**（建议字体大小11号 加粗 右下角放大至330%食用）**

**类的成员不能在类里初始化（不能int a=2;）**

**引用就是对象的别名，对象的值变了，引用的值也一起变**

**类成员的默认访问模式是private**

**内联函数可以减少函数的调用，提高程序的运行速度**

**设置默形参值时，形参名可以省略**

**在函数声明，定义时都可以设置默认形参值**

**重载函数与函数类型无关**

**但const可以重载**

**类的实例化是指创建类的对象**

**调用复制构造函数**

**A. 用一个对象去初始化同一类的另一个新对象时**

**B. 函数的返回值是类的对象，函数执行返回时调用**

**C. 函数的形参是类的对象，调用函数进行形参和实参结合时**

**（B和C参数/返回值是类的对象的引用，不会调用）**

**引用在一开始就要确定引用对象（不能 new ）**

**构造函数是在（ 创建对象 ）时被执行的。**

**类的对象只能访问该类的公有成员**

**普通函数不能直接访问类的公有成员，必须通过对象访问**

**一个类可以将另一个类的对象作为成员**

**有了类的前项引用声明， 但在定义类之前，**

**仍不能定义该类的对象， 也不能在内联成员函数中使用该类的对象**

**（但可以为形参，指针）**

**（因为只能使用被声明的符号， 而不能涉及类的任何细节）**

**作用域和可见性不一致 —— static**

**X::m 访问静态成员**

**Class A; Class B;**

**A aa;**

**B A; 此时A为对象**

**静态数据成员**

**类共有的，类外初始化（加类名限定）（类名紧跟标识符）**

**静态成员函数**

**类共有的，不用对象，可直接访问且只能访问该类的静态数据和函数成员**

**访问类的非静态成员需要指明对象**

**（所以静态成员函数会有参数）**

**友元关系：（数据共享， 不能传递、单向、不被继承）**

**普通函数/类的成员函数 访问 封装在另一个类的数据**

**（friend放最前面）（inline static friend 都放最前面）**

**（const 为常成员函数时 放函数名后面）**

**友元函数**

**可通过对象名访问类的私有和保护成员**

**（只能在类中声明，因为不是类的成员函数， 定义放外面）**

**（不是类的成员函数， 所以也不用类名限定）**

**友元类：**

**Class B**

**｛friend class A｝**

**友元类A的所有成员函数均为类B友元函数，**

**都可以访问B的私有和保护成员**

**Const 既要共享 又要防止改变**

**初始化：定义并指定初值**

**赋值：定义以后 再用=修改值**

**常对象只能调用常成员函数（唯一接口）**

**Void print() const;**

**常函数被调用，目的对象被视为常对象**

**Const 可以用于重载函数区分**

**Static const A::b = 10; //静态常数据成员 类外初始化说明**

**A::A(int i):a(i){} //常数据成员 只能初始化列表初始化**

**对函数中无需改变的参数， 传递常引用**

**（因此复制构造 一般常引用传递）**

**#include**

**<> 按标准方式搜索**

**“ ” 先当前目录 再标准方式**

**多维数组 内存中 顺序连续存储**

**对象数组初始化：**

1. **指定初值 调用与形参相匹配的构造函数**
2. **没有指定初值， 调用默认构造函数（所以一定要写默认构造函数呀）**

**Int a[3][4];**

**a //整个数组首地址**

**&( a[0] ) //整个数组首地址**

**a[0] //第一行首地址**

**虽然值都是一样**

**析构函数**

**在main函数执行完后执行**

**指向常量的指针（对象值不能变）**

**Const int \*p = &a;**

**指针类型的常量 (指针值不能变)**

**Int \* const p = &a**

**指针数组**

**Int \*p[3]; //右结合性，先数组后指针**

**每个元素都是指针**

**(初始化应为 对象地址)**

**P[i][j] = \*( p[i] + j ) = \*( \*(p + i) + j )**

**指针函数（返回指针）（把大量数据从被调函数返回到主调函数）**

**（不能返回非静态地址）**

**（但可返回动态分配中的地址）（因为在堆里）**

**函数指针（调用函数）**

**字符串可以用 char \*p 表示**

**Operator 只能0个或1个参数**

**1、new/delete是C++的操作符，而malloc/free是C中的函数。