

西安电子科技大学计算机科学与技术学院

1. **实验目的**

完成第11章练习1、15，并实现string类。

1. **实验环境**

IDE：clion cmake mingw

os：windows 10

硬件：Dell g15 5511笔记本

1. **实验内容**
2. 给出了结构体与其运算符的重载，说明main函数内给出的逻辑表达式所含有的类型转换。修改代码以输出每个逻辑表达式的值。
3. 定义Vec4类，作为四个float数字组成的向量。重载其各运算符。
4. 实现String类，并自定义实现额外功能。
5. **实验步骤**

1、

在原本的类中再重写<<，其他的保持原样；略修改main函数式子，添加了几个强制类型转换以让表达式合法。

main函数与每个式子中存在的转换如下所示（加了int强制转换的式子则标出了歧义原因）：

cout << i + 10 << endl; // int operator+(int)

cout << int(y) + 10 << endl; // (int)y+10 ? y+X(10) ?

cout << int(y) + 10 \* int(y) << endl; // y+(10\*(int)y) ? y+(X(10)\*y) ?

cout << x + y + i << endl; // Y int() , X operator+() , X operator+(int)

cout << x \* x + i << endl; // Y(X) , X operator\*(X,Y) , X operator+(int)

cout << f(7) << endl; // X(int) , f(X)

cout << f((int) y) << endl; //No X(Y).

cout << y + y << endl; // Y int() , Y int() , int operator+(int).Because No X(Y).

cout << 106 + y << endl; // Y int() , int operator+(int)

2、

定义了三种构造函数：

Vec4();//set default 0

Vec4(const float& i0, const float& i2, const float& i3, const float& i4);

Vec4(const Vec4& vv);

分别为默认0向量（未进行赋值直接输出则为[0,0,0,0]）、使用四个float数初始化向量、使用其他向量对此向量赋值（相当于重载对应的赋值‘=’符号）。

重载了Vec4$Vec4、Vec4$float、float$Vec4三种运算符。此处仅以Vec4$float的加法运算为例。

先重载+=：只需要对四个值进行+=（即使用调用float的+=）即可，返回Vec4参数的引用。然后再重载+：生成临时对象Vec4 ans；再执行ans+=float即可。

这种方式比先重载+再调用+重载+=更好的地方是：使得+=操作无需生成临时对象。也在增加代码复用率的同时保留了独立重载+和+=这一方式的效率。

3、

(1)

大部分对教材上代码进行复制粘贴。同时注意避免using namespace std；。

为了增强空间使用效率，这个string在进行赋值操作的时候，两个string是会共享同一块数据存储区rep的，并且通过n来记录有多少个共享这个存储区的string对象。

每当要删去当前对象时，使得n--。由于n是在对应一系列string共享的，所以这一系列对象都会知道少了一个“伙伴”。当n为0时，说明没有该系列对象了，则删除该存储区。

新建string对象时，n置1。析构时，先n--，再判断n是否为0；若为0，则删除对应存储区。

一旦对一个对象的内容进行改变，则相当于新建一个独立的对象，把内容复制过去，n设为1，然后删除原来的对象（实际上，只是在对这个对象的存储区rep进行删除、复制等操作）。

这样的逻辑可以完美处理string相关的空间。比如在str=String（“abc”）时，赋值号右边会产生一个临时对象，注册其n为1；赋值时，n变为2；赋值结束时，临时对象析构，n转为1。从本质上讲，这种方法嵌入在构造函数和析构函数中，严格把握了对象，只要是应当改变n的就必然会改变，不会有任何疏漏。需要完全复制rep的情况也非常少。

(2)

字符串有一个重要的操作就是对其中的单个元素进行读写。那么，直接返回对应char是绝不可行的。

Cref类则解决了这个问题。Cref记录访问的下标的同时包含了整个字符串信息。使用[i]访问string的一个位时其实会得到整个字符串的引用调用。通过重载operator char()来让它该被当做单一char时就能被当做char。而进行=赋值时，会改变原字符串的数据。

Cref里的friend class String;实在是点睛之笔，使得内部类提供类的对象的引用调用入口成为可能。

(3)

普通运算符的话，不再赘述。另外，同第二题的处理，这里先重载+=，然后利用+=重载+。

(4)

<<的重载比较简单，直接cout<<rep->s；相当于利用char\*的<<。

>>的重载，实现一位一位char地读。首先为了保证输入方式符合cpp自带string的性质，循环条件为：

while (input && input.get(ch) && !isspace(ch))

