头文件和头文件保护符

1. 头文件(header)：使类或其他名字的定义可被多个程序使用的一种机制；程序通过#include指令使用头文件；(P24)；
2. 头文件后缀.h：通常用.h作为头文件后缀，标准库头文件不带后缀；编译器不关心头文件名的形式，IDE可能有要求；(P17)；
3. 头文件保护符(header guard)：①#define指令把一个名字设定为预处理变量；②#ifdef当且仅当变量已定义时为真；③ifndef当且仅当变量未定义时为真；④一旦结果为真，执行后续操作直到#endif为止；⑤头文件必须设置保护符；(P68)；
4. 预处理器(preprocessor)：在C++编译过程中执行的一段程序；(P70)；
5. 预处理变量(preprocessor variable)：①由预处理器管理的变量；②预处理变量有两种状态：已定义和未定义；③在程序编译之前，预处理器负责将程序中的预处理变量替换成它的真实值；④如NULL预处理变量定义在头文件cstdlib中，值为0；(P48,P70)；
6. assert预处理宏和NDEBUG预处理变量：(P216)；

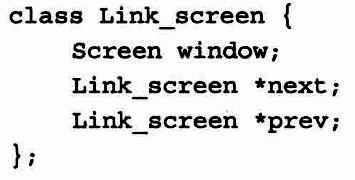
变量的声明和定义

1. 声明(declaration)：声称存在一个变量、函数或别处定义的类型，名字必须在定义或声明后才能使用；(P69)；
2. 变量声明extern：如果声明而非定义一个变量，在变量名前加extern，如 (P41)；
3. 变量定义(definition)：为某一特定类型的变量申请存储空间，可以选择初始化该变量；(P70)；
4. 变量声明和定义的区别：①一个文件文件使用别处定义的名字必须包含对那个名字的声明，定义负责创建与名字关联的实体；②变量声明和定义都会规定变量类型和名字，而定义还会申请存储空间，甚至赋初值；③变量只能定义一次，但可以被多次声明；④如果要在多个文件中使用同一个变量，就必须将声明和定义分离，变量的定义只能出现在一个文件中，其他用到该变量的文件必须对其进行声明；(P41)；

函数的声明和定义

1. 函数原型(function prototype)：函数声明也称为函数原型；函数声明不包括函数体和形参名，只包括函数三要素（返回类型、函数名、形参类型）描述函数接口；(P186)；
2. 在头文件中进行函数声明：含有函数声明的头文件应被包含到定义函数的源文件中，编译器验证函数的定义和声明是否匹配(P186)；
3. 函数定义：由返回类型、函数名字、形参列表、函数体(function body)组成；(P182)；
4. 返回类型(return type)：函数声明的一部分，用于指定函数返回值的类型(P226)；
5. 形参(parameter)：在函数的形参列表中声明的局部变量；形参由逗号隔开位于括号内；用实参初始化形参；(P226)；
6. 实参(argument)：函数调用时提供的值，用于初始化函数的形参(P225)；

类的声明

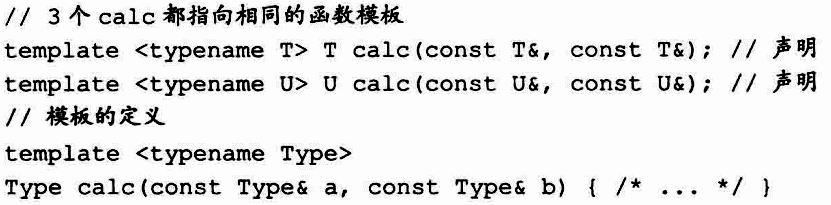
1. 类的声明(class declaration)：首先时关键字class（或struct），随后是类名以及分号。如果类已经声明而尚未定义，则它是一个不完全类；(P273)；
2. 前向声明(forward declaration)：对尚未定义的名字的声明，通常用于表示位于类定义之前的类声

明；(P274)；

1. 不完全类型(incomplete type)：已经声明但是尚未定义的类型；不完全类型不能用于定义变量（创

建对象）或者用引用/指针访问其成员；但是定义指向这种不完全类型的指针或者引用，以及声明（不能

定义）以该类型作为参数或返回类型的函数是合法的；**一个类的成员类型不能是该类自己，但一个类的名字出现后就认为是声明过了，因此类允许包含指向它自身的引用或指针**；(P274)；

函数模板的声明和定义

块、名字的作用域和复合语句

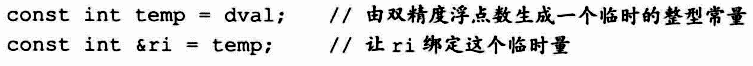
1. 作用域(scope)：程序的一部分，名字在其中有特定含义，作用域以花括号分隔；(P43)；
2. 名字的作用域：有效区域始于名字的声明语句，以声明语句所在的作用域的末端结束；(P43)；
3. 复合语句(compound statement)：用花括号括起的语句和声明的序列，符合语句也被称为块(block)；一个块就是一个作用域，在块中引入的名字只能在块内部以及嵌套在块中的子块里访问；用于语法上需要一条语句，但逻辑上需要多条语句的地方;(P155)；
4. 全局作用域(global scope)：名字定义在所有作用域之外； (P44)；
5. 块作用域(block scope)：名字定义在块内部；(P44)；
6. 内层作用域(inner scope)和外层作用域(outer scope)：嵌套在其他作用域之内的作用域，内层作用域能访问外层作用域中的名字，也可以重新定义外层作用域已有的名字；嵌套着别的作用域的作用域；(P44)；

