|  |  |  |  |
| --- | --- | --- | --- |
| **学号** | 2211410212 | **成绩** |  |





**C++上机报告说明书**

**设计名称**  C++上机报告

**学 院**计算机科学与工程学院

**专 业**计算机科学与技术

**班 级**计算机2202班

**姓 名**吴佳怡

**指导教师**宋晓宇 曹阳

C++上机报告说明书

**目 录**

* 1. **上机一**

**字符串类的设计与使用**

* 1. **上机二**

**派生类的设计与使用**

* 1. **上机三**

**运算符重载**

* 1. **上机四**

**类模板的设计与使用**

**一、上机一**

**1.程序清单**

#include<iostream>

#include<string>

using namespace std;

class CTString

{

static int count;

char\* m\_pData; // 用于保存字符数据

int m\_nLen; // 记录字符长度

public:

// 构造函数和析构函数

CTString()

{

count++;

//cout << "无参构造函数的调用!" << endl;

}

CTString(char\* pDate, int len)

{

if (pDate == NULL)

{

this->m\_pData = NULL;

return;

}

this->m\_pData = new char[strlen(pDate) + 1];

strcpy(this->m\_pData, pDate);

count++;

this->m\_nLen = len;

//cout << "有参构造函数的调用!" << endl;

}

CTString(const char\* pDate, int len)

{

if (pDate == NULL)

{

this->m\_pData = NULL;

return;

}

this->m\_pData = new char[strlen(pDate) + 1];

strcpy(this->m\_pData, pDate);

count++;

this->m\_nLen = len;

//cout << "有参构造函数的调用!" << endl;

}

CTString(const CTString& string)

{

this->m\_nLen = string.m\_nLen;

this->m\_pData = new char[string.m\_nLen + 1];

for (int i = 0; i < string.m\_nLen; i++)

{

this->m\_pData[i] = string.m\_pData[i];

}

count++;

//cout << "拷贝构造函数的调用!" << endl;

}

~CTString()

{

if (m\_pData != NULL)

{

delete[] m\_pData;

m\_pData = NULL;

}

count--;

//cout << "析构函数的调用!" << endl;

}

// 其它成员函数

CTString\* Copy(CTString\* string)// 拷贝

{

this->m\_nLen = string->m\_nLen;

this->m\_pData = new char[string->m\_nLen + 1];

for (int i = 0; i < string->m\_nLen; i++)

{

this->m\_pData[i] = string->m\_pData[i];

}

//cout << "拷贝成员函数的调用!" << endl;

return this;

}

CTString\* Connect(CTString\* string) //连接

{

char\* p = new char[this->m\_nLen + string->m\_nLen];

for (int i = 0; i < this->m\_nLen; i++)

{

p[i] = this->m\_pData[i];

}

for (int i = 0; i < string->m\_nLen; i++)

{

p[i + this->m\_nLen] = string->m\_pData[i];

}

this->m\_nLen += string->m\_nLen;

delete[] this->m\_pData;

this->m\_pData = new char[this->m\_nLen + string->m\_nLen + 1];

for (int i = 0; i < this->m\_nLen; i++)

{

this->m\_pData[i] = p[i];

}

delete[]p;

//cout << "连接成员函数的调用!" << endl;

return this;

}

char\* Find(CTString\* string) // 查找子串

{

if (this->m\_nLen < string->m\_nLen)

{

return NULL;

}

int i;

for (i = 0; i < this->m\_nLen; i++)

{

for (int j = 0; j < string->m\_nLen; j++)

{

if (this->m\_pData[i + j] == string->m\_pData[j])

{

if (string->m\_pData[j + 1] == 0)

{

return string->m\_pData;

}

}

else

break;

}

}

if (i == this->m\_nLen)

{

return NULL;

}

}

char\* Find(char c) //查找字符

{

for (int i = 0; i < this->m\_nLen; i++)

{

if (this->m\_pData[i] == c)

{

return &c;

}

}

return NULL;

}

void Print() // 打印输出字符串内容

{

for (int i = 0; i < this->m\_nLen; i++)

{

cout << this->m\_pData[i];

}

cout << endl;

}

static int to\_count();

