实验二 函数与运算符的重载

**课程名称：** 面向对象技术(C++)

**实验项目：** 函数与运算符的重载

**实验仪器：** 计算机

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **项目** | **报告格式**  **Report format** | **代码质量**  **Code quality** | **注释质量**  **Comment quality** | **逻辑或思想描述**  **Necessitate logical description** | **独创性**  **Originality** | **合计**  **Total** |
| **百分比(%)**  **percentage** | **10** | **25** | **25** | **25** | **15** | **100** |
| 得分（score） |  |  |  |  |  |  |

**系 别：** 计算机系

**专 业：** 计算机大类

**班级姓名：** 计类1805/刘科宏

**学 号：** 2018011252

**日 期：** 4月15日

**成 绩：**

**同组成员：**

**指导教师：** 丁 濛

# 实验目的(Objects)

1. 了解函数重载的概念，学习函数重载的方法，了解C++如何区分不同函数的机制；

Understand the concept of function overloading and how C++ distinguishes different functions. Learn how to define an overloaded function.

1. 学习对类的成员函数的重载方法；

Learn how to define an overloaded member function.

1. 了解操作符重载的概念，学习操作符重载的方法，了解C++中操作符的实质；学会通过成员函数形式和非成员函数形式对操作符进行重载。

Understand the concept of operator overloading and how to overload an operator. Understand the essential of an operator in C++; Learn how to overload an operator in member-function and non-member function way.

# 实验内容(Contents)

1. 利用函数重载，实现函数add, 该函数将返回两个整型变量的和或者两个浮点型变量的和；

Implement different overloading functions add, which can return the sum of two integers or two floats.

1. 定义一个复数类 Complex，该类包含两个私有双精度浮点型变量 double real, double imagine。定义该类的三种构造函数，即默认构造函数，有参构造函数和拷贝构造函数。以Complex类的成员函数的形式操作符+，使其完成一个复数的加法；以Complex类的成员函数的形式操作符-，使其完成一个复数的减法；重载Complex类的赋值操作符=，使其正确完成一个复数的赋值。以友元的形式，重载流操作符<<，完成以 real + image i 的方式打印复数。

用非成员函数形式的方式重载\*操作符，分别完成两个Complex类型变量的加，减与乘法。

Complex operator\* (const Complex& op1, const Complex& op2);

Define a class Complex, which includes two private double type member variables, real and imagine.

Define its three kinds of constructors.

Overload operator +,- through Complex’s member function, performing the addition and subtraction of two complex number respectively.

Overload the assignment operator = to implement the assignment of a complex.

Overload operator << through non-member function way to print a complex in “*real* + *imagine* i” style, where *real* and *imagine* stand for two numbers.

1. 设计一个专门用来处理字符串的类String。要求：
   1. 该类具有一个char \*类型的私有成员变量str用来记录字符串的地址，具有一个unsigned int类型的私有成员变量length用来记录字符串的长度。
   2. 设计相应的构造函数使得该类可以支持如下的初始化方式:  
      String a;  
      String b(a);  
      String c(“This is a test.”);
   3. 在析构函数中定义必要的操作，对申请的内存进行释放；
   4. 设计并实现一个公有函数unsigned int size()，来返回当前字符串的长度；
   5. 设计并实现一个公有函数String& append(const String& other), 将形参所代表的字符串内容“添加”到自己的字符串内容之后，即  
      String a(“This is”);  
      String b(“ a test.”);  
      a.append(b); // ==> a中str指向的内容变为 “This is a test.”
   6. 重载赋值操作符=，即定义成员函数 String& operator= (const String &)，使其正确完成String类型的赋值操作；  
      String a, b;  
      a = b;
   7. 重载必要的操作符，实现形如 String a = “Hello ”; a += “World!”;的功能。
   8. 重载必要的操作符，实现形如 String a, b, c; c = a + b; 的操作过程。
   9. 重载必要的操作符，当完成   
      String a(“Hello ”);   
      a << “world”;  
      的操作过程后，a所代表的字符串为”Hello world”。
   10. 重载必要的操作符，当完成  
       String a(“test”);  
       std::cout << a; 的操作过程后，在屏幕上输入 test
   11. 重载必要的操作符，当完成  
       String a(“test”);  
       a[2] = ’S’; 后，a所代表的字符串为”teSt”。
   12. 编写相应的测试用例，来验证以上功能的正确性（包括内存使用的正确性）。

