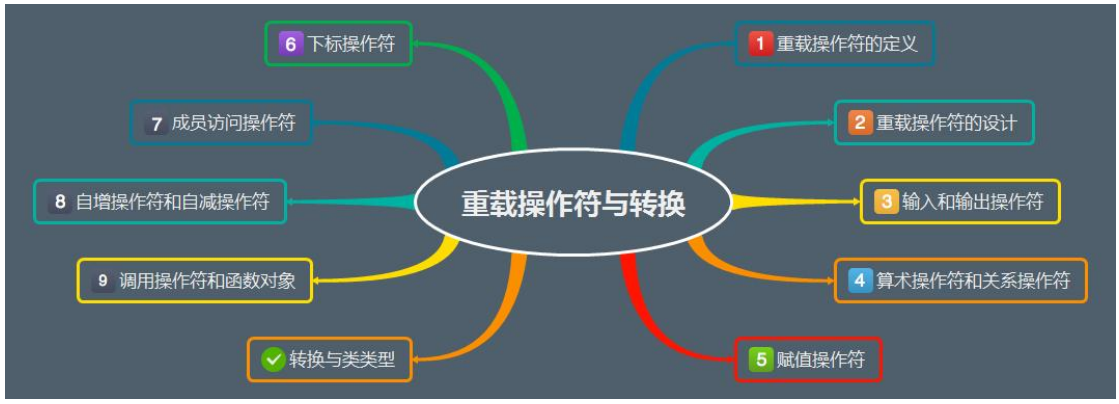
答：标准类型转换为类类型借助：特殊形式的构造函数。用于类型转换的构造函数，即：具有一个标准类型参数的构造函数说明了一种从参数类型到该类类型的转换。类类型转换为基本类型：需要引入一种特殊的成员函数：类型转换函数，它在类对象之间提供一种类似显式类型转换的机制。类型转换函数的功能是将class_Name类型的对象转换为类型为type的实例。type可以是一个预定义类型，也可以是一个用户定义的类型。类型转换函数没有参数，没有返回类型，但这个函数体内必须有一条返回语句，返回一个type类型的实例。类型转换函数不能被重载,因为它没有参数。类型转换函数只能定义为一个类的成员函数，而不能定义为类的友元函数。

