C语言中，头文件和源文件的关系（转）

转载 haibing_blog 2018-01-02 20:34:43 27748 收藏 149
分类专栏： C语言 文章标签： C 头文件 源文件

简单的说其实要理解C文件与头文件（即.h）有什么不同之处，首先需要弄明白编译器的工作过程，一般说来编译器会做以下几个过程：

1.预处理阶段
2.词法与语法分析阶段
3.编译阶段，首先编译成纯汇编语句，再将之汇编成跟CPU相关的二进制码，生成各个目标文件 (.obj文件)
4.连接阶段，将各个目标文件中的各段代码进行绝对地址定位，生成跟特定平台相关的可执行文件，当然，最后还可以用objcopy生成纯二进制码，也就是去掉了文件格式信息。（生成.exe文件）

编译器在编译时是以C文件为单位进行的，也就是说如果你的项目中一个C文件都没有，那么你的项目将无法编译，连接器是以目标文件为单位，它将一个或多个目标文件进行函数与变量的重定位，生成最终的可执行文件，在PC上的程序开发，一般都有一个main函数，这是各个编译器的约定，当然，你如果自己写连接器脚本的话，可以不用main函数作为程序入口！！！！

（main .c文件 目标文件 可执行文件 ）

有了这些基础知识，再言归正传，为了生成一个最终的可执行文件，就需要一些目标文件，也就是需要C文件，而这些C文件中又需要一个main函数作为可执行程序的入口，那么我们就从一个C文件入手，假定这个C文件内容如下：

```
#include <stdio.h>
#include "mytest.h"
int main(int argc,char **argv)
{
test = 25;
printf("test.....%d/n",test);
}
```

头文件内容如下：

```
int test;
```

现在以这个例子来讲解编译器的工作：
1.预处理阶段：编译器以C文件作为一 个单元，首先读这个C文件，发现第一句与第二句是包含一个头文件，就会在所有搜索路径中寻找这两个文件，找到之后，就会将相应头文件中再去处理宏，变量， 函数声明，嵌套的头文件包含等，检测依赖关系，进行宏替换，看是否有重复定义与声明的情况发生，最后将那些文件中所有的东东全部扫描进这个当前的C文件 中，形成一个中间“C文件”

2.编译阶段：在上一步中相当于将那个头文件中的test变量扫描进了一个中间C文件，那么test变量就变成了这个文件中的一个全局变量，此时就将所有这个中间C文件的所有变量，函数分配空间，将各个函数编译成二进制码，按照特定目标文件格式生成目标文件，在这种格式的目标文件中进行各个全局变量，函数的符号描述，将这些二进制码按照一定的标准组织成一个目标文件

3.连接阶段：将上一步生成的各个目标文件，根据一些参数，连接生成最终的可 执行文件，主要的工作就是重定位各个目标文件的函数，变量等，相当于将个目标文件中的二进制码按一定的规范合到一个文件中再回到C文件与头文件各写什么内容的话题上：理论上来说C文件与头文件里的内容，只要是C语言所支持的，无论写什么都可以的，比如你在头文件中写函数体，只要在任何一个C文件包含此头文件就可以将这个函数编译成目标文件的一部分（编译是以C文件为单位的，如果不在任何C文件中包含此头文件的话，这段代码就形同虚设），你可以在C文件中进 行函数声明，变量声明，结构体声明，这也不成问题！！！那为何一定要分成头文件与C文件呢？又为何一般都在头件中进行函数，变量声明，宏声明，结构体声明 呢？而在C文件

1.如果在头文件中实现一个函数体，那么如果在多个C文件中引用它，而且又同时编 译多个C文件，将其生成的目标文件连接成一个可执行文件，在每个引用此头文件的C文件所生成的目标文件中，都有一份这个函数的代码，如果这段函数又没有定 义成局部函数，那么在连接时，就会发现多个相同的函数，就会报错 。

2.如果在头文件中定义全局变量，并且将此全局变量赋初值，那么在多个引用此 头文件的C文件中同样存在相同变量名的拷贝，关键是此变量被赋了初值，所以编译器就会将此变量放入DATA段，最终在连接阶段，会在DATA段中存在多个 相同的变量，它无法将这些变量统一成一个变量，也就是仅为此变量分配一个空间，而不是多份空间，假定这个变量在头文件没有赋初值，编译器就会将之放入 BSS段，连接器会对BSS段的多个同名变量仅分配一个存储空间 。

3.如果在C文件中声明宏，结构体，函数等，那么我要在另一个C文件中引用相 应的宏，结构体，就必须再做一次重复的工作，如果我改了一个C文件中的一个声明，那么又忘了改其它C文件中的声明，这不就出了大问题了，程序的逻辑就变成 了你不可想象的了，如果把这些公共的东东放在一个头文件中，想用它的C文件就只需要引用一个就OK了！！！这样岂不方便，要改某个声明的时候，只需要动一 下头文件就行了 。