//…… // 学生自己扩充成员函数

};

int CTString::count = 0;

int CTString::to\_count()

{

return count;

}

int main()

{

char arr1[] = "hello ";

char arr2[] = "world !";

CTString str1("hello ", strlen(arr1));

CTString str2("world !", strlen(arr2));

str1.Connect(&str2);

str1.Print();

cout << str1.to\_count() << endl;

CTString str3("aabbbcccc", 9);

char arr[] = "bcc";

CTString string("bcc", strlen(arr));

if (str3.Find(&string) == NULL)

{

cout << "Not find!" << endl;

}

else

{

cout << "Find!" << endl;

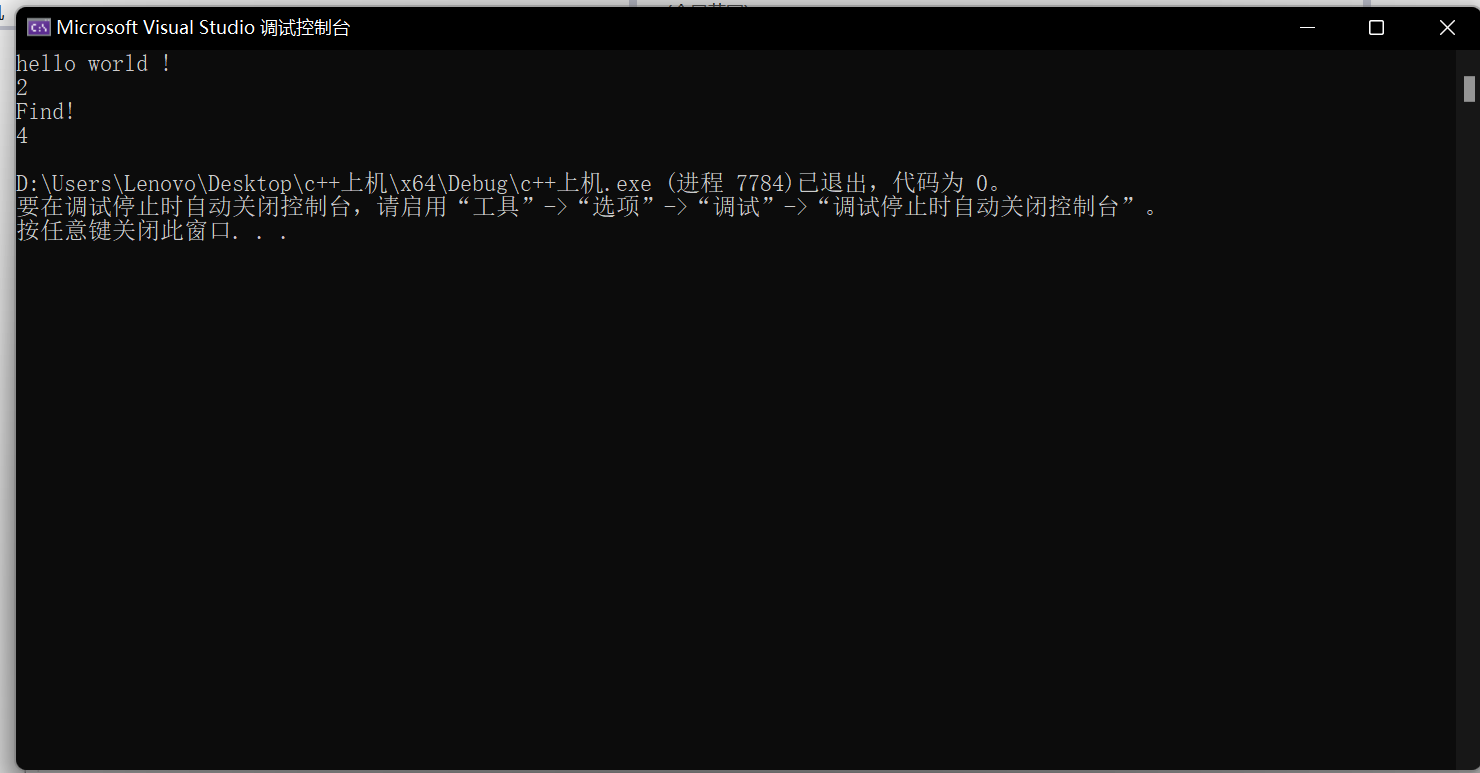
}

cout << str1.to\_count() << endl;

return 0;