**

**2、new做两件事，一是分配内存，二是调用类的构造函数；**

**同样，delete会调用类的析构函数和释放内存。**

**而malloc和free只是分配和释放内存。**

**3、new建立的是一个对象，而malloc分配的是一块内存；**

**new建立的对象可以用成员函数访问，不要直接访问它的地址空间；**

**malloc分配的是一块内存区域，用指针访问，可以在里面移动指针；**

**new出来的指针是带有类型信息的，而malloc返回的是void指针。**

**4、new/delete是保留字，不需要头文件支持；malloc/free需要头文件库函数支持。**

**初始化列表 只能在构造函数里用**

**【补充】（1）当类中含有const（常量）、reference（引用）成员变量时，只能初始化，不能对它们进行赋值；**

**常量不能被赋值，只能被初始化，所以必须在初始化列表中完成，C++引用也一定要初始化，所以必须在初始化列表中完成。**

**（2）基类的构造函数都需要初始化列表；**

**（3）成员类型是没有默认构造函数的类；**

**只有构造函数可以使用这种初始化列表语法（即不能再其他成员函数中用，甚至（） ）**

**复制构造也不用返回值**

**1、运算重载符不改变语法结构。**

**2、运算重载符不改变操作数的个数。**

**3、运算重载符不改变优先级。**

**4、运算重载符不改变结合性。**

**类重载、覆盖、重定义之间的区别：**

**重载指的是函数具有的不同的参数列表，而函数名相同的函数。重载要求参数列表必须不同，比如参数的类型不同、参数的个数不同、参数的顺序不同。如果仅仅是函数的返回值不同是没办法重载的，因为重载要求参数列表必须不同。（发生在同一个类里）**

**覆盖是存在类中，子类重写从基类继承过来的函数。被重写的函数不能是static的。必须是virtual的。但是函数名、返回值、参数列表都必须和基类相同（发生在基类和子类）**

**重定义也叫做隐藏，子类重新定义父类中有相同名称的非虚函数 ( 参数列表可以不同 ) 。（发生在基类和子类）**

# 错题：

**非静态成员函数才有this 指针**

**（static 没有 this）**

**删除数组**

**ClassA \*a = new classA[10];**

**delete [ ]a;**

**由new分配的内存空间是连续的**

**如果当前内存无足够的空间可分配，则new运算符返回NULL**

**由new运算符分配的内存空间，当函数执行结束时系统不会自动收回，必须手动**

**const char \*p；**

**指向字符串常量的指针**

**下面4个选项中，均是合法实数的选项是（ B ）。**

**A. 160. 0.12 e3 B. 123.5 2e-4 123. C. -0.18 2e 0e1 D. 2e3.2 0.234 1e0**

**E前必须有数 e后必须有数 e后必须为整数**

**Static 数据成员 类外初始化，并用类名限定**

**无法通过对象调用**

**用静态成员函数调用**

**（如果是统计个数，则构造，复制构造， 析构都要有）**

**友元函数，不用限定**

**Friend class A 声明为A的友元**

**（好吧程序题还是缩小至200%看吧 = =）**

**有以下程序：**

**class Test**

**{**

**private:**

**static int n; // 静态成员**

**public:**

**Test() { n += 2; } // 构造函数**

**~Test() { n -= 3; } // 析构函数**

**static int GetNum() { return n; } // 返回n**

**};**

**int Test::n = 1; // 初始化n**

**int main() // 主函数main()**

**{**

**Test \*p = new Test; // 定义指针p**

**delete p; // 释放p指向的动太对象**

**cout << "n=" << Test::GetNum() << endl;// 输出n**

**return 0; // 返回值0, 返回操作系统**

**}**

**执行后的输出结果是**

**n=0**

**class MyClass**

**{**

**public:**

**MyClass() { cout << "A"; } // 无参构造函数**

**MyClass(char c) { cout << c; } // 带参构造函数**

**~MyClass() { cout << "B"; } // 析构函数**

**};**

**int main() // 主函数main()**

**{**

**MyClass a, \*p; // 定义变量（定义指针是不会编译 构造并初始化对象的）**

**p = new MyClass('X'); // 分配动态空间**

**delete p; // 释放动态空间**

**cout << endl; // 换行**

**return 0; // 返回值0, 返回操作系统**

**}**

**执行这个程序，屏幕上将显示输出 。