**实验4：类的运算符重载**

**姓名\_\_李飞飞\_\_\_班级\_\_软工2206\_\_学号\_\_\_202105710309\_\_**

* **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、（10分）运行文件夹“4-1 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） 已尝试**

**2）并提交（5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:**

class Date{

public:

    Date(int, int, int);

    int getYear() const { return year; }

    int getMonth() const { return month; }

    int getDay() const { return day; }

private:

    int month;

    int day;

    int year;

};

#include "Friend.hpp"

Clock::Clock(int h, int m, int s) :hour(h), minute(m), second(s) {}

void Clock::display(const Date& d) {

    //引用Date类对象中的私有数据

    cout << d.getMonth() << "/" << d.getDay() << "/" << d.getYear() << endl;

    //引用本类对象中的私有数据

    cout << hour << ":" << minute << ":" << second << endl;

}

Date::Date(int m, int d, int y) :month(m),day(d),year(y){}

**2、（30分）户籍管理部门为了便于管理辖区内人员，想要设计一个姓名类Name。现在有Name的部分设计框架及使用需求。请你按提示完善它。**

请给出：

1）根据使用给出构造函数的声明 1

Name(const char \*s="X");

2）给出拷贝构造函数的声明 1

Name(const Name& obj);

3) 给出赋值重载函数的声明 1

Name& operator=(const Name& obj);

4) 给出下标重载声明 1

char& operator[](int index);

5) 给出姓名等于比较重载声明 1

bool operator==(const Name& obj) const;

6) 给出求平均姓名长度的成员函数AveLen声明 1

static double AveLen();

7) 给出输入流重载的友元声明 1

friend istream& operator>>(istream& in,Name& obj);

8) 给出其他必要的成员函数声明 2

~Name();

operator char\* ();

friend ostream& operator<<(ostream& out,const Name& obj);

9) 给出输出流重载声明1

ostream& operator<<(ostream& out,const Name& obj);

10) 给出count初始化定义1

int Name::count=0;

11)其他构造函数的实现 3

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

    name = new char[strlen(s)+1];

    size = strlen(s);

    strcpy(name,s);

    count++;

    sum += size;

}

12)拷贝构造的实现2

Name::Name(const Name& obj){

    this->name=new char[obj.size+1];

    strcpy(this->name,obj.name);

    this->size=obj.size;

    count++;

    sum += size;

}

13)赋值重载的实现3

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

    sum -= size;

    delete[] name;

    name=new char[obj.size+1];

    strcpy(name,obj.name);

    size=obj.size;

    sum += size;

    return \*this;

}

14)下标重载的实现2

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

    if(index<0||index>=size){

        cout<<"下标越界\n";

        exit(1);

    }

    return name[index];

}

15)姓名等于比较重载实现1

bool Name::operator==(const Name& obj) const{

    return strcmp(this->name,obj.name)==0;

}

16)AveLen的实现2

double Name::AveLen(){

    return 1.0\*sum/count;

}

17)输入流重载的实现2

istream& operator>>(istream& in,Name& obj){

    char \*temp=new char[128];

    in.getline(temp,128);

    delete[] obj.name;

    Name::sum -= obj.size;

    obj.size=strlen(temp);

    obj.name=new char[obj.size+1];

    strcpy(obj.name,temp);

    Name::sum += obj.size;

    return in;

}

18）输出流重载的实现1

ostream& operator<<(ostream& out,const Name& obj){

    obj.display(out);

    return out;

}

19）其他必要成员函数的实现 1+2

Name::~Name(){

    sum -= size;

    delete[] name;

    count--;

}

Name::operator char\* (){

    return name;

}

**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**

//myVector.cpp

#include"myVector.hpp"

#include <iostream>

using namespace std;

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

    for (int i = 0; i < n; i++)

        data[i] = value;

    size = n;

}

myVector::myVector(const myVector& obj) {

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

        data[i] = obj.data[i];

    size = obj.size;

}

//赋值重载

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

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

        data[i] = right.data[i];

    size = right.get\_size();

    return (\*this);

}

//下标运算

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

    if (index < 0 || index >= size) {

        cout << "下标越界\n";

        exit(1);

    }

    return data[index];

}

//调整容量

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

    size = newsize;

}

//获取容量

int myVector::get\_size()const {

    return size;

}

myVector myVector::operator-() {

    myVector mv(this->size, 0);

    for (int i = 0; i < size; i++)

        mv.data[i] = data[size - i - 1];

    return mv;

}

void myVector::sort(){

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

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

        if (data[j] > data[j + 1]) {

            int temp = data[j];

            data[j] = data[j + 1];

            data[j + 1] = temp;

        }

    }

}

}

void myVector::display() const {

    for (int i = 0; i < size; i++) {

        if (i == 0) cout << data[i];

        else cout << ',' << data[i];

    }

    cout << endl;

}

myVector& myVector::operator++() {

    for (int i = 0; i < size; i++)

        data[i] += 1;

    return (\*this);

