**实验4：类的运算符重载**

**姓名\_\_\_\_金宣成\_\_\_\_\_\_班级\_\_\_计科03\_\_\_\_\_\_学号\_202203151310\_**

* **请阅读此说明：实验4满分100分；做完实验后请按要求将代码和截图贴入该文档。然后将此文档、源代码文件（.hpp, .cpp）打包上传到学习通。**

**1、（总分15分）课堂例题巩固。**

* **实验要求：**

**1）（5分）运行文件夹“4-1 static”中的两个程序，体会static的作用。**

**（1）静态成员是跟着整个类生成的，不是跟着某个对象，也只分配一次空间。而且A gg()**

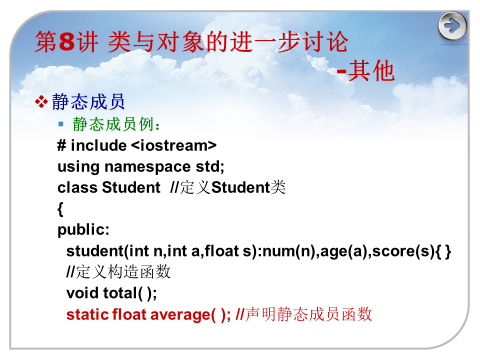
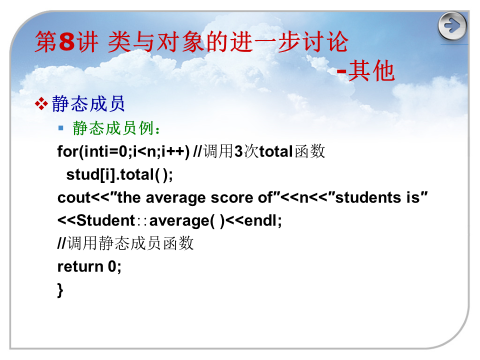
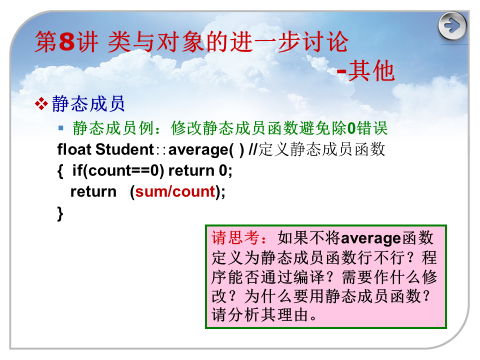
**{   return (\*this);**

**}这样的函数，也会拷贝构造一个然后析构。**

**（2）**普通成员函数可以访问所有成员（包括成员变量和成员函数），静态成员函数只能访问静态成员。

**2）（10分）装配并运行课程ppt上的代码，并思考：❶如果不将average函数定义为静态成员函数行不行？程序能否通过编译？需要作什么修改？ ❷为什么要用静态成员函数？请分析其理由。**

* **ppt附图：**

**** ****  

1. **这里定义静态函数时不用static，**

* **答❶：肯定不能通过编译，去掉static后student::average()这种用法是错的。**

**要改的话，把student::average()改成stu[n-1].average()就可以了，因为非静态成员函数是可以访问静态成员的。所以其实其中[]中的只要是0，1，2中的一个数，都是可以的。**

* **答❷：因为这样可以使average这种对于整个student类型计算的函数更加方便，因为average并不属于任何一个stu，而是属于整个student类。这种时候用static就很合适。**

**2、（总分15分）运行文件夹“4-2 friend”中的程序，体会friend的作用。**

**思考几种解决display需要访问Date私有数据成员的需求：❶将数据的访问控制从private改为public；❷将display设置为Date的友元函数；❸为Date类设计读取私有数据（如在Date类的public内添加 int getYear() const{return Year;}; 这样的成员函数）。体会不同策略的差异以及对数据和应用带来的影响。**

