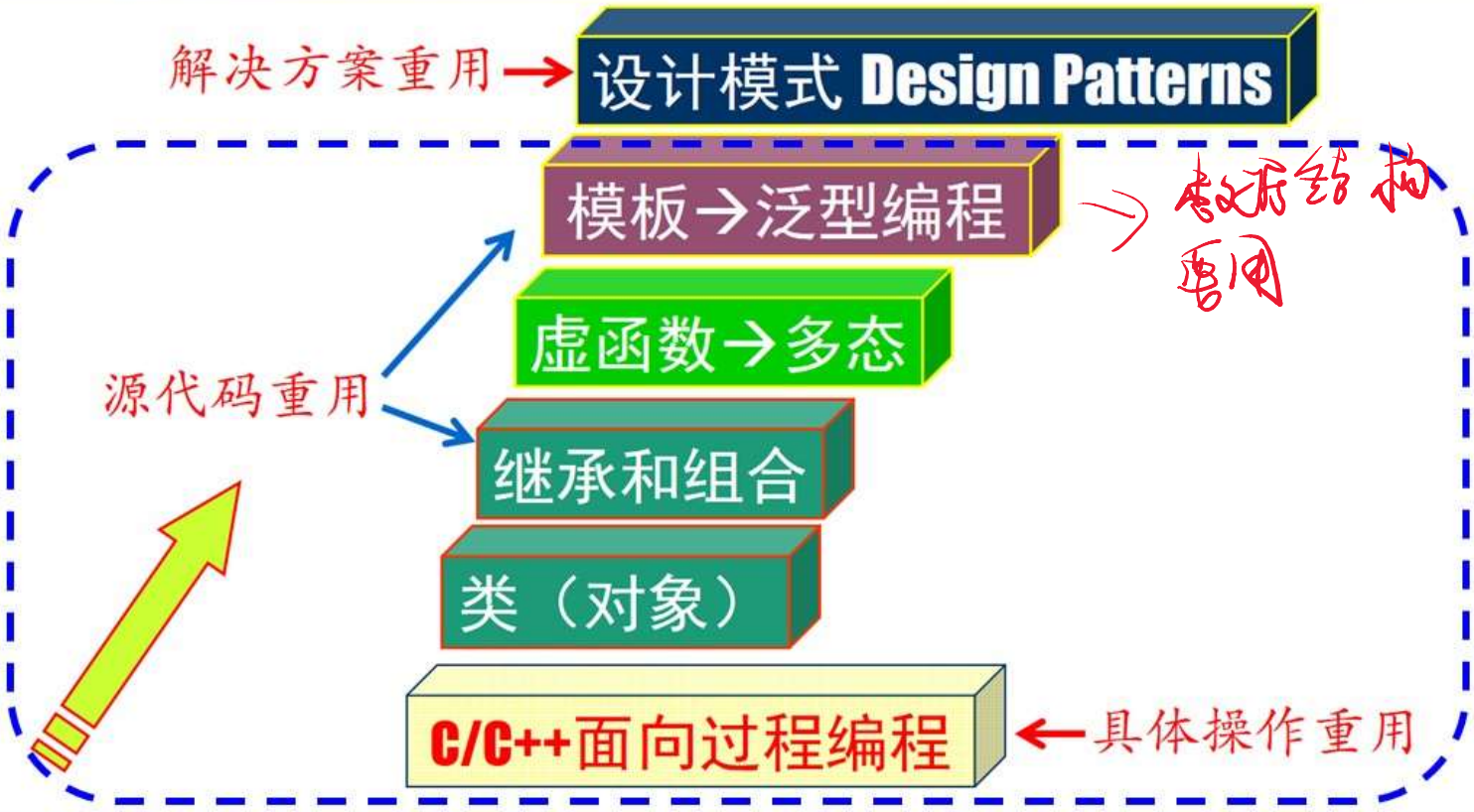
C++内容中的几个台阶


解决方案重用
设计模式 Design Patterns
模板→泛型编程
虚函数→多态
源代码重用
继承和组合
类（对象）
C/C++面向过程编程
具体操作重用

第二章 C++语言简单程序设计

- C++语言概述（与C、Java、Python等的关系）
- 基本数据类型（常量、变量[整型、浮点型、字符型、字符串]）
- 运算符与表达式（算术运算符、关系运算符、逻辑运算符、位运算符、赋值运算符、条件运算符、逗号运算符）
- 数据的输入与输出（cin、cout）
- 算法基本控制结构（顺序、选择、循环）

