实验报告（二）

#### 一、实验目的

(1) 掌握类的声明和使用

(2) 掌握类的声明和对象的声明

(3) 复习具有不同访问属性的成员的访问方式

(4) 观察构造函数和析构函数的执行过程

(5) 学习类的组合使用方法

(6) 观察程序运行中变量的作用域、生存期和可见性。

(7) 学习类的静态成员的使用。

(8) 学习多文件结构在C++程序中的使用。

(9) 学会使用Visual Studio 2008 开发环境中的debuge调试功能：单步执行、设置断点、观察变量值。

#### 二、实验内容

#### 声明一个简单的Computer类，有数据成员芯片(cpu)，内存（ram），光驱（cdrom）,等，有两个共有成员函数run.stop.cpu为CPU类的一个对象，ram为RAM类的一个对象，cdrom为CDROM类的一个对象，声明并实现这个类。

#### 三、实验环境

硬件环境：AMD A12；操作系统：Windows 10；C++环境：Visual Studio 2017

#### 四、程序设计原理和方法

##### 1、以实验一为基础。

CPU类和枚举类型CPU\_RANK延续了实验一的大体的程序，此设计已经在实验一的实验报告中叙述得很详尽，故在此处不再叙述啦。

##### 2、新增对象设置。

根据题意定义了新的类RAM和Cdrom，由于题目没有关于这两个类更多的信息，所以在这两个类中，只写了最简单的构造函数和析构函数。同实验一的设计思路，为了能在输出结果时明确调用了哪些成员函数，所以在成员函数编译时，都使用cout，让其在调用时输出该函数的名称。

##### 3、Computer类的定义。

由于Computer类涉及类的组合（Computer与CPU、RAM、Cdrom），所以在构造函数的地方与实验一有所不同（需要同时初始化没有默认构造函数的CPU），采用Computer::Computer(CPU cpu1) :cpu(cpu1) { cout << "Computer的构造函数"<<endl; }

#### 4、主函数的定义

由于实验二的目的是实现Computer的对象，所以在主函数中定义Computer的对象computer，让其调用自身的成员函数，观察构造函数、成员函数和析构函数的调用情况。

#### 五、程序代码

#include "pch.h"

#include <iostream>

using namespace std;

enum CPU\_RANK { P1 = 1, P2, P3 = 5, P4, P5 };//union是每个数据公用内存；struct和enum不共用内存，而是每个都会占用内存。union和enum每次只能出现一次，而struct可以同时出现许多值

//若给了P1=1，则往后都默认加一。除非自己赋值了

class CPU {

public:

CPU() {

cout << "CPU的构造函数1" << endl;

}

CPU(CPU\_RANK rank, int frequency, float voltage) {

this->rank = rank;

this->frequency = frequency;

this->voltage = voltage;

cout << rank << endl;

cout << frequency << endl;

cout << voltage << endl;

cout << "CPU的构造函数2" << endl;

}

void run();

void stop();

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

private:

CPU\_RANK rank;

int frequency;

float voltage;

};

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

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

class RAM {

public:

RAM() { cout << "RAM的构造函数" << endl; }

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

};

class Cdrom {

public:

Cdrom() { cout << "Cdrom的构造函数" << endl; }

~Cdrom() { cout << "Cdrom的析构函数" << endl; }

};

class Computer {

public:

Computer(CPU cpu1);

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

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

~Computer() { cout << "Computer的析构函数" << endl; }

private:

Cdrom cdrom;

RAM ram;

CPU cpu;

};

Computer::Computer(CPU cpu1) :cpu(cpu1) { cout << "Computer的构造函数"<<endl; }

int main() {

CPU c(P4, 5, 3.5);

Computer computer(c);

computer.run();

computer.stop();

return 0;

}

#### 六、运行结果

6

5

3.5

CPU的构造函数2

Cdrom的构造函数

RAM的构造函数

Computer的构造函数

CPU的析构函数

Computer:run函数

Computer:stop函数

Computer的析构函数

CPU的析构函数

RAM的析构函数

Cdrom的析构函数

CPU的析构函数

#### 七、解释运行结果

“6”、“5”、“3.5”和“CPU的构造函数2”是声明CPU的对象cpu的时候调用了CPU的构造函数的结果，其中：出现6是因为以enum CPU\_RANK中的P4赋值给了rank，同理对于frequency和voltage。

“Cdrom的构造函数”，“RAM的构造函数”和“Computer的构造函数“是因为在调用Computer的构造函数的时候先调用内嵌成员的构造函数，最后调用本类成员的构造函数。且调用顺序根据本类中内嵌对象的声明顺序，先是Cdrom，然后是RAM。

“CPU的析构函数”是因为赋值给computer之后，cpu的使命便结束了，就析构了。

“Computer:run函数”“Computer:stop函数”是在调用computer的成员函数。

最后，computer对象使命结束，它的析构函数的调用是与构造函数时的顺序是完全相反的。

但是我不知道为什么CPU最终还会被释放一次。这是为什么呢？

#### 八、实验小结

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。在观察构造函数的调用顺序的时候，我使用了设置断点和逐语句的方法，清楚地看到了类的定义和主函数之间的联系，成员函数是如何被调用的。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。