Design a class String which is used to describe and process a string:

1. It has a private member variable, char \*str, indicating the initial address of the expressing string; and a private member variable unsigned int length to indicate the length of the expressing string.
2. Define necessary constructors to support the following initializations:

String a;  
String b(a);  
String c(“This is a test.”);

1. Define destroyer to release the allocated memory.
2. Define a public member function unsigned int size() to return the length of the expressing string.
3. Overload the assignment operator =, (means define String& operator= (const String &) ), to let the following assignment runs correctly.

String a, b;  
a = b;

1. Define a public member function *String& append(const String& other)*, which will append the string expressed by *other* to the end of its own string, eg:   
   String a(“This is”);  
   String b(“ a test.”);  
   a.append(b); // ==> the string holding by a will be “This is a test.”
2. Overload the necessary operator to implement the following operation:

String a = “Hello ”;

a += “World!”; //a 🡪 “Hello World”

1. Overload the necessary operator to implement the following operation:

String a, b, c;

c = a + b;

1. Overload the necessary operator to implement the following operation:

String a(“Hello ”);   
a << “world”; //a 🡪 “Hello World”

1. Overload the necessary operator to implement the following operation:

String a(“test”);  
std::cout << a;//will display the string expressed by a on screen.

1. Overload the necessary operator to implement the following operation:

String a(“test”);  
a[2] = ’S’; // a-> “teSt”.

1. Write a main function to test the correctness of your String class.

# 实验内容 (Your steps or codes, and Resultss)

1.

#include <iostream>

using namespace std;

int add(const int x, const int y)//专门对于整型相加的函数，传入的x与y不能修改需要加const

{

printf("int add(const int , const int)\n");//输出所调用的函数类型

return x + y;

}

float add(const float x, const float y)//专门对于浮点型相加的函数

{

printf("float add(const float , const float)\n");

return x + y;

}

int main()

{

int n;

cin >> n;

while(n--)//循环执行n次，最后n为0时退出

{

float f1, f2;

int a, b;

cin >> f1 >> f2 >> a >> b;

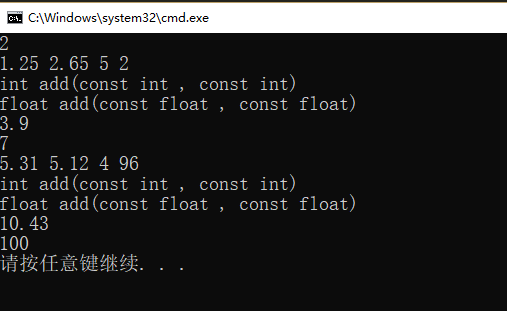
cout << add(f1, f2) << "\n" << add(a, b) << "\n";

}

return 0;

}

执行结果



图表 1：实验1的结果

2.

#include <iostream>

#include <stdio.h>

using namespace std;

class Complex{

private:

double real;

double imagine;

public:

Complex() : real(0),imagine(0){//无参的构造函数，并且使用0去构造初始化成员

}

Complex(double a, double b) : real(a), imagine(b){//有参数的构造函数，并且使用传入的a和b去初始化成员

}

Complex(const Complex& other) : real(other.real),imagine(other.imagine){

}

Complex operator+ ( const Complex& other) const{//重载操作符+号，因为是两个自己定义的类相加，第一个const加上的原因是可能传入的是一个右值不然无法引用，并确保传入的Complex不被修改，第二个const是因为函数不会对任何成员进行修改

return Complex(real + other.real, imagine + other.imagine);

}

Complex operator- ( const Complex& other) const{//重载操作符+号，因为是两个自己定义的类相减，两个const原因同上

return Complex(real - other.real, imagine - other.imagine);