* **实验要求：**

1. **尝试三种方案。（5分）**

**❶就不用friend了，可以直接访问。**

**❷通过friend访问私有数据**

**❸提供接口，这样的话可以确保数据不会被改写，更加安全。而且如果有多个函数要访问private中的内容，这种方法实用性也更好**

**2）并提交（10分）：改写Date类，为其添加读取私有数据的公有接口。并将这些接口应用到display函数中。**

* **改写后的Date类以及改写后的display函数：**

**Date类：#include <iostream>**

**using namespace std;**

**class Date; //对Date类的提前引用声明**

**class Clock //定义clock类**

**{**

**public:**

**Clock(int,int,int);**

**void display(const Date&); //display是成员函数，形参是Date类对象的引用**

**private:**

**int hour;**

**int minute;**

**int second;**

**};**

**class Date //声明Date类**

**{**

**public:**

**Date(int,int,int);**

**int get\_month() const{return month;}**

**int get\_day() const{return day;}**

**int get\_year() const{return year;}**

**//friend void Clock::display(const Date&); //声明clock中的display函数为友元成员函数**

**private:**

**int month;**

**int day;**

**int year;**

**};**

**display函数：**

**void Clock::display(const Date &d) //display的作用是输出年、月、日和时、分、秒**

**{**

**cout<< d.get\_month() << "/" << d.get\_day() << "/" << d.get\_year() << endl; //引用Date类对象中的私有数据**

**cout<<hour<<":"<<minute<<":"<<second<<endl; //引用本类对象中的私有数据**

**}**

**3、（20分）在C++的标准模板库里定义了很多好用的扩展类型，现在我们也来试试吧。我们先来学习做vector类型。根据4-3 myVector 文件夹中的myVector.hpp 的类声明实现该类并通过myVectorTest.cpp的测试。**