4.在头文件中声明结构体，函数等，当你需要将你的代码封装成一个库，让别人来用你的代码，你又不想公布源码，那么人家如何利 用你的库呢？也就是如何利用你的库中的各个函数呢？？一种方法是公布源码，别人想怎么用就怎么用，另一种是提供头文件，别人从头文件中看你的函数原型，这 样人家才知道如何调用你写的函数，就如同你调用printf函数一样，里面的参数是怎样的？？你是怎么知道的？？还不是看人家的头文件中的相关声明啊！！！当然这些东东都成了C标准，就算不看人家的头文件，你一样可以知道怎么使用。

c语言中.c和.h文件的困惑

本质上没有任何区别。只不过一般：
.h文件是头文件，内含函数声明、宏定义、结构体定义等内容.c文件是程序文件，内含函数实现，变量定义等内容。而且是什么后缀也没有关系，只不过编译器会默认对某些后缀的文件采取某些动作。你可以强制编译器把任何后缀的文件都当作c文件来编。

这样分开写成两个文件是一个良好的编程风格。

而且，比方说 我在aaa.h里定义了一个函数的声明，然后我在aaa.h的同一个目录下建立aaa.c， aaa.c里定义了这个函数的实现，然后是在main函数所在.c文件里#include这个aaa.h 然后我就可以使用这个函数了。 main在运行时就会找到这个定义了这个函数的aaa.c文件。这是因为：main函数为标准C/C++的程序入口，编译器会先找到该函数所在的文件。假定编译程序编译myproj.c（其中含main()）时，发现它include了mylib.h（其中声明了函数void test()），那么此时编译器将按照事先设定的路径（Include路径列表及代码文件所在的路径）查找与之同名的实现文件（扩展名为.cpp或.c，此例中为mylib.c），如果找到该文件，并在其中找到该函数（此例中为void test()）的实现代码，则继续编译；如果在指定目录找不到实现文件，或者在该文件及后续的各include文件中未找到实现代码，则返回一个编译错误.其实include的过程完全可以“看成”是一个文件拼接的过程，将声明和实现分别写在头文件及C文件中，或者将二者同时写在头文件中，理论上没有本质的区别。以上是所谓动态方式。对于静态方式，基本所有的C/C++编译器都支持一种链接方式被称为Static Link，即所谓静态链接。在这种方式下，我们所要做的，就是写出包含函数，类等等声明的头文件（a.h,b.h,...），以及他们对应的实现文件（a.cpp,b.cpp,...），编译程序会将其编译为静态的库文件（a.lib,b.lib,...）。在随后的代码重用过程中，我们只需要提供相应的头文件（.h）和相应的库文件（.lib），就可以使用过去的代码了。相对动态方式而言，静态方式的好处是实现代码的隐蔽性，即C++中提倡的“接口对外，实现代码不可见”。有利于库文件的转发.c文件和.h文件的概念与联系。

如果说难题最难的部分是基本概念，可能很多人都会持反对意见，但实际上也确实如此。我高中的时候学物理，老师抓的重点就是概念——概念一定要搞清，于是难题也成了容易题。如果你能分析清楚一道物理难题存在着几个物理过程，每一个过程都遵守那一条物理定律(比如动量守恒、牛II定律、能量守恒)，那么就很轻松的根据定律列出这个过程的方程，N个过程必定是N个N元方程，难题也就迎刃而解。即便是高中的物理竞赛难题，最难之处也不过在于：

(1)、混淆你的概念，让你无法分析出几个物理过程，或某个物理过程遵循的那条物理定律；
(2)、存在高次方程，列出方程也解不出。而后者已经是数学的范畴了，所以说，最难之处还在于掌握清晰的概念；

程序设计也是如此，如果概念很清晰，那基本上没什么难题(会难在数学上，比如算法的选择、时间空间与效率的取舍、稳定与资源的平衡上)。但是，要掌握清晰的概念也没那么容易。比如下面这个例子，看看你有没有很清晰透彻的认识。

```
//a.h
void foo();


//a.c
#include "a.h"   //我的问题出来了：这句话是要，还是不要?
void foo()
{
    return;
}


//main.c
#include "a.h"
int main(int argc, char *argv[])
{
   foo();
    return 0;
}
```

针对上面的代码，请回答三个问题：

a.c 中的 #include "a.h" 这句话是不是多余的?
为什么经常见 xx.c 里面 include 对应的 xx.h?
如果 a.c 中不写，那么编译器是不是会自动把 .h 文件里面的东西跟同名的 .c 文件绑定在一起?
（请针对上面3道题仔细考虑10分钟，莫要着急看下面的解释。:) 考虑的越多，下面理解的就越深。）

