实验二 函数与运算符的重载

Experiment Two: Function and Operator Overloading

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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(Department)： 　　计算机学院**

**专 业(Major)： 　　计算机类**

**班级姓名(Class & Name)：　 计类1710**

**学 号(Student ID)： 2017011471**

**日 期(Data)： 2018.4.9**

**指导教师(Supervisor)：** 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.

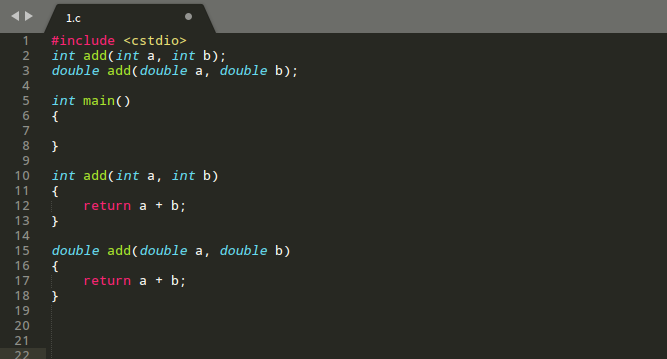
1. Overload the necessary operator to implement the following operation:

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

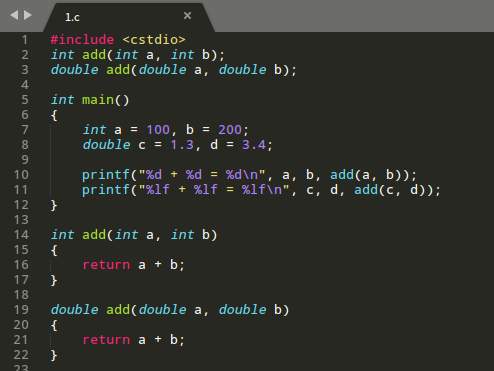
1. Write a main function to test the correctness of your String class.

# 实验步骤(Your steps or codes)

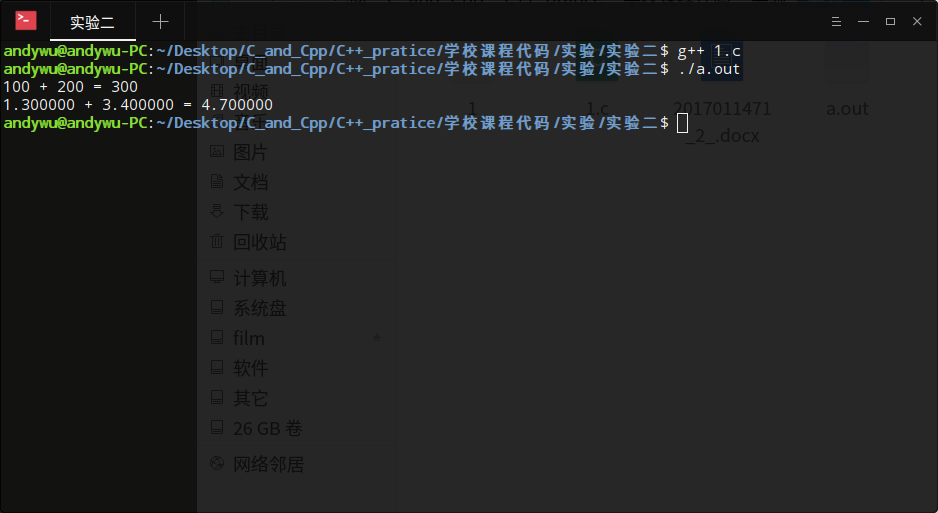
* 1. 实验一
     1. 打开IDE或者文本编辑器（本次实验用sublime３）
     2. 搭好程序框架，编写函数



* + 1. 在main函数定义变量，检验函数运行结果。



* + 1. 编译运行



* + 1. 代码

#include <cstdio>

int add(int a, int b);

double add(double a, double b);

int main()

{

int a = 100, b = 200;

double c = 1.3, d = 3.4;

printf("%d + %d = %d\n", a, b, add(a, b));

printf("%lf + %lf = %lf\n", c, d, add(c, d));

}

int add(int a, int b)

{

return a + b;

}

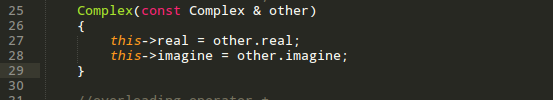
double add(double a, double b)

{

return a + b;

}

* 1. 实验二
     1. 构建类
     2. 编写各个函数，其中要注意，拷贝构造函数的参数必须要加const前缀，否则将无法接收右值引用



* + 1. 代码如下

#include <cstdio>

#include <iostream>

using namespace std;

//declare a class

class Complex

{

//private available

private:

double real;

double imagine;

public:

//The default constructor

Complex()

{

}

//The constuctor with availables

Complex(double real\_, double imagine\_):real(real\_), imagine(imagine\_)

{

}

//Copy constructor

//"const" must be write, otherwise the constructor can't accept rvalue reference

Complex(const Complex & other)

{

this->real = other.real;

this->imagine = other.imagine;

}

//overloading operator +

Complex operator+(const Complex other) const

{

//declare a temporary variable to save the value of "this + other"

Complex t;

t.real = this->real + other.real;

t.imagine = this->imagine + other.imagine;

return t;

}

//overloading operator -

Complex operator-(const Complex & other) const

{

//declare a temporary variable to save the value of "this - other"

Complex t;

t.real = this->real - other.real;

t.imagine = this->imagine - other.imagine;

return t;

}

//overloading operator =

Complex & operator=(const Complex & other)

{

this->real = other.real;

this->imagine = other.imagine;

return \*this;

}

friend Complex operator\*(const Complex &a, const Complex & b);

friend std::ostream & operator<<(std::ostream &o, Complex a);

};

Complex operator\*(const Complex &a, const Complex & b)

{

Complex t;

t.real = a.real \* b.real - a.imagine \* b.imagine;

t.imagine = a.real \* b.imagine + a.imagine \* b.real;

return t;

}

std::ostream & operator<<(std::ostream &o,const Complex a)

{

o<< a.real << " + " << a.imagine << " i";

return o;

}

int main()

{

Complex a(1, 3), b(3, 4), c(5, 6);

cout << "a:" << a << endl;

cout << "b:" << b << endl;

cout << "c:" << c << endl;

cout << "a + b = " << a + b << endl;

cout << "a - b = " << a - b << endl;

cout << "a \* b = " << a \* b << endl;

return 0;

}

* 1. 实验三

# 实验结果(Results)

# 结论(Conclusions)