C++程序设计期末考试-参考PPT节选 2022.11.25

第2章

只读变量：使用**const**或**constexpr**说明或定义的变量，定义时必须同时初始化。当前程序只能读不能修改其值。

**引用变量只能在定义时初始化1次，初始化后对引用变量的赋值运算是对该引用变量所代表的存贮单元进行赋值。**

**用左值初始化的引用变量：被引用的对象（内存单元）的别名（引用变量本身不占用内存）。**

**用右值初始化的引用变量：编译器为被引用的对象申请一个缓存，该引用变量就是这个缓存的别名（引用变量本身不占用内存）。**

**理解引用变量：逻辑上：变量的别名（不占用存储单元）；实现上：被编译为指针（占用存储单元）**

**const int x = 1本质上是左值，因为x占有存贮单元。通过特殊方法可以修改x的值： \*(int \*)&x = 0;**

int &&q = (int)x //对，**（int）x是右值** int & &u; //错：引用变量u去引用无内存的引用，**int &&u; ?**

int & \*v; //错，指针不能指向引用变量(无内存) int \*p; int \*&v = p //对，v是指针p的引用，可以使用\*v, v[1]等

**左值与右值**：x+3计算结果为传统右值。x=5是赋值运算，计算结果为传统左值x(x的值为5) 。由于计算结果为传统左值x，故还可对x赋值7，相当于运算(x=x+3)=7，结果为左值。（课本P41）

★前置运算（++x，--x）先运算后取值，结果为传统**左值**；后置运算（x++，x--）先取值后运算，结果为传统**右值**。

第3章

调用开销是指为完成调用所进行的实参传递、重要寄存器保护及恢复以及返回时的栈指针恢复到调用前的值所额外编译或执行的指令。若f1、f2的调用开销分别为10、7，函数体指令数分别为5、20，程序对f1和f2均有100个位置调用。则调用f1、f2编译后的指令数：

成功内联f1=100\*5，不内联f1=10\*100+5=1005: 函数体小，内联合算

成功内联f2=100\*20，不内联f2=7\*100+20=720: 函数体大，调用合算

由此可见：函数体相对较小的函数，使用内联更合算。若函数为虚函数、或包含分支(if, switch,?:,循环,调用)，或取函数地址，或调用时未见函数体，则内联失败。内联失败不代表程序有错，只是被编译为函数调用指令。

constexpr函数的函数体可能被优化掉，其作用域相当于static。

**inline、static修饰的变量和函数作用域是模块文件内部，不同模块可定义同名的全局或inline、static变量和函数**

第4章

类定义体的最后一个花括号后要跟有**分号**作为定义体结束标志。

同一个对象仅自动构造一次。构造函数是唯一不能被显式调用的函数成员；析构函数既能被显式调用，也能被隐式（自动）调用。由于只有一个固定类型的this，故不可能重载，只能有一个析构函数。若实例数据成员有指针，应当防止反复析构（用指针是否为空做标志）。

**程序不同结束形式对对象的影响：**

exit：局部自动对象不能自动执行析构函数，故此类对象资源不能被释放。静态和全局对象在exit退出main时自动执行收工函数析构。

abort：所有对象自动调用的析构函数都不能执行。局部和全局对象的资源都不能被释放，即abort退出main后不执行收工函数。

return：隐式调用的析构函数得以执行。局部和全局对象的资源被释放。

提倡使用 return。如果用 abort和 exit，则要显式调用析构函数；使用异常处理时，自动调用的析构函数都会执行。

**接受与删除编译自动生成的函数：**（课本P111）若类A没有定义任何构造函数，则编译器会自动提供无参的构造函数A( )（实际上是空操作，相当于一条return语句）；如果A定义了任何构造函数，编译器就不会自动提供无参构造函数。不管A是否定义了构造函数，A类只要没有定义带引用参数A &的构造函数A(const A &a)，编译器会自动提供该构造函数A(const A &a)（利用浅拷贝实现该构造函数）。可以利用 default、delete强制说明接受、删除编译器提供的构造函数，如 A() = default、 A(const A &a) = delete。

用new分配数组内存时，数组仅第一维下标可为任意表达式，其它维为常量表达式：int (\*q)[6][8]=new int[x+20][6][8];

**delete [ ]<数组指针>的细节：若指针指向任意维的数组，delete [ ] q；若为对象数组，对所有对象(元素)调用析构函数。若数组元素为简单类型，则可用delete <指针>代替。**

**new和malloc、delete和free的用法总结：如果是简单类型（不是类的对象，或类的对象没有构造、析构函数），malloc和free等价，delete和free等价；如果类有构造函数则要用new，有析构函数则要用delete。（课本P105）**

