Advanced Programming* Homework *III* LaTeX

* Teacher: Shujian Huang. TA: Yiyu Zhang

1st 张逸凯 171840708

Department of Computer Science and Technology

Nanjing University

zykhelloha@gmail.com

1. 概念题(从课本中回答, 回归课本)

一、 C++ 中为什么需要对操作符进行重载？除了常用的操作符外，还可以对哪些特殊的操作符进行重载？

解:

对于能够直接使用的操作符, 语言规定的含义有时不能满足实际要求, 对于不能直接使用的操作符, 把它们用于类的对象有时会给程序设计带来方便.

除了常用的操作符, 还可以对赋值操作符"=", 下标操作符"[]", 类成员访问操作符"-¿", 动态存储分配与去配操作符new和delete, 类型转换操作符(通过类型名)以及函数调用操作符"()"等.

二、C++ 中系统提供的隐式赋值操作存在什么问题？如何解决？

解: 如果某些对象指向了一个动态区域, 提供的隐式赋值操作会让两个对象指向同一块内存区域, 这是十分危险的, 除了会产生与隐式拷贝构造函数类似的问题(两个对象相互干扰以及一块空间被归还两次)以外, 它还会导致内存泄露, 因为被赋值的那个对象之前指向的动态内存区域找不到了.

解决办法就是自己定义赋值操作符重载函数: 这里说一下整体的思路: 首先防止自身赋值, 把原来的空间delete之后, 在重新申请一块空间, 然后将需要赋值的内容复制到这块空间上. 这样就保证了两个对象的指正不会指向同一块内存区域.

2. 编程题

现有一个 CustomString 类，请按照要求思考需要重载的运算符操作或者构造函数相关的操作，补全string_operator.h以及string_operator.cpp文件，得到相应的输出结果。

见附件 `string_operator.h`, `string_operator.cpp`.

*谢谢老师和助教哥的耐心批改.

高级程序设计 操作符重载 程序部分作业

张逸凯 171840708

`string_operator.cpp`:

```
#include "string_operator.h"


///////////////// 自己写的部分:
// 构造函数
CustomString::CustomString() : p(NULL), len(0) { }
CustomString::CustomString(const char* str) : len(strlen(str) + 1) {
    p = new char[len];
    strcpy(p, str);
}

// 注意不要漏了拷贝构造函数:
CustomString::CustomString(const CustomString& a) : len(a.len) {
    p = new char[a.len];
    strcpy(p, a.p);
}



char& CustomString::operator [](int idx) {
    if (idx >= len) {
        printf("index ERROR!");
        exit(0);
    }
    return p[idx];
}

// 重载加号, 返回和传进去都是const A&, 返回的都是函数内创建的临时对象
CustomString CustomString::operator +(const CustomString& a) {
    char* arr = new char[this->len + a.len];
    int cnt = 0;
    for (int i = 0; p[i] != '\0'; ++i) {
        arr[cnt++] = p[i];
    }
    for (int i = 0; a.p[i] != '\0'; ++i) {
        arr[cnt++] = a.p[i];
    }
    arr[cnt++] = '\0';

    CustomString tmp(arr);

    delete[]arr;
    return tmp;
}
// 重载+=, 返回和传进去和重载+一样, 但是是直接修改本身(就是第一个参数), 返回*this
CustomString CustomString::operator +=(const CustomString& a) {
```

```
    char *arr = new char[this->len + a.len];
    int cnt = 0;
    for (int i = 0; p[i] != '\0'; ++i) {
        arr[cnt++] = p[i];
    }
    for (int i = 0; a.p[i] != '\0'; ++i) {
        arr[cnt++] = a.p[i];
    }
    arr[cnt++] = '\0';
    this->len = cnt;

    delete[]this->p;
    p = new char[this->len];
    strcpy(p, arr);

    return *this;
}

bool CustomString::operator ==(const CustomString& a) {
    if (strcmp(this->p, a.p) == 0) {
        return true;
    }

    return false;
}

bool CustomString::operator !=(const CustomString& a) {
    return !(*this == a);
}
// 重载 << 操作符函数应把ostream&作为其第一个参数，对类类型const对象的引用作为第二个参数，并返回对ostream形参的引用，类内声明为友元.
ostream& operator <<(ostream& out, const CustomString& a) {
    out << a.p;
    return out;
}

// 注意输入这里就不需要 const A& 了.
istream& operator >>(istream& in, CustomString& a) {
    delete[]a.p;
    char arr[100010] = "";
    in >> arr;
    a.len = strlen(arr) + 1;

    a.p = new char[a.len];
    strcpy(a.p, arr);
    return in;
}

////////////////// 自己写的部分 (上面)

// 析构函数
CustomString::~CustomString()
{
    delete[] p;
    p = NULL;
}

int main() {
```

```
    CustomString mystr("this is e CustomString class for testing!");
    cout << mystr[8] << endl;
    mystr[8] = 'a';
    cout << mystr << endl;
    CustomString mystr2 = mystr;            // 注意这里是调用拷贝构造函数，不是赋值操
作符重载!
    cout << mystr2 << endl;
    CustomString mystr3;
    mystr3.operator=(mystr + mystr2);
    cout << mystr3 << endl;
    cout << mystr + mystr2 << endl;
    mystr3 += mystr;
    cout << mystr3 << endl;
    cout << (mystr == mystr2) << endl;
    cout << (mystr != mystr3) << endl;
    CustomString mystr4;
    cout << "Input any string to test the overloaded input operator >>: " <<
endl;
    cin >> mystr4;
    cout << mystr4 << endl;
    cout << "Congratulations! testing passed!" << endl;

    return 0;
}
```

`string_operator.h`:

```
#pragma once
#include <iostream>
#include <string.h>

using namespace std;

class CustomString
{
public:
    //构造函数
    CustomString();
    CustomString(const char* str);
    CustomString(const CustomString& a);        // 不要忘了拷贝构造函数.
    CustomString operator +(const CustomString& a);
    CustomString operator +=(const CustomString& a);
    bool operator ==(const CustomString& a);
    bool operator !=(const CustomString& a);
    friend ostream& operator <<(ostream& os, const CustomString& a);
    friend istream& operator >>(istream& in, CustomString& a);

    CustomString& operator=(const CustomString& a) {
        delete[]this->p;

        p = new char[a.len];
        strcpy(p, a.p);
        len = a.len;

        return *this;
    }
```

```
    //析构函数
    ~CustomString();


    // 自己写的部分:
    char& operator [](int idx);
private:
    char* p; // 字符串的起始地址
    int len; // 字符串的长度
};
```