}

**2.运行结果**



**二、上机二**

**1.程序清单**

#include<iostream>

using namespace std;

class CEmpoyee

{

char\* m\_pName;//姓名

int m\_nAge; //年龄

float m\_fSalary;//薪水

public:

//构造函数和析构函数

CEmpoyee(char\* pName = NULL, int age = 0, float salary = 0.0);

CEmpoyee(const CEmpoyee&);

~CEmpoyee();

//其他成员函数

void SetName(char\*);

char\* Getname();

void SetAge(int);

int GetAge();

void SetSalary(float);

float Getsalary();

void Print();

};

class CManager :public CEmpoyee

{

int m\_nlevel;//级别

public:

//构造函数和析构函数

CManager(char\* pName = NULL, int age = 0, float salary = 0.0, int nLevel = 0);

CManager(const CEmpoyee&, int);

~CManager();

//其他成员函数

void SetLevel(int);

int GetLevel();

void Print();

};

//构造函数

CManager::CManager(char\* pName, int age, float salary, int nLevel) :CEmpoyee(pName, age, salary)

{

this->m\_nlevel = nLevel;

}

//拷贝构造函数

CManager::CManager(const CEmpoyee& Empoyee, int level) :CEmpoyee(Empoyee)

{

this->m\_nlevel = level;

}

//析构函数

CManager::~CManager()

{}

//设置级别

void CManager::SetLevel(int level)

{

this->m\_nlevel = level;

}

//获取级别

int CManager::GetLevel()

{

return this->m\_nlevel;

}

//打印输出

void CManager::Print()

{

cout << "姓名：" << this->Getname() << " 年龄：" << this->GetAge() <<

" 薪资：" << this->Getsalary() << " 级别：" << this->GetLevel() << endl;

}

//构造函数

CEmpoyee::CEmpoyee(char\* pName, int age, float salary)

{

if (pName == NULL)

{

return;

}

this->m\_pName = new char[strlen(pName) + 1];

strcpy(this->m\_pName, pName);

//for (int i = 0; i <= strlen(pName); i++)

//{

// this->m\_pName[i] = pName[i];

//}

this->m\_nAge = age;

this->m\_fSalary = salary;

}

//拷贝构造函数

CEmpoyee::CEmpoyee(const CEmpoyee& empoyee)

{

this->m\_pName = new char[strlen(empoyee.m\_pName) + 1];

for (int i = 0; i <= strlen(empoyee.m\_pName); i++)

{

this->m\_pName[i] = empoyee.m\_pName[i];

}

this->m\_nAge = empoyee.m\_nAge;

this->m\_fSalary = empoyee.m\_fSalary;

}

//析构函数

CEmpoyee::~CEmpoyee()

{

if (this->m\_pName != NULL)

{

delete[] this->m\_pName;

this->m\_pName = NULL;

}

}

//设置名字

void CEmpoyee::SetName(char\* p\_name)

{

if (this->m\_pName != NULL)

{

delete[] this->m\_pName;

this->m\_pName = NULL;

}

this->m\_pName = new char[strlen(p\_name) + 1];

strcpy(this->m\_pName, p\_name);

}

//获取名字

char\* CEmpoyee::Getname()

{

return this->m\_pName;

}

//设置年龄

void CEmpoyee::SetAge(int age)

{

this->m\_nAge = age;

}

//获取年龄

int CEmpoyee::GetAge()

{

return this->m\_nAge;

}

//设置薪水

void CEmpoyee::SetSalary(float salary)

{

this->m\_fSalary = salary;

}

//获取薪水

float CEmpoyee::Getsalary()

{

return this->m\_fSalary;

}

//打印信息

void CEmpoyee::Print()

{

cout << "姓名：" << this->Getname() << " 年龄：" << this->GetAge() << " 工资：" << this->Getsalary() << endl;

}

int main()