注意：可用\*this来引用或访问调用该函数成员的普通、const或volatile对象；**类的静态函数成员（static）没有隐含的this指针**。

**创建对象时，不调用构造函数的情况（满足下面条件）：**

(1)类中没有定义任何构造函数、没有虚函数；(2)该类不是派生类（别的类可以是该类的派生类）；

(3)定义对象时用 { ... } 给对象赋值，{ ... }中的元素个数必须小于等于类中实例成员变量（非static变量）的个数。这相当于在构造函数的参数初始化列表中给实例成员变量初始化；

(4)用非空的 { ... } 给对象赋值时, 类中对应的实例成员变量可访问。

第6章

**在派生类中，可使用“基类名::成员”或“using 基类名::成员”修改基类成员的访问权限。**using 特定基类数据成员后，不允许再在派生类中定义同名数据成员；using 特定基类函数成员后，还可以再在派生类中定义同名函数成员。

标识符的作用范围从小到大：①作用于函数成员内；②作用于类或派生类内；③作用于基类内；④作用于虚基类内。标识符的作用范围越小，被访问到的优先级越高。如果希望访问作用范围更大的标识符，则可用类名和作用域运算符进行限定。

派生类不能访问基类私有成员，除非将派生类声明为基类的**友元类**，或将要访问基类私有成员的派生类函数成员声明为基类的友元。

**例：class B;**

**class A {** **…**

**friend B; };** **//声明B为A的友元类，B类成员可以访问A任何成员**

**以下情况派生类必须定义自己的构造函数：**虚基类或基类只定义了带参数的构造函数；派生类自身定义了引用成员或只读成员；派生类需要使用带参数构造函数初始化的对象成员。（课本P143）

**在派生类函数成员内部，基类指针可直接指向该派生类对象；如果函数声明为派生类的友元，则该友元定义的基类指针也可以直接指向该基类的派生类对象**，不必强制类型转换。

第8章

**一旦父类(基类)定义了虚函数，即使没有 virtual 声明，所有派生类中原型相同的非静态成员函数自动成为虚函数(虚函数特性的无限传递性)**

**对于父类A中声明的虚函数f( )，若在子类B中重定义f( )，必须确保子类B::f( )与父类A::f( )具有完全相同的函数原型，才能覆盖原虚函数f( )而产生虚特性，执行动态联编机制。**只要有一个参数不同，编译系统就认为它是一个全新的（函数名相同时重载）函数，而不实现动态联编。

**用父类引用实现动态多态性时需要注意，若被(new产生)引用对象自身不能析构，则必须用delete &析构：**

A &z = \*new B(“123"); delete &z; //析构对象z并释放对象z占用的内存

上述delete &z完成了两个任务：①调用该对象析构函数~B( )，释放其基类和对象成员各自为字符指针s分配的空间；②释放B对象自身占用的存储空间。如将delete &z改为z.~A( )，则只完成①没完成②；如改为free(&z)，则只完成②没完成①，造成内存泄露。

**当用常量对象、类型为&&的返回对象作为实参调用函数时，优先调用的函数是带有&&参数的函数。**

* **虚函数友元与晚期绑定**

**虚函数动态绑定：**C++使用虚函数地址表(VFT)来实现虚函数的动态绑定。VFT是一个函数指针列表，存放对象的所有虚函数的入口地址。编译程序为有虚函数的类创建一个VFT，其首地址通常存放在对象的起始单元中。调用虚函数的对象通过起始单元的VFT动态绑定相应的函数成员，从而使虚函数随调用对象的不同而表现多态特性。

动态绑定比静态绑定多一次地址访问，在一定程度上降低了程序的执行效率，但同虚函数的多态特性带来的优点相比，效率降低所产生的影响是微不足道的。

**虚函数动态绑定过程：**设基类A和派生类B对应的虚函数表分别为VFTA和VFTB。则派生类对象b的虚函数动态绑定过程：

对象构造：先将VFTA的首地址存放到b的起始单元，在A类构造函数的函数体执行前甚至初试化前，使A类对象调用的虚函数与VFTA绑定，可使A类构造函数执行A的虚函数；在B类构造函数的函数体执行前（甚至初试化前），将VFTB的首地址存放到b的起始单元，使B类对象调用的虚函数与VFTB绑定，可使B类构造函数执行B的虚函数。

对象使用(生成期间)：b的起始单元指向VFTB，执行B的虚函数。

对象析构：由于b的起始单元已指向VFTB，故析构函数调用的是B的虚函数；然后将VFTA的首地址存放到b的起始单元，使基类析构函数调用的虚函数与VFTA绑定，使基类析构函数调用基类A的虚函数。

