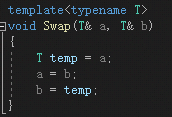
1. 必须先声明原型，原型的变量名可以省略。
2. ,泛型的定义，如果参数有两个类型，想统一为一个类型，则显示调用
3. 通常，编译器使用三块独立的内存，一块用于静态变量（可能再细分），一块用于自动变量，一块用于动态存储。
4. 初始化
   1. int\* pi = new int(6);
   2. int\* ar = new int[3]{1, 2, 3};
5. 内联函数可以直接写在类中，也可以只声明原型，然后在实现时前面加inline
6. 构造方式
   1. A a = A(“1”);
   2. A a(“1”);
   3. A\* a = new A(“1”);
   4. A a = {“1”};
   5. A a{“1”};
   6. A\* a = new A{“1”};
7. 在成员函数后加const，表示此方法不允许修改成员值(非静态)
8. 声明对象数组 A a[3]，并且可以用构造函数初始化数组：

A a[3] = {A(“1”), A(“2”), A(“3”)};

1. 多态，当不适用virtual时，将根据引用类型或指针类型选择方法，使用virtual时将根据引用或指针指向的对象的类型来选择方法
2. 虚函数工作原理，给对象增加一个隐藏成员指针，这个指针指向虚函数表，虚函数表中存储了这个类的所有虚函数的地址，派生类对象将新弄一个指针，然后复制一份虚函数表，如果重新定义了虚函数，则替换为新的地址
3. 友元不能是虚函数，因为友元不是类成员
4. 抽奖类，virtual void Test() = 0; 等于0即表示为纯虚函数，存在纯虚函数的类即为抽象类，无法实例化
5. 公有、私有、保护继承：
   1. 公有继承：父类的所有权限维持原状
   2. 私有：父类所有都变私有
   3. 保护：父类公有和保护都变保护
6. 虚基类，防止菱形继承时底部有两个根节点父类拷贝的情况
7. 类模板：
   1. 在类前加 template<typename T>,然后类的所有实现前都要加这个（想吐槽啊。。。真麻烦）
   2. 类模板必须显示的提供所需的类型，A<int> a;
   3. 可以有默认值 template<typename T1, typename T2 = int>，这样的话，如果声明时不加T2，就也可以，默认就是Int了
   4. 数组模板：template<class T, int n>，用法：T ar[n];
   5. 也可在类成员内的新类中加模板，也就是可以模板套模板
   6. 也可以将模板当做参数：template<template<typename T> class Thing>
   7. 可以给模板设置别名：typedef array<double, 12> arrd; arrd就可以代表了
8. 友元类
   1. 假如一个电视和一个遥控器，不是is a也不是has a，可以设为友元
   2. 写法：在TV类中，加入friend class Remote；
   3. 可以互为友元类
   4. 共同友元，比如一个类想同时访问两个类的私有数据，也需要类似下面的前向生命，详见 c++ primer plus 610页；
9. 友元成员函数，当比如遥控器只需要一个方法称为TV的友元的话，就不需要把整个类声明为友元，但是做法很复杂，这里写不明白，详见c++ primer plus 607页，设计到循环依赖的解决方案，挺重要的。
   1. 写个大概吧，先前向声明 class Tv;，注意这里不实现
   2. 写class Remote类，注意这里所有的内联函数都不允许出现
   3. 完整实现Tv类
   4. 然后实现内联函数，例如：

inline bool Remote:volup(Tv& t){return t.volip();}

* 1. 新加一个需要注意的地方，比如在Tv把Remote声明为友元类，则必须按照如上顺序，因为程序读到友元类的声明时，此类必须实现

1. 智能指针：正常的指针需要在无用时delete，但是如果能有一个指针有析构函数，然后在析构函数中调用delete就不用用户管理了，这就是智能指针的思想.
   1. 需包含头文件 #include <memory>
   2. auto\_prt<string> ps (new string(str));
   3. 如下语句

auto\_ptr<string> p1(new string(“auto”));

auto\_prt<string> p2;

p2 = p1;

p1的所有权将被转移至p2，智能指针的这个设计可以防止两个指针指向同一个变量，然后析构两次

如果用unique的话，编译器将认为第三句是非法的

总之，程序试图将一个unique\_ptr赋值给另一个时，如果源unique\_ptr是个临时右值，编译器则会允许这样做，比如函数返回一个unique\_ptr，然后用一个unique\_ptr接收，这是允许的，详见 673页

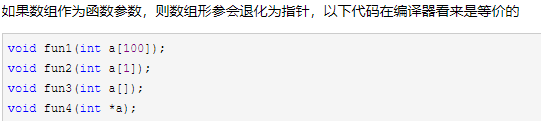
* 1. Auto\_ptr和shared\_ptr只有使用new时才能用智能指针，使用new[]时不能用
  2. Unique\_ptr都可以用

Unique\_ptr<double[]> pda(new double(5));

1. STL
   1. 迭代器：vector<double>::iterator pd;

for(pd = scores.begin(); pd != scores.end(); pd++)

* 1. For(double x : prices)
  2. For\_each(books.begin(), books.end(), ShowReview)

1. 数组作为参数时，会退化为指针

也就是说，数组作为参数时，获取数组的长度是无效的，这是几乎必中的坑啊….

1. 使用new可以在运行时分配内存，不使用new则在编译时就分配内存空间，不管程序最终是否使用，他都在那里，占用内存。在编译时给数组分配内存被称为静态联编，运行阶段再创建，还可以在程序运行时选择数组的长度，被称为动态联编；
2. 对空指针delete是安全的
3. 引用必须声明时就初始化
   1. Int& rodents = rats 实际上是 int\* const pr = &rats
4. 抽象类不能实例化，所以不能把抽象类作为容器，即vector和list这种都不能直接用，找了半天解决方案，还是老外靠谱，用指针就可以，引用都不行