封装 C 字符串课程项目

陆宇涵组

陆宇涵 刘逸儒 邱姜铭 卫祎懿

(上海大学 计算机工程与科学学院，上海 200444)

摘　要　根据作业要求，本报告介绍了自定义字符串类 MyString 的设计和实现。该类提供了字符串的基本操作，包括构造、拷贝、移动、赋值、比较、连接和其他常见操作。此外，它还提供了异常处理机制来处理索引越界异常。

关键词　自定义字符串类；字符串操作；运算符重载

Encapsulates the C String Course Project

Yuhan Lu's group

Yuhan Lu, YiRu Liu, JiangMing Qiu, Yiyi Wei

(School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China)

Abstract　According to the job requirements, this report describes the design and implementation of the custom string class MyString. This class provides basic operations for strings, including constructing, copying, moving, assigning, comparing, concatenating, and other common operations. In addition, it provides an exception handling mechanism to handle index out-of-bounds exceptions.

Key words Custom string class, string manipulation, operator overloading.

目录

1 引言

1.1 介绍

本报告介绍了自定义字符串类 MyString 的设计和实现。该类提供了字符串的基本操作，包括构造、拷贝、移动、赋值、比较、连接和其他常见操作。此外，它还提供了异常处理机制来处理索引越界异常。

1.2 项目结构

表 1 C-字符串项目的结构设计与对应功能

	功能
数据域	内部字符数组 字符串长度
功能函数	字符串连接 比较字符串 复制字符串 追加字符串 删除字符串 替换字符串

2 程序细节介绍

2.1 功能结构介绍

本项目创建了一个名为 MyString 的自定义字符串类。可以封装 C-字符串，具有对象构造、析构、赋值相关的四大函数；以及其他成员函数（如：length、c_str 等）；同时重载了运算符；有异常处理功能与测试用例设计。

2.2 程序清单

2.2.1 数据成员设计

```
char *data;
int size;
```

2.2.2 成员函数原型设计

序号	函数名	作用
	构造函数、析构函数	
1	`MyString();`	默认构造函数
2	`MyString(const char *c_str, int length);`	构造函数
3	`MyString(MyString const &string);`	（深）拷贝构造函数
4	`~MyString();`	析构函数
	运算符重载	
7	`MyString &operator=(MyString const &string);`	赋值运算
8	`char operator[](int index) const noexcept(false) ;`	下标运算
9	`MyString &operator+(MyString const &string) const;`	字符串相加
10	`MyString &operator+=(MyString const &string);`	字符串相加并赋值
11	`bool operator==(MyString const &string) const;`	字符串比较
12	`bool operator<(MyString const &string) const;`	字符串小于
13	`bool operator<=(MyString const &string) const;`	字符串小于等于
14	`bool operator>(MyString const &string) const;`	字符串大于
15	`bool operator>=(MyString const &string) const;`	字符串大于等于

	输入、输出	
16	`friend std::ostream &operator<<(std::ostream &os, MyString const &string);`	输出
17	`friend std::istream &operator>>(std::istream &is, MyString &string) = delete;`	读入字符串

3 MyString 类的设计

3.1部分函数

3. 1. 1 运算符重载

```
MyString &MyString::operator+=(const MyString &string) {
    char *temp = new char[this->size];
    for (int i = 0; i < this->size; ++i) {
        temp[i] = this->data[i];
    }
    for (int i = 0; i < string.size; ++i) {
        temp[this->size + i] = string.data[i];
    }
    delete[] this->data;
    this->size += string.size;
    this->data = temp;
    return *this;
}
```

返回原对象，在原对象基础上修改

```
MyString &MyString::operator+(const MyString &string) const {
    auto *temp = new MyString();
    temp->size = this->size + string.size;
    temp->data = new char[temp->size];
    for (int i = 0; i < this->size; ++i) {
        temp->data[i] = this->data[i];
    }
    for (int i = 0; i < string.size; ++i) {
        temp->data[this->size + i] = string.data[i];
    }
    return *temp;
}
```

