C语言中的.h文件和我认识由来已久，其使用方法虽不十分复杂，但我却是经过了几个月的“不懂”时期，几年的“一知半解”时期才逐渐认识清楚他的本来面目。揪其原因，我的 驽钝和好学而不求甚解固然是原因之一，但另外还有其他原因。原因一：对于较小的项目，其作用不易被充分开发，换句话说就是即使不知道他的详细使用方法，项 目照样进行，程序在计算机上照样跑。原因二：现在的各种C语言书籍都是只对C语言的语法进行详细的不能再详细的说明，但对于整个程序的文件组织构架却只字不提，找了好几本比较著名的C语言著作，却没有一个把.h文件的用法写的比较透彻的。下面我就斗胆提笔，来按照我对.h的认识思路，向大家介绍一下。

让我们的思绪乘着时间机器回到大学一年级。C原来老师正在讲台上讲着我们的第一个C语言程序: Hello world!

 文件名 First.c

main()

{

     printf(“Hello world!”);

}

     例程-1

看看上面的程序，没有.h文件。是的，就是没有，世界上的万物都是经历从没有到有的过程的，我们对.h的认识，我想也需要从这个步骤开始。这时确实不需要.h文件，因为这个程序太简单了，根本就不需要。那么如何才能需要呢？让我们把这个程序变得稍微复杂些，请看下面这个，

文件名 First.c

 printStr()

{

     printf(“Hello world!”);

}

main()

{

printStr();

}

     例程-2

还是没有, 那就让我们把这个程序再稍微改动一下.

文件名 First.c

main()

{

printStr();

}

 printStr()

{

     printf(“Hello world!”);

}

     例程-3

等等，不就是改变了个顺序嘛, 但结果确是十分不同的. 让我们编译一下例程-2和例程-3,你会发现例程-3是编译不过的.这时需要我们来认识一下另一个C语言中的概念:作用域.

我们在这里只讲述与.h文件相关的顶层作用域, 顶层作用域就是从声明点延伸到源程序文本结束, 就printStr()这个函数来说，他没有单独的声明,只有定义,那么就从他定义的行开始,到first.c文件结束, 也就是说,在在例程-2的main()函数的引用点上,已经是他的作用域. 例程-3的main()函数的引用点上，还不是他的作用域,所以会编译出错. 这种情况怎么办呢? 有两种方法 ,一个就是让我们回到例程-2, 顺序对我们来说没什么, 谁先谁后不一样呢，只要能编译通过,程序能运行, 就让main()文件总是放到最后吧. 那就让我们来看另一个例程,让我们看看这个方法是不是在任何时候都会起作用.

文件名 First.c  
   play2()  
   {

 ……………….

 play1();

 ………………..

   }  
   play1(){  
    ……………..

 play2();         
    ……………………  
   }

main()

{

play1();

}

例程-4

也许大部分都会看出来了，这就是经常用到的一种算法, 函数嵌套, 那么让我们看看, play1和play2这两个函数哪个放到前面呢?

这时就需要我们来使用第二种方法,使用声明.

文件名 First.c

play1();

play2();

play2()

{

 ……………….

 play1();

 ………………..  
    }  
    play1()  
    {  
     …………………….

 play2();

 ……………………  
    }

main()

{

play1();

}

例程-4

经历了我的半天的唠叨, 加上四个例程的说明,我们终于开始了用量变引起的质变, 这篇文章的主题.h文件快要出现了。

一个大型的软件项目,可能有几千个,上万个play, 而不只是play1,play2这么简单, 这样就可能有N个类似 play1(); play2(); 这样的声明,这个时候就需要我们想办法把这样的play1(); play2(); 也另行管理, 而不是把他放在.c文件中, 于是.h文件出现了.

文件名 First.h

play1();

play2();

文件名 First.C

#include “first.h”

play2()

{

 ……………….

 play1();

 ………………..

}  
    play1();

{

……………………..

     play2();

  ……………………

}

main()

{

play1();

}

例程-4

各位有可能会说,这位janders大虾也太罗嗦了，上面这些我也知道, 你还讲了这么半天, 请原谅, 如果说上面的内容80%的人都知道的话,那么我保证,下面的内容,80%的人都不完全知道. 而且这也是我讲述一件事的一贯作风,我总是想把一个东西说明白,让那些刚刚接触C的人也一样明白.

上面是.h文件的最基本的功能, 那么.h文件还有什么别的功能呢? 让我来描述一下我手头的一个项目吧.