{

CEmpoyee one;

char arr[] = "Chiyo Hoang";

char\* p = arr;

CEmpoyee two(p, 20, 13.14);

CEmpoyee three(two);

cout << sizeof(two) << endl;;

char name[] = "bule rose";

char\* ptr = name;

one.SetName(ptr);

one.SetAge(20);

one.SetSalary(12.34);

two.SetName(ptr);

one.Print();

two.Print();

three.Print();

CManager four(ptr, 14, 2345.67, 100);

CManager five(three, 200);

five.SetLevel(300);

four.Print();

five.Print();

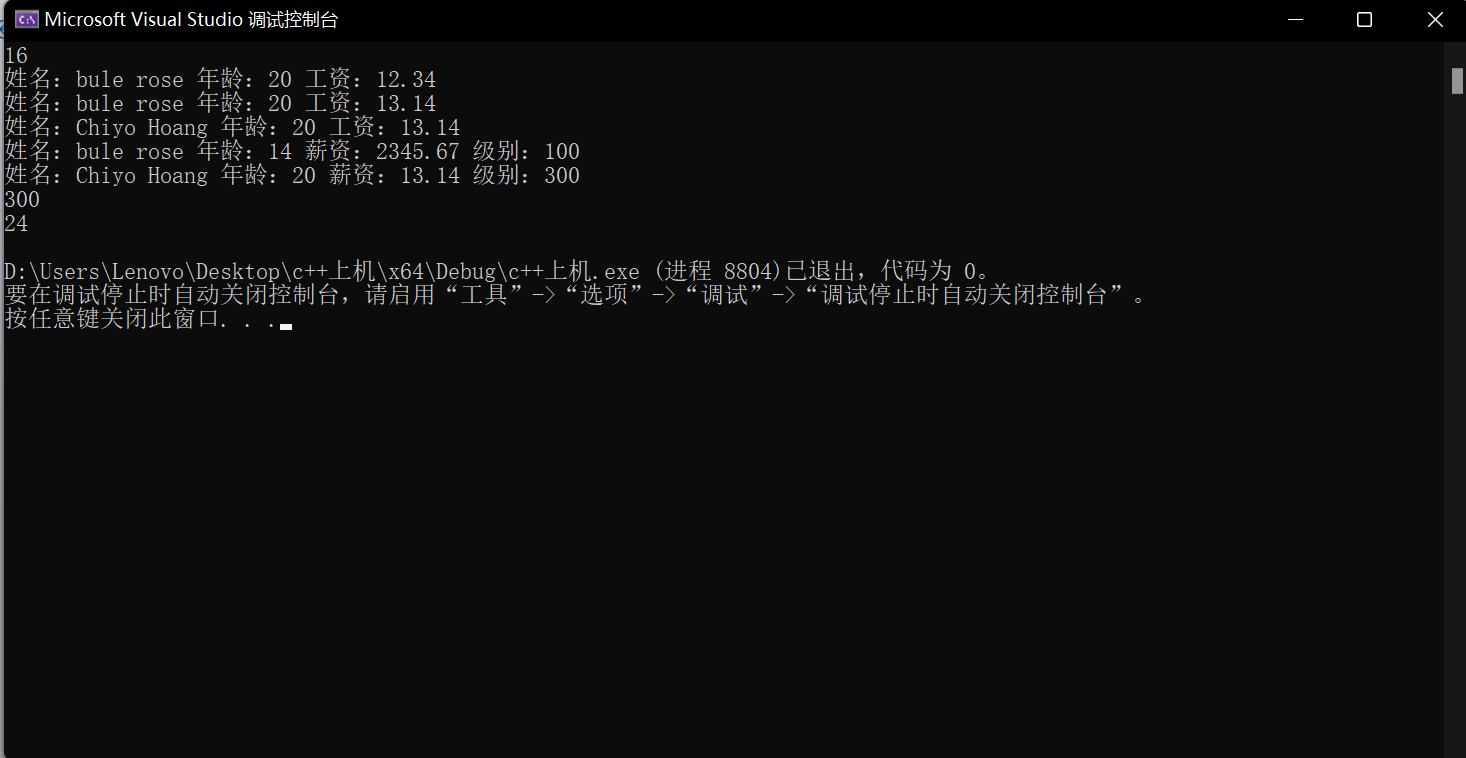
cout << five.GetLevel() << endl;;

cout << sizeof(five) << endl;

return 0;