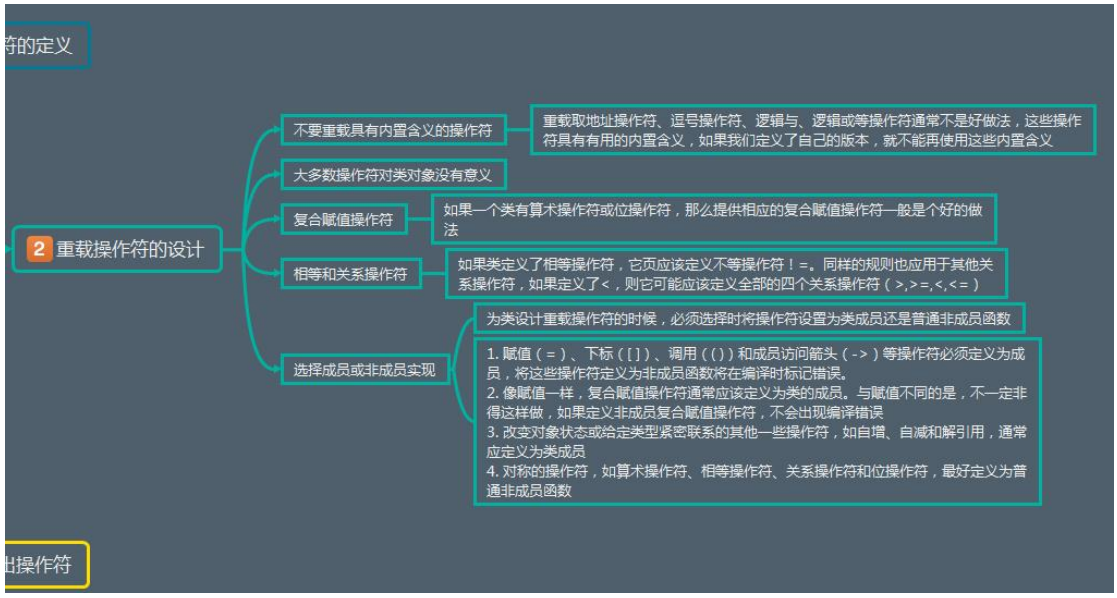
7、请描述“类类型向基本类型”转换的实现思路。

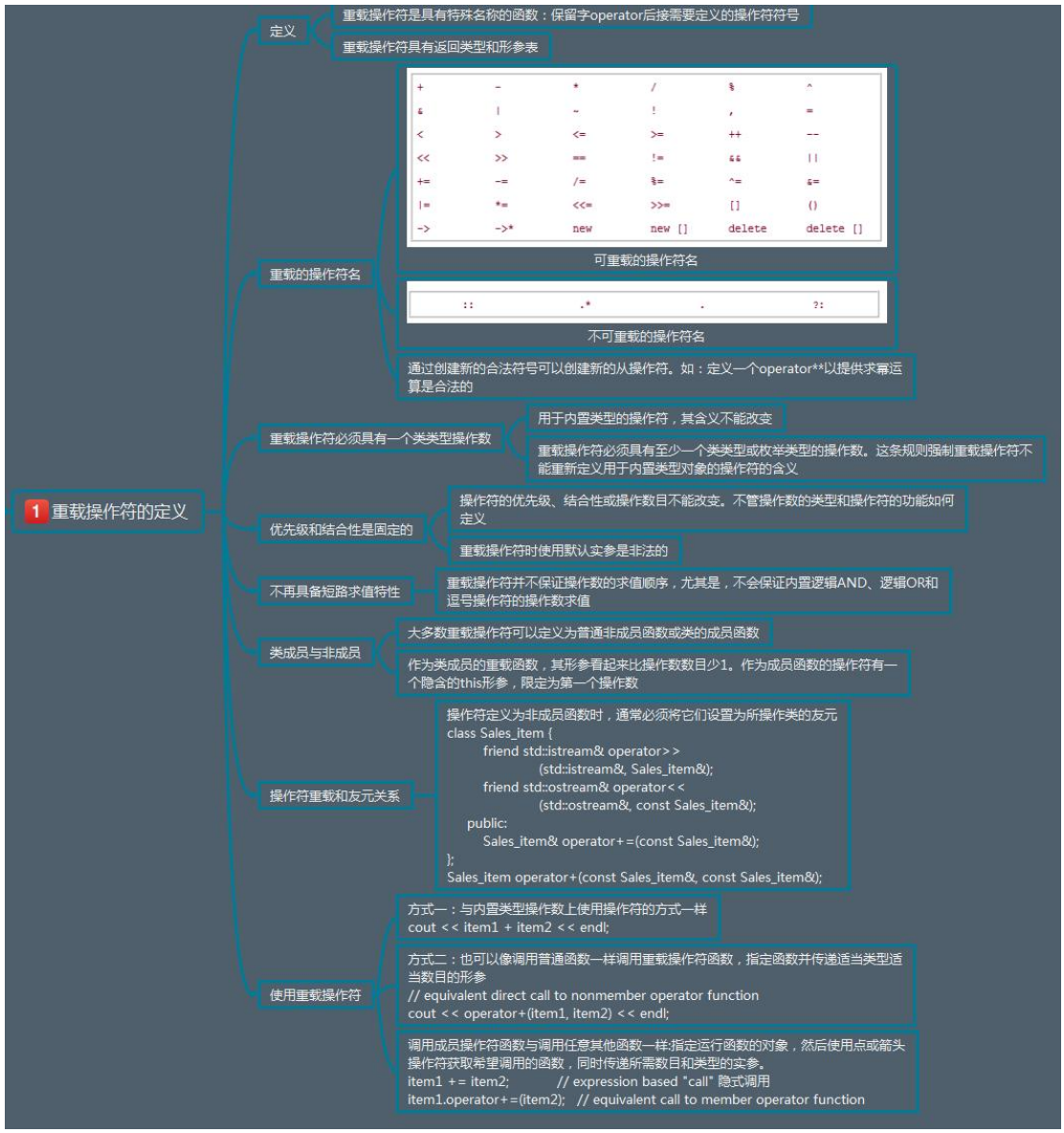
答：类型转换函数，它在类对象之间提供一种类似显式类型转换的机制。类型转换函数的功能是将class_Name类型的对象转换为类型为type的实例。type可以是一个预定义类型，也可以是一个用户定义的类型。类型转换函数没有参数，没有返回类型，但这个函数体内必须有一条返回语句，返回一个type类型的实例。类型转换函数不能被重载,因为它没有参数。类型转换函数只能定义为一个类的成员函数，而不能定义为类的友元函数。

