封装C-字符串

2021年10月13日

# **1 封装C-字符串项目概述**

## **1.1 项目内容及要求**

(1) 可以封装C-字符串（即含串结束标志字符'\0'），也可以不含串结束标志；

(2) 对象构造、析构、赋值相关的四大函数；

(3) 其他成员函数（如：length、c\_str等）；

(4) 重载适当的运算符（如：<<、>>、+、+=、<、<=、>、>=、==、!=、[ ]等）；

(5) 有异常处理功能（如：方括号运算时下标越界等）；

(6) 测试用例设计（含异常处理）。

## **1.2 研究人员及分工**

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 学号 | 姓名 | 角色及具体贡献 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# **2 MyString类的设计**

## **2.1 数据成员设计**

char \*p; // 指向数据区域的节点

## **2.2 成员函数（友元函数）原型设计**

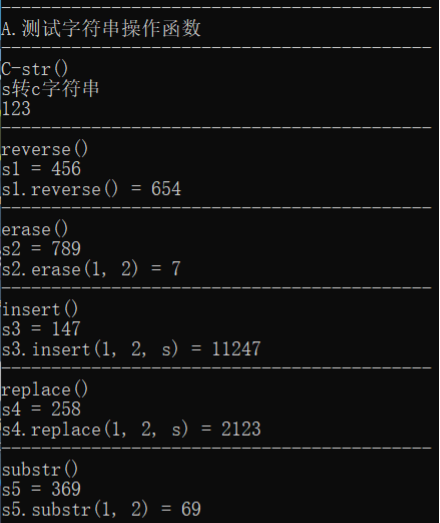
|  |  |  |
| --- | --- | --- |
| 序号 | 函数名 | 作用 |
|  | 构造函数、析构函数 | |
| 1 | MyString(const char \*str = ""); | 转换构造函数、默认构造函数 |
| 2 | MyString(const char \*str, const unsigned int n); | 指定长度的构造转换函数 |
| 3 | MyString(const MyString &str); | 拷贝构造函数 |
| 4 | ~MyString(); | 析构函数 |
|  | 静态成员函数 | |
| 5 | static void New(char\* &str, const size\_t &len); | 申请堆空间 |
| 6 | static void Free(char\*& str); | 释放堆空间 |
|  | 运算符重载 | |
| 7 | MyString &operator=(const MyString &str); | 赋值运算 |
| 8 | char &operator[](const unsigned int &index); | 下标运算 |
| 9 | MyString operator+(const MyString &str) const; | 字符串相加 |
| 10 | MyString &operator+=(const MyString &str); | 字符串相加并赋值 |
| 11 | bool operator==(const MyString &str) const; | 字符串判等 |
| 12 | bool operator<(const MyString &str) const; | 字符串小于 |
| 13 | bool operator<=(const MyString &str) const; | 字符串小于等于 |
| 14 | bool operator>(const MyString &str) const; | 字符串大于 |
| 15 | bool operator>=(const MyString &str) const; | 字符串大于等于 |
|  | 输入、输出 | |
| 16 | friend ostream & operator<<(ostream &out, const MyString &str); | 输出 |
| 17 | friend istream & operator>>(istream &in, MyString &str); | 读入字符串，遇到空白字符结束 |
| 18 | friend istream & getline(istream &in, MyString &str, const char delim = '\n'); | 读入字符串，遇到指定字符结束 |
|  | 其他友元 | |
| 19 | friend int stoi(const MyString &str); | 将字符串转换为整型 |
| 20 | friend void swap(MyString &str1, MyString &str2); | 交换两个字符串 |
| 21 | friend MyString to\_MyString(const char str[]); | 将一个char数组转换为字符串 |
|  | 功能类函数 | |
| 22 | void clear(); | 清空字符串 |
| 23 | MyString copy() const; | 返回一个该字符串的拷贝 |
| 24 | unsigned int size() const; | 返回字符串的长度 |
| 25 | const char \*c\_str() const; | 返回字符串的char数组拷贝 |
| 26 | bool empty() const; | 返回字符串是否为空 |
| 27 | MyString &reverse(); | 反转字符串 |
| 28 | MyString &append(const MyString &str); | 添加段字符串到末尾 |
| 29 | MyString &erase(const unsigned int &index, const unsigned int &len); | 将下标起始于index,长度为len的部分擦除 |
| 30 | MyString &replace(const unsigned int &index, const unsigned int &len, const MyString &str); | 将下标起始于index,长度为len的部分替换为str |
| 31 | MyString &insert(const unsigned int &index, const unsigned int &len, const MyString &str); | 在下标起始于index处插入str,插入的长度为len |
| 32 | MyString substr(const unsigned int &index, const unsigned int &len) const; | 返回下标起始于index,长度为len的子串 |
| 33 | void swap(MyString &str); | 交换两个MyString |

