**实验报告**

**实验1：**

（1）程序：

#include <iostream>

using namespace std;

enum CPU\_Rank{P1=1,P2,P3,P4,P5,P6,P7};

class CPU {

public:CPU(CPU\_Rank r, int f, float v);

~CPU() { cout << "destructing CPU..." << endl; }

void run() { cout << "running!" << endl; }

void stop() { cout << "stoping!" << endl; }

private:CPU\_Rank rank;

int frequency;

float voltage;

};

CPU::CPU(CPU\_Rank r, int f, float v) {

rank = r;

frequency = f;

voltage = v;

cout<<"constructing CPU..." << endl;

cout << "Rank: " << rank << endl;

cout << "Frequency: " << frequency << " MHz" << endl;

cout << "Voltage: " << voltage << endl;

}

int main() {

CPU cpu(P1, 5, 11.28);

cpu.run();

cpu.stop();

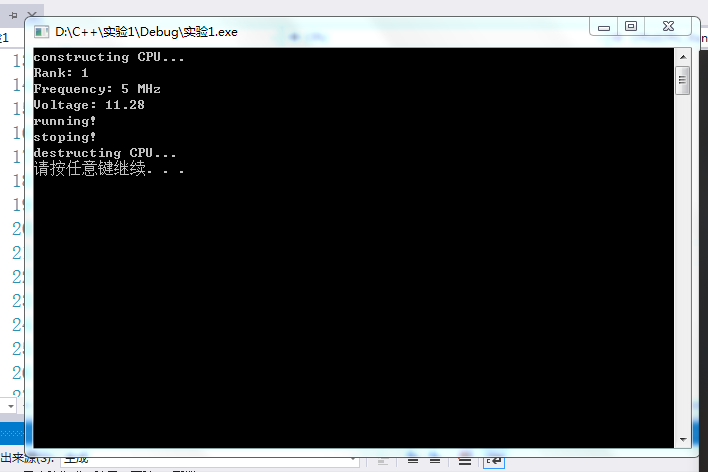
cpu.~CPU();

system("pause");

return 0;

}

（2）运行结果：



（3）该程序运用了类与对象、枚举类型enum的知识。

首先声明了枚举类型enumCPU\_Rank并为P1赋值为1，之后按顺序加1,。接着声明了CPU类，其中公有成员函数包括类的构造函数和析构函数，run、stop函数，私有成员包括枚举类型的rank，整型的frequency和浮点型的voltage。

主函数中，先声明并初始化了类的对象cpu，程序调用其构造函数，接着依次调用了run、stop函数，最后调用了其析构函数。程序结果如（2）所示。

**实验2：**

（1）程序：

#include <iostream>

using namespace std;

class CPU {

public:CPU() { cout << "constructing CPU..." << endl; }

~CPU() { cout << "destructing CPU..." << endl; }

};

class RAM {

public:

RAM() { cout << "constructing RAM..." << endl; }

~RAM() { cout << "destructing RAM..." << endl; }

};

class CDRAM {

public:

CDRAM() { cout << "constructing CDRAM..." << endl; }

~CDRAM() { cout << "destructing CDRAM..." << endl; }

};

class Computer {

public:

Computer(int cpu1,int ram1,int cdram1) {

int cpu = cpu1;

int ram = ram1;

int cdram = cdram1;

cout << "constructing Computer..." << endl; }

~Computer() { cout << "destructing Computer..." << endl; }

void run() { cout << "running!" << endl; }

void stop() { cout << "stoping!" << endl; }

private:

CPU cpu;

RAM ram;

CDRAM cdram;

};

int main() {

Computer computer(1,2,3);

computer.run();

computer.stop();

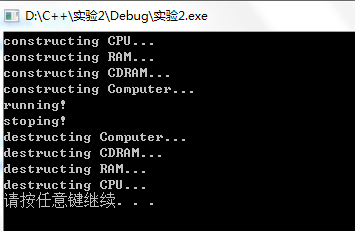
computer.~Computer();

system("pause");

return 0;