（二）分析题

1、应用思维导图整理“运算符重载”知识点；







2、围绕“=”、“>>”、“<<”、“+”、“+=”、“++”、“--”、“[ ]”等运算符重载函数，着重分析：

1、运算符重载函数的函数原型；

答：

（1）赋值运算符=能把给定类型的对象复制到同一类型的另一个对象中。Cube& operator= (const Cube& aBox) { }。

（2）<<和>>运算符重载的声明形式：istream & operator >> (istream &,自定义类 &); ostream & operator << (ostream &,自定义类 &);

（3）“+”运算符，是一个二元运算符，且涉及到创建并返回新对象；可以定义为成员函数，且产生一个无名的对象，用值来初始化：inline Box Box::operator+(const Box& aBox) const也可以定义为一般函数，且产生一个无名的对象，用值来初始化。 inline Box operator+(const Box& aBox,const Box& bBox)
（4）“+=”：Box& Box::operator+=(const Box& right)
（5）++和--只有一个操作数,是单目运算符；++和--运算符有两种使用方式，前置自增运算符和后置自增运算符，作用是不一样的，在自增（自减）运算符重载函数中，增加一个int型虚拟形参，就是后置自增运算符函。Object & operator++( ); //前置++a const Object operator++(int) //后置 a++。
（6）重载函数调用运算符( ) ，xobj.operator( )(arg1,arg2)。重载下标运算符[ ]：掌握连续空间下，可判断下标是否超界，xobj.operator[](arg)。

2、函数的形参及函数返回值描述形式的分析；

答：

（1）operator=( )的参数是一个常对象引用，避免对原对象的修改；operator=( )的返回类型为对象的引用(引用,可避免不必要的复制操作，能作为左值)。
（2）重载运算符“<<”和“>>”的函数的第一个参数和函数类型都必须是*stream &类型，第二个参数是要进行输入操作的类；返回是引用，当需要将运算结果作为左值时，实现“连续操作” ，用引用返回 cin<<a<<b; 参数是引用， 用户可通过定义一个ostream的对象的引用s，则引用s 也可以使用运算符“<<。
（3）“+”运算符，是一个二元运算符，且涉及到创建并返回新对象；可以定义为成员函数，且产生一个无名的对象，用值来初始化。也可以定义为一般函数，且产生一个无名的对象，用值来初始化。
（4）+=运算符重载。函数功能:给左操作数*this加上右操作数,修改了左操作数;函数返回值:涉及赋值,需要返回一个引用;
（5）前缀形式的返回类型一般是当前对象递增后的引用；后缀形式，先在修改之前创建原对象的副本，再将执行递增后的原对象返回。后缀运算符的返回值常声明为const。
（6）[ ]与数组（字符数组）等关联；用户可以自定义添加上新的功能，如：在进行下标访问时。