第9章

**派生类可能有虚基类、基类、对象成员、const成员以及引用成员。**当虚基类、基类和对象成员只有带参数的构造函数时，派生类必须定义自己的构造函数，而不能利用C++提供的缺省构造函数。类有非静态对象成员、const 成员时，也必须定义构造函数。对于虚基类、基类和对象成员来说，如果它们没有定义自己的构造函数，则编译程序就会为它们提供缺省的无参构造函数。对于虚基类、基类和对象成员的无参构造函数，无论它们是自定义的还是由编译程序提供的，可被派生类构造函数按定义顺序自动地调用。

**重点：多继承构造顺序（课本P203） 注意：构造必须是从上向下构造，但是搜索的过程是从下向上、从左至右**

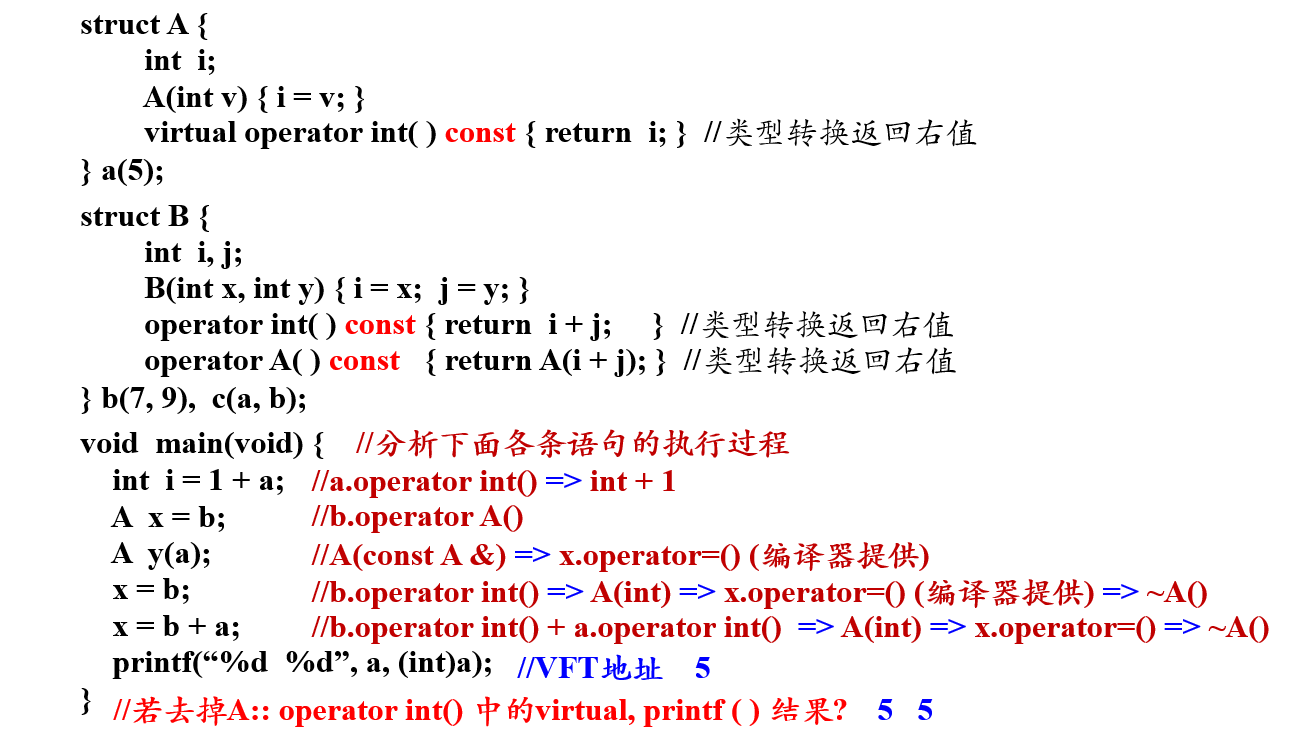
* 1. 按定义顺序自左至右、自下而上地构造所有**虚基类**（如果虚基类上面还有虚基类，就继续向上递归）；
  2. 按定义顺序构造派生类的所有**直接基类**（实际上就是从左到右遍历）；
  3. 按定义顺序构造（初始化）派生类的所有**数据成员**，包括对象成员、const成员和引用成员；
  4. 执行派生类**自身**的构造函数体。

**如果虚基类、基类、对象成员、const成员以及引用成员又是派生类对象，重复上述派生类对象的构造过程，但同名虚基类对象在同一棵派生树中仅构造一次。**

析构派生类对象的顺序是构造的逆序。

第11章

**强制类型转换的结果通常为右值，故最好不要将类型转换函数的返回值定义为左值，也不应该修改当前被转换的对象 (参数表后用const说明this)。**

****

