**实验5：运算符重载与模板类**

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

* **请阅读此说明：实验5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100分）在实验4的基础上工作：（20分）❶考虑将myVector类调整为模板类，注意体会模板类和非模板类在程序设计和装配时的差异；（30分）❷实例化一个myVector<myString>的字符串向量类型 ，并按测试程序的要求调整myString的设计 通过测试程序，注意体会模板类实例化的方法和实例化类使用的特点；（30分）❸尝试用该类型写一个主函数完成一个基于选择排序的字符串降序排列工作，注意比较和直接使用myVector内的sort有什么不同？（20分）❹若字符串集合的排序要求为按照字符串的第2个字符至第6个字符构成的子串来完成降序排序（假设每个原始字符串长度都超过10），则考虑你的程序该做怎么样的调整？**

**实验解答：**

**❶黏贴模板类myVector的实现代码和调整后的myString的定义代码：**

**myVector.cpp:**

#include "../thpp/myVector.hpp"

#include "../thpp/myString.hpp"

#include <iostream>

#include <string.h>

using namespace std;

//构造函数

template <class T>

myVector<T>::myVector(int size){

    this->size = size;

    data = new T[size];

}

//无参构造

template <class T>

myVector<T>::myVector(){

    size = 0;

    data = NULL;

}

//拷贝构造函数

template <class T>

myVector<T>::myVector(const myVector<T> &right){

    size = right.size;

    data = new T[size];

    for(int i = 0;i < size;i++){

        data[i] = right.data[i];

    }

}

//赋值重载

template <class T>

myVector<T>& myVector<T>::operator=(const myVector<T> &right){

    if(this == &right){

        return \*this;

    }

    delete[] data;

    size = right.size;

    data = new T[size];

    for(int i = 0;i < size;i++){

        data[i] = right.data[i];

    }

    return \*this;

}

//下标运算

template <class T>

T& myVector<T>::operator[](int index){

    return data[index];

}

//减号的赋值重载

template <class T>

myVector<T> myVector<T>::operator-(){

    myVector<T> temp(size);

    for(int i = 0;i < size;i++){

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

    }

    return temp;

}

template <class T>

void myVector<T>::display() const {

    for(int i = 0;i < size;i++){

        cout << data[i] << " ";

    }

    cout << endl;

}

//调整容量

template <class T>

void myVector<T>::set\_size(unsigned newsize){

    //如果新的容量大，那就扩容，如果新的容量小，那就缩容，把最后几个删去，其他保留

    if(newsize > size){

        T\* temp = new T[newsize];

        for(int i = 0;i < size;i++){

            temp[i] = data[i];

        }

        delete[] data;

        data = new T[newsize];

        data = temp;

        size = newsize;

    }

    else{

        size = newsize;

        T\* temp = new T[newsize];

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

            temp[i] = data[i];

        }

        delete[] data;

        data = new T[newsize];

        data = temp;

    }

}

//获取容量

template <class T>

int myVector<T>::get\_size() const{

    return size;

}

//输入流重载

template <class T>

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

    for(int i = 0;i < vec.size;i++){

        in >> vec.data[i];

    }

    return in;

}

//输出流重载

template <class T>

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

    for(int i = 0;i < vec.size;i++){

        out << vec.data[i] << ",";

    }

    return out;

}

//减法重载，即求两个集合的差集

template <class T>

myVector<T> operator-(const myVector<T> &left, const myVector<T> &right){

    myVector<T> temp(left.size);

    int count = 0;

    for(int i = 0;i < left.size;i++){

        bool flag = true;

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

            //cout << left.data[i] << " " << right.data[j] << endl;

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

                flag = false;

                break;

            }

        }

        if(flag){

            temp.data[count++] = left.data[i];

        }

    }

    temp.set\_size(count);

    return temp;

}

template myVector<myString> operator-(const myVector<myString> &left, const myVector<myString> &right);

//后置自增重载

template <class T>

myVector<T>& myVector<T>::operator++(int){

    for(int i = 0;i < size;i++){

        data[i]++;