}

myVector myVector::operator++(int) {

    myVector mv(\*this);

    for (int i = 0; i < size; i++)

        data[i] += 1;

    return mv;

}

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

    myVector mv;

    int sum = 0;

    int flag;

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

        flag = 1;

        for (int j = 0; j < sum; j++)

            if (left.data[i] == mv.data[j])

                flag = 0;

        if (flag) {

            mv.data[sum] = left.data[i];

            sum++;

        }

    }

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

        flag = 1;

        for (int j = 0; j < sum; j++)

            if (right.data[i] == mv.data[j])

                flag = 0;

        if (flag) {

            mv.data[sum] = right.data[i];

            sum++;

        }

    }

    mv.set\_size(sum);

    return mv;

}

myVector  operator-(const myVector& left, const myVector& right) {//表示求left和right的差集

    myVector mv;

    int sum = 0, flag;

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

        flag = 1;

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

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

                flag = 0;

        if (flag) {

            mv.data[sum] = left.data[i];

            sum++;

        }

    }

    mv.set\_size(sum);

    return mv;

}

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

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

        if (i == 0) out << vec.data[i];

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

    return out;

}

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

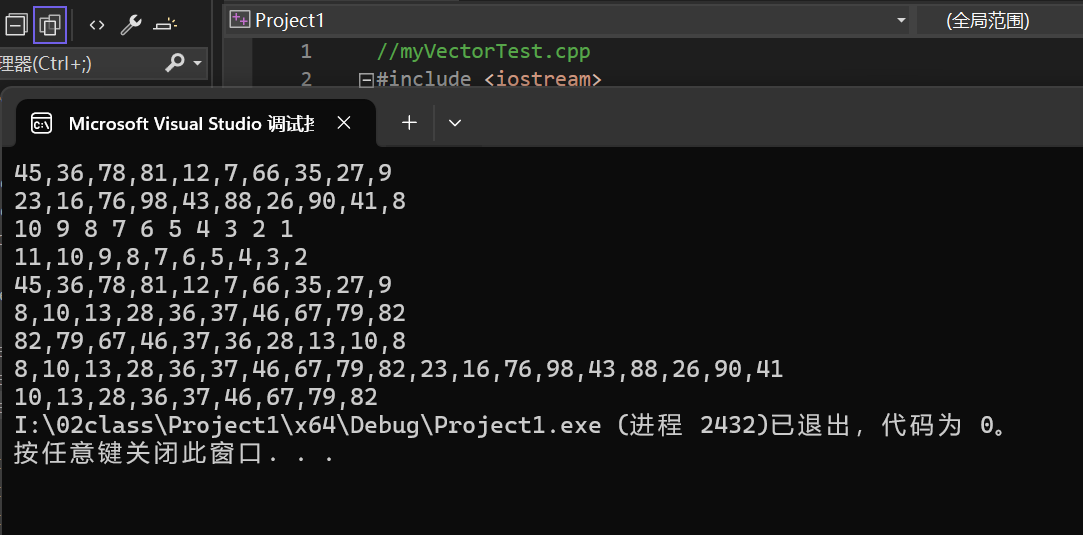
    for (int i = 0; i < 10; i++)

        in >> vec.data[i];

    return in;

}

* **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:**

//myString.hpp

#pragma once

#include <iostream>

#include <cstring>

using namespace std;

class myString {

public:

    //根据测试程序写构造函数原型

    myString();//默认构造函数

    myString(const myString& a); //拷贝构造函数

    myString(const char\* str); //构造函数

    myString(int length, const char c); //构造函数

    myString(const char\* s, int index, int length); //构造函数

    ~myString(); //析构函数

    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);

    //运算符重载

    myString& operator=(const myString& a); //赋值重载

    bool operator!=(const myString& a);//字符串不等于比较

    bool operator==(const myString& a);//字符串等于比较

    bool operator>(const myString& a); //字符串大于比较

    myString operator+(const myString& a); //字符串拼接

private:

    char\* str;

    int size;

};

//考虑为myString添加输入输出流重载

//重载<<运算符

ostream& operator<<(ostream& out, const myString& a);

//重载>>运算符

istream& operator>>(istream& in, myString& a);

* + **myString.cpp**

//myString.cpp

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "myString.hpp"

#include <iostream>

#include <cstring>

using namespace std;

//根据测试程序写构造函数原型

myString::myString() {

    str = new char[1];

    str[0] = '\0';

    size = 1;

}

myString::myString(const myString& a) {

    str = new char[a.size];

    size = a.size;

    strcpy(str, a.str);

}

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

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

    strcpy(this->str, str);

    size = strlen(str)+1;

}

//构造函数

myString::myString(int length, const char c) {

    str = new char[length + 1];

    size = length;

    for (int i = 0; i < size; ++i)

        str[i] = c;

    str[size] = '\0';

