字符串类的实现

2023 年 10 月 17 日

1 字符串项目概述

1.1 项目内容及要求

本项目主要实现了对字符串类的包装，设计成员函数对字符串进行处理，实现一些基础功能，比如增删改查，以及取子串，对运算符进行重载。

1.2 研究人员及分工

序号	学号	姓名	角色及具体贡献
1	杨弈开	22122528	组员（功能函数设计与编写，程序可靠性测试）
2	文婷	22121435	组长（类的设计，构造析构函数，测试函数，报告撰写，讲解视频的录制）
3	党昊天	22120051	组员（异常处理，代码检查与 bug 修复）
4	李宗恒	22120680	组员（部分异常处理）

2 CPPString 类的设计

2.1 数据成员设计

本类中设计了两个成员，分别保存字符串及其长度，将长度作为成员方便了使用长度，但是在构造时加上对长度的赋值语句，同时，在要改变一个对象时，它的长度也要根据实际情况加以修改。

```
char *s;
int l;
```

2.2 成员函数（友元函数）原型设计

2.2.1 构造、析构函数

我们所设计的字符串对象，不包含\0 终止符，这在构造函数中有所体现。

```
CPPString();                                    //默认构造
CPPString(const char *x);                       //构造
{
    int k=0;
    while(x[k]!='\0')k++;
    s=new char[k];
    for(int i=0;i<k;i++)
        s[i]=x[i];
```

```
    l=k;
}
CPPString(const CPPString &x);                    //拷贝构造
{
    s=new char[x.l];
    for(int i=0;i<x.l;i++)s[i]=x.s[i];
    l=x.l;
}
~CPPString()                                      //析构
{
    if(s!=NULL)
        delete []s;
    l = 0;
}
```

2.2.2 基础功能函数：增删改查

增加(ins)：x 为插入的字符串序列，k 为插入位置，无返回值。

对于增加这一功能，我们一共设计了两个成员函数，两个函数的区别只有传入形参的类型不同，一个为 char 型指针，指向一个字符串，另一个为我们自定义的 CPPString 类型对象，这两个函数不仅能实现对已知字符串指定位置插入一个字符串，同时还能实现插入单个字符，如果插入位置超出字符串的长度，则会进行异常处理。

同时 ins()函数作为本类中一个重要的成员函数，rep()、cut()、operator+()等多个函数中都应用了这一函数。

```
void CPPString::ins(const CPPString &x,int k)
{
    if (k < 0 || k >l)
        throw(int)k;
    char *t=new char[l+x.l];
    for(int i=0;i<k;i++)
        t[i]=s[i];
    for(int i=k;i<=k+x.l;i++)
        t[i]=x.s[i-k];
    for(int i=k+x.l;i<=l+x.l;i++)
        t[i]=s[i-x.l];
    delete []s;
    s=t;
    l+=x.l;
}
void CPPString::ins(const char *x,int k)
{
    if (k < 0 || k > l)
        throw(int)k;
    ins(CPPString(x),k);
}
```

删除(del)：k 为开始删除的位置，len 为删除字符串序列的长度，无返回值。

```
void CPPString::del(int k,int len){
```

```
    if (k > l||k<0)
        throw (int)1;
    if (l < len + k)
        len = l - k ;
    char *t = new char[l - len];
    for(int i=0;i<k;i++)
        t[i]=s[i];
    for(int i=k+len;i<l;i++)
        t[i-len]=s[i];
    delete []s;
    s = t;
    l = l - len;
}
```

查找(loc)：x 为想要查找的字符串序列，k：字符串中第 k 次出现查找序列，返回值为第 k 个所查找的字符串序列首字母所对应的下标。

```
int CPPString::loc(const CPPString &x,int k)
{
    int r = -1;
    for (int i = 0; i < l && k>0; i++)
    {
        if (s[i] == x.s[0])
            k--;
        if (k == 0)
            r = i;
    }
    cout << r << endl;
    if (r == -1)
        return -1;
    else
        return r;
}
int CPPString::loc(const char* x, int k);
```