3、上述运算符有些只能采用成员函数的形式，有些只能采用普通函数的形式，请分析缘由列表整理。

答：=，()，[]，->，->*必须定义为成员函数。如果是一般函数（友元）可能会造成左边的值是个常量的问题。只能将重载“<<”和“>>”的函数作为友元函数或普通函数，而不能将它们定义为成员函数。

三、程序基本题

（一）

```
有以下类：
class Complex
    { private:
        double real;
        double im;
      public:
        Complex(double r, double i)          {   real=r;
            im=i;
         }
         Complex(const Complex &t)              {   real=t.real;
            im=t.im;
           cout<<"HAHA       ";
         }
        ____________//要求写函数，实现复数的加，对“+”进行重载
        ____________///要求写函数，实现对Complex对象的>>和<<操作

};
```

```cpp
#include <iostream>
using namespace std;

class Complex
{
private:
    double real;
    double im;
public:
    Complex& operator++();//前置++a
    Complex operator++(int);//后置a++
    Complex(double r, double i) {
        real = r;
        im = i;
    }
    //拷贝构造函数
    Complex(const Complex& t) {
        real = t.real;
        im = t.im;
        cout << "HAHA    " << endl;
    }
    Complex() {
        real = 0;
        im = 0;
    };
```

```
    friend ostream& operator <<(ostream&, Complex&);
    friend istream& operator >>(istream&, Complex&);
    //要求写函数，实现复数的加，对“+”进行重载
    //最后的const表明调用函数对象不会被修改
    //括号中的const表示参数对象不会被修改
    Complex operator +(const Complex& x)const {
        Complex temp;
        temp.real = real + x.real;
        temp.im = im + x.im;
        return temp;
    }
    double getReal()const {
        return real;
    }
    double getIm()const {
        return im;
    }
    Complex& operator+=(const Complex& c) {
        real += c.real;
        im += c.im;
        return *this;
    }
};
```

```
Complex& Complex::operator++() {
    real++;//先增量
    return *this;//再返回对象
}
Complex Complex::operator++(int) {
    Complex temp(*this);//复制构造函数，对象存放原有对象值
    real++;
    return temp;
}

//要求写函数，实现对Complex对象的>>和<<操作
ostream& operator <<(ostream& output, Complex& c) {
    output << "(" << c.real << "+" << c.im << "i)" << endl;
    return output;
};
istream& operator >>(istream& input, Complex& c) {
    cout << "请输入复数的实数部分和虚数部分" << endl;
    input >> c.real >> c.im;
    return input;
};
int main() {
    Complex c1(2,3), c2(3,4), c3(5,6),c4;
    c4 = c1 + c2 + c3;
    cout << c4 << endl;

    Complex c5, c6;
    cin >> c5 >> c6;
    cout << c5 << c6 << endl;

    c2 = c1++;
    c2 = ++(++c1);
    cout << c2 <<c3<< endl;

    c1 += c2 += c3 += c4;
    cout << c4 << endl;
    return 0;
}
```

（1）完成代码后，在main中添加”对象1+对象2+对象3”，看看程序的执行

```
Complex c1(2,3), c2(3,4), c3(5,6),c4;
c4 = c1 + c2 + c3;
cout << c4 << endl;
```

```
Microsoft Visual Studio 调试控制台
HAHA
HAHA
(10+13i)
```

（2）完善Complex类，增加成员函数来读取私有成员信息，如：double getReal() const;

```
double getReal()const {
    return real;
}
double getIm()const {
    return im;
}
```

（3）基于（2）的结果，实现cin>>c1>>c2；并且能cout<<直接输出c1和c2的求和结果

```
Complex c5, c6;
cin >> c5 >> c6;
cout << c5 << c6 << endl;
```

```
请输入复数的实数部分和虚数部分
1 2
请输入复数的实数部分和虚数部分
2 3
(1+2i)
(2+3i)
```

