**一，开始**

**1，声明和定义**

C++语言支持"分别编译"。在a.cpp中定义了一个全局函数"void a(){}"，而在b.cpp中需要调用这个函数。可以分别地对它们进行编译，编译成目标文件之后再链接，程序可以正常运行。在b.cpp中，在调用函数之前，先声明一下这个函数"void a();"就可以了。

这是因为编译器在编译b.cpp的时候会生成一个符号表，像 "void a()" 这样的看不到定义的符号，就会被存放在这个表中。在进行链接的时候，编译器就会在别的目标文件中去寻找这个符号的定义。一旦找到了，程序也就可以顺利地生成了。

简单地说，"声明"只是声明这个符号的存在，"定义"则是把一个符号完完整整地描述出来。需要注意的是，一个符号在整个程序中可以被声明多次，但却要且仅要被定义一次。

**2，头文件的作用**

如果有一个很常用的函数在许多.cpp文件中都会被调用，只需要在一个文件中定义这个函数，而在其他的文件中声明这个函数就可以了。一个函数还好，声明起来也就一句话。但是，如果函数多了，有好几百个，那怎么办？

头文件便可以发挥它的作用了。头文件不用被编译。我们把所有的函数声明全部放进一个头文件中，当某一个.cpp 源文件需要它们时，可以通过一个宏命令 "#include" 包含进这个.cpp文件中，从而把它们的内容合并到.cpp文件中去。

**3，#include**

#include是一个来自C语言的宏命令，它在编译器进行编译之前，即在预编译的时候就会起作用。#include本身是没有其它任何作用与副功能的，它的作用是把它后面所写的那个文件的内容，完完整整地，一字不改地包含到当前的文件中来。

#ifndef SALES\_DATA\_H

#define SALES\_DATA\_H

XXXXXX

#endif

使用#pragma once可以获得相同的效果。但是，这种方式不支持跨平台！

**4，头文件内容**

头文件的作用就是被其他的.cpp包含进去的。它们本身并不参与编译，但它们的内容却在多个 .cpp 文件中得到了编译。

通过"定义只能有一次"的规则，我们很容易可以得出，头文件中应该只放变量和函数的声明，而不能放它们的定义。只能在头文件中写形如：extern int a; 和 void f(); 的句子。这些才是声明。如果写上 int a;或者 void f() {}这样的句子，那么一旦这个头文件被两个或两个以上的 .cpp 文件包含的话，编译器会立马报错。

**5，三个例外**

一，头文件中可以写const对象的定义。因为全局的const对象默认是没有extern的声明的，所以它只在当前文件中有效。static对象的定义也可以放进头文件。

二，头文件中可以写内联函数的定义。因为inline函数是需要编译器在遇到它的地方根据它的定义把它内联展开的，而并非是普通函数那样先声明再链接的（内联函数不会链接）

三，头文件中可以写类（class）的定义。关于类的定义的要求，跟内联函数是基本一样的。

一般，我们把类的定义放在头文件中，而把函数成员的实现代码放在.cpp文件中。不过，还有另一种办法。那就是直接把函数成员的实现代码也写进类定义里面。如果函数成员在类的定义体中被定义，那么编译器会视这个函数为内联的。

**6，综述**

.h文件中能包含：

类：

成员数据的声明，但不能赋值

静态数据成员的声明和定义，但不建议，只是个声明就好。

成员函数的声明

内联函数的定义

非类：

非类成员函数的声明

常数的定义：如：constint a=5;

静态函数的定义

**7，头文件中的保护措施**

如果头文件中只包含声明语句的话，它被同一个.cpp文件包含再多次都没问题。但是，一旦头文件中出现了上面三个例外中的一个，它被一个.cpp包含多次的话，问题就大了。因为这三个例外中的语法元素在一个源文件中只能出现一次。

使用和#ifndef，#endif和 #define使得编译器可以根据这个名字是否被定义，再决定要不要继续编译该头文中后续的内容。这个方法虽然简单，但是写头文件时一定记得写进去。