* **源代码粘贴处：myVector.cpp 的源代码**

**#include <iostream>**

**#include <algorithm>**

**#include "myVector.hpp"**

**using namespace std;**

**myVector::myVector(unsigned n,int value){**

**size = n;**

**for(int i = 0;i < n;i++){**

**data[i] = value;**

**}**

**}**

**myVector::myVector(const myVector& right){**

**size = right.size;**

**for(int i = 0;i < size;i++){**

**data[i] = right.data[i];**

**}**

**}**

**myVector& myVector::operator=(const myVector& right){**

**size = right.size;**

**for(int i = 0;i < size;i++){**

**data[i] = right.data[i];**

**}**

**return (\*this);**

**}**

**int & myVector::operator[](unsigned index){**

**return this->data[index];**

**}**

**int myVector::get\_size()const{**

**return size;**

**}**

**void myVector::set\_size(unsigned newsize){**

**size = newsize;**

**}**

**void myVector::display()const{**

**for(int i =0;i < size;i++){**

**cout << this->data[i] << ',';**

**}**

**cout << endl;**

**}**

**myVector myVector::operator-(){**

**myVector a;**

**for(int i = 0;i < size;i++){**

**a.data[i] = this->data[size - 1 - i];**

**}**

**return a;**

**}**

**void myVector::sort() {**

**for (int i = 0; i < size - 1; i++) {**

**for (int j = 0; j < size - i - 1; j++) {**

**if (data[j] > data[j + 1]) {**

**// ½»»» data[j] ºÍ data[j+1]**

**int temp = data[j];**

**data[j] = data[j + 1];**

**data[j + 1] = temp;**

**}**

**}**

**}**

**}**

**myVector myVector::operator++(){**

**for(int i = 0;i < size;i++){**

**data[i]++;**

**}**

**return (\*this);**

**}**

**myVector myVector::operator++(int){**

**myVector res = (\*this);**

**for(int i = 0;i < size;i++){**

**data[i]++;**

**}**

**return res;**

**}**

**ostream& operator<<(ostream& out,const myVector& vec){**

**for(int i = 0 ;i < vec.size;i++){**

**out << vec.data[i] << ',';**

**}**

**out << endl;**

**return out;**

**}**

**istream& operator>>(istream& in,myVector& vec){**

**for(int i = 0;i < vec.size;i++){**

**in >> vec.data[i];**

**}**

**return in;**

**}**

**myVector operator+(const myVector& left,const myVector& right){**

**myVector res;**

**res.size = left.get\_size() + right.get\_size();**

**for(int i = 0;i < res.size;i++){**

**if(i < left.get\_size()){**

**res[i] = left.data[i];**

**}else{**

**res[i] = right.data[i - left.get\_size()];**

**}**

**}**

**return res;**

**}**

**myVector operator-(const myVector& left, const myVector& right){**

**myVector res;**

**res.size = left.get\_size();**

**int flag = 1,index = -1;**

**for(int i = 0;i < left.get\_size();i++){**

**flag = 1;**

**for(int j = 0;j < right.get\_size();j++){**

**if(right.data[j] == left.data[i]){**

**flag = 0;**

**res.size--;**

**}**

**}**

**if(flag){**

**res[++index] = left.data[i];**

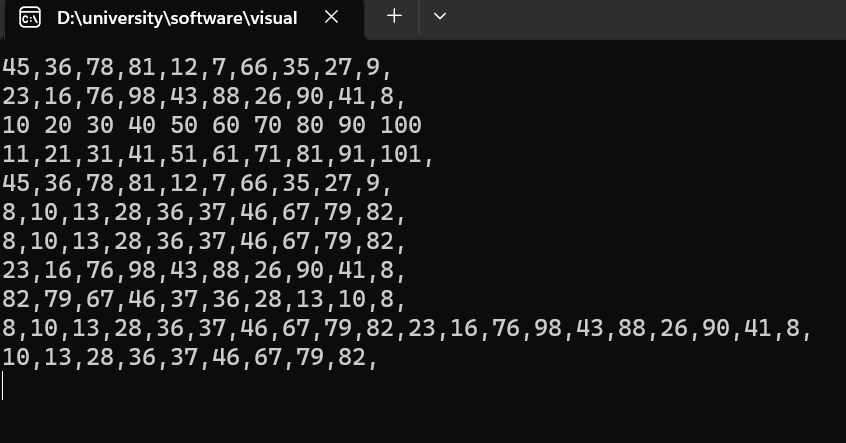
**}**

**}**

**return res;**

**}**

* **程序测试截图：**

****

**4、（30分）在C++的标准模板库里定义了很多好用的扩展类型，现在我们也来试试吧。然后我们来学习做string类型。根据4-4 myString文件夹中的myStringTest.cpp的测试需求将myString.hpp的类声明补充完整，并实现myString类，通过myStringTest.cpp的测试。**