第三章 函数

- 函数的定义、声明和调用
- 函数间的参数传递（值传递、指针传递、引用传递）
- 内联函数（与宏定义的区别）
- 带默认参数值的函数（默认形参值必须自右向左连续地定义）
- 函数重载（形参个数不同或形参类型不同）
- C++系统函数

- 函数重载：同名函数的不同实现
  - 至少有一个函数参数的类型有区别；
  - 返回值、参数名称不能作为区分标识。

- 形参缺省值
  - 自动设置参数值：可不提供实参
  - 位置：在无缺省值参数后
  - 二义性：可能与函数重载冲突，导致编译错误

类与对象

第四章 类与对象

- 面向对象的基本概念、类和对象的区别与联系
- 构造函数负责对象的"生"，析构函数负责对象的"死"
  - **How(程序员决定) 和 When(编译器决定)**
  - **理解构造函数和析构函数的作用和调用过程**
- 复制构造函数：对象之间的拷贝复制（语法，**调用情形P111**）
- 右值引用：延长临时对象的生命周期
- 移动构造函数：偷临时对象的资源，避免频繁的拷贝
- 类组合（语法、构造函数与析构函数调用过程）

- 对象
  - 属性数据(静态特征) + 特定操作(动态特征)
  - 状态记忆(数据成员) + 活动能力(函数成员)

没有提到

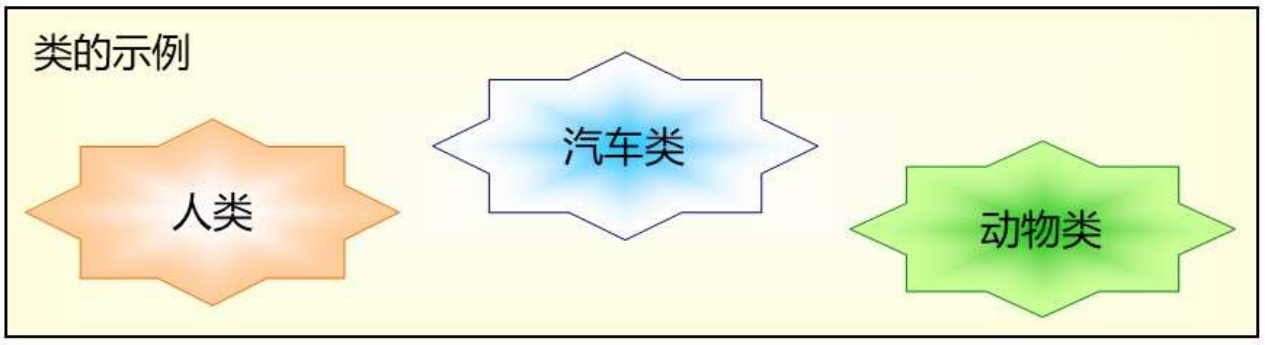
- 封装
  - oop基本特征：数据 + 函数
  - class：“用户自定义类型” / “抽象数据类型”
  - 头文件（类内声明）+ 实现文件（类外定义）

- 类成员的访问权限：public/private(缺省)/protected

- this指针：指向当前对象的指针变量

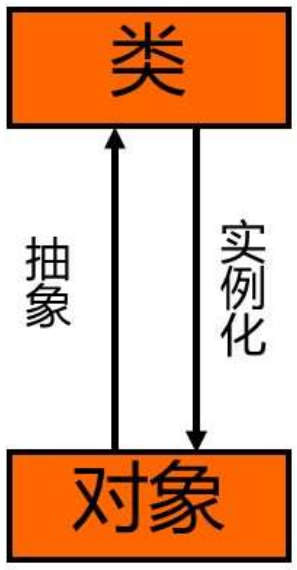
非静态

- 类是具有相同属性和行为的一组对象的集合



类和对象的区别

- 类是具有共同属性和行为的对象的抽象；
- 类可以定义为数据和方法的集合；
- 类也称为模板，因为它们提供了对象的基本框架；



构造函数与析构函数

	构造函数	析构函数
返回值类型	无	无
函数名	与类名相同	~类名（无参数）
能否重载	能	不能
初始化	使用初始化列表初始化数据成员或调用其他构造函数	—
编译器合成	未定义任何构造函数时，编译器会根据自身需要来合成无参的默认构造函数	未定义析构函数时，编译器会根据自身需要合成
删除	delete	—

- default关键字可用于手动指定默认构造函数的生成
- 即使用户显式定义析构函数，编译器也可能会拓展该部分代码，如在派生类析构函数中插入析构基类的代码
- 对象数组初始化：与构造函数相对应

