C++作业

罗悦 2016220304022

习题 5.1：

C 风格字符串的类型是 char*，使用起来不是很方便。请读者设计一个类 cstring 来封装这种字符串。这里只考虑类的属性和操作接口，暂时不考虑操作的实现。

所编写的程序如下所示：

```
#ifndef __MYSTRING_H__
#define __MYSTRING_H__
#endif
#include <iostream>
using namespace std;

class cstring                                   //类 cstring
{
    public:
        cstring();                              //构造函数
        cstring(const char *_str);      //重载构造函数
    public:
        const cstring operator +(const cstring &_str);
        void operator =(const cstring &_str);
    public:
        friend istream & operator >>(istream &_in, cstring &_str);
        friend ostream & operator <<(ostream &_out, cstring &_str);
    private:
        char *str;
        int length;
};

cstring::cstring()
{
    str = new char;
    str[0] = '\0';
    length = 0;
}
```

```
cstring::cstring(const char *_str)
{
	length = strlen(_str);
	str = new char[length + 1];
	str[length] = '\0';
	strncpy(str, _str, length+1);
}

const cstring cstring::operator +(const cstring &_str)//运算符“+”的重载
{
	int newLength = length + _str.length;
	char * newStr = new char[newLength + 1];
	newStr[newLength] = '\0';
	strncpy(newStr, str, length);
	strncat(newStr, _str.str, _str.length);

	return cstring(newStr);
}

void cstring::operator =(const cstring &_str)		//运算符“=”的重载
{
	str = _str.str;
	length = _str.length;
}

istream & operator >>(istream &_in, cstring &_str) //重载“>>”
{
	_in >> _str.str;
	_str.length = strlen(_str.str);
	return _in;
}

ostream & operator <<(ostream &_out, cstring &_str)//“<<”,打印字符串和长度
{
	_out << _str.str << endl << "length: " << _str.length;
	return _out;
}
```