字面值常量

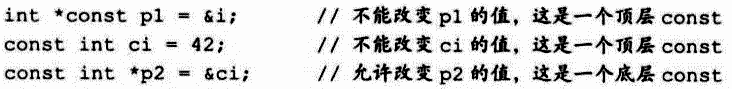
1.字面值(literal)：是一个不能改变的值，如数字、字符、字符串等；单引号内的是字符字面值，双引号内的时字符串字面值(P70)；

const限定符

1. const类型修饰符: const对象一旦创建后其值不能改变，所以const对象必须初始化；(P53)；
2. 临时值(temporary)：编译器在计算表达式的结果时创建的无名对象；为某表达式创建了一个临时值将一直存在直到包含有该表达式的最大的表达式计算完成为止；(P71)；
3. 默认情况下，const对象仅在文件内有效：①多个文件出现同名const变量等于在不同文件分别定义独立的变量；②如果想在多个文件之间共享const对象，必须在变量声明和定义之前加上extern；(P54)；
4. 对常量的引用(reference to const)：①是一个引用，不能用来改变它所绑定对象的值；②对常量的引用可以绑定常量对象，或者非常量对象，或者表达式的结果；(P54)；



1. const默认作用于其左边的东西，否则作用于其右边的东西；
2. 常量指针(const pointer)：必须初始化，初始化后值不能改变；不变的是指针本身的值而非指向的值；(P56)；
3. 顶层(top-level) const和底层(low-level) const：①顶层const规定某对象的值不能变，如表示指针本身是个常量；②底层const不能被忽略，如表示指针所指向的对象是个常量；③指针和引用才有底层const，其它所有类型都是顶层const；④引用不存在顶层const，因为引用必须初始化绑定一个对象，绑定后无法重新绑定，因此本身自带顶层const的特性；执行拷贝时，顶层const会被忽略，而底层const则要求拷入和拷出的对象必须具有相同的底层const资格；(P57)；



1. 常量表达式(const expression)：是指值不会改变并且在编译过程就能得到计算结果的表达式；字面值属于常量表达式。用常量表达式初始化的const对象也是常量表达式；C++11允许将变量生命为constexpr类型来验证是否为常量表达式；(P59)：

for语句和范围for语句

1. 传统for语句：①语法形式为for(init-statement; condition; expression) {statement}；②init-statement必须是三种情况之一：声明语句、表达式语句、空语句；③init-statement可以定义多个对象，但只能有一条声明语句，因此所有变量的基础类型必须相同（但可以同时声明指针引用等）；(P166)；
2. 范围for语句：①语法形式为for(declaration : expression) {statement}；②expression必须是一个序列，如花括号括起来的初始列表、数组、vector和string等类型的能返回迭代器的对象;③declaration每次迭代都会重新定义循环变量，并将其初始化为序列中下一个值；④如果需要对序列中元素执行写操作，循环变量必须声明成引用类型；(P168)；
3. 范围for循环的实现：不能通过范围for语句增加vector或其他容器的元素，因为for语句预存了end()的值，一旦添加或删除元素，end函数的值就可能变得无效；(P169)；

函数基础

1. ()运算符(() operator)：调用运算符，用于执行某函数，括号前面是函数名或函数指针，括号内是以逗号隔开的实参列表(P226)；
2. 函数调用和返回：①用实参初始化对应的形参；②主调函数(calling function)将控制权转移给被调函数(called function)；③返回return语句的值；④控制权转移(P182)；
3. 实参和形参：实参类型必须和形参类型匹配；函数调用规定实参数量应与形参数量；(P183)；
4. 形参列表：可以为空但不能省略；形参通常用逗号隔开，每个形参都是含有一个声明符的声明；形参名不能相同，也不能与函数最外层局部变量相同；(P183)；
5. 局部变量(local variable)：定义在块中的变量，形参和函数体内部定义的变量统称为局部变量；(P184)；
6. 自动对象(automatic object)：仅存在于函数执行过程中的对象；当程序的控制流经过此类对象的定义语句时，创建该对象；当到达定义所在的块的末尾时，销毁该对象；形参是一种自动对象，函数开始为形参申请空间，因为形参定义在函数体作用域之内，函数终止，形参销毁；(P225)；
7. 局部静态对象(local static object)：他的值在函数调用结束后仍然存在；第一次使用局部静态对象前创建并初始化它，当程序结束时局部静态对象才被销毁；如果局部静态变量没有显示的初始值，它将执行值初始化，内置类型的局部静态变量初始化为0；(P185)；