}

Complex& operator= ( const Complex& other) {//重载操作符=号，便于让自己定义的complex类赋值时进行操作

real = other.real;

imagine = other.imagine;

return \*this;

}

friend ostream& operator<< ( ostream&,const Complex&);//友元函数，由于非成员重载<<函数会对类中的private的成员进行修改，所以必须在类中加入此友元函数，这里的const必须加上，防止传入的值为右值

friend Complex operator\* ( const Complex&, const Complex&);//友元函数，同样是因为重载\*函数会对类中的private的成员进行修改，这里的两个const也同样是防止引用右值，并且确保不会修改传入的两个Complex

};

ostream& operator<< ( ostream& output, const Complex& other){//对于操作符<<的重载,在使用cin输入时会使用到，以确定输出的格式

cout << other.real << " + " << other.imagine << " i\n";

return output;

}

Complex operator\* ( const Complex& a, const Complex& other){//重载操作符\*，使得自己定义的类之间可以进行相乘

return Complex(a.real \* other.real -a.imagine \* other.imagine, a.imagine \* other.real + a.real \* other.imagine);

}

int main(){

Complex c1(4.234234, -27.6), c2(3.5, -4.7);

Complex c3, c4;

c3 = c2 + c1;

c4 = c3 \* c1;

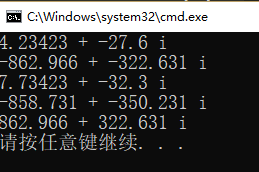
c2 = c4 - c1;

cout << c1 << c2 << c3 << c4 << c1 - c4;//最后输出的是一个右值，所以对应了前面的重载<<的const，必须时刻注意

return 0;

}

执行结果：



图表 2实验2的执行结果

3.

#include <stdio.h>

#include <iostream>

#include <string.h>

#include <assert.h>

using namespace std;

class String{

private:

char \*str;

unsigned int l;

public:

String() : str(NULL), l(0){//无参的构造函数，并且构造初始化\*str与l

}

String(const char \*s)//带有一个字符串为参数的构造函数

{

if(s)//先判断传入的字符串是否为空

{

l = strlen(s);

str = new char[strlen(s) + 1];

strcpy(str, s);

}

else//字符串为空则还是把指针\*str指向空

{

str = NULL;

l = 0;

}

}

String(const String& other) : l(other.l)//拷贝构造函数，可以直接使用需要拷贝构造的l来构造this的l

{

if(other.str)//判断需要拷贝的对象str是否为空

{

str = new char[other.l + 1];

strcpy(str, other.str);

}

else//为空的话，让构造出来的对象str也指向空

str = NULL;

}

~String()

{

delete [] str;//str的空间是由new开辟出来，所以最后得使用delete删去空间

l = 0;

}

unsigned int size() const//用于返回对象的\*str中的字符串的长度

{

unsigned int i;

if(str)

{

for(i = 0;;i++)//一个个检测str的每一位是否为'\0'，并且计数，最后检测到'\0'时返回计的数

{

if(str[i] =='\0')

break;

}

return i;

}

return 0;

}

String& append(const String& other)//在进行两个String类对象相加时的函数，为的是让自己定义的类的字符串相加并且长度相加

{

l = other.l + l;//长度相加

if(other.str)//判断相加的字符串是否为空

{

char \*x = new char[l + other.l + 1];//定义一个指针并指向一个开辟出两个字符串长度之和的空间

if(str)//判断被加的对象字符串是否为空

{

strcpy(x, str);

strcat(x,other.str);//把拷贝好的x字符串与other的字符串相加，strcat函数可以用于两字符串相加，更加方便

delete [] str;

}

else

strcpy(x, other.str);//若自己的字符串为空，直接拷贝other的字符串

str = x;

return \*this;

}

else

return \*this;//若other的字符串为空，直接原封不动的返回

}

String& operator= (const String& other)//重载操作符=，加const是为了防止传入的为右值

{

l = other.l;

if(this == &other)//若是和自己赋值，则原封不动返回自己，以免误删自己的str

return \*this;

if(str)//先判断Str是否为空

{

delete [] str;//不为空则删除原本str指向的开辟出的空间，反正空间泄漏

}

if(other.str)