}

**2.运行结果**



**三、上机三**

**1.程序清单**

#include<iostream>

using namespace std;

class CComplex

{

float real;

float imag;

public:

CComplex()

{

this->real = 0.0;

this->imag = 0.0;

}

CComplex(float real, float imag)

{

this->real = real;

this->imag = imag;

}

CComplex& operator=(CComplex& complex)

{

this->real = complex.real;

this->imag = complex.imag;

return \*this;

}

CComplex operator\*(CComplex& complex)

{

CComplex temp;

temp.real = this->real \* complex.real;

temp.imag = this->imag \* complex.imag;

return temp;

}

CComplex operator+(CComplex& complex)

{

CComplex temp;

temp.real = this->real + complex.real;

temp.imag = this->imag + complex.imag;

return temp;

}

CComplex operator-(CComplex& complex)

{

CComplex temp;

temp.real = this->real - complex.real;

temp.imag = this->imag - complex.imag;

return temp;

}

CComplex operator/(CComplex& complex)

{

if (complex.imag == 0 || complex.real == 0)

{

cout << "zero can't be div" << endl;

exit(0);

}

CComplex temp;

temp.real = this->real / complex.real;

temp.imag = this->imag / complex.imag;

return temp;

}

CComplex operator[](int index)

{

}

friend bool operator==(CComplex& complex1, CComplex& complex2);

friend bool operator!=(CComplex& complex1, CComplex& complex2);

void print()

{

cout << "实部为：" << this->real << " 虚部为：" << this->imag << endl;

}

};

bool operator==(CComplex& complex1, CComplex& complex2)

{

if (complex1.real == complex2.real && complex1.imag == complex2.imag)

return true;

else

return false;

}

bool operator!=(CComplex& complex1, CComplex& complex2)

{

if (complex1.real != complex2.real || complex1.imag != complex2.imag)

return true;

else

return false;

}

int main()

{

CComplex complex1;

complex1.print();

CComplex complex2(2.3, 1.3);

complex2.print();

CComplex complex3 = complex2 + complex2;

complex3.print();

CComplex complex4 = complex2 \* complex2;

complex4.print();

CComplex complex5 = complex4 - complex2;

complex5.print();

//CComplex complex6 = complex4 / complex1;

//complex6.print();

CComplex complex7 = complex4 / complex2;

complex7.print();

if (complex7 == complex2)

{

cout << "complex7 = complex2" << endl;

}

else

{

cout << "complex7 != complex2" << endl;

}

CComplex complex8(1.2, 2.3);