函数参数传递

1. 传递参数方法：①传值调用；②传引用调用；③传对常量引用的调用；④传右值引用调用；
2. 传值调用(called by value)和值传递(passed by value)：描述如何讲实参传递给非引用类型的形参；非引用类型的形参实际上是相应实参的一个副本；指针形参也属于值传递，拷贝的是指针的值；(P187)；
3. 传引用调用(called by reference)和引用传递(passed by reference)：描述如何将实参传递给引用类型的形参；形参被绑定到相应的实参上；可以避免拷贝较大的实参，还可以通过引用形参+返回额外信息；(P226)；
4. const形参和实参：形参的顶层const会被忽略，当形参有顶层const

时实参可以是常量对象也可以是非常量对象；因为顶层const被忽略因

此如图两种函数的参数完全一样；(P190)；

1. 指针或引用形参与const：可以用非常量初始化一个底层const形参，但是不能用一个底层const对象初始化非常量形参；(P191)；
2. 尽量使用常量引用：使用普通引用会限制函数所能接受的实参类型，如不能把const对象传给普通引用形参；函数嵌套中，外层函数的形参时常量引用，则内层函数如果对该形参进行引用传递或值传递中的指针传递必须也是常量引用或指向常量的指针；(P192)；
3. 传递多维数组：数组第二维以及后面所有维度的大小都是数组类型的一部分，不能省略；如void print (int (\*matrix)[10], int rowSize){};或者void print(int matrix[][10], int rowSize){};；(P196)；
4. Main处理命令行选项：int argc和char \*argv[]；(P196)；
5. 含有可变参数的函数：如果实参数量未知但类型相同可以使用initializer\_list类型的形参；该对象中的元素永远是常量值initializer\_list<int>相当于vector<const int>；如果想向该类型形参中传递一个值的序列必须放在花括号内；省略符形参(P189)；

函数的返回类型和return语句

1. return语句形式：有两种形式①return;②return expression;(P199)；
2. 无返回值函数：返回void的函数不要求有return而会隐式地执行return；通常用return实现在中间位置提前退出；
3. 有返回值函数：return语句返回值的类型必须与函数返回类型相同，或者能隐式地转换成函数的返回类型；
4. 返回值的实现：返回的值用于初始化调用点的一个临时量，该临时量就是函数调用的结果，返回值将被拷贝到调用点，即返回一个副本或临时对象；函数返回引用时，不会真正拷贝对象；不要返回局部对象的引用或指针，如retuen “Empty”；(P201)；
5. 引用返回左值：函数的返回类型决定函数调用是否是左值，调用返回引用的函数得到左值，其它返回类型得到右值；可以为返回类型是非常量引用的函数的结果赋值；(P202)；
6. 列表初始化返回值：①C++11规定,函数可以返回花括号包围的值得列表；②列表表示对函数返回的临时量进行初始化；③列表为空，临时量执行值初始化，否则，返回的值由函数的返回类型决定；④若返回内置类型，则花括号包围的列表最多包含一个值，且该值不应大于目标类型的空间；⑤**vector<int>{}和{}没有区别**；(P203)；
7. Main函数返回值：允许没有return，会隐式插入一条return 0;语句；(P204)；



1. 返回数组指针：①使用类型别名；②返回数组指针的函数形式如下；



1. 尾置返回类型(trailing return type)：尾置返回类型跟在形参列表后并以->符号开头，并在原本返回类型的位置放置auto；(P206)；
2. 使用decltype：(P206)；

函数重载和默认实参

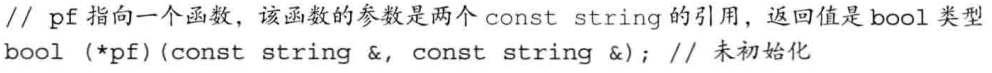
1. 重载函数(overloaded function)：函数名与其他函数相同的函数，多个重载函数必须在形参数量或形参类型上有所区别，返回类型不做区别；(P207)；
2. 重载和const形参：顶层const不影响传入函数的对象，不能实现重载；形参是指针或引用的底层const，可以重载；因为const不能转换成其他类型，因此只能把const对象传递给const形参；传递一个非常量对象或者指向非常量对象的指针时，编译器会优先选用非常量函数；(P208)；
3. const\_cast和重载：(P209)；
4. 函数匹配(function matching)和重载确定(overload resolution)：编译器解析重载函数调用的过程，在此过程中，实参的每个重载函数的形参列表逐一比较；
5. 调用重载函数的三种结果：最佳匹配；无匹配；二义性调用；
6. 最佳匹配(best match)：从一组重载函数中为调用选出的一个函数；如果存在最佳匹配，则选出的函数与其他所有可行函数相比，至少在一个实参上是更有的匹配，同时在其他实参的匹配上不会更差；(P225)；
7. 无匹配(no match)：是一种编译时发生的错误，原因是在函数匹配过程中所有函数的形参都不能与调用提供的实参匹配；(P226)；
8. 二义性调用(ambiguous call)：是一种编译时发生的错误，造成二义性调用的原因是在函数匹配时两个或多个函数提供的匹配一样好，编译器找不到唯一的最佳匹配；(P225)；
9. 重载与作用域：不同作用域中无法重载函数，外层作用于中的同名函数声明会被隐藏；(P210)；
10. 默认实参(default argument)：当调用缺少了某个实参时，为该实参指定的默认值，一旦某个形参被赋予了默认值，它后面所有形参都必须有默认值；函数调用时，实参按其位置解析，默认实参负责填补缺少的尾部实参；(P212)；
11. 默认实参声明：应该在函数声明中指定默认实参，并将该声明放在头文件中；在给定作用域中一个形参只能被赋予一次默认实参，后续声明只能给没有默认值的形参添加默认实参，且该形参的右侧所有形参必须都有默认值，如string screen(sz=24, sz=80, char);；(P212)；
12. 默认实参初始值：用作默认实参的名字在函数声明所在的作用域内解析，而这些名字的求值过程发生在函数调用时；在调用所在作用域定义相同名字的局部变量不会影响默认实参值，因为默认实参的变量名在声明所在作用域内解析不在调用所在作用域解析；(P213)；