对象的构造与析构时机

	构造	析构
局部对象	程序执行到该局部对象的代码时	所在作用域结束后
全局对象	main()函数调用之前	main()函数执行完return之后
函数静态对象	程序执行到该局部对象的代码时	main()函数执行完return之后
参数对象 (传递形参)	函数被调用时	函数结束时
参数对象 (传递引用)	不构造	不析构

拷贝/移动构造函数

	拷贝构造函数	移动构造函数
定义	ClassName (const ClassName&)	ClassName (ClassName&&)
参数	同类对象的常量左值引用	同类对象的右值引用
常见调用场景	1. 用一个对象初始化另一个同类对象 2. 函数调用时以类的对象为形参 3. 函数返回类对象 (注：考虑显式定义拷贝构造而未显式定义移动构造的情况)	1.用一个对象初始化另一个同类对象 2. 函数返回类对象 3. std::move返回参数的右值，便于调用移动构造函数 (注：考虑显式定义移动构造函数的情况)
效率	低	高

- 编译器自动合成的拷贝构造函数采用位拷贝，拷贝地址而非内容，遇到指针类型成员时会出错

组合

■ **has-a：整体–部分关系**

- 已有类的对象作为新类的公有/私有数据成员

■ **构造与析构**

- 子对象构造时若需要参数，应在当前类的构造函数的初始化列表中进行
- 构造次序：先子对象，再当前对象。子对象的构造按照类中声明的次序进行
- 析构次序：与构造次序相反

12

第五章　数据的共享与保护

- **作用域**（函数原型作用域、局部作用域、类作用域、命名空间作用域）及可见性
- **对象生存期**（动态生存期、静态生存期【全局变量、函数内部用static关键字声明的局部变量】——只第一次进入函数时被初始化）
- 为了实现一个类的不同对象之间数据和函数共享：**静态成员**。
- **静态数据成员**是一种特殊的数据成员，它表示类属性，而不是某个对象单独的属性，它在程序开始产生，在程序结束时消失。静态数据成员具有静态生存期。
- **静态数据成员**初始化时前面不加static关键字；由于是类的成员，因此在初始化时必须使用类作用域运算符::限定它所属的类。
- **静态成员函数**属于整个类，是该类的所有对象共享的成员函数。可通过类名、对象调用静态成员函数访问静态函数成员。
- **静态成员函数**可直接访问类的静态成员，但不能直接访问类的非静态成员；若要访问非静态成员，必须通过参数传递的方式得到相应的对象，再通过对象来访问。

- **友元函数**不是类的成员（在函数实现时不需要friend关键字），但它可以访问类的任何成员。一个类的友元类可以访问该类的任何成员。
- 友元关系是**不能传递、不继承、单向性。**友元提高了数据的**共享性**，加强了函数与函数、类与类之间的相互联系，大大提高程序的效率；但**破坏了数据隐蔽和数据封装。**
- 为了增强C++程序的**安全性和可控性：**常对象、常数据成员、常成员函数、常引用
- 关键字const来定义的对象称为**常对象**，常对象的成员不允许被修改。常对象只能调用它的常成员函数，而不能调用其他成员函数。
- 使用const定义的数据成员称为**常数据成员**，必须进行初始化，并且**不能被更新。**但常数据成员的初始化只能通过构造函数的初始化列表进行。
- 在定义时使用const关键字修饰的成员函数称为**常成员函数**，用于访问类的常对象。const关键字可以用于参与重载函数的区分。
- **常引用**所引用的对象不能被更新。如果用常引用做形参，调用函数时又不必建立实参的拷贝，可以提高程序效率，不会意外地发生对实参的更改。

静态成员与常量成员

	静态	常量
关键字	static	const
数据成员	所有对象共享	在对象的生命周期中不可更改
成员函数	只能访问静态数据成员	不能修改类的数据成员
访问方式	类/对象	对象

初始化方式	就地初始化	初始化列表初始化	构造函数体内初始化	类外初始化
静态				√
常量	√	√		
常量静态（除int,enum）				√
常量静态（int,enum）	√			√

第六章 数组、指针与字符串

- 数组（一维、二维、多维、对象数组）
- 指针（*与&、指针赋值、指针常量、常量指针、对象指针、this指针）
- 动态存储分配（new、delete、深复制）
- 指针与数组（指针运算、用指针来处理数组元素、指针数组）
- 指针与函数（指针作为函数参数及与引用做函数参数的联系、指针型函数【函数返回指针】、指向函数的指针【函数代码的首地址】）
- 字符串（字符数组和字符串的区别）