# **3 测试情况**

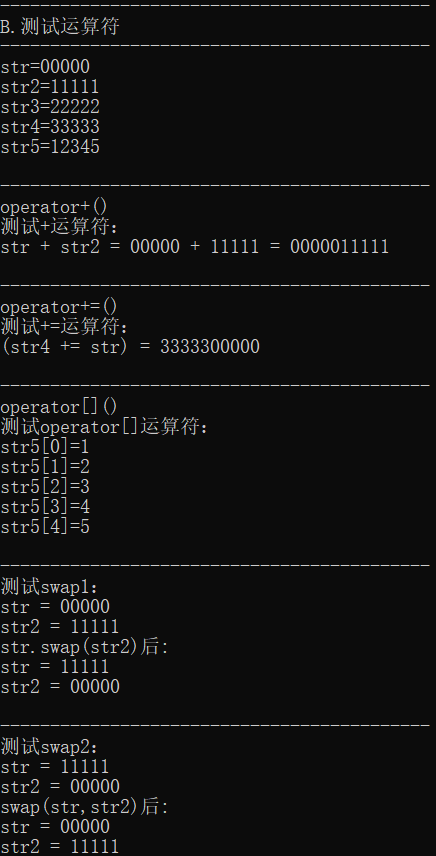
## **3.1 测试样例设计**

**3.1.1 基本功能测试**

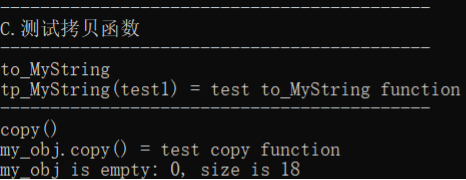
A. 字符串操作函数测试：



B. 运算符重载测试：



C. 拷贝函数测试：



D. 运算符重载函数测试：

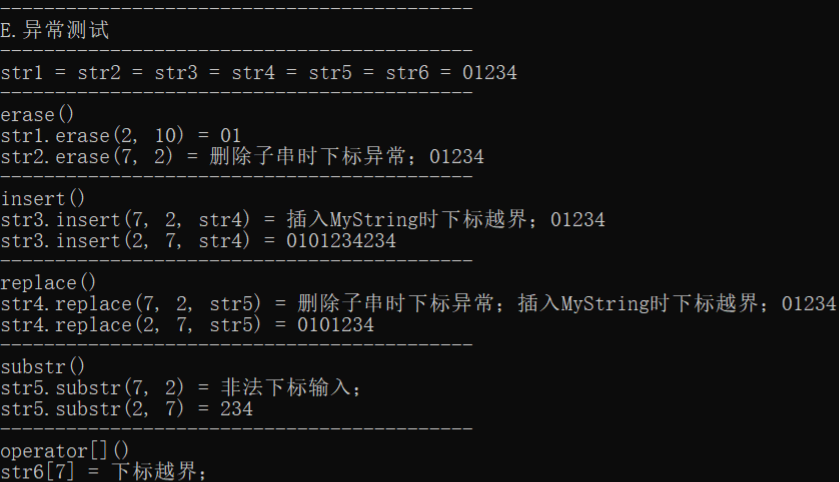
电脑屏幕的照片

中度可信度描述已自动生成

**3.1.2 可靠性测试**

【提示】尽量不要出现运行时错而导致成效崩溃。如0做除数、内存访问出错（如指针使用错误、数组下标越界）等。如果程序中有异常处理，也设计一些测试用例测试对异常的处理。

E. 异常测试：



3.1.2.1 对substr() 函数的异常处理：

try {

if (index >= size()) throw 1.0;

else if (index + len > size()) throw - 1;

New(t\_char, len + 1);

for (size\_t i = index; i < index + len; ++i)

t\_char[i - index] = p[i];

t\_char[len] = '\0';

}

catch (int e) {

size\_t tlen = size();

New(t\_char, tlen - index + 1);

for (size\_t i = index; i < tlen; ++i)

t\_char[i - index] = p[i];

t\_char[tlen - index] = '\0';

}

catch (double e) {

t\_char = new char;

t\_char[0] = '\0';

cerr << "非法下标输入；";

}

此处我们分别处理了子串过长（throw -1）和数组越界（throw 1.0）两种情况。

3.1.2.2 对 erase() 函数的异常处理：

try {

if (index >= tlen) throw - 1;

if (tlen <= len + index) {

char\* tp = NULL;

New(tp, index + 1);

for (size\_t i = 0; i < index; ++i) tp[i] = p[i];

tp[index] = '\0';

Free(p);

p = tp;

return \*this;

}

char\* tp = NULL;

New(tp, tlen - len + 1);

for (size\_t i = 0; i < index; ++i) tp[i] = p[i];

for (size\_t i = index + len; i < tlen; ++i) tp[i - len] = p[i];

tp[tlen - len] = '\0';

Free(p);

p = tp;

}

catch (int e) {

cerr << "删除子串时下标异常；";

}

此处我们处理了删除时删除长度超出字符串长度的问题。

3.1.2.3 对 insert() 函数的异常处理：

try {

if (index > tlen) throw - 1;

for (size\_t i = 0; i < index; ++i) tp[i] = p[i];

}

catch (int e) {

Free(tp);

cerr << "插入MyString时下标越界；";

return \*this;

}

此处我们处理了插入字符串时插入下标过大的情况。

## **3.2 测试结果对程序的改进情况**

使用静态成员函数Free，在释放空间前检测空指针，防止对空指针使用delete；

使用静态成员函数New，传入指针和需要申请的空间的大小，在申请空间前调用Free以确保原指针指向的空间被提前释放。

在所有构造函数中加入了p = new char;确保p永远指向堆空间，使Free不会出错。