**实验4：类的运算符重载**

**姓名\_\_\_徐佳辉\_\_\_班级\_\_计科1902\_\_学号\_\_201906080621 \_\_**

* **请阅读此说明：实验4满分110分；其中10分为附加，可选做；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。**

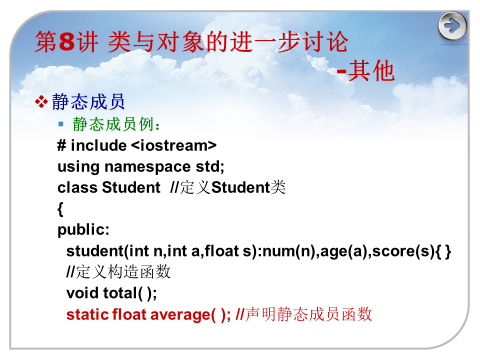
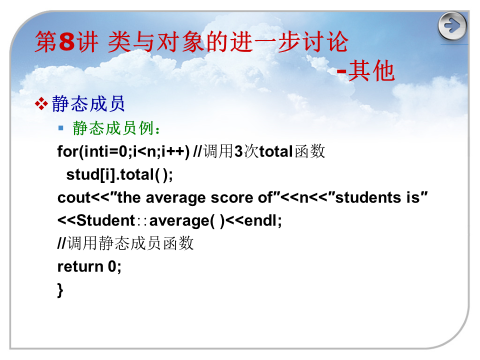
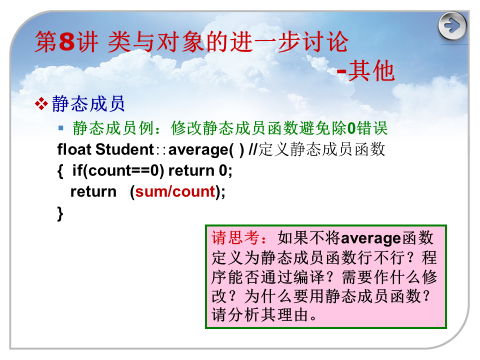
**1、（总分15分）课堂例题巩固。**

* **实验要求：**

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

**2）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？（5分）❷为什么要用静态成员函数？请分析其理由。（5分）**

* **ppt附图：**

**** ****  

**2）**

* **答❶：**
* 类体中的数据成员的声明前加static关键字，该数据成员就成为了该类的静态数据成员。
* 静态数据成员实际上是类域中的全局变量。
* 静态数据成员被类的所有对象所共享，包括该类派生类的对象。即派生类对象与基类对象共享基类的静态数据成员。
* 静态函数不包含有编译器提供的隐藏的this指针，所以不可以调用类的非静态成员，它在类没有实例化的时候就存在
* **答❷：**
* **不行，程序无法通过编译。**
* **修改：将Student::average()的调用改为stud[0].average()**
* **静态成员函数是整个类的函数，而不是对象的函数，在类没有实例化之前，静态成员函数就已经存在。静态成员函数只能调用静态成员变量，刚好sum和count是静态成员变量，用静态成员函数计算average更加方便**

**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。**

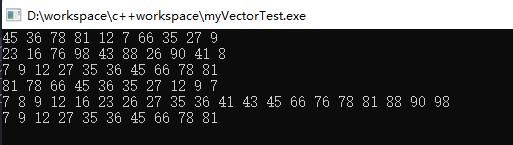
* **实验要求：**

**1）尝试三种方案。（5分）**

**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：**
* **方案一：**
* class Date { //声明Date类
* public:
* Date(int, int, int);
* int month;
* int day;
* int year;
* };
* **方案二：**
* class Date { //声明Date类
* public:
* Date(int, int, int);
* 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;
* };
* **方案三：**
* class Date { //声明Date类
* public:
* Date(int, int, int);
* int getmonth() const {
* return month;
* }
* int getday() const {
* return day;
* }
* int getyear() const {
* return year;
* }
* private:
* int month;
* int day;
* int year;
* };
* void Clock::display(const Date &d) { //display的作用是输出年、月、日和时、分、秒
* cout << d.getmonth() << "/" << d.getday() << "/" << d.getyear() << 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、（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。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 "myVector.hpp"
* #include <iostream>
* using namespace std;
* //用指定值value初始化n个单元 ,n<=CAPACITY
* myVector::myVector(unsigned n, int value) {
* size = n;
* for (int i = 0; i < n; i++) {
* data[i] = value;
* }
* }
* //拷贝构造
* myVector::myVector(const myVector &obj) {
* size = obj.size;
* for (int i = 0; i < obj.size; i++) {
* data[i] = obj.data[i];
* }
* }
* //冒泡排序
* void myVector::sort() {
* for (int i = 0; i < size - 1; i++) {
* for (int j = 0; j < size - 1 - i; ++j) {
* if (data[j] > data[j + 1]) {
* int temp = data[j + 1];
* data[j + 1] = data[j];
* data[j] = temp;
* }
* }
* }
* }
* //赋值重载
* myVector &myVector::operator=(const myVector &right) {
* this->size = right.size;
* for (int i = 0; i < right.size; i++) {
* this->data[i] = right.data[i];
* }
* return \*this;
* }
* //下标运算
* int &myVector::operator[](unsigned index) {
* if (index < 0 || index > size - 1) {
* cout << "下标越界！" << endl;
* exit(1);
* }
* return data[index];
* }
* //返回元素逆置存放的向量，即将原向量元素首尾交换的结果，注意原向量保持不变。
* myVector myVector::operator-() {
* myVector res;
* for (int i = 0; i < size; i++) {
* res.data[i] = data[size - i - 1];
* }
* return res;
* }
* //调整容量
* void myVector::set\_size(unsigned newsize) {
* size = newsize;
* }
* //获取容量
* int myVector::get\_size() const {
* return size;
* }
* //从0开始显示向量元素
* void myVector::display() const {
* for (int i = 0; i < size; i++) {
* cout << data[i] << ' ';
* }
* cout << endl;
* }
* //表示求复杂可以求left和right的并集, 并集元素个数不超过CAPACITY
* myVector operator+(const myVector &left, const myVector &right) {
* myVector res(left);
* int newsize = left.size;
* for (int i = 0; i < right.size; i++) {
* bool pd = 1;
* for (int j = 0; j < left.size; j++) {
* if (right.data[i] == left.data[j]) {
* pd = 0;
* break;
* }
* }
* if (pd == 1) {
* res.data[newsize] = right.data[i];
* newsize++;
* }
* }
* res.set\_size(newsize);
* res.sort();
* return res;
* }
* //表示求left和right的差集
* myVector operator-(const myVector &left, const myVector &right) {
* myVector res;
* int newsize = 0;
* for (int i = 0; i < left.size; i++) {
* bool pd = 1;
* for (int j = 0; j < right.size; j++) {
* if (left.data[i] == right.data[j]) {
* pd = 0;
* break;
* }
* }
* if (pd == 1) {
* res.data[newsize] = left.data[i];
* newsize++;
* }
* }
* res.set\_size(newsize);
* res.sort();
* return res;
* }
* **程序测试截图：**
* 