**

**AXBB**

**Quiz3**

**在公有和保护继承方式下，派生类的对象可以对基类的保护成员进行访问**

**虚基类不能创建对象实例**

**在C++中，编译时的多态性是通过函数和运算符的重载实现的。**

**运算符=、()、[]和->可作为类成员运算符, 不能作为友员运算符**

**抽象类只能作为基类使用，其纯虚函数的实现由派生类来完成**

**运算符重载以后，其优先级和结合性都不能改变**

**抽象类不能创建对象实例（存在纯虚函数没有被定义）**

**抽象类指针可以指向不同的派生类**

**静态成员函数没有this指针**

**将用于输出的插入运算符函数operator<<设计成友元函数的根本原因**

**不是因为进行输出操作时需要访问对象的内部数据成员。**

**而是不是友元则为重载，一定有this，即左操作数一定是this，则无法输出数据到cout里**

**虚析构函数的作用是delete动态对象时释放资源**

**int \*const p; 说明不能修改p指针**

**假定AB为一个类，则该类的拷贝构造函数的声明语句为AB(AB& x);**

**友元运算符obj1>obj2被C++编译器解释为operator > (obj1, obj2)**

**基类和派生类 先构造基类 再构造派生类**

**class B**

**{ int a,b;**

**public:**

**B(int aa=0, int bb=0) { a=aa; b=bb; }**

**B operator+ ( int x )**

**{ B r;**

**r.a=a+x;**

**r.b=b+x;**

**return r;**

**}**

**};**

**void main( )**

**{ B x(3,5), y(8,4), z1, z2;**

**z1=x+5;**

**z2=10+y;　　　　　　不能为左值**

**}**

**class B**

**{**

**public:**

**virtual void Show() const { cout << "B"; } // 输出信息**

**};**

**class D: public B**

**{**

**public:**

**void Show() const { cout << "D"; } // 输出信息**

**};**

**void Fun1(const B \*p) { p->Show(); } // 定义Fun1()**

**void Fun2(const B &obj) { obj.Show(); } // 定义Fun2()**

**void Fun3(const B obj) { obj.Show(); } // 定义Fun3()**

**int main() // 主函数main()**

**{**

**B \*p = new D; // 指针**

**D d; // 对象**

**Fun1(p); // 调用Fun1()**

**Fun2(d); // 调用Fun2()**

**Fun3(d); // 调用Fun3()**

**cout << endl; // 换行**

**return 0; // 返回值0, 返回操作系统**

**}**

**程序的输出结果是 。**

**DDB 形参虚基类 传指针和引用 实参派生类 则执行派生类同名函数**

**#include <iostream>**

**using namespace std;**

**class Test**

**{**

**public:**

**Test(){}**

**~Test()**

**{**

**cout<<"#";**

**}**

**};**

**int main()**

**{**

**Test temp[2];**

**Test \*ptemp[2]; //根本没被编译 （打了断点 也被跳过）**

**//cout << "Hello world!" << endl;**

**return 0;**

**}**