即输入流正常、输入未结束、未遇到空格时才接着读入下一位。顺便，当读到空格时，能够不把空格读入字符串的同时吞掉空格。

由于输入的每个char是保留在数组里的，若避免使用STL的话，我们无法直接得到无限变量数量上限的数组，因此在输入大量数据时必然会越界，或者在输入小数据时产生空间浪费。于是需要手动产生动态数组。

定义const int ATOM\_INPUT\_SIZE = 16;作为读入字符串尺寸的基准值。刚开始时，maxsize=ATOM\_INPUT\_SIZE，临时数组只有17位。但是，若发现没有空间再留个‘0’作为结尾了，则要进行数组扩展：max\_size += ATOM\_INPUT\_SIZE;新建max\_size+1大小的数组，复制原数组，然后删除原数组。这使得临时数组变为33位。因此，每次越界都会给数组再加16位，在理论上可以无限添加。

通过改变扩展表达式max\_size += ATOM\_INPUT\_SIZE;可以实现不同的扩展方法，比如改成max\_size\*=2;变成指数型扩展。

(5)

实现了几个额外功能。

size()返回rep->sz得到尺寸。

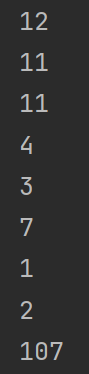
c\_str()返回rep->s得到c风格字符串。

重载operator()(int start, int n)作为substring子串截取。新建n+1大小的char数组，使用strncpy进行复制即可。

(6)

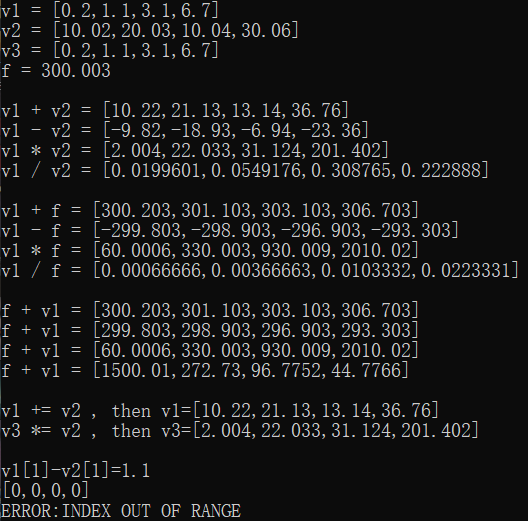
若访问越界则throw出OutOfRangeException()。这一判断被整合进了check()函数内。

1. **实验结果**

1、

计算无误。

2、



所有计算通过验算并未发现错误。正确运行所有重载的运算符。

倒数第二行通过

Vec4 vt;//default

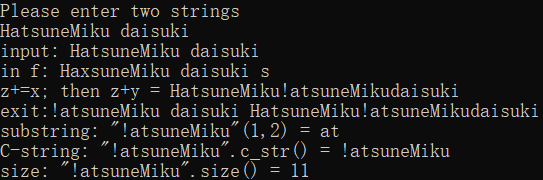
cout<<vt<<endl;

输出，发现成功默认初始化为[0,0,0,0]。

越界时，正常报错。

3、

注：String z=x；



基本测试无误，IO正常、已重载的运算符正常，额外实现的三个功能也正常。

1. **实验总结**

首先体会到的就是完备的重载运算符的困难（甚至可以说在某些情况下的不可能的），必须要在某些地方进行额外的强制类型转换。不过，有没有一种逻辑，可以使得11.1这样的两个类可以进行放心地使用运算符构成任意表达式，并且在表达式确立的时候，使用者就能知道自己要不要加强制类型转换？毕竟从11.1来看，只有简单的表达式能看清楚到底怎么强制类型转换，并且往往是在报错之后才去思考，这在大型开发中显然是不应当的。

第二题是非常标准化的重载运算符，我觉得所有人都应当熟悉这个流程。

string的实现真的十分巧妙，特别是rep和Cref这两个东西让我大开眼界。前者非常良好的节约了空间，并且严格执行，不留一点漏洞；后者给访问对应下标提供了方便又功能齐全的入口，涵盖读写操作，又能直接当char使用。也许在其他许多地方可以借鉴这样的设计理念。另外，对未知长度输入流的输入函数设计也使我对输入的方式的理解更加深刻了。