如上程序所示，我设计了一个 string 类，这个类对简单的运算符进行了重载，实现了可以对字符串的简单操作。

习题 5.2：请编码实现习题 5.1 中设计的 cstring 类。注意内存资源的申请和释放。

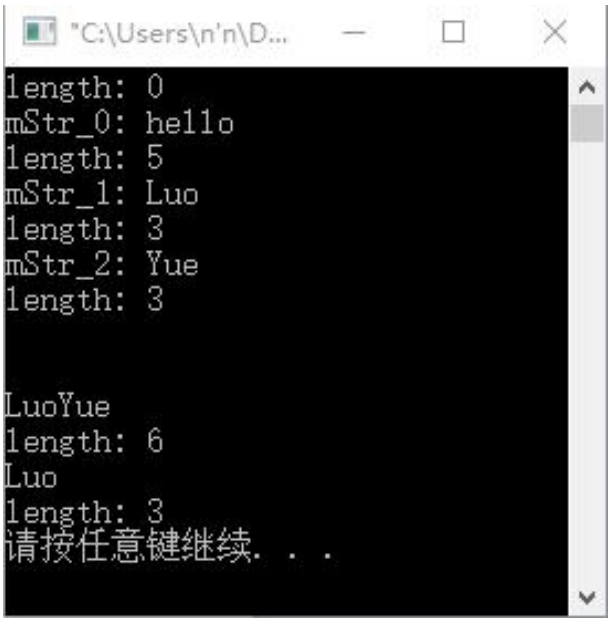
所编写的程序如下所示：

```
#ifndef __MYSTRING_H__
#define __MYSTRING_H__
#endif
#include <iostream>
using namespace std;

class cstring                                    //类 cstring
{
    public:
        cstring();                               //构造函数
        cstring(const char *_str);       //重载构造函数
    public:
        const cstring operator +(const cstring &_str);
        void operator =(const cstring &_str);
    public:
        friend istream & operator >>(istream &_in, cstring &_str);
        friend ostream & operator <<(ostream &_out, cstring &_str);
    private:
        char *str;
        int length;
};

cstring::cstring()
{
    str = new char;
    str[0] = '\0';
    length = 0;
}

cstring::cstring(const char *_str)
{
    length = strlen(_str);
    str = new char[length + 1];
    str[length] = '\0';
    strncpy(str, _str, length+1);
}

const cstring cstring::operator +(const cstring &_str)//运算符“+”的重载
```

```
{
	int newLength = length + _str.length;
	char * newStr = new char[newLength + 1];
	newStr[newLength] = '\0';
	strncpy(newStr, str, length);
	strncat(newStr, _str.str, _str.length);

	return cstring(newStr);
}

void cstring::operator =(const cstring &_str)		//运算符“=”的重载
{
	str = _str.str;
	length = _str.length;
}

istream & operator >>(istream &_in, cstring &_str) //重载“>>”
{
	_in >> _str.str;
	_str.length = strlen(_str.str);
	return _in;
}

ostream & operator <<(ostream &_out, cstring &_str)//“<<”,打印字符串和长度
{
	_out << _str.str << endl << "length: " << _str.length;
	return _out;
}

int main()
{
	cstring mStr;				//定义对象 mStr
	cstring mStr_0("hello");		//定义对象 mStr_0
	cstring mStr_1("Luo");			//定义对象 mStr_1
	cstring mStr_2("Yue");			//定义对象 mStr_2

	cout << "mStr: " << mStr << endl;
	cout << "mStr_0: " << mStr_0 << endl;
	cout << "mStr_1: " << mStr_1 << endl;
	cout << "mStr_2: " << mStr_2 << endl;
	cout << "\n" << endl;

	mStr_0 = mStr_1;
	mStr = mStr_1 + mStr_2;
```

```
    cout << mStr << endl;
    cout << mStr_0 << endl;

    return 0;
}
```

如上程序所示，我设计了一个 string 类，这个类对简单的运算符进行了重载，实现了可以对字符串的简单操作，并设计了几个对象进行验证。测试结果如下图所示：



其中定义对象 mStr 时，长度为 0，定义 mStr_0 时，把“hello”赋给了私有变量*str。定义 mStr_1 和 mStr_2 时，分别把字符串“Luo”和“Yue”赋给了私有变量*str。最后把 mStr_1 和 mStr_2 两“字符串”相加得到 mStr_3 并打印出来。

习题 5.3：请编写一个雇员 employee 类，其中包含如姓名、薪酬等属性，以及一些相关操作。为该类定义一个静态数据成员来保存所有雇员的薪酬总额，并定义一些静态成员函数来操作这个成员。

所编写程序如下所示：

```
#include<iostream>
using namespace std;

class employee                          //类 employee
{
    private:
        char *name;                     //雇员姓名
        int pay;                        //雇员薪酬
        static int n_pay;               //静态变量 n_pay，用于存放雇员总薪酬额
    public:
        employee(){                     //构造函数
            name=new char;
            name[0]='\0';
        }
        employee(char *na,int pa){      //重载构造函数
            name=na;
            pay=pa;
            n_pay+=pa;
        }
        char *getname(){
            return name;
        }
        static int getn_pay();          //静态成员函数，用于操作静态数据成员 n_pay;
};

int employee::n_pay=0;
int employee::getn_pay(){               //返回静态数据成员 n_pay;
    return n_pay;
}
int main(){
    employee A("Luo",50);
    employee B("Yue",60);
    employee C("LuoYue",70);    //定义三个雇员对象，薪酬分别为 50、60、70；
    cout<<A.getname()<<"、"<<B.getname()<<"、"<<C.getname()<<endl;
    cout<<"三位雇员的总薪酬为："<<A.getn_pay()<<endl;
    return 0;
}
```

