20220329-C++&计算机网络

1.过程描述

动态内存-1 C++ 复制代码

```
static变量将被所有对象共享，比如可以用来记录所创建的对象数目；注意不能再类声明中初始化静态成员变量。可以在声明之外使用单独的语句进行初始化（在方法文件中）。但如果静态成员是整型或枚举型const，则可以在类声明中初始化

strlen(someString)返回的长度不包含空字符

strcpy(char* destination,const char* sourcestring)是将sourcestring复制到destination内存位置上

如果定义了构造函数，还想在创建对象时不显式地对它进行初始化，则必须显式地定义默认构造函数。
带有默认值的参数的构造函数也可以认为是默认构造函数
ClassDemo(int n=0){para=n};

复制构造函数用于将一个对象复制到新创建的对象中，其原型为
Class_name(const Class_name &);
新建一个对象并将其初始化为同类现有对象时，复制构造函数都将被调用：
假设bb为已有的StringBad对象
StringBad aa(bb);
StringBad aa=bb;
StringBad aa=StringBad(bb);
StringBad* paa=new StringBad(bb);

按值传递将调用复制构造函数，因此应该按引用传递对象，这样可以节省调用构造函数的时间以及存储新对象的空间
```

StringBad.h C++ 复制代码

```cpp
#pragma once
#ifndef STRINGBAD_H
#define STRINGBAD_H

#include <iostream>

class StringBad
{
private:
    char* str;//使用char指针来表示姓名，意味着类声明没有为字符串本身分配存储空间
    int len;
    static int num_strings;
public:
    StringBad();
    StringBad(const char* s);
    StringBad(const StringBad& st);
    //必须定义复制构造函数的原因在于，一些类成员是使用new初始化的、指向数据的指针，而不是数据本身
    StringBad& operator=(const StringBad& st);
    ~StringBad();
    friend std::ostream& operator<<(std::ostream& os, const StringBad& st);
};
#endif
```

StringBad.cpp C++ 复制代码

```cpp
#include "StringBad.h"
int StringBad::num_strings = 0;
StringBad::StringBad()
{
    len = 4;
    str = new char[4];
    std::strcpy(str, "C++");
    num_strings++;
    std::cout << num_strings << ": " << str;
}

StringBad::StringBad(const char* s)
{
    //字符串并不保存在对象中，字符串单独保存在堆内存中，对象仅
    //保存了指出到哪里去查找字符串的信息
    len = std::strlen(s);
    str = new char[len + 1];
    std::strcpy(str, s);
    num_strings++;
    std::cout << num_strings << ": " << str << "\n";
}

StringBad::StringBad(const StringBad& st)
{
    num_strings++;
    len = st.len;
    str = new char[len + 1];
    std::strcpy(str, st.str);
    std::cout << num_strings << ": " << str << "\n";
}

StringBad& StringBad::operator=(const StringBad& st)
{
    if (this == &st)//检查赋值运算符右边的地址(&st)是否与接收对象的地址相同
    {
        return *this;
    }
    else
    {
        delete[] str;//释放str指向的内存，后面把新字符串的地址赋给str
        len = st.len;
        str = new char[len + 1];
        std::strcpy(str, st.str);
        return *this;
    }
```

```
46
47  }
48  StringBad::~StringBad()
49  {
50      std::cout << str << " deleted" << "\n";
51      --num_strings;
52      std::cout << num_strings << " left\n";
53      delete[] str;//前面用了new[]
54  }
55
56  std::ostream& operator<<(std::ostream& os, const StringBad& st)
57  {
58      os << st.str;
59      return os;
60  }
```

main.cpp C++ 复制代码

```cpp
#include <iostream>
#include "StringBad.h"
using std::cout;

void callme1(StringBad&);
void callme2(StringBad);
int main()
{
    using std::endl;
    {
        cout << "Starting an innner block.\n";
        StringBad headline1("aaaaaaaaa");
        StringBad headline2("bbbbbbbbbb");
        StringBad sports("ccccccccc");

        cout << "headline1: " << headline1 << endl;
        cout << "headline2: " << headline2 << endl;
        cout << "sports: " << sports << endl;

        callme1(headline1);
        cout << "headline1: " << headline1 << endl;
        callme2(headline2);//由于callme2按值传递，所以这里会调用复制构造函数，而在函数调用完成之后对象也随之销毁
        cout << "headline2: " << headline2 << endl;

        cout << "Initialize one object to another:\n";
        StringBad sailor=sports;
        //等价于StringBad sailor=StringBad(sports),当用一个对象初始化另一个对象时，编译器将自动生成StringBad(const StringBad &)这个构造函数（称为赋值构造函数，因为它创建对象的一个副本
        //相当于StringBad sailor;sailor.str=sports.str，逐个复制非静态成员的值(这里复制的不是字符串，而是一个指向字符串的指针
        //这相当于将sailor初始化为sports后，得到的是两个指向同一个字符串的指针，这样当sailor的析构函数调用时会导致有害的结果，同时有可能试图释放内存两次（需要对复制构造函数进行深度定义

        cout << "sailor: " << sailor << endl;
        cout << "Assign one object to another:\n";
        StringBad knot;
        knot = headline1;//这里的赋值操作不创建新的对象
        //上面的sailor是一个新创建的对象，被初始化为sports的值，因此使用复制构造函数
        //这里的knot同样会出现存在两个指向同一字符串的指针的问题
        cout << "knot: " << knot << endl;
        cout << "Exiting the block.\n";
```

```
39        }
40        cout << "End of main()\n";
41        return 0;
42    }
43
44    void callme1(StringBad& rsb)
45    {
46        cout << "String passed by reference\n";
47        cout << rsb << "\n";
48    }
49
50    void callme2(StringBad sb)//按值传递或函数返回对象时，都将使用复制构造函数
51    {
52        cout << "String passed by value\n";
53        cout << sb << "\n";
54    }
```

2.结果输出

上午主要看了C++动态内存的一小部分（起太晚了），关于对象复制过程中的内存处理，事实证明真要把prime书上的代码跟每个细节都锄一遍确实太花时间了。但内存这块又感觉不太能囫囵吞枣，所以还是得老老实实地啃下去。下午跟晚上主要把计算机网络的网络层跟运输层看完的，这两块内容也是TCP/IP协议的核心，看的速度倒是不慢，但这种看小说式的阅读到底有多少能留在脑子里还真不好说，后面势必还得再复习一遍。

任务项	时间	预期	碎片时间关注	实际情况（截至29日
C++ prime	3月24–26	掌握C++基本语法知识	• The Cherno c++视频 • 计算机图形学 • 游戏引擎架构 • 即兴演讲+结构化写作	还没看完，类和动态内存分配、类继承以及STL三大块待完成
计算机网络	3月27–28	看完全书		未完成，还差应用层、网络安全、音频视频、无线网络四部分
计算机网络习题	3月29	利用习题检验并回顾		未完成
数据结构c语言版	3月30–4月3	看完全书		
数据结构与算法分析	4月4–4月10	看完全书		
TCP/IP网络编程	4月11–12	看完全书	• 数学之美 • 大图景 • 复杂性思维	
C++深度学习框架	4月13–15	看完全书		
C和指针	4月16–18	看完全书		
计算机网络自顶向下	4月19–20	看完全书		