实验一：

// 实验一-.cpp : 定义控制台应用程序的入口点。

//

#include "stdafx.h"

#include<iostream>

using namespace std;

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

class CPU {

public:

CPU(CPU\_Rank rank, int frequency, float voltage);

void run();

void stop();

~CPU() { cout << "调用析构函数" << endl; };

private:

CPU\_Rank rank;

int frequency;

float voltage;

};

CPU::CPU(CPU\_Rank rank, int frequency, float voltage)

{

cout << "调用CPU的构造函数" << endl;

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

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

cout << "voltage:" << voltage << "V"<<endl;

}

void CPU::run() { cout << "run函数" << endl; }

void CPU::stop() { cout << "stop函数" << endl; }

int main()

{

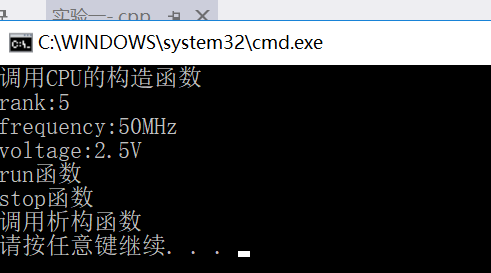
CPU cpu(P5, 50, 2.5);

cpu.run();

cpu.stop();

return 0;

}



因为我在构造函数体中写了cout语句，所以能显示类的对象的各参数数值，并且类的对象的创造、函数实现、对象删除过程中，是先调用构造函数，再调用类的相关成员函数，最后调用析构函数，这与运行结果一致。

这一实验让我复习了静态生存期、enum枚举类型、类的声明，对类的对象建立过程中的赋值过程有了更具体的认识。

实验二：

// 实验二.cpp : 定义控制台应用程序的入口点。

//

#include "stdafx.h"

#include<iostream>

using namespace std;

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

class CPU {

public:

CPU() { cout << "调用CPU默认构造函数" << endl; }

CPU(CPU\_Rank rank, int frequency, float voltage);

void run();

void stop();

~CPU() { cout << "调用CPU析构函数" << endl; };

private:

CPU\_Rank rank;

int frequency;

float voltage;

};

CPU::CPU(CPU\_Rank rank, int frequency, float voltage)

{

cout << "调用CPU的构造函数" << endl;

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

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

cout << "voltage:" << voltage << "V" << endl;

}

void CPU::run() { cout << "run函数" << endl; }

void CPU::stop() { cout << "stop函数" << endl; }

class RAM {

public:

RAM() { cout << "调用RAM构造函数" << endl; }

~RAM() { cout << "调用RAM析构函数" << endl; }

};

class CDROM {

public:

CDROM() { cout << "调用CDROM构造函数" << endl; }

~CDROM() { cout << "调用CDROM析构函数" << endl; }

};

class COMPUTER {

public:

COMPUTER(CPU cpu, RAM ram, CDROM cdrom);

void run() { cout << "COMPUTER::run" << endl; }

void stop() { cout << "COMPUTER::stop" << endl; }

~COMPUTER() { cout << "调用COMPUTER析构函数" << endl; }

private:

CPU cpu;

RAM ram;

CDROM cdrom;

};

COMPUTER::COMPUTER(CPU cpu, RAM ram, CDROM cdrom) :cpu(), ram(), cdrom() {

cout << "调用COMPUTER构造函数" << endl;

}

int main()

{

CPU cpu(P5,50,2.5);

RAM ram;

CDROM cdrom;

cout << "创建computer对象" << endl;

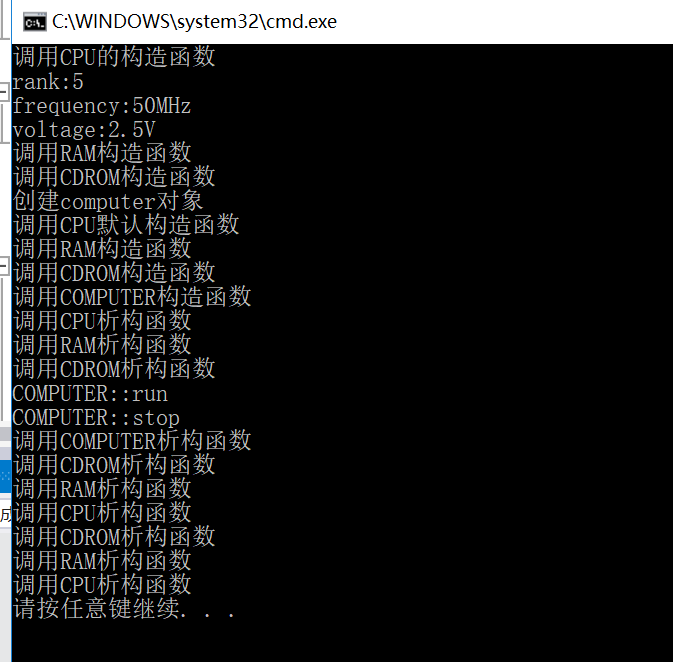
COMPUTER computer(cpu,ram,cdrom);