如上程序所示，我定义了一个类 employee，定义了私有变量 name，和 pay，同时也定义了一个静态数据 n_pay，主要用于存放雇员的总薪酬，在构造函数中，用 n_pay 加上新的雇员薪酬作为新的 n_pay 的

值，这样在 main 函数中，每定义一个对象，即能够把该对象的薪酬加到总薪酬上去，同时也定义了一个返回静态数据 n_pay 的静态成员函数 getn_pay()；用于返回 n_pay 的值。实验结果如下图所示：

定义了三位雇员：“Luo”，薪酬 50；“Yue”，薪酬 60；“LuoYue”，薪酬 70；所以三位雇员总薪酬为 180。

习题 5.5：请读者编写一个整型堆栈类，内部实现使用数组。

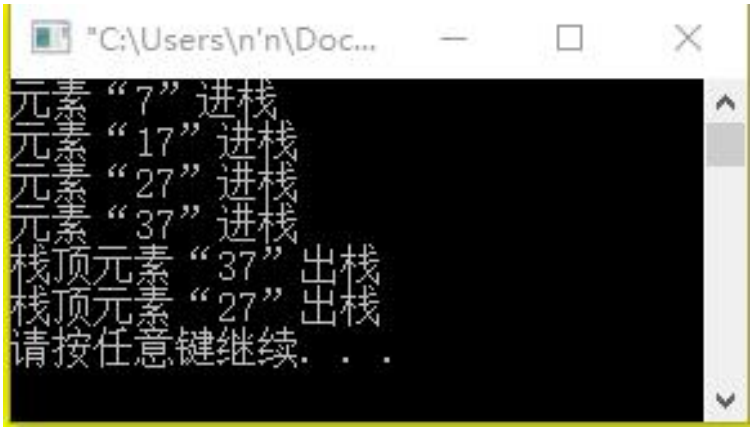
编写程序如下所示：

```
#include<iostream>
using namespace std;
class Stack
{
    public:
        Stack()        //构造函数
        {
            top=-1;
        }
        bool push(int n)//压栈
        {
            if(!isfull())
                data[++top]=n;
            else
                return false;
            return true;
        }
        bool pop()//退栈
        {
            if(!isempty())
```

```
                top--;
            else
                return false;
            return true;
        }
        int gettop()//得到栈顶元素
        {
            return data[top];
        }
        bool isempty()//判断是否为空
        {
            return top==-1?true:false;
        }
        bool isfull()//判断是否已满
        {
            return top==9?true:false;
        }
    private:
        int data[10];
        int top;
};

int main()
{
    Stack s;//建立一个栈
    if(!s.push(7))//将 7 入栈；
    {
        cout<<"栈溢出"<<endl;
        return 0;
    }
    cout<<"元素“"<<"7"<<"”进栈"<<endl;
    if(!s.push(17))//将 17 入栈；
    {
        cout<<"栈溢出"<<endl;
        return 0;
    }
    cout<<"元素“"<<"17"<<"”进栈"<<endl;
    if(!s.push(27))//将 27 入栈；
    {
        cout<<"栈溢出"<<endl;
        return 0;
    }
    cout<<"元素“"<<"27"<<"”进栈"<<endl;
    if(!s.push(37))//将 37 入栈；
```

```
    {
        cout<<"栈溢出"<<endl;
        return 0;
    }
    cout<<"元素“"<<"37"<<"”进栈"<<endl;
    cout<<"栈顶元素“"<<s.gettop()<<"”出栈"<<endl;//读出并输出栈顶元素；
    s.pop();//退栈
    cout<<"栈顶元素“"<<s.gettop()<<"”出栈"<<endl;//读出并输出栈顶元素；
    s.pop();//退栈
}
```

如上程序所示，我设计了一个类 Stack，并且以数组作为内部实现，且定义栈的大小为 10，在 main 函数中将 7、17、27、37 分别压入栈，再将栈顶元素取出并打印。实验结果如下图所示：