返回新对象

3.1.1 字符串比较

```
bool MyString::operator==(const MyString &string) const {
    return *this - string == 0;
}

bool MyString::operator<(const MyString &string) const {
    return *this - string < 0;
}

bool MyString::operator>(const MyString &string) const {
    return *this - string > 0;
}

bool MyString::operator<=(const MyString &string) const {
    return *this - string <= 0;
}

bool MyString::operator>=(const MyString &string) const {
    return *this - string >= 0;
}

bool MyString::operator!=(const MyString &string) const {
    return *this - string != 0;
}
```

}

‘operator==’函数将调用‘-’运算符来比较调用它的对象‘(*this)’和传入的参数‘string’。如果它们之间的差异为 0，即两个字符串相等，‘operator==’函数返回‘true’，否则返回‘false’。

3.1.1 移动构造函数

```
MyString::MyString(MyString &&string) noexcept {
    this->size = string.size;
    this->data = string.data;
    string.data = nullptr;
    string.size = 0;
}
```

}

这是 C++中自定义字符串类 `MyString` 的移动构造函数（`MyString` 的右值引用版本）的实现。这个移动构造函数用于将一个临时创建的 `MyString` 对象的内容移动到另一个 `MyString` 对象中，而不是进行深层次的复制。目的是尽可能高效地将资源所有权转移到新对象，而不进行不必要的数据复制。

这个移动构造函数的主要目的是实现资源的所有权转移。在这个实现中:

1. `string` 是一个右值引用，表示它是一个临时对象或即将销毁的对象。
2. `this->size` 和 `this->data` 分别代表当前对象（`*this`）的成员变量，即将接收 `string` 对象的内容。
3. `string.size` 和 `string.data` 是被移动的对象的成员变量。
4. 首先，将 `string` 的 `size` 成员赋给当前对象的 `size` 成员，以表示新对象的大小。
5. 然后，将 `string` 的 `data` 成员（指向字符串数据的指针）赋给当前对象的 `data` 成员。这样，当前对象将共享 `string` 对象的数据，而不是复制它。
6. 接着，将 `string.data` 设为 `nullptr`，以确保 `string` 对象的析构函数不会释放已经移动的数据。
7. 最后，将 `string` 的 `size` 成员设为 0，因为它不再拥有数据。

这个移动构造函数有一个 `noexcept` 说明符，表示这个函数不会抛出异常。

3.2异常处理

3.2.1 对下标访问函数的异常处理：

```
char MyString::operator[](int index) const noexcept(false) {
    if (index < 0 || index >= this->size) {
        throw std::out_of_range("Index out of range");
    }
    return this->data[index];
}
```

4 测试结果展示

4.1 默认构造函数测试

```
默认构造函数测试
Running MyString default_construct = MyString()
Expected: ("") Actual: (default_construct) Passed
Expected: (0) Actual: (default_construct.length()) Passed
```

4.2 空指针测试

```
空指针测试
Running MyString nullptrString(nullptr, 0)
Expected: ("") Actual: (nullptrString) Passed
Expected: (0) Actual: (nullptrString.length()) Passed
```

4.3 默认构造和空指针等价测试

```
默认构造和空指针等价测试
Expected: (true) Actual: (default_construct == nullptrString) Passed
=======================================
Running MyString helloWord("Hello Word!", 11)
```

4.4 字符串拼接测试

```
字符串拼接测试
operator+运算符测试
Expected: ("Hello Word!") Actual: (nullptrString + helloWord) Passed
operator+=运算符测试
Running nullptrString += helloWord
Expected: ("Hello Word!") Actual: (nullptrString) Passed
operator=运算符测试
Running nullptrString = MyString(nullptr, 0)
Expected: ("") Actual: (nullptrString) Passed
```