函数匹配

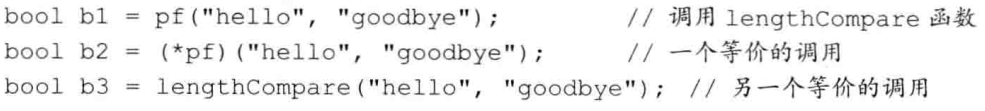
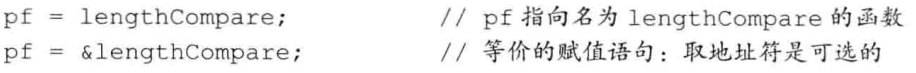
1. 候选函数(candidate function)：解析某次函数调用时考虑的一组函数；候选函数的名字应该在与函数调用使用的名字一致，并且候选函数的声明在在调用点所在的作用域之内；(P227)；
2. 可行函数(viable function)：是候选函数的子集；可行函数能匹配本次调用，它的形参数量与调用提供的实参数量相等，并且每个实参类型都能转换成相应的形参类型或相同；(P226)；
3. 匹配步骤：确定候选函数；确定可行函数，如果没有可行函数则报告无匹配函数错误，如果只有一个可行函数则匹配结束；如果有多个可行函数，在可行函数中选择与本次调用最匹配的函数，如果无法找到最佳匹配则报告二义性错误；(P218)；
4. 实参类型转换等级：①精确匹配（实参形参类型相同；实参为数组或函数类型变为对应指针类型；向实参添加或删除顶层const）；②通过const转换实现的匹配；③通过类型提升实现匹配（short转换为int或long）；④通过算数类型转换或指针转换实现匹配；⑤通过类类型转换实现的匹配；(P219)；
5. 函数匹配与const实参：传入非常量对象时，非常量对象初始化常量引用需要类型转换，而非常量形参则与非常量对象精确匹配；(P220)；

函数指针

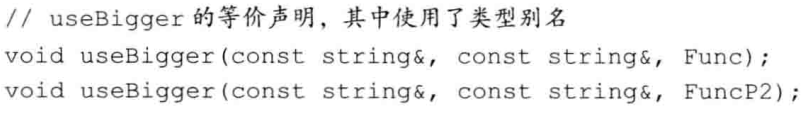
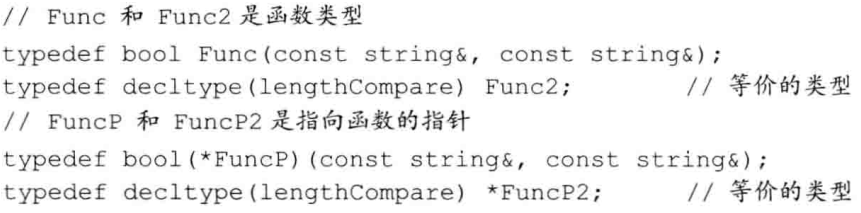
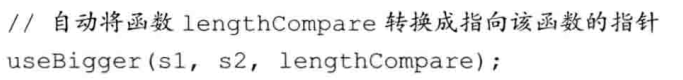
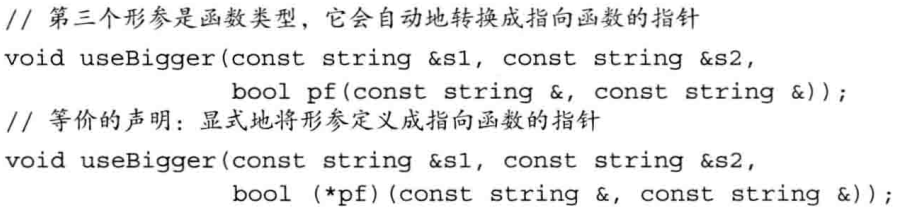
1. 函数的类型：由函数返回类型和形参类型决定，与函数名无关；(P221)；
2. 函数指针的声明：用指针替换函数名(P221)；