如上图所示为进栈、出栈的过程。

习题 6.3：请读者为第五章设计的 cstring 类添加构造函数和析构函数。注意构造函数的重载。注意：如果读者的编译器支持 C++ 1y 标准，那么也请读者主动使用 C++ 1y 语法。

编写程序如下所示：

```
#ifndef __MYSTRING_H__
#define __MYSTRING_H__
#endif
#include <iostream>
using namespace std;
```

```
class cstring                                   //类 cstring
{
    public:
        cstring();                              //构造函数
        cstring(const char *_str);      //重载构造函数
        ~cstring();
    public:
        const cstring operator +(const cstring &_str);
        void operator =(const cstring &_str);
    public:
        friend istream & operator >>(istream &_in, cstring &_str);
        friend ostream & operator <<(ostream &_out, cstring &_str);
    private:
        char *str;
        int length;
};

cstring::cstring()
{
    str = new char;
    str[0] = '\0';
    length = 0;
}

cstring::cstring(const char *_str)
{
    length = strlen(_str);
    str = new char[length + 1];
    str[length] = '\0';
    strncpy(str, _str, length+1);
}

cstring::~cstring(){}

const cstring cstring::operator +(const cstring &_str)//运算符“+”的重载
{
    int newLength = length + _str.length;
    char * newStr = new char[newLength + 1];
    newStr[newLength] = '\0';
    strncpy(newStr, str, length);
    strncat(newStr, _str.str, _str.length);

    return cstring(newStr);
```

```
}

void cstring::operator =(const cstring &_str)            //运算符“=”的重载
{
    str = _str.str;
    length = _str.length;
}

istream & operator >>(istream &_in, cstring &_str) //重载“>>”
{
    _in >> _str.str;
    _str.length = strlen(_str.str);
    return _in;
}

ostream & operator <<(ostream &_out, cstring &_str)//“<<”,打印字符串和长度
{
    _out << _str.str << endl << "length: " << _str.length;
    return _out;
}

int main()
{
    cstring mStr;                        //定义对象 mStr
    cstring mStr_0("hello");        //定义对象 mStr_0
    cstring mStr_1("Luo");           //定义对象 mStr_1
    cstring mStr_2("Yue");           //定义对象 mStr_2

    cout << "mStr: " << mStr << endl;
    cout << "mStr_0: " << mStr_0 << endl;
    cout << "mStr_1: " << mStr_1 << endl;
    cout << "mStr_2: " << mStr_2 << endl;
    cout << "\n" << endl;

    mStr_0 = mStr_1;
    mStr = mStr_1 + mStr_2;

    cout << mStr << endl;
    cout << mStr_0 << endl;

    return 0;
}
```

如上程序所示，我在 string 类中添加了析构函数和重载构造函数。

这个类对简单的运算符进行了重载，实现了可以对字符串的简单操作，并设计了几个对象进行验证。测试结果如下图所示：

其中定义对象 mStr 时，长度为 0，定义 mStr_0 时，把“hello”赋给了私有变量*str。定义 mStr_1 和 mStr_2 时，分别把字符串“Luo”和“Yue”赋给了私有变量*str。最后把 mStr_1 和 mStr_2 两“字符串”相加得到 mStr_3 并打印出来。