    }

    return \*this;

}

//析构函数

template <class T>

myVector<T>::~myVector(){

    delete[] data;

}

**myString.hpp:**

#ifndef MYSTRING\_HPP

#define MYSTRING\_HPP

#include <iostream>

using namespace std;

class myString

{

public:

   //根据测试程序写构造函数原型

   //无参构造函数

   myString() ;

   //拷贝构造函数

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

   //有参构造函数

   myString(int len,const char r);

   //有参构造函数

   myString(const char\* r);

   //拷贝构造函数

   myString(const myString& right);

   //显示字符串

   void display() const;

   //输入字符串

   void input();

   //求字符串长度

   int len() const;

   //补充下标重载运算

   char& operator[](int i);

   //字符串等于比较

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

   //字符串大于比较

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

   //字符串拼接

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

   //输出流重载运算

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

   //输入流重载运算

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

   //补充赋值重载运算

   myString& operator=(const myString&right);

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

   //后置自增重载运算

   myString operator++(int);

   //类型转换

   operator const char\*();

   //获取字符串长度

   int get\_size() const;

   //补充析构函数

   ~myString();

private:

   char \*str;

   int size;

};

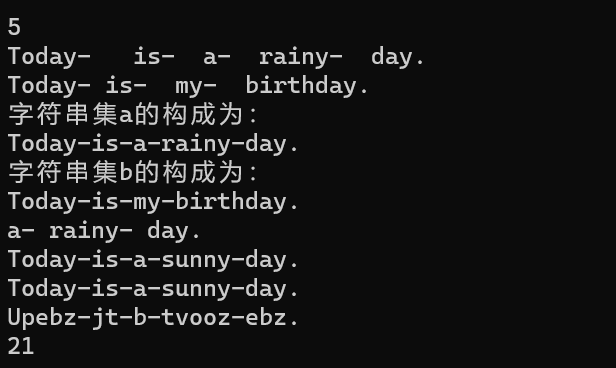
//考虑为myString添加输入输出流重载

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

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

#endif

**❷黏贴程序测试运行结果窗口（运行结果截屏）：**

****

**❸黏贴排序实现代码：**

void sort(myVector<myString> & a){

    int i = 0 , j = 0, max = 0;

    myString temp;

    for(int i = 0; i < a.get\_size() - 1;i++){

        max = i;

        for(int j = i + 1;j < a.get\_size();j++){

            if(a[j] > a[max]){

                max = j;

            }

        }

        if(max != i){

            temp = a[i];

            a[i] = a[max];

            a[max] = temp;

        }

    }

}

**❹黏贴基于子串比较规则的排序实现代码 以及你测试的数据和测试的结果（运行结果截屏）；以及说明原来设计的类是否做了调整？若有调整，调整了什么？**

myString substr(myString& b){

    char \* a;

    a = new char[6];

    for(int i = 1;i < 6;i++){

        a[i - 1] = b[i];

    }

    a[5] = '\0';

    myString temp(a);

    return temp;

}

void sort(myVector<myString> & a){

    int i = 0 , j = 0, max = 0;

    myString temp;

    for(int i = 0; i < a.get\_size() - 1;i++){

        max = i;

        for(int j = i + 1;j < a.get\_size();j++){

            if(substr(a[j]) > substr(a[max])){

                max = j;

            }

        }

        if(max != i){

            temp = a[i];

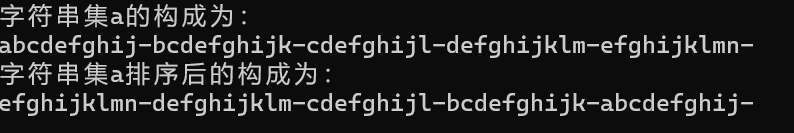
            a[i] = a[max];

            a[max] = temp;

        }

    }

}

****

对比第三小问，这里也没有调整。就是增加了一个函数：substr来得到myString类型中的2-6个字符构成的子串，再用第三小问的代码跑一样也是可以成功的。