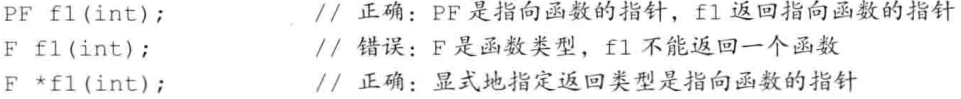
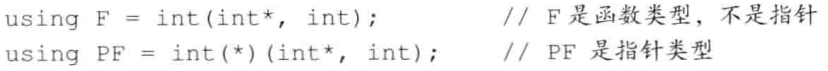
1. 分析：pf前面有\*表明pf是指针；右侧是形参列表表明pf指向的是函数；左侧返回值类型表明函数返回布尔值；\*pf须加括号(P221)；
2. 函数指针的使用：①将函数名作为一个值使用时，函数名变为指针；②指向函数的指针可以直接调用该函数，无须解引用；③不同函数类型指针不存在转换规则，但可以赋nullptr或0；(P221)；



1. 重载函数的指针：指针类型必须与重载函数中的一个精确匹配，编译器通过指针类型决定选用那个函数(P222)；
2. 函数指针形参：和数组类似，函数不能做形参但函数的指针可以；作为形参时，函数类型会自动转换为指向函数的指针；decltype返回函数类型不会自动转换；函数名可以直接作为实参使用，会自动转换成指针(P222)；



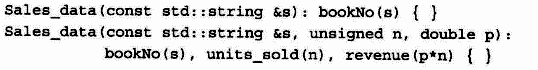
1. 返回指向函数的指针：和数组类似，不能返回函数但可以返回指向函数类型的指针；必须把返回类型写成指针形式，编译器不会自动转换；



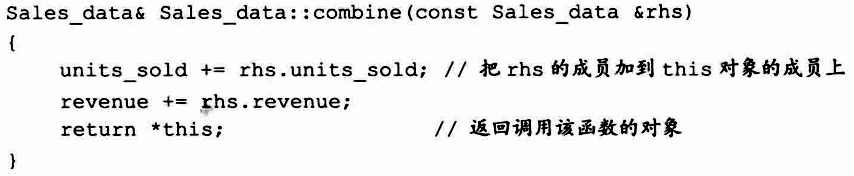
作用域运算符

1. 编译器处理类的步骤：先编译成员的声明，再编译函数体；因此，**成员函数可以随意使用类中的其它成员而无需在意这些成员出现的次序**；(P232)；
2. 在类外部定义成员函数：函数名Sales\_data::avg\_price使用作用域运算符来说明：定义一个名为avg\_price的函数，并且该函数被声明在类Sales\_data的作用域内，编译器识别函数名后会认为后面剩余代码是位于类的作用域内，因此代码块内可以直接访问成员；(P232)；

类基础

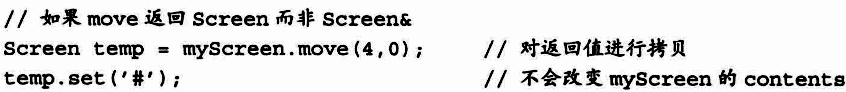
1. 定义类：以关键字struct开始，紧跟着类名和类体；类体由花括号包围形成了一个新的作用域；类体由侧的表示结束的花括号必须写一个分号，因为类体后面可以紧跟变量名以示对该类型对象的定义，分号表示声明符的结束；(P65)；
2. 类内初始值(in-class initializer)：在声明类的数据成员时同时提供的初始值，必须置于等号右侧或花括号内；(P70)；
3. 抽象数据类型(abstract data type)：封装（隐藏）了实现细节的数据结构；(P273)；
4. 成员函数(member function)：类的函数成员；普通的成员函数通过隐式的this指针与类对象绑定在一起；静态成员函数不与对象绑定也没有this指针；成员函数可以重载，此时隐式的this指针参与函数匹配的过程；(P274)；
5. 成员函数的定义：所有成员函数必须在类内声明，但成员函数体可以定义在类内也可以定义在类外；(P230)；
6. 类相关的非成员函数：如果非成员函数是类接口的组成部分，则这些函数的声明应该与类在同一头文件内；(P234)；
7. IO类不能被拷贝，只能通过引用来传递；(P234)；
8. 五大函数：析构函数、拷贝构造函数、移动构造函数、拷贝赋值运算符、移动赋值运算符；
9. 构造函数( constructor)：①用于初始化对象的一种特殊的成员函数，函数名与类名相同且没有返回类型；②构造函数应该给每个数据成员都赋一个合适的初始值；③构造函数不能被声明称const的，当创建类的一个const对象时，构造函数完成初始化后对象才取得常量属性，因此构造过程中可以写值；(P273)；
10. 默认构造函数( default constructor)：当没有提供任何实参时使用的构造函数；(P274)；
11. 合成默认构造函数(synthesized default constructor)：对于没有显式地定义任何构造函数的类，编译器为其创建（合成）的默认构造函数；该构造函数检查类的数据成员，如果提供了类内初始值，就用它执行初始化操作，否则就对数据成员执行默认初始化；(P274)；
12. 某些类不能依赖默认构造函数：①当类没有声明任何构造函数时，编译器才会自动地生成默认构造函数；②如果类包含内置类型或符合类型，则只有这些成员被赋予了类内初始值后才适合默认构造函数，否则内置类型或复合类型被默认初始化的值是未定义的；③类中包含其他类类型的成员，且这个成员没有默认构造函数，那么编译器无法初始化该成员；(P236)；
13. = default：一种语法形式，位于类内部默认构造函数声明语句的参数列表之后，要求编译器生成构造函数，而不管类是否已经有了其他构造函数；(P274)；
14. 构造函数初始值列表( constructor initialize list)：①说明一个类的数据成员的初始值，由形参列表后的冒号和冒号与花括号见得代码组成；②在构造函数体执行之前首先用初始值列表中的值初始化数据成员；③构造函数初始值是成员名字的一个列表，每个名字后面紧跟括号括起来的（或花括号内的）成员初始值；④未经初始值列表初始化的成员将被默认初始化；⑤构造函数不应该轻易覆盖类内初始值，除非新赋的值与原值不同；⑥若不能使用类内初始值，则所有构造函数都应该显示初始化每个内置类型的成员；(P273)；