习题 6.4：请读者为 cstring 类添加一个复制构造函数。如有可能，再添加转移复制构造函数。

编写程序如下所示：

```
#ifndef __MYSTRING_H__
#define __MYSTRING_H__
#endif
#include <iostream>
using namespace std;

class cstring                                    //类 cstring
{
      public:
```

```
        cstring();                              //构造函数
        cstring(const char *_str);              //重载构造函数
        cstring(const cstring &_str);           //复制构造函数
        ~cstring();
    public:
        const cstring operator +(const cstring &_str);
        void operator =(const cstring &_str);
    public:
        friend istream & operator >>(istream &_in, cstring &_str);
        friend ostream & operator <<(ostream &_out, cstring &_str);
    private:
        char *str;
        int length;
};

cstring::cstring()
{
    str = new char;
    str[0] = '\0';
    length = 0;
}

cstring::cstring(const char *_str)
{
    length = strlen(_str);
    str = new char[length + 1];
    str[length] = '\0';
    strncpy(str, _str, length+1);
}

cstring::cstring(const cstring &_str)
{
    length = _str.length;
    str = new char[length + 1];
    str[length] = '\0';
    strncpy(str, _str.str, length+1);
}

cstring::~cstring(){}

const cstring cstring::operator +(const cstring &_str)//运算符“+”的重载
{
    int newLength = length + _str.length;
    char * newStr = new char[newLength + 1];
```

```
	newStr[newLength] = '\0';
	strncpy(newStr, str, length);
	strncat(newStr, _str.str, _str.length);

	return cstring(newStr);
}

void cstring::operator =(const cstring &_str)		//运算符“=”的重载
{
	str = _str.str;
	length = _str.length;
}

istream & operator >>(istream &_in, cstring &_str) //重载“>>”
{
	_in >> _str.str;
	_str.length = strlen(_str.str);
	return _in;
}

ostream & operator <<(ostream &_out, cstring &_str)//“<<”,打印字符串和长度
{
	_out << _str.str << endl << "length: " << _str.length;
	return _out;
}

int main()
{
	cstring mStr;			//定义对象 mStr
	cstring mStr_0(mStr);		//定义对象 mStr_0
	cstring mStr_1("Luo");		//定义对象 mStr_1
	cstring mStr_2("Yue");		//定义对象 mStr_2

	cout << "mStr: " << mStr << endl;
	cout << "mStr_0: " << mStr_0 << endl;
	cout << "mStr_1: " << mStr_1 << endl;
	cout << "mStr_2: " << mStr_2 << endl;
	cout << "\n" << endl;

	mStr_0 = mStr_1;
	mStr = mStr_1 + mStr_2;

	cout << mStr << endl;
	cout << mStr_0 << endl;
```

```
    return 0;
}
```

如上程序所示，我在 string 类中添加了析构函数和重载构造函数和复制构造函数。这个类对简单的运算符进行了重载，实现了可以对字符串的简单操作，并设计了几个对象进行验证。测试结果如下图所示：

其中定义对象 mStr 时，长度为 0，定义 mStr_0 时，把“hello”赋给了私有变量*str。定义 mStr_1 和 mStr_2 时，分别把字符串“Luo”和“Yue”赋给了私有变量*str。最后把 mStr_1 和 mStr_2 两“字符串”相加得到 mStr_3 并打印出来。