**4、（40分+附加5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.cpp

#include "myString.hpp"

#include <iostream>

#include <string.h>

using namespace std;

//构造函数

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

if (s == NULL) {

size = 0;

str = new char[size + 1];

strcpy(str, "");

} else {

size = strlen(s);

str = new char[size + 1];

strcpy(str, s);

}

}

myString::myString(const myString &a, int start, int len) {

str = new char[len + 1];

memcpy(str, a.str + start, len);

str[len + 1] = '**\0**';

}

myString::myString(int n, char s) {

size = n;

str = new char[n + 1];

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

str[i] = s;

}

str[n + 1] = '**\0**';

}

//拷贝构造函数

myString::myString(const myString &obj) {

size = obj.size;

str = new char[size + 1];

strcpy(str, obj.str);

}

//析构函数

myString::~myString() {

if (str != NULL) {

delete[] str;

str = NULL;

size = 0;

}

}

//显示字符串

void myString::display() const {

cout << str << endl;

}

//输入字符串

void myString::input() {

char s[100];

gets(s);

size = strlen(s);

strcpy(str, s);

}

//求字符串长

int myString::len() const {

return size;

}

//补充下标重载运算

char &myString::operator[](int index) {

return str[index];

}

//补充赋值重载运算

myString myString::operator=(const myString &obj) {

if (str != NULL) {

delete[] str;

str = NULL;

size = 0;

}

size = obj.size;

str = new char[size + 1];

strcpy(str, obj.str);

return \*this;

}

myString myString::operator=(const char \*s) {

if (str != NULL) {

delete[] str;

str = NULL;

size = 0;

}

size = strlen(s);

str = new char[size + 1];

strcpy(str, s);

return \*this;

}

//相等运算符重载

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

if (a.size != b.size)

return 0;

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

if (a.str[i] != b.str[i])

return 0;

}

return 1;

}

int operator==(const myString &a, const char \*s) {

if (a.size != strlen(s))

return 0;

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

if (a.str[i] != s[i])

return 0;

}

return 1;

}

//>大于运算符重载

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

if (strcmp(a.str, b.str) > 0)

return 1;

return 0;

}

//+运算符重载

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

myString c;

c.size = a.size + b.size;

c.str = new char[c.size + 1];

strcpy(c.str, a.str);

strcat(c.str, b.str);

return c;

}

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

myString c;

c.size = a.size + strlen(b);

c.str = new char[c.size + 1];

strcpy(c.str, a.str);

strcat(c.str, b);

return c;

}

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

myString c;

c.size = strlen(a) + b.size;

c.str = new char[c.size + 1];

strcpy(c.str, a);

strcat(c.str, b.str);

return c;

}

//myString.hpp

#include <iostream>

using namespace std;

class myString {

public:

//根据测试程序写构造函数原型

myString(const char \*s = NULL);

myString(const myString &a, int start, int len);

myString(int n, char s);

myString(const myString &obj);

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 index);

//相等运算符重载

friend int operator==(const myString &a, const myString &b);

friend int operator==(const myString &a, const char \*s);

//>大于运算符重载

friend int operator>(const myString &a, const myString &b);

//+运算符重载

friend myString operator+(const myString &a, const myString &b);

friend myString operator+(const myString &a, const char \*b);

friend myString operator+(const char \*a, const myString &b);

//补充赋值重载运算

myString operator=(const myString &obj);

myString operator=(const char \*s);

//补充析构函数

~myString();

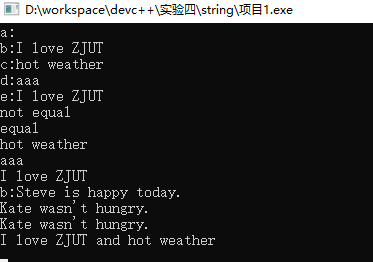
private:

char \*str;

int size;

};

* **程序测试截图：**



**\*拓展思考：（附加10分）5分-1）为什么myVector不需要重写类的可缺省部分，而myString需要？**

**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) 合理，因为关系比较符一般都是两个**myString比较，左值保证是myString类型例如：**

**myString A, B;**

**A == B 🡪 A.operator==(B)**

**而+运算符可能会出现”and” + A的情况，如果是类内重载，左值无法转换成myString类型，所以需要依靠类外重载实现，myString operator+(const char \*a, const myString &b)。**