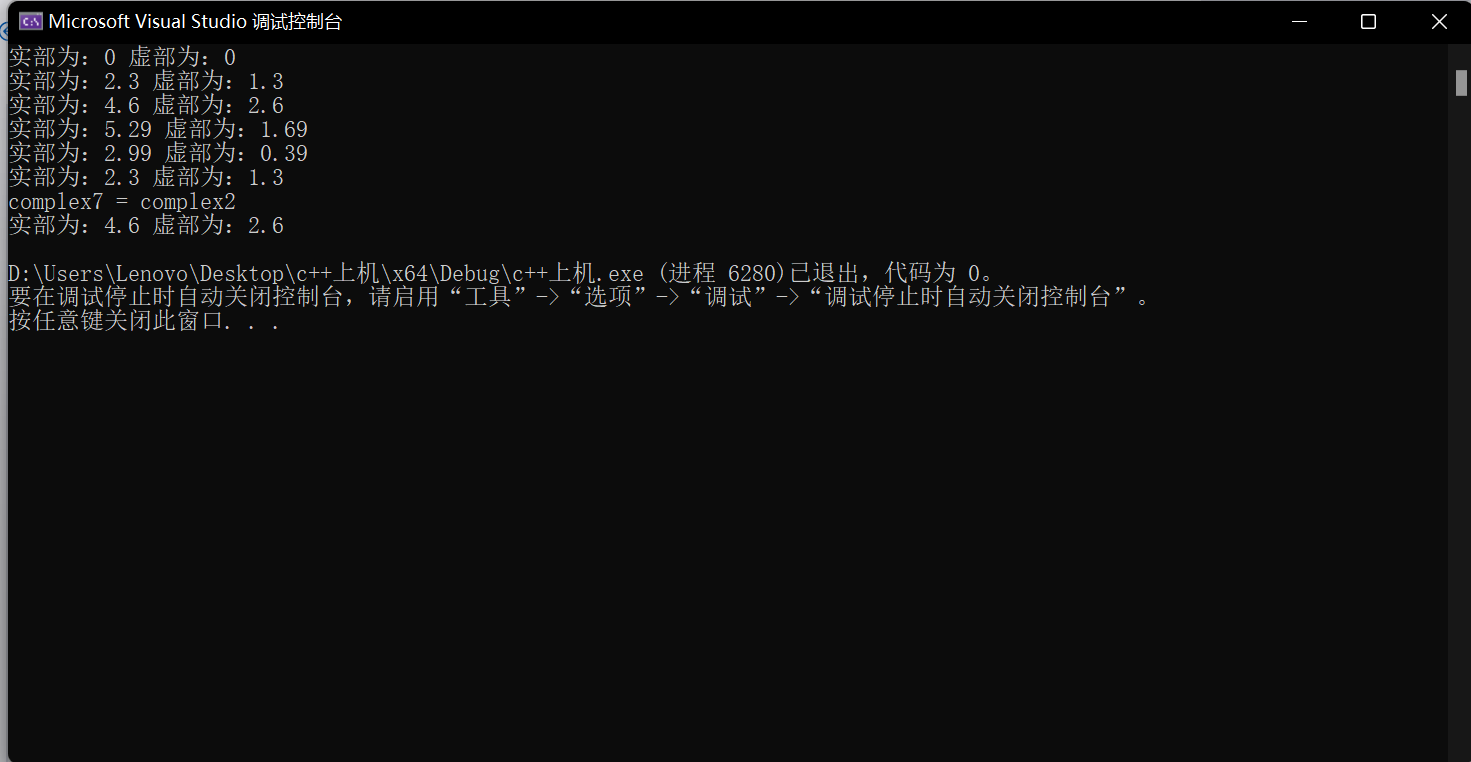
complex8 = complex3;

complex8.print();

return 0;

}

**2.运行结果**



**四、上机四**

**1.程序清单**

#include <iostream>

using namespace std;

template<class T>

class LinkStack

{

private:

struct Node // 定义链表节点结构体

{

T data; // 节点存储的数据

Node\* next; // 指向下一个节点的指针

Node(const T& d = T(), Node\* n = nullptr) : data(d), next(n) {}

};

Node\* top; // 栈顶指针

public:

LinkStack(); // 构造函数

~LinkStack(); // 析构函数

bool IsEmpty() const; // 判断栈是否为空

void Push(const T& x); // 入栈操作

T Pop(); // 出栈操作

};

template<class T>

LinkStack<T>::LinkStack() : top(nullptr) {}

template<class T>

LinkStack<T>::~LinkStack()

{

if (top != nullptr) {

Node\* p;

p = top;

top = top->next;

delete p;

}

}

template<class T>

bool LinkStack<T>::IsEmpty() const

{

return top == nullptr;

}

template<class T>

void LinkStack<T>::Push(const T& x)

{

Node\* p;

p = new Node;

p->data = x;

if (top) p->next = top;

top = p;

}

template<class T>

T LinkStack<T>::Pop()

{

Node\* p;

T i;

i = top->data;

p = top;

top = top->next;

delete p;

return i;

}

int main()

{

LinkStack<int> s;

s.Push(1);

s.Push(2);

s.Push(3);

while (!s.IsEmpty())

{

cout << s.Pop() << " ";

}

cout << endl;

LinkStack<char> m;

m.Push('a');

m.Push('b');

m.Push('c');

while (!m.IsEmpty())

{

cout << m.Pop() << " ";

}

cout << endl;

return 0;

}

**2.运行结果**