习题 6.6：请读者将 putstr()放在某个类中，并使这个类成为 string 类的友元类。

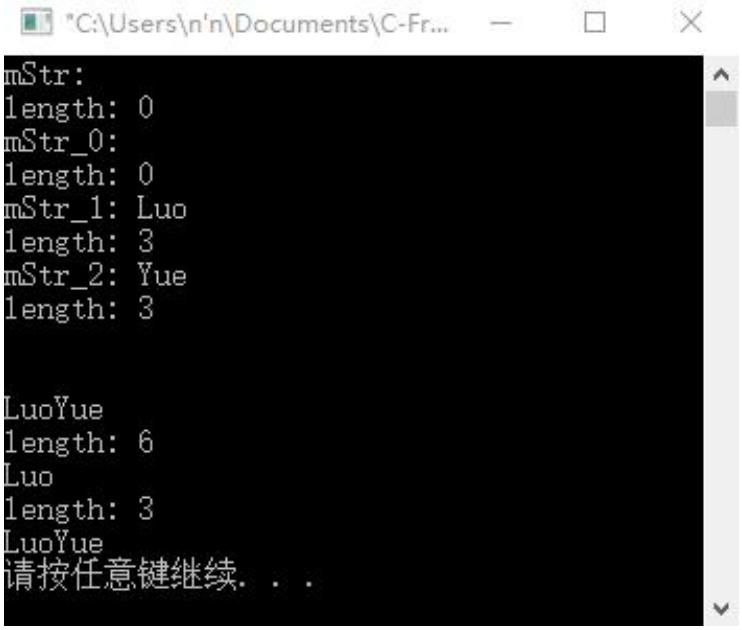
所编写程序如下所示：

```
#ifndef __MYSTRING_H__
#define __MYSTRING_H__
#endif
#include <iostream>
using namespace std;

class PP;

class cstring                                    //类 cstring
{
    public:
        cstring();                               //构造函数
        cstring(const char *_str);               //重载构造函数
        cstring(const cstring &_str);            //复制构造函数
        friend class PP;                         //定义 PP 作为友元类
        ~cstring();
    public:
        const cstring operator +(const cstring &_str);
        void operator =(const cstring &_str);
    public:
        friend istream & operator >>(istream &_in, cstring &_str);
        friend ostream & operator <<(ostream &_out, cstring &_str);
    private:
        char *str;
        int length;
};

cstring::cstring()
{
    str = new char;
    str[0] = '\0';
    length = 0;
}

cstring::cstring(const char *_str)
{
    length = strlen(_str);
    str = new char[length + 1];
    str[length] = '\0';
    strncpy(str, _str, length+1);
}

cstring::cstring(const cstring &_str)
{
```

```
        length = _str.length;
        str = new char[length + 1];
        str[length] = '\0';
        strncpy(str, _str.str, length+1);
}

cstring::~cstring(){}

const cstring cstring::operator +(const cstring &_str)//运算符“+”的重载
{
        int newLength = length + _str.length;
        char * newStr = new char[newLength + 1];
        newStr[newLength] = '\0';
        strncpy(newStr, str, length);
        strncat(newStr, _str.str, _str.length);

        return cstring(newStr);
}

void cstring::operator =(const cstring &_str)             //运算符“=”的重载
{
        str = _str.str;
        length = _str.length;
}

istream & operator >>(istream &_in, cstring &_str) //重载“>>”
{
        _in >> _str.str;
        _str.length = strlen(_str.str);
        return _in;
}

ostream & operator <<(ostream &_out, cstring &_str)//“<<”,打印字符串和长度
{
        _out << _str.str << endl << "length: " << _str.length;
        return _out;
}

class PP                                                  //类 PP
{
      public:
            void putstr(cstring &s){          //putstr 函数
                  cout<<s.str<<endl;
            }
```

```
};
int main()
{
	cstring mStr;                    //定义对象 mStr
	cstring mStr_0(mStr);            //定义对象 mStr_0
	cstring mStr_1("Luo");           //定义对象 mStr_1
	cstring mStr_2("Yue");           //定义对象 mStr_2

	cout << "mStr: " << mStr << endl;
	cout << "mStr_0: " << mStr_0 << endl;
	cout << "mStr_1: " << mStr_1 << endl;
	cout << "mStr_2: " << mStr_2 << endl;
	cout << "\n" << endl;

	mStr_0 = mStr_1;
	mStr = mStr_1 + mStr_2;

	cout << mStr << endl;
	cout << mStr_0 << endl;
	PP p;                                //定义一个 PP 类的对象 p
	p.putstr(mStr);                      //调用 putstr 函数打印对象 mStr 的字符串

	return 0;
}
```

如上程序所示，在原程序上，我定义了一个类 PP,并且在类 cstring 把类 PP 声明为友元类，这样在 PP 类中的成员函数 putstr 就可以调用类 PP 中的私有成员变量了，上程序用 putstr 函数打印了 cstring 类的对象 mStr 的字符串。实验结果如下所示：



最后一行字符串“LuoYue”时通过友元类函数实现打印。