4.5 自赋值测试

```
自赋值测试
Running nullptrString = nullptrString
Expected: ("") Actual: (nullptrString) Passed
```

4.6 空字符串测试

```
空字符串应和空指针等价
Running MyString emptyString("", 0)
Expected: ("") Actual: (emptyString) Passed
=======================================
Running MyString biggerString("abc", 3)
```

4.7 拷贝构造函数测试

```
拷贝构造函数测试
Running MyString anotherBiggerString(biggerString)
Expected: ("abc") Actual: (anotherBiggerString) Passed
```

4.8 测试移动构造函数

```
移动构造函数测试
Running MyString movedBiggerString(std::move(anotherBiggerString))
Expected: ("abc") Actual: (movedBiggerString) Passed
Running MyString smallerString("ab", 2)
```

4.9 测试比较运算符重载

```
比较测试
Expected: (false) Actual: (biggerString == smallerString) Passed
Expected: (false) Actual: (biggerString < smallerString) Passed
Expected: (true) Actual: (biggerString > smallerString) Passed
Expected: (false) Actual: (biggerString <= smallerString) Passed
Expected: (true) Actual: (biggerString >= smallerString) Passed
Expected: (true) Actual: (biggerString != smallerString) Passed
```

4.10 下标访问异常测试

```
下标访问测试
Expected: ('a') Actual: (biggerString[0]) Passed
下标访问异常测试
Expecting Exception: std::out_of_range
Running biggerString[3]
Exception Matched
```

5 总结

MyString 类的设计和实现演示了 C++中类的基本概念，包括构造函数、析构函数、运算符重载、异常处理等。以下是对整个项目关键的总结:

1、使用动态分配的字符数组来存储字符串数据。

2、在拷贝构造函数和赋值运算符中进行深拷贝，以避免浅拷贝导致的资源共享问题。

3、移动构造函数通过移交数据所有权来提高效率，避免不必要的数据复制。

4、使用异常处理机制，当索引越界时，抛出`std::out_of_range`异常。

5、通过友元函数重载`<<`运算符，实现了方便的字符串输出。

致谢！

附录 完整代码

MyString.h

```
//
// Created by 邱姜铭 on 2023/9/27.
//

#ifndef MYSTRING_MYSTRING_H
#define MYSTRING_MYSTRING_H

#include <iostream>


class MyString {
public:

    MyString();

    MyString(const char *c_str, int length);

    MyString(MyString const &string);

    MyString(MyString &&string) noexcept;

    ~MyString();

    MyString &operator=(MyString const &string);

    std::ostream &operator<<(std::ostream &os) const;

    friend std::ostream &operator<<(std::ostream &os, MyString
const &string);

    std::istream &operator>>(std::istream &is) = delete;

    friend std::istream &operator>>(std::istream &is, MyString
&string) = delete;

    MyString &operator+(MyString const &string) const;
```

```
    MyString &operator+=(MyString const &string);

    bool operator==(MyString const &string) const;

    bool operator<(MyString const &string) const;

    bool operator>(MyString const &string) const;

    bool operator<=(MyString const &string) const;

    bool operator>=(MyString const &string) const;

    bool operator!=(MyString const &string) const;

    char operator[](int index) const noexcept(false) ;

    const char *c_str() const;

    int length() const;

private:
    int operator-(MyString const &string) const;

    char *data;
    int size;
};


#endif //MYSTRING_MYSTRING_H
```