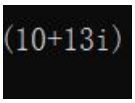
（4）完善代码，以实现++c1，++(++c1)

```
c2 = c1++;
c2 = ++(++c1);
cout << c2 <<c3<< endl;
```

```
HAHA
HAHA
(5+3i)
(5+6i)
```

（5）实现+=运算符重载，能够实现类似: c1+=c2+=c3+=c4; 并进一步分析比较：返回值是个对象或是引用之间的区别

```
c1 += c2 += c3 += c4;
cout << c4 << endl;
```



分析：引用,可避免不必要的复制操作，能作为左值。返回值为对象引用不需要初始化内存临时对象，返回对象的引用，可以作为左值，能实现连续的书写。

（二）运算符重载下的的函数重载

程序员决定了运算符重载函数，运算符重载函数决定了运算符究竟如何运算。无论是什么运算符，要仔细地考量：该运算符对于该类的实际意义是什么。比如：下面题目中，+运算符，究竟是加的是什么，是否有意义？当然，如果仅仅是为了训练，那就另当别论；在>中，需要考虑用户可能的多种比较：对象>对象　对象>数值　数值>对象

设计一个长方形类Triangle，包含长和宽两个私有数据成员。重载运算符>

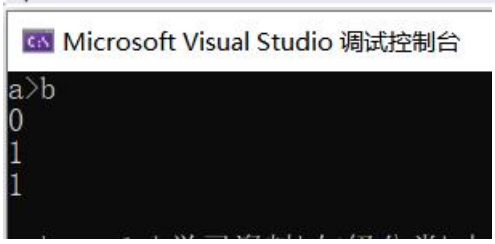
要求实现如下功能：

1、不但可以实现两个矩形对象面积大小的比较，同时可以实现矩形对象面积和某个数值的比较。这就决定了需要实现多个运算符重载（函数重载）

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

class Rectangle {
private:
    double w, h;
    double area;
public:
    Rectangle(double width = 0, double height = 0)  {
        w = width, h = height, area = height * width;
    }
    operator double(){
        return area;
    }
    //面积比较
    double operator>(Rectangle& a){
        if (w * h <= a.w * a.h) return 0;
        else  return w * h;
    }
    double operator>(double a){
        if (w * h <= a) return 0;
        return w * h;
    }
};
```

2、需要考虑用户书写的多种可能：对象>对象　对象>数值　数值>对象 对象>对象>对象

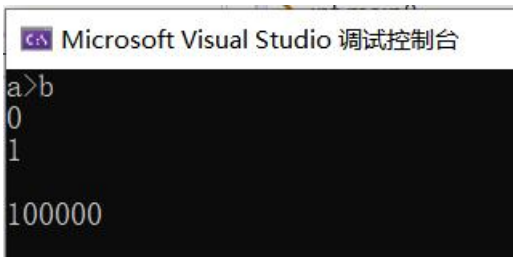
```
int main()
{
    Rectangle a(4, 5), b(4, 4),c(1,2);
    if (a > b) cout << "a>b" << endl;
    else cout << "a<=b" << endl;

    bool one = (10.1 > a);
    bool two = (a > 10.1);
    cout << one << endl<< two << endl;

    bool three = (a > b > c);
    cout << three << endl;
    return 0;
}
```



3、请描述在main中执行“对象>对象>对象”的执行过程

以a>b>c为例，首先比较a和b，有a>b，接着比较b和c有b>c，最后输出a>b>c为true。

