## 前言

面向过程，不停修改的是一个一个的方法；

面向对象，不停修改的是一个一个的类；

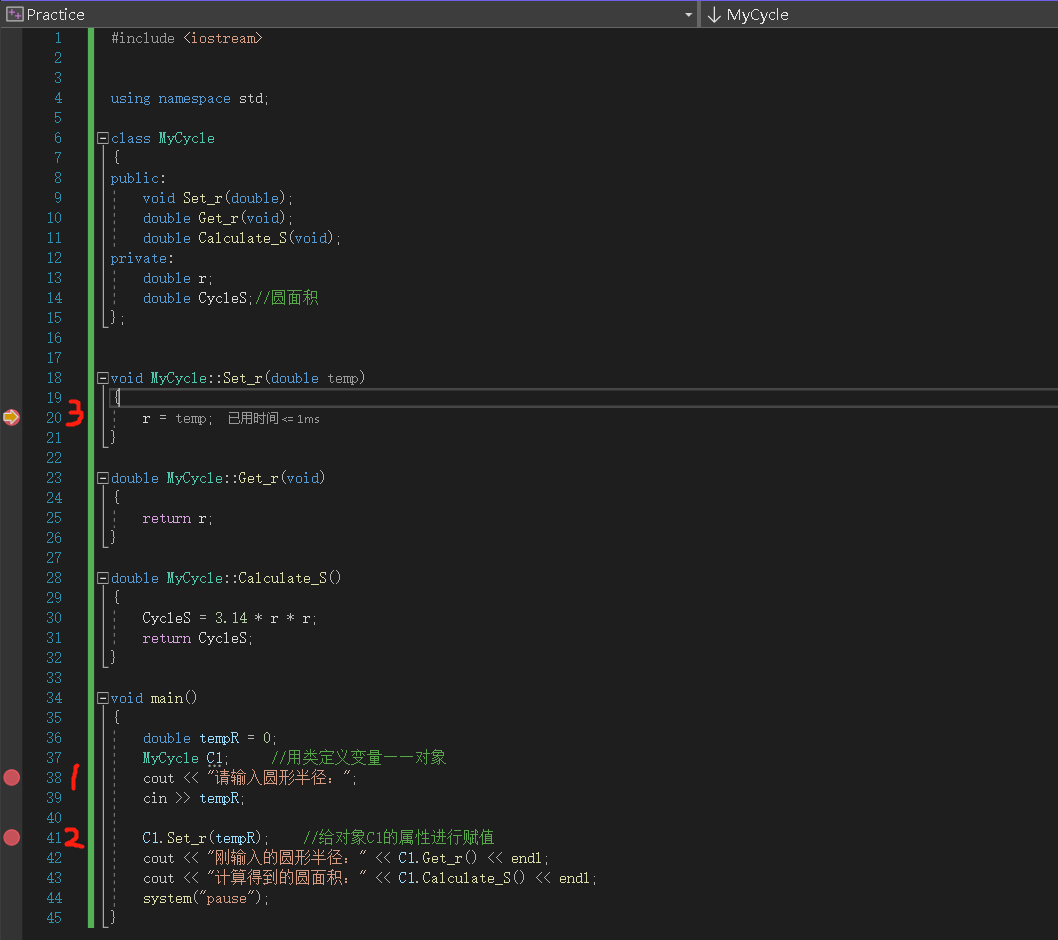
类是一个数据类型（是固定大小内存块的别名）；所以定义一个类，是一个抽象的概念，编译器并不会给用户分配内存。只有用数据类型定义变量的时候，才会分配内存

## 一、C++新概念

### 实例化

实例化：用类定义变量（得到对象）

|  |
| --- |
| #include <iostream>  using namespace std;  class MyCycle  {  public:  void Set\_r(double);  double Get\_r(void);  double Calculate\_S(void);  private:  double r;  double CycleS;//圆面积  };  void MyCycle::Set\_r(double temp)  {  r = temp;  }  double MyCycle::Get\_r(void)  {  return r;  }  double MyCycle::Calculate\_S()  {  CycleS = 3.14 \* r \* r;  return CycleS;  }  void main()  {  double tempR = 0;  MyCycle C1; //用类定义变量——对象  cout << "请输入圆形半径：";  cin >> tempR;  C1.Set\_r(tempR); //给对象C1的属性进行赋值  cout << "刚输入的圆形半径：" << C1.Get\_r() << endl;  cout << "计算得到的圆面积：" << C1.Calculate\_S() << endl;  system("pause");  } |
|  |