MyString.cpp

```
//
// Created by 邱姜铭 on 2023/9/27.
//

#include "MyString.h"


MyString::MyString() {
    this->size = 0;
    this->data = new char{};
}
```

```
MyString::MyString(const MyString &string) {
    this->size = string.size;
    this->data = new char[this->size];
    for (int i = 0; i < this->size; ++i) {
        this->data[i] = string.data[i];
    }
}

MyString::MyString(MyString &&string) noexcept {
    this->size = string.size;
    this->data = string.data;
    string.data = nullptr;
    string.size = 0;
}

MyString::MyString(const char *c_str, int size) {
    if (c_str == nullptr || size == 0) {
        this->size = 0;
        this->data = new char{};
    } else {
        this->size = size;
        this->data = new char[this->size];
        for (int i = 0; i < this->size; ++i) {
            this->data[i] = c_str[i];
        }
    }
}

MyString::~MyString() {
    delete[] this->data;
}

MyString &MyString::operator=(const MyString &string) {
    if (this == &string) {
        return *this;
    }
    delete[] this->data;
    this->size = string.size;
    this->data = new char[this->size];
    for (int i = 0; i < this->size; ++i) {
        this->data[i] = string.data[i];
```

```
    }
    return *this;
}

std::ostream &MyString::operator<<(std::ostream &os) const {
    for (int i = 0; i < this->size; ++i) {
        os << this->data[i];
    }
    return os;
}

std::ostream &operator<<(std::ostream &os, const MyString &string)
{
    return string.operator<<(os);
}

const char *MyString::c_str() const {
    char *temp = new char[this->size + 1];
    for (int i = 0; i < this->size; ++i) {
        temp[i] = this->data[i];
    }
    temp[this->size] = '\0';
    return temp;
}

MyString &MyString::operator+(const MyString &string) const {
    auto *temp = new MyString();
    temp->size = this->size + string.size;
    temp->data = new char[temp->size];
    for (int i = 0; i < this->size; ++i) {
        temp->data[i] = this->data[i];
    }
    for (int i = 0; i < string.size; ++i) {
        temp->data[this->size + i] = string.data[i];
    }
    return *temp;
}

MyString &MyString::operator+=(const MyString &string) {
    char *temp = new char[this->size];
    for (int i = 0; i < this->size; ++i) {
        temp[i] = this->data[i];
```

```
    }
    for (int i = 0; i < string.size; ++i) {
        temp[this->size + i] = string.data[i];
    }
    delete[] this->data;
    this->size += string.size;
    this->data = temp;
    return *this;
}

bool MyString::operator==(const MyString &string) const {
    return *this - string == 0;
}

bool MyString::operator<(const MyString &string) const {
    return *this - string < 0;
}

bool MyString::operator>(const MyString &string) const {
    return *this - string > 0;
}

bool MyString::operator<=(const MyString &string) const {
    return *this - string <= 0;
}

bool MyString::operator>=(const MyString &string) const {
    return *this - string >= 0;
}

bool MyString::operator!=(const MyString &string) const {
    return *this - string != 0;
}

char MyString::operator[](int index) const noexcept(false) {
    if (index < 0 || index >= this->size) {
        throw std::out_of_range("Index out of range");
    }
    return this->data[index];
}

int MyString::operator-(const MyString &string) const {
```

```
    if (this == &string) {
        return 0;
    }
    if (this->size != string.size) {
        return this->size - string.size;
    }
    for (int i = 0; i < this->size; ++i) {
        if (this->data[i] != string.data[i]) {
            return this->data[i] - string.data[i];
        }
    }
    return 0;
}

int MyString::length() const {
    return this->size;
}
```