```
bool three = (a > b > c);
bool four = (a > c > b);
bool five = (b > c > a);
bool six = (b > a > c);
bool seven = (c > a > b);
bool eight = (c > b > a);
cout << three<<four<<five<<six<<seven<<eight << endl;
return 0;
```



四、赋值运算符专辑

（一）对于赋值运算符，分析下列代码执行结果，详细探讨存在的不足，着重

从“obj1=obj2=obj3”以及“(obj1=obj2)=obj3”分析对该运算符重载函数参数和返回值描述的影响。

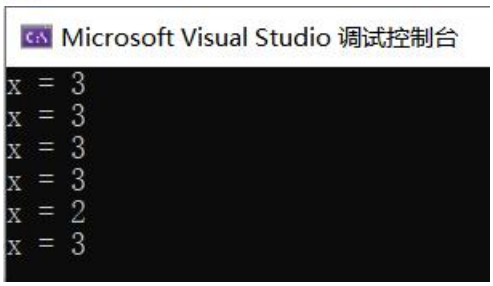
```
#include <iostream>
using namespace std;

class Sample {
private:
    int x;
public:
    Sample() {} //考虑为何要写这个
    Sample(int a) { x = a; }
    void disp() { cout << "x = " << x << endl; }
    void operator =(Sample s);
};
void Sample::operator =(Sample s) {
    x = s.x;
}
void main() {
    Sample obj1(10);
    Sample obj2;
    obj2 = obj1;
    obj1.disp();
    obj2.disp();
}
```

答：可以将operator=( )的参数改为常对象引用，避免对原对象的修改；operator=( )的返回类型改为对象的引用(引用,可避免不必要的复制操作，能作为左值) 函数体return语句：return *this。

在第一条语句中，先让obj2=obj3，然后返回对obj2对象的引用，再让obj1=obj2。最后输出obj1、obj2、obj3全为3。

在第二条语句中，先让obj4=obj5，然后返回obj4的引用，并让obj4再等于obj6，所以最后obj4和obj6都为3，而obj5的值未改变仍然为2。

```cpp
#include <iostream>
using namespace std;

class Sample {
private:
    int x;
public:
    Sample() { x = 0; } //考虑为何要写这个
    Sample(int a) { x = a; }
    void disp() { cout << "x = " << x << endl; }
    Sample& operator =(const Sample& s);
};
Sample& Sample::operator =(const Sample& s) {
    x = s.x;
    return *this;
}
void main() {
    Sample obj1(1),obj2(2), obj3(3);
    obj1 = obj2=obj3;
    obj1.disp();
    obj2.disp();
    obj3.disp();
    Sample obj4(1), obj5(2), obj6(3);
    (obj4 = obj5) = obj6;
    obj4.disp();
    obj5.disp();
    obj6.disp();
    return;
}
```



五、设计一个字符串类MyString，并能完成以下功能：

完成：构造函数、复制构造函数、=运算符重载和析构函数等。

能使用+=运算符进行两个字符串的连接运算

能使用+运算符进行两个字符串的复制

能使用==运算符对两个字符串判断是否相等
能使用=运算符进行两个字符串的赋值运算
能使用[ ]运算符返回字符信息，并能做越界判断

1. 头文件，类的声明与定义，成员函数和友元函数的声明

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

class MyString
{
private:
    char* str;
    int len;
    void memError();
public:
    MyString() { str = NULL; len = 0; }
    MyString(const char*);
    MyString(MyString&);
    ~MyString() {if (len != 0)delete[]str;}
    int length() { return len; }
    char* getValue() { return str; }
    MyString operator +=(MyString&);
    char* operator+=(const char*);
    MyString operator=(MyString&);
    char* operator=(const char*);
    bool operator ==(MyString&);
    bool operator ==(const char*);
    MyString operator +(MyString&);
    char* operator+(const char*);
    // 重载流操作符和提取符，流操作符必须重载为友元
    friend ostream& operator <<(ostream&, MyString&);
    char & operator[](int idx);
};
```