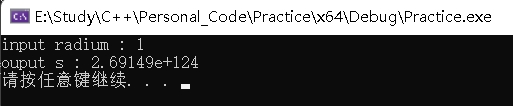
从调试过程可以看出，按步骤执行时可以看出，会发现C++程序跳转和C程序跳转并不一致，C++通过调用对象的类的成员函数进行追踪。（而C是一步一步进行追踪指向）

思考：C++编译器是如何处理多个对象，调用类的成员函数？（如何区分是哪个成员调用的哪个方法？）

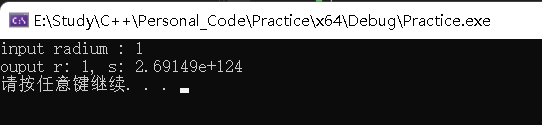
### 易犯错模型

案例一：

|  |
| --- |
| #include <iostream>  class cycle  {  public:  double r;  double pi = 3.14159;  double s = pi \* r \* r;  };  using namespace std;  int main(void)  {  cycle c1;  cout << "input radium : " ;  cin >> c1.r;  cout << "ouput s : " << c1.s << endl;  system("pause");  return 0;  } |



输出这个结果的原因，正是因为没有编写成员函数，用户在对类进行实例化（初始化）时，对象中的成员s已经根据原来存放在内存空间中的垃圾值r，计算得出。并不是在只有用户输入r后，才去计算得出s；调用c1.s时，只是从变量所标识的内存空间中拿值，并没有重新执行计算。（可以把半径也输出，以证明）



和成员函数的调用现象。原理完全不同！！！

### 程序设计方法的发展历程

#### 3.1、面向过程的结构化程序设计方法

·设计思路

* 自顶向下，采用模块分解与功能抽象

·程序结构

* 按功能划分为若干个基本模块，形成一个树状结构
* 各模块间的关系尽可能简单，功能上相对独立，每一个模块内部均是由顺序、选择和循环三种基本结构组成
* 其模块化实现的具体方法是使用子程序

·优点

* 有效地将一个较为复杂的程序系统设计任务分解成许多易于控制和处理的子任务，便于开发和维护

·缺点

* 可重用性差，数据安全性差、难以开发大型软件和图形界面等应用软件
* 把数据和处理数据的过程分离为相互独立的实体
* 当数据结构改变时，所有相关的处理过程都要进行相应的修改
* 每一种相对于老问题的新方法，都要带来额外的开销
* 图形化用户界面的应用程序，很难用过程来描述和实现，开发和维护也都很困难

#### 3.2、面向对象的方法

·特性

* 将数据及数据的操作方法封装到一起，作为一个相互依存，不可分离的整体——对象
* 对同类型对象抽象出其共性，形成类
* 类通过一个简单的外部接口，可以与外界发生关系
* 对象与对象之间通过消息进行通信

·对象

* 面向对象方法中的对象：是系统中用来描述客观事物的一个实体，它是用来构成系统的一个基本单位。对象由一组属性（用来描述对象静态特征的数据项）和一组行为（用来描述对象动态特征的操作序列）构成；

·类

* 面向对象方法中的类：具有相同属性和服务的一组对象的集合；为属于该类的全部对象提供了抽象的描述，包括属性和行为两个主要部分；

·类与对象的关系：犹如模具与铸件之间的关系，一个属于某类的对象称为该类的一个实例

·封装

* 也就是把客观事物封装成抽象的类，并且类可以把自己的数据和方法只让可信的类或者对象操作，对不可信的进行信息隐藏
* 把对象的属性和服务结合成一个独立的系统单元
* 尽可能隐藏对象的内部细节，对外形成一个边界（或者说一道屏障），只保留有限的对外接口，使之与外部发生联系
* **继承**对于软件复用具有重要意义，是面对对象技术能够提高软件开发效率的重要原因之一（特殊类的对象拥有一般类的全部属性与服务，称作特殊类对一般类的继承）
* **多态**是指一般类中定义的属性或行为，被特殊类继承之后，可以具有不同的数据类型或表现出不同的行为。这使得同一个属性或行为在一般类及其各个特殊类中具有不同的语义