my_assert.h

```
//
// Created by 邱姜铭 on 2023/10/8.
//

#ifndef MYSTRING_MY_ASSERT_H
#define MYSTRING_MY_ASSERT_H

#include <iostream>
#include <sstream>

#ifdef _WIN32
#include "windows.h"
#define print_red(str)
"";SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
FOREGROUND_RED); std::cout << str;
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE);std::cout
#define print_green(str)
"";SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
FOREGROUND_GREEN); std::cout << str;
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE);std::cout
#define print_yellow(str)
```

```
"";SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
FOREGROUND_RED | FOREGROUND_GREEN); std::cout << str;
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE);std::cout
#else
#define print_red(str) "\033[31m" << str << "\033[0m"
#define print_green(str) "\033[32m" << str << "\033[0m"
#define print_yellow(str) "\033[33m" << str << "\033[0m"
#endif
#define run(x) std::cout << "Running " << print_yellow(#x) <<
std::endl; x;

#define my_assert_throws(x, exception) \
try {                                  \
    std::cout << print_green("Expecting Exception: ") << #exception
<< std::endl;    \
    run(x);                                \
    std::cout << print_red("No Exception throws Failed") <<
std::endl; \
} catch (exception &e) {               \
    std::cout << print_green("Exception Matched") << std::endl; \
}

#define my_assert(actual, expected) \
std::cout << print_green("Expected:") << " (" << #expected << ") "
\
     << print_red("Actual:") <<" (" << print_yellow(#actual) <<
")"; \
my_assert_impl<decltype(actual)>(actual, expected);          \


template<typename T>
void my_assert_impl(T actual, T expected) {
    if (actual == expected) {
        std::cout << print_green(" Passed") << std::endl;
    } else {
        std::cout << print_red(" Failed") << std::endl;
    }
}

template<typename T>
void my_assert_impl(T actual, std::string_view expected) {
```

```
    std::ostringstream oss;
    oss << actual;
    if (oss.str() == expected) {
        std::cout << print_green(" Passed") << std::endl;
    } else {
        std::cout << print_red(" Failed") << std::endl;
    }
}


#endif //MYSTRING_MY_ASSERT_H
```

MyStringTest.cpp

```
//
// Created by 邱姜铭 on 2023/9/27.
//
#include "MyString.h"
#include "my_assert.h"


using namespace std;


int main() {
    cout << "默认构造函数测试" << endl;
    run(MyString default_construct = MyString())
    my_assert(default_construct, "")
    my_assert(default_construct.length(), 0)
    cout << "空指针测试" << endl;
    run(MyString nullptrString(nullptr, 0))
    my_assert(nullptrString, "")
    my_assert(nullptrString.length(), 0)
    cout << "默认构造和空指针等价测试" << endl;
    my_assert(default_construct == nullptrString, true)

    cout << "========================================" << endl;
    run(MyString helloWord("Hello Word!", 11))
    cout << "字符串输出测试" << endl;
    my_assert(helloWord, "Hello Word!")
    cout << "字符串拼接测试" << endl;
    cout << "operator+运算符测试" << endl;
    my_assert(nullptrString + helloWord, "Hello Word!")
```

```
    cout << "operator+=运算符测试" << endl;
    run(nullptrString += helloWord)
    my_assert(nullptrString, "Hello Word!")
    cout << "operator=运算符测试" << endl;
    run(nullptrString = MyString(nullptr, 0))
    my_assert(nullptrString, "")
    cout << "自赋值测试" << endl;
    run(nullptrString = nullptrString)
    my_assert(nullptrString, "")
    cout << "空字符串应和空指针等价" << endl;
    run(MyString emptyString("", 0))
    my_assert(emptyString, "")

    cout << "========================================" << endl;
    run(MyString biggerString("abc", 3))
    cout << "拷贝构造函数测试" << endl;
    run(MyString anotherBiggerString(biggerString))
    my_assert(anotherBiggerString, "abc")
    cout << "移动构造函数测试" << endl;
    run(MyString movedBiggerString(std::move(anotherBiggerString)))
    my_assert(movedBiggerString, "abc")
    run(MyString smallerString("ab", 2))
    cout << "比较测试" << endl;
    my_assert(biggerString == smallerString, false)
    my_assert(biggerString < smallerString, false)
    my_assert(biggerString > smallerString, true)
    my_assert(biggerString <= smallerString, false)
    my_assert(biggerString >= smallerString, true)
    my_assert(biggerString != smallerString, true)
    cout << "下标访问测试" << endl;
    my_assert(biggerString[0], 'a')
    cout << "下标访问异常测试" << endl;
    my_assert_throws(biggerString[3], std::out_of_range)
    return 0;
}
```