修改(rep)：x 为代替部分原字符串的字符串序列，k 为开始替换的位置，len 为修改的长度。

```
void CPPString::rep(const CPPString &x,int k,int len)
{
    if (k < 0 || k > l)
        throw(int)1;
    if (len + k >= l)
        len = l - k ;
    del(k,len);
    ins(x,k);
}
void CPPString::rep(const char *x,int k,int len);
```

取子串(cut)：h 为截取字符串首字母在原字符串中的位置（下标），len 为截取长度。

```
CPPString CPPString::cut(int h,int len)
{
    CPPString temp;
    if (h > l || h <0)
        throw (int) 1;
    if (l <h + len)
    {
        len = l - h ;
        cout << "输出长度越界，自动取到最后一位" << endl;
    }
    temp.s = new char[len];
    for (int i = 0; i <len; i++)
        temp.s[i] = s[i+h];
    temp.l = len;
    return temp;
}
```

2.2.3 运算符重载

对运算符的重载中，我们做了对[]、+、+=、*、*=、=的重载，对+号的重载函数实现了对字符串的拼接，类似于 Cstring 库中的 strcat()函数，不同的是，我们所设计的字符串对象不包含‘\0’终止符，因此在函数设计过程中不需要考虑对‘\0’的处理；对*号的重载实现了对字符串的复制操作，对*号的重载中设计了两个函数，一个是成员函数，另一个是作为友元函数的全局函数，两者略有差别，前者的运算中，字符串对象在前而整形数据在后（str*int）；而后者则是整形数据在前，字符串对象在后（int*str）。

[]重载：

当输入下标大于等于本身长度时，会抛出异常，正常情况下返回下标对应的字符。

```
char &CPPString::operator[](const int &x)
{
    if (x > l)
        throw (int)1;
    return s[x];
}
CPPString &operator=(const CPPString &x)
{
    if(s!=NULL)    delete []s;
    s=new char[x.l];
    for(int i=0;i<x.l;i++)s[i]=x.s[i];
    l=x.l;
    return *this;
}
}
```

+和+=运算符重载：

在+和+=的运算符重载的函数中，都用到了 ins()函数，实现字符串之间的拼接。

```
CPPString CPPString::operator+(const CPPString &x)   // +号运算符重载作为成员函数
{
    CPPString t=*this;
    t.ins(x,l);
    return t;
}
CPPString &operator+=(const CPPString &x)
{
    ins(x,l);
    return *this;
}
```

和=运算符重载：

在*和*=运算符重载的函数中，同样使用 ins()函数，实现了对字符串先复制后拼接的功能，最终达到乘法的结果。

```
CPPString CPPString::operator*(int y)
{
    CPPString t = *this;
    for(int i=1;i<y;i++)t.ins(*this,t.l);
    return t;
}
CPPString operator*(int y,const CPPString &x)
{
    CPPString t=x;
    for(int i=1;i<y;i++)
        t.ins(x,t.l);
    return t;
}
CPPString &CPPString::operator*=(int y)
{
    CPPString t=*this;
    for(int i=1;i<y;i++)
        *this+=t;
    return *this;
}
```

比较运算符重载（==、!=、>=,>,<=,<）：

对以上六个比较运算符的重载函数实现的内部逻辑相同，都是通过遍历字符串，比较每一位的字符是否相同，最终得到结果（对于字母型字符串，认为大小写字母不相同）。

```
bool operator==(const CPPString &x)const;            //关系运算符重载
{
    if(l!=x.l)
    return false;
    for (int i = 0; i < l; i++)
    {
     if (s[i] != x.s[i])
         return false;
```

```
    }
    return true;
    }
bool operator!=(const CPPString &x)const;
bool operator>(const CPPString &x)const;
bool operator<(const CPPString &x)const;
bool operator>=(const CPPString &x)const;
bool operator<=(const CPPString &x)const;
```

