程序设计基础小班

饶淙元 2020年10月16日

1.面向对象的设计思想

1.1把大象塞进冰箱里

如果要设计一个流程把大象塞进冰箱里，你会怎么做?

面向过程

面向过程(Procedure Oriented)是一种对多数人来说比较自然的设计方法:

1. 打开冰箱门
2. 把大象塞进冰箱
3. 关上冰箱门

如果再把大象2塞进冰箱里?

1. 打开冰箱门
2. 把大象2塞进冰箱
3. 关上冰箱门

再把大象从冰箱取出来，并塞进冰箱2?

1. 打开冰箱门
2. 把大象取出冰箱
3. 关上冰箱门
4. 打开冰箱2门
5. 把大象塞进冰箱2
6. 关上冰箱2门

以此类推，当我们有m头大象，n台冰箱，根据实际需求需要用若干种方法塞进/取出时，反复的ctrl+c和ctrl+v将让你极为头疼。

资源打通的面向过程

当然，你也可以避免反复开关冰箱门，让问题简单一点。

1. 打开所有冰箱门
2. 完成所有大象操作
    1. 将大象x从冰箱a中取出
    2. 将大象y从塞进冰箱b中
    3. 将大象z从冰箱c转移到冰箱d
    4. 继续循环直到完成所有操作
3. 关闭所有冰箱门

而同时打开所有冰箱门，让所有冰箱对所有大象随时开放，无疑会消耗不少的电力与注意力，如果你在编程时这么做，满篇全据局变量和高内存占用可能让你的程序充满风险。

面向对象

面向对象(Object Oriented)则是一些做事考虑周全，更有条理的人可能想到的方法。

既然我们要反复塞进大象(注意是反复，考虑下如果你只需要放入一次大象是否适合这么做？)，那么我们考虑冰箱可以有放入大象这样一个基本操作，这是一个一气呵成的过程，无需多做说明。这个操作不妨起名为“放入”，那么对于任何给定大象，我们都有以下过程：

1. 打开冰箱门
2. 塞进指定大象
3. 关上冰箱门

实际上放入大象是一个具体的大象，如放入（大象1），而这个所谓的“放入”不单单是对大象进行空间位移，还包括开门、关门，甚至是检查冰箱是否满了、制冰是否正常等一系列操作，因此我们把它归入冰箱的专有操作，所有的冰箱都可以有这个操作，而我们实际上要放入一个特定的冰箱，如冰箱1，那么调用时用`冰箱1.放入(大象1)`这样表示。同样的也可以有`冰箱1.取出(大象1)`。这种构造对所有冰箱所有大象都有效的过程，就是一种抽象的思想。

实际上我们可能不只放大象，还会放任何动物，比如放个猴子放头牛陪陪大象什么的，因此我们希望这个放入对任何动物都有效，大象不过是动物的一种；再者大象在里面可能会饿，我们可能放些食物，比如草、苹果什么的，而这些食物和动物都可以叫做东西，我们的放入操作实际上是对东西都有效的。那么在这个设想下，大象和猴子都是来自自动物这个基本类型，动物和食物都来自东西，只要我们约定东西可以被放入冰箱了，那么来自它的动物食物以及来自动物的大象猴子都可以被放进冰箱，这就是继承。

大象可能并不想被放进冰箱里，也许把它放进动物园是个不错的选择，我们把冰箱和动物园都叫做容器，那么可以上升到容器.放入(xx)，容器.取出(xx)。但不同容器的放入取出可能不一样，因此我们或许需要根据具体的容器实现不同的放入操作，这即是多态的一种表现。

1.2面向对象编程

面向对象编程(Object Oriented Programming)即是用上面的思想进行编程，而电子系的程设课中则是选择用C++进行OOP，因此 用C++OOP $\in$ OOP $\neq$ 面向对象

C++将pop改为oop是一种重要的重构技术，学会这一点，不一定有高技术，但一定有高分数:D

例：编程实现上面的设计内容

（演示内容：面向过程实现、OOP实现、virtual的使用）

2.期末考试

期末考试分填空、补全、写输出

2.1 填空

填空题主要考察课件上的原题，这部分随缘复习，不建议作为重点，我根据去年考题给出了示例。

注：期末考试的权重没有想象的那么大，一般地说期末考试不会阻拦你拿A，但是会阻拦你拿A+。

2.2 代码补全

这里给出的主要基本和课件上的例子一样，常让写声明语句，把握要点即可（关键字、作用域、变量类型、~~书写规范~~）。

例子

```
class A
{
    static const int f(int a = 3, int b = 4);
};

const int A::f(int a, int b)  // 这行注释符号左边是一个大填空
{
    return a + b;
}
```

2.3 写输出

此处主要涉及到类的构造析构顺序、文件操作等

做好的办法：大脑当处理器过一遍过程，确保对于老师课件上的代码可以顺利读出输出。

注：电子程设不考错误代码，因此所有代码都应该是可以编译的，且理想情况下没有UB，如果自己测试记得不要开优化，降低报警等级

关于构造析构系列的内容，基于以下基本架构进行演示。

```
class A
{
public:
    int a;
    A(int a = 0) :a(a)
    {
        cout << "constructor A " << a << endl;
    }
    A(const A& p)
    {
        a = p.a;
        cout << "copy constructor A " << a << endl;
    }
    ~A()
    {
        cout << "destructor A " << a << endl;
    }
    A& operator=(const A& p)
    {
        a = p.a;
        cout << "assign to A " << a << endl;
        return *this;
    }
};

class B : public A
{
```

```
public:
    int b;
    B(int a = 0, int b = 0) :A(a), b(b)
    {
        cout << "constructor B:" << a << ',' << b << endl;
    }
    B(const B& p)
    {
        a = p.a;
        b = p.b;
        cout << "copy constructor B:" << a << ',' << b << endl;
    }
    ~B()
    {
        cout << "destructor B:" << a << ',' << b << endl;
    }
    B& operator=(const B& p)
    {
        a = p.a;
        b = p.b;
        cout << "assign to B " << a << ',' << b << endl;
        return *this;
    }
};
```

（演示内容：临时变量返回、构造函数不同形式、对象构造与析构）

```
B f()
{
    B b2{ 1,2 };
    return b2;
}

int main()
{
    B b1 = B(3, 4);
    b1 = f();
    b1.a = 5;
    return 0;
}
```