好了，时间到！请忘掉上面的3道题，以及对这三道题引发出你的想法，然后再听我慢慢道来。正确的概念是：从C编译器角度看，.h和.c皆是浮云，就是改名为.txt、.doc也没有大的分别。换句话说，就是.h和.c没啥必然联系。.h中一般放的是同名.c文件中定义的变量、数组、函数的声明，需要让.c外部使用的声明。这个声明有啥用？只是让需要用这些声明的地方方便引用。因为 #include "xx.h" 这个宏其实际意思就是把当前这一行删掉，把 xx.h 中的内容原封不动的插入在当前行的位置。由于想写这些函数声明的地方非常多（每一个调用 xx.c 中函数的地方，都要在使用前声明一下子），所以用 #include "xx.h" 这个宏就简化了许多行代码——让预处理器自己替换好了。也就是说，xx.h 其实只是让需要写 xx.c 中函数声明的地方调用（可以少写几行字），至于 include 这个 .h 文件是谁，是 .h 还是 .c，还是与这个 .h 同名的 .c，都没有任何必然关系。
　　这样你可能会说：啊？那我平时只想调用 xx.c 中的某个函数，却 include了 xx.h 文件，岂不是宏替换后出现了很多无用的声明？没错，确实引入了很多垃圾，但是它却省了你不少笔墨，并且整个版面也看起来清爽的多。鱼与熊掌不可得兼，就是这个道理。反正多些声明（.h一般只用来放声明，而放不定义，参见拙著“过马路，左右看”）也无害处，又不会影响编译，何乐而不为呢?
翻回头再看上面的3个问题，很好解答了吧?

答：不一定。这个例子中显然是多余的。但是如果.c中的函数也需要调用同个.c中的其它函数，那么这个.c往往会include同名的.h，这样就不需要为声明和调用顺序而发愁了（C语言要求使用之前必须声明，而include同名.h一般会放在.c的开头）。有很多工程甚至把这种写法约定为代码规范，以规范出清晰的代码来。
答：1中已经回答过了。
答：不会。问这个问题的人绝对是概念不清，要不就是想混水摸鱼。非常讨厌的是中国的很多考试出的都是这种烂题，生怕别人有个清楚的概念了，绝对要把考生搞晕。
搞清楚语法和概念说易也易，说难也难。窍门有三点：

不要晕着头工作，要抽空多思考思考，多看看书；
看书要看好书，问人要问强人。烂书和烂人都会给你一个错误的概念，误导你；
勤能补拙是良训，一分辛苦一分才；

（1）通过头文件来调用库功能。在很多场合，源代码不便（或不准）向用户公布，只要向用户提供头文件和二进制的库即可。用户只需要按照头文件中的接口声明来调用库功能，而不必关心接口怎么实现的。编译器会从库中提取相应的代码。
（2）头文件能加强类型安全检查。如果某个接口被实现或被使用时，其方式与头文件中的声明不一致，编译器就会指出错误，这一简单的规则能大大减轻程序员调试、改错的负担。
头文件用来存放函数原型。

头文件如何来关联源文件?
这个问题实际上是说，已知头文件“a.h”声明了一系列函数(仅有函数原型,没有函数实现)，“b.cpp”中实现了这些函数，那么如果我想在“c.cpp”中使用“a.h”中声明的这些在“b.cpp”中实现的函数，通常都是在“c.cpp”中使用#include “a.h”,那么c.cpp是怎样找到b.cpp中的实现呢?
其实.cpp和.h文件名称没有任何直接关系，很多编译器都可以接受其他扩展名。
谭浩强老师的《C程序设计》一书中提到，编译器预处理时，要对#include命令进行“文件包含处理”：将headfile.h的全部内容复制到#include “headfile.h”处。这也正说明了，为什么很多编译器并不care到底这个文件的后缀名是什么----因为#include预处理就是完成了一个“复制并插入代码”的工作。
程序编译的时候，并不会去找b.cpp文件中的函数实现，只有在link的时候才进行这个工作。我们在b.cpp或c.cpp中用#include “a.h”实际上是引入相关声明，使得编译可以通过，程序并不关心实现是在哪里，是怎么实现的。源文件编译后成生了目标文件（.o或.obj文件），目标文件中，这些函数和变量就视作一个个符号。在link的时候，需要在makefile里面说明需要连接哪个.o或.obj文件（在这里是b.cpp生成的.o或.obj文件），此时，连接器会去这个.o或.obj文件中找在b.cpp中实现的函数，再把他们build到makefile中指定的那个可以执行文件中。
在VC中，一帮情况下不需要自己写makefile，只需要将需要的文件都包括在project中，VC会自动帮你把makefile写好。
通常，编译器会在每个.o或.obj文件中都去找一下所需要的符号，而不是只在某个文件中找或者说找到一个就不找了。因此，如果在几个不同文件中实现了同一个函数，或者定义了同一个全局变量，链接的时候就会提示“redefined”.