输入、输出运算符重载：

```
std::ostream &operator<<(std::ostream &out,const CPPString &x)
{
     for(int i=0;i<x.l;i++)
    out<<x.s[i];
     return out;
}
std::istream &operator>>(std::istream &in, CPPString &x)
{
     char* t = new char[100];
     in.getline(t, 100, '\n');
     x = CPPString(t);
     delete[]t;
     return in;
```

3 测试情况

3.1 测试样例设计

3.1.1 基本功能测试

A.增删改查、获取长度、取子串功能测试

B.基本运算符测试

```
----------------------------------------
                B.测试基本运算符
----------------------------------------
str1=00000
str2=11111
str3=22222
str4=33333
str5=12345

----------------------------------------
operator+()
测试+运算符:
str + str2 = 00000 + 11111 = 0000011111

----------------------------------------
operator+=()
测试+=运算符:
str4 += str1    str4=3333300000

----------------------------------------
operator[]()
测试operator[]运算符:
str5[0]=1
str5[1]=2
str5[2]=3
str5[3]=4
str5[4]=5

请按任意键继续. . .
```

C.比较运算符重载测试

```
----------------------------------------
                C.测试比较运算符重载
----------------------------------------
operator==()  operator!=()
 s2: 789 s3: 147
比较s2 s3是否相等，测试==运算符重载: 1是，0否
结果: 0
比较s2 s3是否不等，测试!=运算符重载: 1是，0否
结果: 1

 s: 123 s4: 123
比较s s4是否相等，测试==运算符重载: 1是，0否
结果: 1

比较s s4是否不等，测试!=运算符重载: 1是，0否
结果: 0

----------------------------------------
operator<()  operator>()
 s2: 789 s3: 147
比较s2是否<s3，测试<运算符重载: 1是，0否
结果: 0

比较s2是否>s3，测试>运算符重载: 1是，0否
结果: 1

 s1: 456 s2: 789
比较s1是否<s2，测试<运算符重载: 1是，0否
结果: 1

比较s1是否>s2，测试>运算符重载: 1是，0否
结果: 0
```

```
结果: 1

比较s1是否>s2，测试>运算符重载: 1是，0否
结果: 0

----------------------------------------
operator<=()   operator>=()
 s2: 789 s3: 147
比较s2是否<=s3，测试<=运算符重载: 1是，0否
结果: 0

比较s2是否>=s3，测试>=运算符重载: 1是，0否
结果: 1

 s1: 456 s2: 789
比较s1是否<=s2，测试<=运算符重载: 1是，0否
结果: 1

比较s1是否>=s2，测试>=运算符重载: 1是，0否
结果: 0

 s: 123 s4: 123
比较s是否<=s4，测试<=运算符重载: 1是，0否
结果: 1

比较s是否>=s4，测试>=运算符重载: 1是，0否
结果: 1

请按任意键继续. . .
```

D.*和*=运算符重载测试

```
------------------------------------------
            E.测试*以及*=运算符
------------------------------------------
s1=abcd s2=1234
s1*=2   s1=abcdabcd
s2*2=12341234
2*s2=12341234
```

3.1.2 可靠性测试

主要对下标越界进行异常处理。对于函数，如果传入的下标大于字符串本身长度或者本身数值非法（小于 0），即会引发异常处理。

```
----------------------------------------
           D.异常处理
----------------------------------------

str1 = str2 = str3 = str4 = str5 = str6 = abcd

----------------------------------------
insert()
str1=abcd
str1.ins(str2,9)
输入下标错误

----------------------------------------
delete()
str2=abcd
str2.del(5,4)
输入下标错误

----------------------------------------
replace()
str4=abcd
str4.rep(str1, 5,4)
输入下标错误

----------------------------------------
operator[]
str6[5]=
输入下标错误

请按任意键继续. . .
```

3.2 测试结果对程序的改进情况

在写 ins()函数时，我们本来的想法是将原本的字符串按照插入位置用 cut()函数分割为两个字符串，再将分割后的两个字符串和插入的字符串进行拼接，达到插入的效果，但经过测试，发现运行时间过长，之后我们就选用了比较朴素的方法，即通过遍历对其赋值。