{

str = new char[other.l + 1];

strcpy(str, other.str);

}

else

str = NULL;

}

String& operator+= (const String& other)//重载操作符+=号

{

return append(other);//效果和append函数相似，可以直接使用其与other相加并返回

}

String operator+ (const String& other) const//重载操作符+号，加const说明函数对成员不会进行修改

{

String A(\*this);//先用this来构造出一个A，由于+的返回值不是自己，所以得创建出A来返回

A += other;

return A;

}

char& operator[] (const int i) const//重载操作符[]，这样可以使得调用具体字符会方便很多

{

assert(i >= 0 && i < l);//加入了assert函数，来判断str[i]是否存在，不存在就返回错误报告

return str[i];

}

bool isNull()const{//用于判断str是否为空的函数

if(str)

return true;

else

return false;

}

String& operator<< ( const String& other){//重载操作符<<，效果与append()函数相似

return this->append(other);

}

friend ostream& operator<< ( ostream& output, const String& other);//友元函数，因为<<将会使用到类的private成员

};

ostream& operator<< ( ostream& output, const String& other){//重载函数cout加操作符<<，加const防止传入的值为右值

if(other.str)//判断other的str是否为空，防止输出null

cout << other.str;

else

return output;

}

int main(){

String a("This is "), b(a);

String c;

cout << "a:" << a << "\n" << "b:" << b << "\n" << "c:" << c << "\n";

c = "a test.";

cout << "c:" << c << "\n";

a.append(c);

cout << "a:" << a << "\n";

{

String a("Hello"), b;

a += " World";

b << (a + b);

cout << "b:" << b << "\n";

String c;

c = b + "!";

cout << "c:" << c << "\n";

}

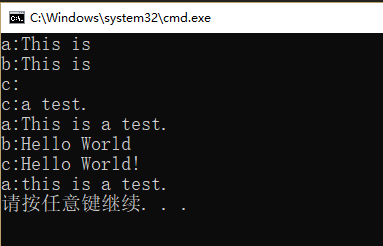
a[0] = 't';

cout << "a:" << a << "\n";

return 0;

}

执行结果：



图表 3：实验3的执行结果

# 实验总结 (Conclusion)

通过了这一次的实验，又再一次学到了许多的东西。

1. 首先对函数的重载又有了更深的理解，与c不同的是c++函数名称是可以重复的，从实验1就可以看出，只要我们传入的参数种类不同，函数是可以重新定义的。
2. 然后引入了一个新的概念，左值与右值，这个概念对于一个东西能否被引用有着巨大的联系，来说一般const Type& 可以解决大部分的问题，但对于是否会遇到是左值还是又是右值问题的判断必须得达到精细而及快速，如同在实验2当中就遇到了这个问题，必须得马上反应出问题的原因，做出正确的改正，接着在实验3中就很好的避免了。
3. 学习到了使用<iostream>库当中的cin与cout，在多数情况下都可以达到方便输入与输出的目的，并且cin与cout可以通过操作符的重载来很简便的输出或输入由自己定义的类。
4. 对于操作符的重载印象是最深的，完全改变了我对操作符的认识，不同的操作符有时重载起来会有些许的不同，就比如重载+=与+猛一看以为相识，其实还是会很大的不同，对于+=来说返回值就是被加数自己本身，被加数会发生改变，而+仅仅代表两个类相加，都不会发生改变，因此+=的类型必须为Type&。而又对于cin与cout来说也重载很特殊，由于它们的构造函数是private的等原因，因而与其他重载很不同。
5. 还学习到了使用初始化列表来初始化类成员，这样可以省去许多类的构造与赋值，使得程序运行更加流畅。
6. 学习到了要简化程序的内容，在开始的实验3的程序，有许多的繁杂的步骤，通过老师的指点，改为了一些函数来代替或者删去了繁杂无意义的内容，使程序更加流畅，这也是我明白了程序运行正确是远远不够的，说不定还蕴含着一些隐藏的问题或是代码不够优化，这也同样需要花费很多的时间去发现与寻找。