computer.run();

computer.stop();

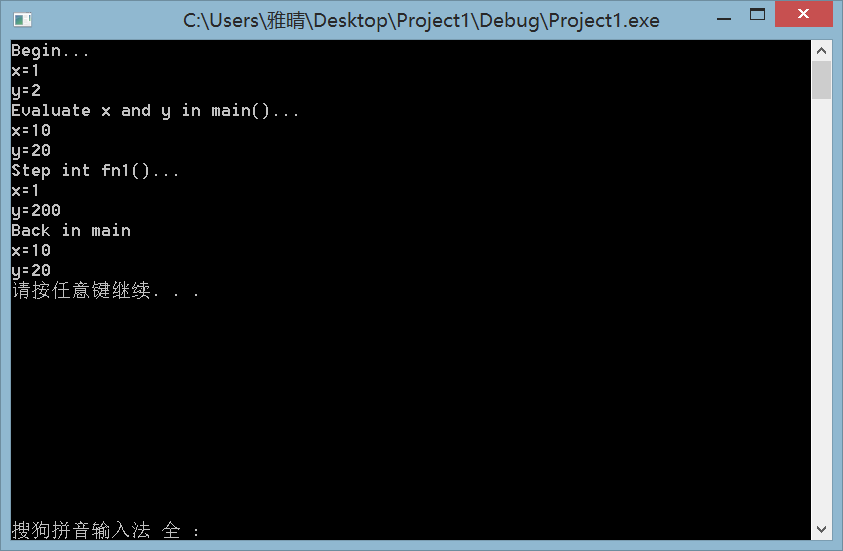
return 0;

}



通过实验二，我重新类的组合，类的组合中构造函数应该如何声明，构造函数的调用情况又是如何的。这样的观察加深了我对类的组合内嵌对象构造函数调用的理解。

实验三：（1）



解释：最开始的x和y是全局变量，所以输出的值分别为1、2。

第二个x和y在main函数中重新声明了一次，因此之前的x、y在函数中不可见，所以输出的值为10,20。

然后调用fn1函数，输出了值为1的全局变量x和值为200的局部变量y。

最后由于fn1中的变量具有动态生存期，随着函数返回而失效，所以输出的x、y依然是main函数中声明的变量，值为10、20。

（2）

//client.h

#ifndef CLIENT\_H

#define CLIENT\_H

class CLIENT

{

public:

CLIENT() {}

CLIENT(int id, int age);

~CLIENT() {}

int getid() const;

int getage() const;

static int gettotal();

void show() const;

private:

int id;

int age;

static int total;

};

#endif // CLIENT\_H

//client.cpp

#include"stdafx.h"

#include "client.h"

#include <iostream>

using namespace std;

int CLIENT::total = 0;

CLIENT::CLIENT(int id, int age) :id(id), age(age){

cout << " #" << id << " is created" << endl;

total++;

}

int CLIENT::getid() const { return id; }

int CLIENT::getage() const { return age; }

int CLIENT::gettotal() { return total; }

void CLIENT::show() const {

cout << "id#" << id << " age:" << age << endl;

}

// lab3-2.cpp : 定义控制台应用程序的入口点。

#include"stdafx.h"

#include "client.h"

#include <iostream>

using namespace std;

int main()

{

CLIENT client1(992231,23);

CLIENT client2(112233,33);

CLIENT client3(990022,19);

cout << client1.getid() << endl;

cout << client2.getage() << endl;

cout << client2.gettotal() << endl;

cout << client3.gettotal() << endl;

client3.show();

return 0;

}