常量成员函数、this指针与返回\*this的成员函数

1. this指针(this pointer)：①是一个隐式的值，作为额外的实参传递给类的每个非静态成员函数；②this指针指向代表函数调用者的对象，是一个常量指针；③在成员函数内部，可以直接使用调用该函数的对象的成员而无需成员访问运算符（.和->）；④任何对类成员的直接访问都被看做this的隐式引用；(P274)；
2. 常量成员函数(const member function)：一个成员函数，在其中不能修改对象的普通数据成员（即既不是static也不是mutable）；const成员函数的this指针是指向常量的指针，通过区分函数是否是const可以进行重载；由于this是隐式的并且不会出现在参数列表中，因此const在参数列表()之后；(P231)；
3. 定义一个返回this对象的函数：return语句解

引用this指针以获得执行该函数的对象，即上面

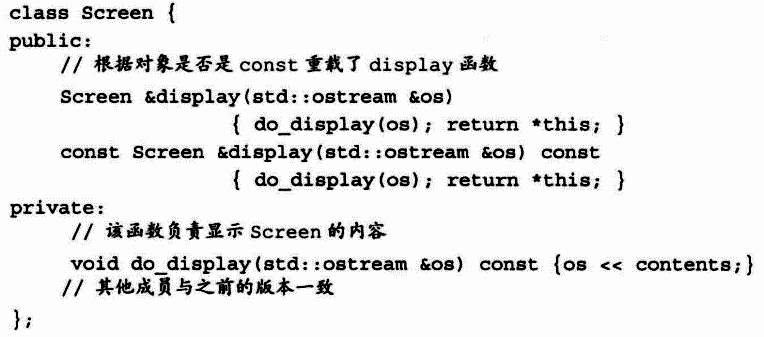
这个调用返回total的引用(P233)；

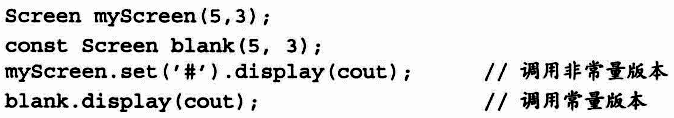
1. 返回引用的函数是左值的，意味着这些函数返回的是

对象本身而非对象副本；如果定义的返回类型不是引用，

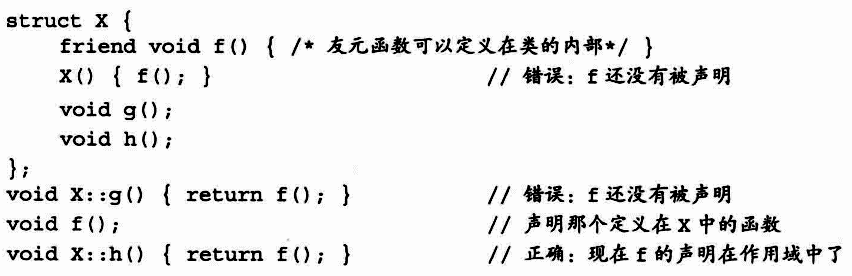
则move的返回值将是\*this的副本，因此调用set只能

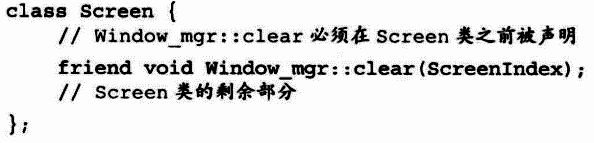
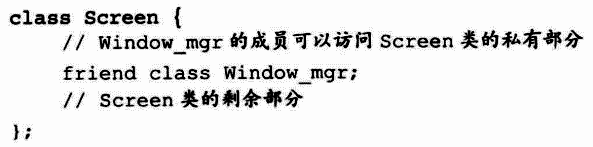
改变临时副本，而不能改变myScreen的值；(P247)；

1. 从const成员函数返回\*this：成员函数不改变参数时可定义为const成员，this将是一个指向const的指针，而\*this是const对象；当返回一个const的引用，就不能把该成员函数嵌入到一组动作的序列中去，如：myScreen.display(cout).set(‘\*’)；(P247)；
2. 基于const的重载：①常量对象上只能使用const成员函数，非常量对象上优先匹配非常量版本；当一个成员调用另一个成员时，this指针在其中隐式地传递；②const成员函数调用const成员函数，this指针直接隐式传递；③非const成员函数调用const成员函数，this指针隐式地从指向非常量的指针转换为指向常量的指针；当内层const成员函数执行完成，外层函数返回各自返回解引用this所得对象，在非常量版本中this指向非常量对象，因此非const成员函数返回非常量引用，而const成员则返回常量引用；(P248)；