* **源代码粘贴处：myString.hpp的源代码，myString.cpp的源代码**

**myString.hpp的源代码:**

**#include <iostream>**

**using namespace std;**

**class myString**

**{**

**public:**

**//根据测试程序写构造函数原型**

**myString() ;**

**myString(const char\* str,int s,int end);**

**myString(int len,const char r);**

**myString(const char\* r);**

**void display() const; //显示字符串**

**void input();                              //输入字符串**

**int len() const;                           //求字符串长**

**//补充下标重载运算**

**char& operator[](int i);**

**friend int operator==(const myString &a, const myString &b);     //字符串等于比较**

**friend int operator>(const myString &a, const myString &b);      //字符串大于比较**

**friend myString operator+(const myString &a, const myString &b); //字符串拼接**

**//补充赋值重载运算**

**myString& operator=(const myString&right);**

**myString& operator=(const char \* right);**

**//补充析构函数**

**~myString();**

**private:**

**char \*str;**

**int size;**

**};**

**myString.cpp的源代码**

**#include <iostream>**

**#include <string>**

**#include <string.h>**

**#include "myString.hpp"**

**using namespace std;**

**myString::myString() {**

**str = new char ('\0');**

**size = 0;**

**}**

**myString::myString(const myString& right) {**

**str = new char[strlen(right.str) + 1];**

**strcpy(str, right.str);**

**size = right.size;**

**}**

**myString::myString(const char\* rstr,int s = 0,int len = 0) {**

**if (rstr[0] == '\0') {**

**str[0] = '\0';**

**size = 0;**

**return;**

**}**

**str = new char[len + 1];**

**int i = 0;**

**for(i = 0;i < len ;i++){**

**str[i] = rstr[s + i];**

**}**

**str[i] = '\0';**

**size = strlen(str);**

**}**

**myString::myString(const char\* r){**

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

**strcpy(str,r);**

**size = strlen(str);**

**return;**

**}**

**myString::myString(int len,const char r = '\0'){**

**size = len;**

**str = new char[size + 1];**

**int i;**

**for(i = 0;i < len;i++){**

**str[i] = r;**

**}**

**str[i] = '\0';**

**}**

**myString::~myString() {**

**delete[] str;**

**}**

**void myString::display() const {**

**cout << str << endl;**

**}**

**void myString::input() {**

**cout << "请输入字符";**

**getchar();//读取上一个回车键**

**gets(str);**

**}**

**int myString::len() const {**

**return size;**

**}**

**int operator==(const myString &a, const myString &b){**

**if(strcmp(a.str,b.str) == 0){**

**return 1;**

**}**

**return 0;**

**}**

**char& myString::operator[](int i){**

**return this->str[i];**

**}**

**int operator>(const myString& a, const myString& b) {**

**return strcmp(a.str, b.str) > 0;**

**}**

**myString& myString::operator=(const char \* right){**

**if(str != NULL){**

**delete [] str;**

**}**

**this->str = new char [strlen(right) + 1];**

**strcpy(str,right);**

**size = strlen(str);**

**return (\*this);**

**}**

**myString& myString::operator=(const myString& other) {**

**if(str != NULL){**

**delete [] str;**

**}**

**str = new char [other.size + 1];**

**strcpy(str,other.str);**

**size = other.size;**

**return (\*this);**

**}**

**myString operator+(const myString &a, const myString &b){**

**myString res;**

**res.size = a.size + b.size;**

**int i;**

**for( i = 0;i < res.size;i++){**

**if(i < a.size){**

**res.str[i] = a.str[i];**

**}else{**

**res.str[i] = b.str[i-a.size];**

**}**

**}**

**res.str[i] ='\0';**

**return res;**

**}**

**ostream& operator<<(ostream& out,const myString& right){**

**out << right.str ;**

**return out;**

**}**

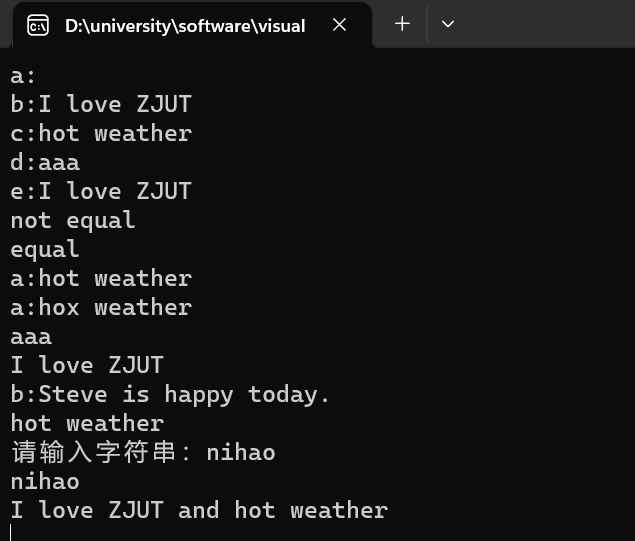
**istream& operator>>(istream& in, myString& right){**

**in >> right.str;**

**return in;**

**}**

**程序测试截图：**

****

**（20分）思考：**

**（5分）-1）为什么myVector不需要重写类的可缺省部分，而myString需要？**

1. **因为myString中的str是指针类型，其中的空间使用new来创造的。所以最后一定要自己用delete[]来删除内存。**

**（5分）-2）在myString的设计中，我们将关系比较（==，>）写在类内作为类的成员，而将+写在类外作为普通函数，请问这样的设计合理吗？说说你的判断结论和理由？如果不合理的话，更合适的设计应该是什么模样？请描述你的设计方案。**

**2)我觉得很不合理，所以我在上面设计函数的时候就已经改成了类外的形式。就如上课所举的那个重载+例子，如果实在类内定义的话，那么a+3是可以的，但是3+a就不行了（a是上课ppt里面的那个complex类型）.但是如果放在类外定义的话，两种都是可以的。所以这样更优。我的设计就在上面的代码部分已经体现，就是用friend将这个类外定义的重载设为友元，这样更加方便，适用范围也比之前更广。**

**（10分）-3）请为myString类设计输入输出流重载，并在主函数中测试它们。**

1. **定义部分：**

**friend ostream& operator<<(ostream& out,const myString& right);**

**friend istream& operator>>(istream& in,myString& right);**

**ostream& operator<<(ostream& out,const myString& right);**

**istream& operator>>(istream& in, myString& right);**

**实现部分：**

**ostream& operator<<(ostream& out,const myString& right){**

**out << right.str ;**

**return out;**

**}**

**istream& operator>>(istream& in, myString& right){**

**in >> right.str;**

**return in;**

**}**

**测试部分：**

**cout << "tesy >> and <<" << "input to m:";**

**myString m;**

**cin >> m;**

**测试截图：**

****

成功！