第七章 类的继承

- 通过继承，派生类在原有的类的基础上派生出来，它继承原有类的属性和行为，并且可以扩充新的属性和行为，或者对原有类中的成员进行更新，从而实现了软件重用。
- 继承分为：单继承和多继承。
- 继承方式有public、protected、private，各种继承方式下，基类的私有成员在派生类中不可存取。public继承方式基类成员的访问控制属性在派生类中不变，protected继承方式基类成员的访问控制属性在派生类中为protected, private继承方式基类成员的访问控制属性在派生类中为private。

继承表		继承方法					
		public		private		protected	
基类中成员类型	public	OK	pub/yes	OK	prv/no	OK	pro/no
	private	NO	prv/no	NO	prv/no	NO	prv/no
	protected	OK	pro/no	OK	prv/no	OK	pro/no

派生类成员函数能否访问基类成员

基类成员在派生类中的成员类型，派生类对象能否访问基类成员

- **构造、析构顺序**：在派生类建立对象时，会调用派生类的构造函数，在调用派生类的构造函数前，先调用基类的构造函数。派生类对象消失时，先调用派生类的析构函数，然后再调用基类的析构函数。
- **类型兼容**是指在公有派生的情况下，一个派生类对象可以作为基类的对象来使用：派生类对象可以赋值给基类对象，派生类对象可以初始化基类的引用，派生类对象的地址可以赋给指向基类的指针。
- 多继承时，多个基类中的同名的成员在派生类中由于标识符不唯一而出现二义性。在派生类中采用**成员名限定**或**重定义具有二义性的成员**来消除二义性。
- 在多继承中，当派生类的部分或全部直接基类又是从另一个共同基类派生而来时，可能会出现间接二义性。消除间接二义性除了采用消除二义性的两种方法外，可以采用虚基类的方法。

例 赋值兼容规则实例。

```
#include <iostream.h>
class base{
public:
int i;
base(int x)  {i=x;}
void show()
{cout<<"base "<<i<<endl;}
};

class derive:public base{
public:
derive(int x):base(x) { };
void show()
{cout<<"derive "<<i<<endl;}
};
```

```
void main()
{
  base b1(11); b1.show();
  derive d1(22);
  b1=d1;   b1.show();
  derive d2(33);
  base &b2=d2;  b2.show();
  derive d3(44);
  base *b3=&d3; b3->show();
  derive *d4=new derive(55);
  base *b4=d4;  b4->show();
 delete d4;
 }
```

```
Base 11
Base 22
Base 33
Base 44
Base 55
```

第八章 多态性

- 多态性是面向对象程序设计的重要特性之一，多态是指同样的消息被不同类型的对象接收时导致完全不同的行为。
- 联编可以在编译和连接时进行，称为静态联编；联编也可以在运行时进行，称为动态联编。
- 运算符重载是通过重载运算符函数实现的，其实质是函数重载。
- 运算符可以作为普通函数、类的友元函数、类的成员函数重载。作为成员函数重载时，第一个参数为调用对象自身，因此可以在函数定义时省略第一个参数。
- 虚函数是在基类中定义的以virtual关键字作为开头的成员函数，需要在派生类中重新定义。通过指向基类的指针或引用来调用虚函数实现动态联编。
- 纯虚函数是一个在基类中声明的没有定义具体实现的虚函数，带有纯虚函数的类称为抽象类。
- 抽象类是为了抽象的目的而建立的，通过抽象类多态地使用其中的成员函数，为一个类族提供统一的操作界面。抽象类处于类层次的上层，自身无法实例化，只能通过继承机制，生成抽象类的具体派生类，然后再实例化。通过抽象类的指针和引用，可以指向并访问各派生类成员，实现动态多态性。

第九章 模板

- 函数模板与类模板
- 函数模板/类模板/成员函数模板
  - 当模板类的成员函数具有其他模板参数时，类外定义时不能写成多参数形式，因为两者的作用域不同
- 模板特化
  - 函数模板不能部分特化（区别特化与重载）
  - 类模板允许部分特化
- 模板原理：静多态(编译时多态)
  - 模板库必须在头文件中实现，否则会出现链接错误

线上考试注意事项

- 浏览器：火狐或谷歌浏览器
- 主观题：答案不要错序
- 用白纸黑笔作答、答案多图上传图片时注意顺序、图片要旋转正确后再上传