**实验5：运算符重载与模板类**

**姓名：陈王子 学号：202103150503**

**班级：数据科学与大数据技术2101班**

* **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）**

#include <iostream>

using namespace std;

template<class T>

class myVector {

public:

    //构造系列

    myVector(int n = 10, T value = 0); //用指定值value初始化n个单元  ,n<=CAPACITY

    myVector(const myVector<T>& obj);//拷贝构造

    //~myVector();

    myVector<T>& operator=(const myVector<T>& right);//赋值重载

    T& operator[](int index);  //下标运算

    void set\_size(int newsize); //调整容量到newsize ,newsize<=CAPACITY

    int get\_size()const;//获取容量

    //也可以用下面两个函数代替

    //void expand(unsigned delta);//调整容量到size+delta

    //void subtract(unsigned delta);//调整容量到size-delta

    myVector<T> operator-(); //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。

    void sort();//升序排序

    void display() const;//从0开始显示向量元素,以逗号分隔每个单元值

    myVector<T> operator++(); //前置增量重载，将vector内每个单元值增加1，并返回修改后的vector

    myVector<T> operator++(int);//后置增量重载，将vector内每个单元值增加1，并返回修改前的vector

    friend myVector<T> operator+ (const myVector<T>& left, const  myVector<T>& right);

    //friend myVector<T> operator- (const myVector<T>& left,const  myVector<T>& right);

    myVector<T> operator-(const myVector<T>& right);

    //输入输出流重载

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

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

private:

    T data[1000];

    int size; //有效单元个数

};

template<class T>

myVector<T>::myVector(int n, T value)

{

    size = n;

    for (int i = 0; i < n; i++)

        data[i] = value;

}

template<class T>

myVector<T>::myVector(const myVector<T>& obj)//拷贝构造

{

    size = obj.size;

    for (int i = 0; i < size; i++)

        data[i] = obj.data[i];

}

template<class T>

myVector<T>& myVector<T>::operator=(const myVector<T>& right)//赋值重载

{

    size = right.size;

    for (int i = 0; i < size; i++)

        data[i] = right.data[i];

    return \*this;

}

template<class T>

T& myVector<T>::operator[](int index) //下标运算

{

    if (index < 0 || index >= size)

    {

        cout << "Out of index";

    }

    return data[index];

}

template<class T>

void myVector<T>::set\_size(int newsize) //调整容量到newsize ,newsize<=CAPACITY

{

    size = newsize;

    if (newsize < size)

    {

        for (int i = newsize; i < size; i++)

            data[i] = '\0';

    }

    else

    {

        for (int i = newsize; i < size; i++)

            data[i] = '\0';

    }

}

template<class T>

int myVector<T>::get\_size()const//获取容量

{

    return size;

}

template<class T>

myVector<T> myVector<T>::operator-() //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 v;

    v.size = size;

    for (int i = 0; i < size; i++)

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

    return v;

}

template<class T>

void myVector<T>::sort()//升序排序

{

    int t;

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

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

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

            {

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

            }

}

template<class T>

void myVector<T>::display() const//从0开始显示向量元素  ,以逗号分隔每个单元值

{

    for (int i = 0; i < size; i++)

        cout << data[i] << ",";

    cout << endl;

}

template<class T>

myVector<T> myVector<T>::operator++()//前置增量重载，将vector内每个单元值增加1，并返回修改后的vector

{

    for (int i = 0; i < size; i++)

        data[i] = data[i] + 1;

    return \*this;

}

template<class T>

myVector<T> myVector<T>::operator++(int)//后置增量重载，将vector内每个单元值增加1，并返回修改前的vector

{

    myVector v = \*this;

    v.size = size;

    for (int i = 0; i < size; i++)

        v.data[i] = data[i];

    for (int i = 0; i < size; i++)

        data[i] = data[i] + 1;

    return v;

}

template<class T>

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

{

    out << "size:" << vec.size << endl;

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

    {

        out << vec.data[i] << ",";

    }

    out << endl;

    return out;

}

template<class T>

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

{

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

        in >> vec.data[i];

    return in;

}

template<class T>

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

//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<T> v;

    v.size = left.size;

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

        v.data[i] = left.data[i];

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

    {

        int f = 0;

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

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

            {

                f = 1;

                break;

            }

        if (f == 0)

        {

            v.size++;

            v.data[v.size - 1] = right.data[i];

        }

    }

    return v;

}

template<class T>

myVector<T> myVector<T>::operator-(const myVector<T>& right)

//表示求left和right的差集

{

    int q = 0;

    myVector<T> v;

    for (int i = 0; i < size; i++)

    {

        int f = 0;

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

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

            {

                f = 1;

                break;

            }

        if (f == 0)

        {

            v.data[q] = data[i];

            q++;

        }

    }

    v.size = q;

    return v;

}

**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）**

class myString{

public:

    //根据测试程序写构造函数原型

    myString(char\* s="",int start=0,int len=0);

    myString(int len,char s=' ');