2、函数的定义

```cpp
#include "5.h"

//内存分配失败，调用exit()终止
void MyString::memError() {
    cout << "内存分配出错" << endl;
    exit(0);
}
//构造函数
MyString::MyString(const char* sptr) {
    len = strlen(sptr);
    str = new char[len + 1];
    if (str==NULL){
        memError();
    }
    strcpy(str, sptr);
}
//拷贝构造函数
MyString::MyString(MyString& right) {
    str = new char[right.length() + 1];
    if (str == NULL) memError();
    strcpy(str, right.getValue());
    len = right.length();
}
```

```
//能使用 += 运算符进行两个字符串的连接运算
//返回值是自身调用的对象
MyString MyString::operator+=(MyString& right) {
    char* temp = str;
    str = new char[strlen(str) + right.length() + 1];
    if (str == NULL)memError();
    strcpy(str, temp);
    strcat(str, right.getValue());
    if (len != 0)delete[]temp;
    len = strlen(str);
    return *this;
}
char* MyString::operator +=(const char* right)
{
    char* temp = str;
    str = new char[strlen(str) + strlen(right) + 1];
    if (str == NULL)  memError();
    strcpy(str, temp);
    strcat(str, right);
    if (len != 0) delete[] temp;
    return str;
}

//能使用 + 运算符进行两个字符串的复制
//返回值是调用对象
MyString MyString::operator +(MyString& right) {
    MyString temp(*this);
    temp += right;
    return temp;
};
```

```
char* MyString::operator+(const char* a){
    MyString temp(* this);
    temp += (const char*)a;
    char* ans = temp.str;
    return ans;
};

//能使用 == 运算符对两个字符串判断是否相等
// 如果调用对象和参数对象的 str 内容相同，返回true，否则返回 false
bool MyString::operator ==(MyString& right)
{
    return strcmp(str, right.getValue()) == 0 ? true : false;
}
bool MyString::operator ==(const char* right)
{
    return strcmp(str, right) == 0 ? true : false;
}

//能使用 = 运算符进行两个字符串的赋值运算
//返回值是调用对象自身
char* MyString::operator=(const char* right) {
    if (len != 0)  delete[] str;
    len = strlen(right);
    str = new char[len + 1];
    if (str == NULL) memError();
    strcpy(str, right);
    return str;
}
```

```
MyString MyString::operator =(MyString& right)
{
    if (len != 0)  delete[] str;
    str = new char[right.length() + 1];
    if (str == NULL)  memError();
    strcpy(str, right.getValue());
    len = right.length();
    return *this; // 返回调用对象本身
}

//能使用[]运算符返回字符信息，并能做越界判断
char& MyString::operator[](int idx) {
    if (idx < len) return str[idx];
    else {
        cout << "数组越界" << endl;
        char zero[] = "\0";
        *this += (const char*)zero;
        return str[len];
    }
};
// 重载流插入符<<，返回一个引用
ostream& operator <<(ostream& strm, MyString& obj)
{
    strm << obj.str;
    return strm; // 将当前流对象返回
}
```

3、main函数的定义

```
int main()
{
    MyString obj1("I "), obj2("love "),obj3("China");
    MyString obj4 = obj1; // 调用拷贝构造函数
    cout << obj1[3] << endl;
    char str[] = "!";
    cout << "对象 1: " << obj1 << endl;
    cout << "对象 2: " << obj2 << endl;
    cout << "对象 3: " << obj3 << endl;
    cout << "对象 4: " << obj4 << endl;
    cout << "字符数组: " << str << endl;
    // 演示对象 += 操作
    obj1 += obj2;
    obj1 += obj3;
    obj1 += str;
    cout << "对象 1: " << obj1 << "\n\n";
    // 演示关系运算
    if (obj1 == str)  cout << obj1 << " 等于字符数组 " << str << endl;
    else  cout << obj1 << " 不等于字符数组 " << str << endl;
    if (obj3 == "China") cout << obj3 << " 等于 China\n";
    else cout << obj3 << " 不等于 China\n";
    return 0;
}
```

4、运行结果

```
Microsoft Visual Studio 调试控制台
数组越界

对象 1: I
对象 2: love
对象 3: China
对象 4: I
字符数组: !
对象 1: I love China!

I love China! 不等于字符数组 !
China 等于 China
```

三、 附录

源程序文件项目清单：src/homework

3.1.cpp 3.2.cpp 4.cpp 5.cpp