}

（2）运行结果：



（3）该实验使我复习到了类的组合的知识。在Computer类中的数据成员cpu、ram、cdram分别是类CPU、类RAM、类CDRAM的对象。因此在创建Computer类的对象computer时，不仅调用了它本身的构造函数，还调用了其内前对象的构造函数，调用顺序是Computer类中定义的顺序；同样，在调用析构函数时，也调用了内嵌对象的析构函数，调用顺序与构造函数相反，即与它们在组合类的定义中出现的次序刚好相反。

**实验3（1）：**

（1）程序：

#include<iostream>

using namespace std;

void fn1();

int x = 1, y = 2;

int main()

{

cout << "Begin..." << endl;

cout << "x=" << x << endl;

cout << "y=" << y << endl;

cout << "Evaluate x and y in main()..." << endl;

int x = 10, y = 20;

cout << "x=" << x << endl;

cout << "y=" << y << endl;

cout << "Step int fn1()..." << endl;

fn1();

cout << "Back in main" << endl;

cout << "x=" << x << endl;

cout << "y=" << y << endl;

system("pause");

return 0;

}

void fn1()

{

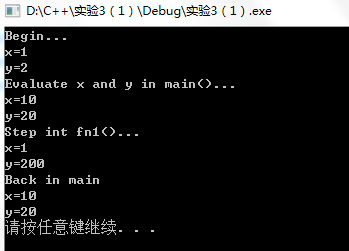
int y = 200;

cout << "x=" << x << endl;

cout << "y=" << y << endl;

}

（2）运行结果：



（3）该实验涉及到了标识符的作用域和可见性以及对象的生存期的问题。在main函数中，没有定义x和y的时候，输出的x、y为1和2，x=1，y=2为全局变量；当main函数中为x和y重新赋值为10和20后，输出的x、y为10和20，因为此时内层覆盖了外层的x、y；当调用函数fn1时，y被重新赋值为200，而x依然为全局变量x=1；最后输出的x和y为main函数中已被赋值成为10和20的x和y。

实验3（2）：

（1）程序：

client.h:

#pragma once

class CLIENT {

public:

CLIENT(double a=0, double b=0) {

age = a;

height = b;

}

double getage()const { return age; }

double getheight()const { return height; }

static void showCount();

private:

double age;

double height;

static int count;

};

client.cpp;

#include<iostream>

#include"client.h"

using namespace std;

int CLIENT::count = 1;

void CLIENT::showCount() {

cout << "Object count=" << count << endl;

}

lab5\_2.cpp:

#include<iostream>

#include"client.h"

using namespace std;

int main() {

CLIENT client(18, 180.5);

cout << "The client's age is " <<client.getage() << endl;

cout << "The client's height is " << client.getheight() << endl;

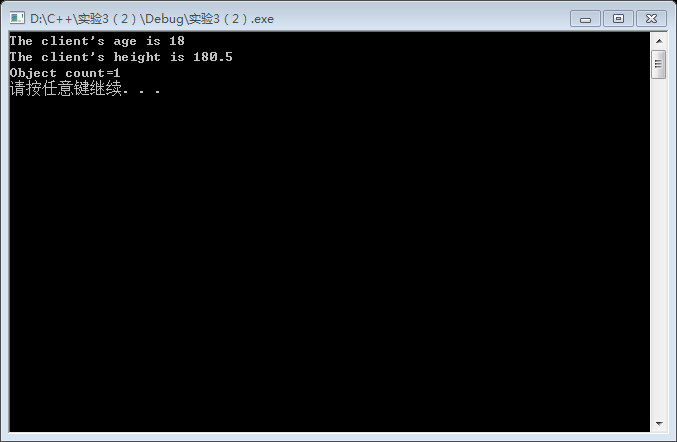
CLIENT::showCount();

system("pause");

return 0;

}

（2）运行结果：



（3）该题涉及了多文件结构和编译预处理命令的知识。首先建立了头文件client.h，并在其中定义了类CLIENT；接着建立了client.cpp文件，在该文件中实现了类CLIENT；最后在lab5\_2.cpp的main函数内测试了CLIENT类各个成员变量的结果。

（总结：在写构造函数对私有成员初始化时，不应该规定其类型，否则最后运行结果会把私有成员的值给覆盖，出现垃圾数据。）