访问控制与封装

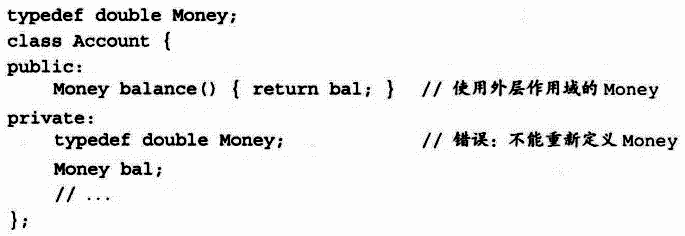
1. 访问说明符(access specifier)：包括关键字public和private；用于定义成员对类的用户可见还是只对类的友元和成员可见；在类中说明符可以出现多次，每个说明符的有效范围从它自身开始，到下一个说明符为止；(P273)；
2. 友元( friend)：类向外部提供其非公有成员访问权限的一种机制，只需要在返回类型前加friend关键字；友元的访问权限与成员函数一样；友元声明只能出现在类定义内部，但是在类内出现的具体位置不限；友元可以是类，也可以是函数；(P274)；
3. 友元的声明：有缘的生命仅仅制定了访问权限，调用某个友元函数还需要在友元声明外对函数进行一次声明；通常为使友元对类用户可见，把友元的函数声明与类放置在同一头文件中；(P242)；
4. 类之间的友元关系：如果一个类指定了友元类，如上图，则友元类的成员函数可以访问此类包括非公有成员在内所有成员；有缘关系不存在传递性，每个类负责控制自己的友元类或友元函数；(P251)；
5. 令成员函数作为友元：把一个成员函数声明成友元时，必须指明该函数属于哪个类，且必须组织结构以满足声明和定义的彼此依赖关系；首先定义 Window\_mgr类，其中声明clear函数，但是不能定义它,在 clear使用 Screen的成员之前必须先声明 Screen；接下来定义Screen,包括对于clear的友元声明；最后定义c1ear,此时它才可以使用 Screen的成员；(P252)；

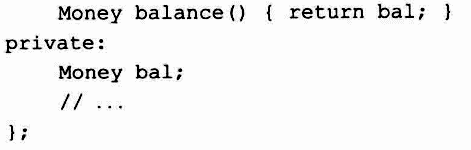
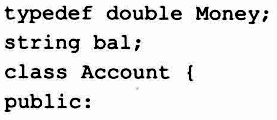


1. 函数重载和友元：如果一个类想把一组重载函数声明成它的友元，它需要对这组函数中的每一个分别声明；(P252)；
2. 友元声明和作用域：**即使在类内部定义友元函数，即友元声明加定义，也必须在类外部提供相应的声明才能使用该函数**；(P252)；
3. 类型成员(type member)：除定义数据和函数成员外，类还可以定义某种类型在类中的别名，称为类型成员；由类定义的类型名字也存在访问限制，可以是public或private；类型成员必须先定义后使用，因此一般放在类开头(P243)；
4. 可变数据成员( mutable data member)：这种成员水远不是const,即使它属于const对象；在const函数内可以修改可变数据成员；(P274)；

类的作用域

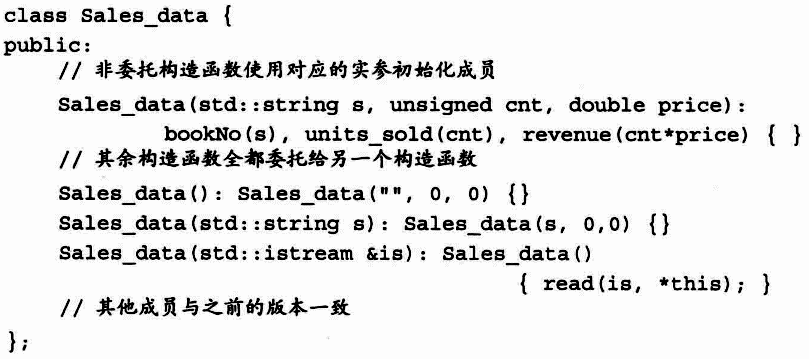
1. 作用域和定义在类外部的成员：一旦遇到类名，定义的剩余部分就在类的作用域之内了，这里的剩余部分包括参数列表和函数体；而返回类型出现在类名之前，所以它在类的作用域之外，因此如果要返回类成员必须明确指定哪个类定义了它；(P254)；
2. 两阶段：在名字所在块中寻找声明语句，只考虑在名字使用之前出现的生命；如果没有找到，继续查找外层作用域；成员函数体直到整个类可见后才会被处理，所以它能使用类中定义的任何名字；如果函数的定义和成员的声明被同时处理，那么只能在函数中使用已经出现过的名字；(P254)；
3. 用于类成员声明的名字查找：声明中使用的名字，包括返回类型或者参数列表中使用的名字都必须在使用前确保可见，而且只会查找类中在使用该名字之前出现的声明，若没找到则回到外层作用域查找；
4. 类型名要特殊处理：一般来说，内层作用域可以重新定义外层作用域中的名字，即使该名字已经在内层作用域中使用过；然而在类中如果成员使用了外层作用域中某个名字，则类不能在之后重新定义该名字；类型名通常出现在类的开始处，这样就能确保所有使用该类型的成员都出现在类名的定义之后；(P255)；