    //myString();

    void display() const; //显示字符串

    void input(); //输入字符串

    int len() const ;//求字符串长

    char& operator[](int index);//补充下标重载运算

    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

    myString operator=(const myString& a);//补充赋值重载运算

    ~myString();//补充析构函数

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

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

    operator const char\*();//定义myString的类型转换操作

    myString operator++(int);//friend myString operator++(const myString a); //定义myString的增量操作

    //friend myString operator++(const myString &a);

private:

   char \*str;

   int size;

};

myString::myString(char\* s,int start,int len)

{

    int k=0;

    if(start==0&&len==0)

    {

        size=strlen(s);

        str=new char(size);

        for(int i=0;i<size;i++)

            str[i]=s[i];

        str[size]='\0';

    }

    else

    {

        size=len;

        str=new char(size);

        for(int i=start;i<start+len;i++)

            str[k++]=s[i];

        str[k]='\0';

    }

}

myString::myString(int len,char s)

{

    size=len;

    str=new char(size);

    for(int i=0;i<len;i++)

        str[i]=s;

    str[size]='\0';

}

//考虑为myString添加输入输出流重载

void myString::display() const //显示字符串

{

    for(int i=0;i<size;i++)

        cout<<str[i];

    cout<<endl;

}

void myString::input()//输入字符串

{

    cin>>str;

    size=strlen(str);

}

int myString::len() const //求字符串长

{

    return size;

}

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

{

    if(strcmp(a.str,b.str)==0)

        return 1;

    return 0;

}

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

{

    if(strcmp(a.str,b.str)>0)

        return 1;

    return 0;

}

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

{

    myString c;

    int f=0;

    for(int i=0;i<a.size;i++)

        c.str[f++]=a.str[i];

    for(int i=0;i<b.size;i++)

        c.str[f++]=b.str[i];

    c.size=a.size+b.size;

    c.str[c.size]='\0';

    return c;

}

myString myString::operator=(const myString& a)//补充赋值重载运算

{

    size=a.size;

    str=new char(a.size);

    for(int i=0;i<size;i++)

        str[i]=a.str[i];

    str[size]='\0';

    return \*this;

}

myString::~myString()//补充析构函数

{

    //delete[] str;   //delete[] str;

    size=0;

}

char& myString::operator[](int index)//补充下标重载运算

{

    return str[index];

}

ostream&operator<<(ostream&out, const myString&s)

{

    out<<s.str;

    return out;

}

istream& operator>>(istream&in,myString&s)

{

    s.str=new char(100);

    cin.getline(s.str,100);

    s.size=strlen(s.str);

    return in;

}

myString::operator const char\*()

{

    return str;

}

myString myString::operator++(int)

{

    for(int i=0;i<size;i++)

    {

        if((str[i]>='a'&&str[i]<'z')||(str[i]>='A'&&str[i]<'Z'))

            str[i]=char(str[i]+1);

        else if(str[i]=='z')

            str[i]='a';

        else if(str[i]=='Z')

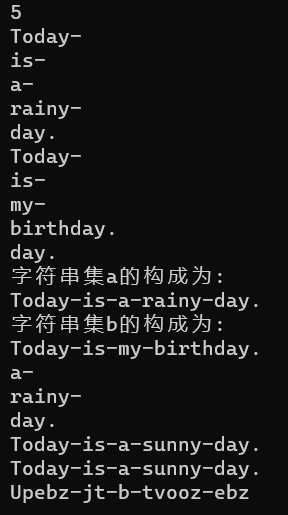
            str[i]='A';

    }

    return \*this;

}

**❷黏贴程序测试运行结果窗口（运行结果截屏）：**



**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）**

int main()

{

    string a[100];

    int n;

    cin>>n;

    cout<<"before sort:"<<endl;

    for(int i=0;i<n;i++)

        cin>>a[i];

    cout<<"after sort:"<<endl;

    for(int i=0;i<n-1;i++)

    {

        int m=i;

        for(int j=i+1;j<n;j++)

            if(a[j]>a[m])

                m=j;

        swap(a[m],a[i]);

    }

    for(int i=0;i<n;i++)

        cout<<a[i]<<endl;

    getchar();

    getchar();

}

**排序方式不同。主函数中的sort使用的是选择排序，而myVector中的sort使用的是冒泡排序。**

**主函数中的sort实现的是降序排列，而myVector中sort默认实现的是升序排列。**

**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？**

template<class T>

void myVector<T>::sort()

{

    int t;

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

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

            if (data[j].substr(1, 5) < data[j + 1].substr(1, 5)) // 比较子串并改为降序

            {

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

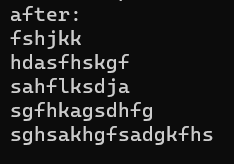
            }

}

**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？**

**调整后的按照字符串的第2个字符至第6个字符构成的子串来完成降序排序。**

**原来：**



**现在：**