#### 3.3、C语言和C++语言的关系

C语言是在实践过程中逐步完善起来的：没有深思熟虑的设计过程、使用时存在很多“灰色地带”、残留量过多低级语言的特征、直接利用指针进行内存操作，但高效；（当面向过程方法论暴露问题越来越多哦时，业界开始考虑往工程项目中引入面向对象的设计方法）

C++和C语言不是对立的竞争关系，C++是以C为基础，完全兼容C语言的特性，是C语言的加强，是一种更好的C。（类型加强、函数加强、异常处理）

学习C++并不会影响原有的C知识，相反会更加加深对C的认知。

学习C++可以接触到更多的软件设计方法，并带来更多机会。

### C++命名空间

所谓的namespace，是指标识符的各种可见范围。C++标准程序库中，所有标识符都被定义在一个名为std的namespace中。

<iostream>和<iostream.h>格式不同，而且是不同的两个文件，代码也不同。后缀为.h的头文件C++标准已明确提出不支持。

1. 当使用<iostream.h>，相当于在c中调用库函数，使用的是全局命名空间，也就是早期的C++实现。
2. 当使用<iostream>的时候，该头文件没有定义全局命名空间，必须使用namespace std，这样才能正确使用 cout 等标准输入输出函数。
3. 由于namespace的概念，使用C++标准程序库的任何标识符时，有三种选择：①直接指定标识符（如：std::cout）②使用using关键字（声明using std::cout; 后便可直接使用cout）③最方便则是使用 using namespace std（使std内所有标识符都曝光，则可使用全部标识符）
4. 由于std库非常庞大，程序员在选择类的名称或函数名时，很有可能和标准库中的某个名字相同。所以为了避免这种情况所造成的名字冲突，就把标准库中的一切放在命名空间std中。

在C++中，名称（name）可以是符号常量、变量、宏、函数、结构、枚举、类和对象等，为了避免在大规模程序设计中，以及在程序员使用各种各样的C++库时，这些标识符的命名发生冲突，标准C++引入关键字namespace（命名空间/名称空间），可以更好地控制标识符的作用域。

C语言中的命名空间：只有一个全局作用域，C语言中所有的全局标识符共享一个作用域，标识符之间很容易发生冲突。

C++中提出了命名空间的概念：命名空间将全局作用域分成不同的部分，不同命名空间中的标识符可以同名而不发生冲突，命名空间可以相互嵌套，全局作用域可叫默认命名空间

C++命名空间的定义：

|  |
| --- |
| namespace name{…} |

C++命名空间的使用：

|  |
| --- |
| using namespace name;//使用整个命名空间  using name::variable;//使用命名空间中的变量  ::variable;//使用默认命名空间中的变量  //默认情况下可以世界使用默认命名空间中的所有标识符 |

如：由于iostream没有引入标准的std，如果不声明 using namespace std，而后调用标识符时，需要显示引入std；

如果同时使用（简便方式表示）同名标识符时，编译器会提示“指定不明确”

|  |
| --- |
| #include <iostream>  namespace testA {  int a = 0;  }  namespace testB {  int a = 1;  namespace testC {  struct MyStruct  {  char name[10];  int age;  };  }  }  int main(void)  {  using namespace testA;  using namespace testB;  std::cout << "a = " << a << std::endl;//"a不明确符号"  std::cout << "a = " << testB::a << std::endl;  testB::testC::MyStruct mystruct;  mystruct.age = 10;  system("pause");  } |

C++标准为了和C区别开，也为了正确使用命名空间，规定头文件不使用后缀.h