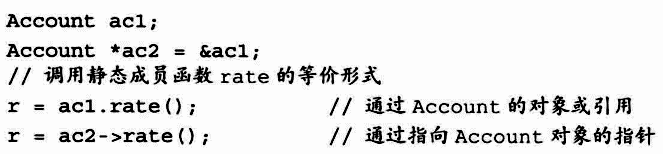


1. 成员定义中的普通块作用域的名字查找：在成员函数内（包括形参列表）查找该名字的声明，且只有在函数使用之前出现的声明才被考虑；如果在成员函数内没找到，在类内继续查找，且所有类成员都被考虑；类内没找到，则在成员函数定义之前的作用域内查找；(P255)；

构造函数

1. 数据成员的初始化和赋值：如果没有在构造函数的初始值列表中显式地初始化成员，则该成员在构造函数体之前执行默认初始化；函数体中只能通过“=”进行赋值操作；(P258)；
2. 构造函数的初始值有时必不可少：如果成员是const、引用或者属于某种未提供默认构造函数的类类型，必须通过构造函数初始值列表为这些成员提供初始值，因为随着构造函数体一开始执行初始化就完成了；(P259)；
3. 成员初始化顺序：成员初始化顺序与它们在类定义中出现的顺序一直；最好令构造函数初始值的顺序与成员声明的顺序保持一致；尽量避免使用某些成员初始化其他成员；(P259)；
4. 默认实参和构造函数：如果一个构造函数为所有实参都提供了默认实参，则实际上也定义了默认构造函数；(P260)；
5. 委托构造函数( delegating constructor)：委托构造函数的初始值列表只有一个入口，就是类名本身，指定类的另一个构造函数执行初始化操默认初始化；当一个构造函数委托给另一个构造函数时，受委托的构造函数的初始值列表和函数体被一次执行，然后才是委托者的函数体；(P274)；
6. 默认初始化：①当我们在块作用域内不使用任何初始值定义一个非静态变量或者数组时；②当一个类本身含有类类型的成员且使用合成的默认构造函数时；③当类类型的成员没有在构造函数初始值列表中显式地初始化时； (P262)；
7. 值初始化：①在数组初始化的过程中如果我们提供的初始值数量少于数组的大小时；②当我们不使用初始值定义一个局部静态变量时；③当我们通过书写形如T()的表达式显式地请求值初始化时，其中T是类型名；(P262)；
8. 默认构造函数的使用：去掉对象名后的空括号；(P263)；
9. 转换构造函数( converting constructor)：可以用一个实参调用的非显式构造函数；这样的函数隐式地将参数类型转换成类类型；编译器只会自动执行一步类型转换；(P264)；
10. 显式构造函数( explicit constructor)：替可用一个单独的实参调用但是不能用于隐式转换的构造函数；②通过在构造函数的声明之前加上explicit关键字就可以将其声明成显式构造函数；③关键字explicit只对一个实参的构造函数有效，因为需要多个实参的构造函数不能用于执行隐式转换；④只能在类内声明构造函数时使用explicit关键字，类外部定义时不能重复；(P265)；
11. 聚合类：所有成员都是public的；没有定义任何构造函数；没有类内初始值；(P266)；
12. 字面值常量类：数据成员都是字面值类型的聚合类；或者满足下列条件，数据成员都必须是字面值类型；类必须至少含有一个 constexpr构造函数；如果一个数据成员含有类内初始值，则内置类型成员的初始值必须是一条常量表达式，或者如果成员属于某种类类型，则初始值必须使用成员自己的constexpr构造函数；类必须使用析构函数的默认定义，该成员负责销毁类的对象；(P267)；

类的静态成员

1. 静态成员只需要在返回类型前或变量类型前加static关键字；
2. 静态成员存在于任何对象之外；静态成员函数不与任何对象绑定在一起，因此不含this指针，静态成员不能声明为const的；(P269)；
3. 使用类的静态成员：①使用作用域运算符直接访问静态成员；②使用类的对象、引用或指针来访问静态成员；③成员函数不通过作用域运算符可以直接使用静态成员；



1. 定义静态成员：既可以在类内部也可以在类外部定义，当在类外部定义静态成员时，不能重复static关键字，该关键字只能出现在类内部的声明语句中；通常不能在类内部初始化静态成员，且一旦定义就存在于整个函数生命周期中；
2. 静态成员的类内初始化：若静态成员时字面值常量类型，可以为静态成员提供const整数类型的类内初始值；即使类内初始化，也最好在类外部定义一下该成员；(P271)；
3. 静态数据成员可以是不完全类型；(P271)；