}

//构造函数

myString::myString(const char\* s, int index, int length) //构造函数

{

    str = new char[length + 1];

    size = length + 1;

    for (int i = 0; i < length; ++i)

        str[i] = s[index + i];

    str[length] = '\0';

}

myString::~myString() {

    if (str != nullptr) {

        str = nullptr;

        delete[] str;

    }

}

void myString::display() const {

    cout << str << endl;

}

void myString::input() {

    // 使用getline读取一整行

    char c[1000];

    cin.getline(c, 1000);

    size = strlen(c)+1;

    strcpy(str, c);

    str[size - 1] = '\0';

}

int myString::len() const {

    return size;

}

//补充下标重载运算

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

    return str[index];

}

//运算符重载

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

    if (this != &a) {

        if (str != nullptr)

            delete[] str;

        str = nullptr;

        str = new char[a.size];

        strcpy(str, a.str);

        size = a.size;

    }

    return \*this;

}

bool myString::operator!=(const myString& a) //字符串不等于比较

{

    if (strcmp(a.str, str) != 0) return true;

    return false;

}

bool myString::operator==(const myString& a) //字符串等于比较

{

    if (strcmp(a.str, str) == 0) return 1;

    return 0;

}

bool myString::operator>(const myString& a) //字符串大于比较

{

    if (strcmp(str, a.str) > 0) return 1;

    return 0;

}

myString myString::operator+(const myString& a) {

    int len = a.size + size-1;

    char\* c = new char[len];

    size = len;

    strcpy(c, str);

    strcat(c, a.str);

    delete[] str;

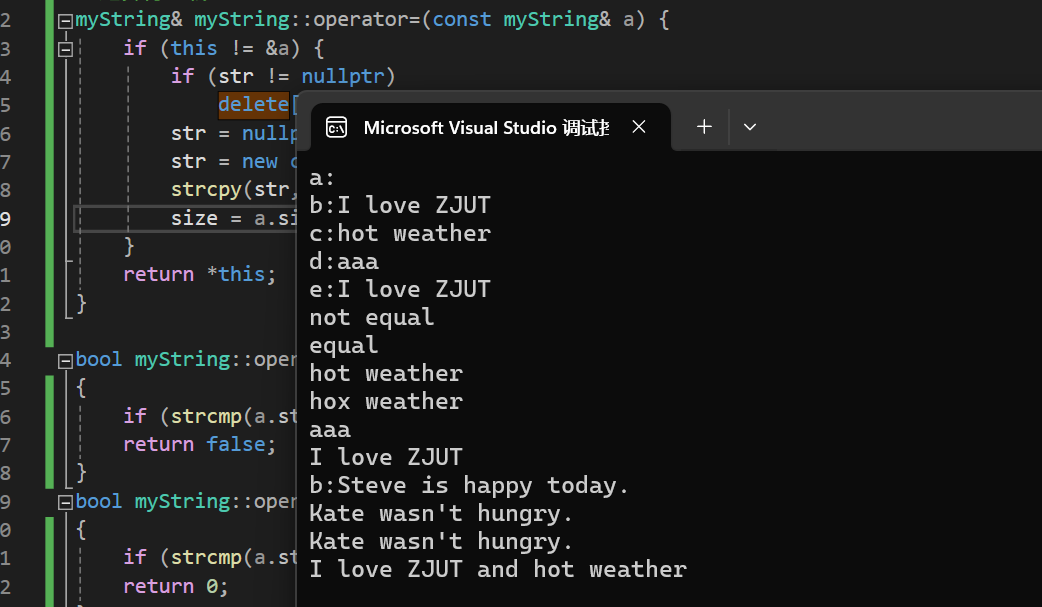
    str = nullptr;

    str = c;

    return (\*this);

}

* **程序测试截图：**

****

**（10分）思考：**

**（2分）-1）为什么myVector不需要重写类的可缺省部分，而myString需要？**

**myString中需要用new动态分配内存，初始化char\*，需要重载默认构造函数，析构函数也需要用delete释放内存，而myVector中没有这个需求。**

**myString要动态管理内存，初始化资源，myVector不需要。涉及动态内存管理，就需要手动定义构造函数、析构函数，以及其他需要保证内存安全的运算符或方法。而如果一个类不需要管理动态分配的内存，编译器生成的默认构造函数和析构函数可能已经足够**

**（2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。**

**合理。**

**关系比较运算符作为成员函数涉及比较当前对象与另一个同类型对象的内部状态。因此，将其作为成员函数是合理的，因为它们通常访问对象的私有数据。加法运算符作为非成员函数，涉及合并两个对象，可能需要支持不同类型的操作数，并且通常需要对称性。因此，**

**-3）（6分）请为myString类设计输入输出流重载，并在主函数中测试它们。**

//重载<<运算符

ostream& operator<<(ostream& out, const myString& a) {

    a.display();

    return out;

}

//重载>>运算符

istream& operator>>(istream& in, myString& a) {

    a.input();

    return in;

}