这个项目已经做了有10年以上了，具体多少年我们部门的人谁都说不太准确,况且时间并不是最主要的，不再详查了。是一个通讯设备的前台软件, 源文件大小共 51.6M, 大小共1601个文件, 编译后大约10M, 其庞大可想而知, 在这里充斥着错综复杂的调用关系,如在second.c中还有一个函数需要调用first.c文件中的play1函数, 如何实现呢?

Second.h 文件

play1();

second.c文件

\*\*\*()

{

…………….

Play();

……………….

}

例程-5

在second.h文件内声明play1函数，怎么能调用到first.c文件中的哪个play1函数中呢? 是不是搞错了，没有搞错, 这里涉及到c语言的另一个特性:存储类说明符.

C语言的存储类说明符有以下几个, 我来列表说明一下

|  |  |
| --- | --- |
| **说明符** | **用    法** |
| **Auto** | *只在块内变量声明中被允许, 表示变量具有本地生存期.* |
| ***Extern*** | *出现在顶层或块的外部变量函数与变量声明中，表示声明的对象具有静态生存期, 连接程序知道其名字.* |
| ***Static*** | *可以放在函数与变量声明中,在函数定义时,只用于指定函数名,而不将函数导出到链接程序,在函数声明中,表示其后边会有定义声明的函数,存储类型static.在数据声明中,总是表示定义的声明不导出到连接程序.* |

无疑, 在例程-5中的second.h和first.h中,需要我们用extern标志符来修饰play1函数的声明,这样,play1()函数就可以被导出到连接程序,也就是实现了无论在first.c文件中调用,还是在second.c文件中调用,连接程序都会很聪明的按照我们的意愿,把他连接到first.c文件中的play1函数的定义上去, 而不必我们在second.c文件中也要再写一个一样的play1函数.

但随之有一个小问题, 在例程-5中,我们并没有用extern标志符来修饰play1啊, 这里涉及到另一个问题, C语言中有默认的存储类标志符.C99中规定, 所有顶层的默认存储类标志符都是extern . 原来如此啊, 哈哈. 回想一下例程-4, 也是好险, 我们在无知的情况下, 竟然也误打误撞,用到了extern修饰符, 否则在first.h中声明的play1函数如果不被连接程序导出,那么我们在在play2()中调用他时, 是找不到其实际定义位置的 .

那么我们如何来区分*哪个头文件中的声明在其对应的.c文件中有定义,而哪个又没有呢?*这也许不是必须的，因为无论在哪个文件中定义，聪明的连接程序都会义无返顾的帮我们找到，并导出到连接程序, 但我觉得他确实必要的. 因为我们需要知道这个函数的具体内容是什么,有什么功能, 有了新需求后我也许要修改他，我需要在短时间内能找到这个函数的定义, 那么我来介绍一下在C语言中一个人为的规范**:**

*在.h文件中声明的函数,如果在其对应的.c文件中有定义,那么我们在声明这个函数时,不使用extern修饰符, 如果反之,则必须显示使用extern修饰符.*

这样,在C语言的.h文件中,我们会看到两种类型的函数声明. 带extern的,还不带extern的, 简单明了,一个是引用外部函数，一个是自己生命并定义的函数.

最终如下:

Second.h 文件

Extern play1();

上面洋洋洒洒写了那么多都是针对函数的，而实际上.h文件却不是为函数所御用的. 打开我们项目的一个.h文件我们发现除了函数外,还有其他的东西, 那就是全局变量.

在大型项目中，对全局变量的使用不可避免, 比如,在first.c中需要使用一个全局变量G\_test, 那么我们可以在first.h中,定义 TPYE G\_test. 与对函数的使用类似, 在second.c中我们的开发人员发现他也需要使用这个全局变量, 而且要与first.c中一样的那个, 如何处理? 对,我们可以仿照函数中的处理方法, 在second.h中再次声明TPYE G\_test, 根据extern的用法,以及c语言中默认的存储类型, 在两个头文件中声明的TPYE G\_test,其实其存储类型都是extern, 也就是说不必我们操心, 连接程序会帮助我们处理一切. 但我们又如何区分全局变量哪个是定义声明,哪个是引用声明呢?这个比函数要复杂一些, 一般在C语言中有如下几种模型来区分:

1、**初始化语句模型**

顶层声明中，存在初始化语句是，表示这个声明是定义声明，其他声明是引用声明。C语言的所有文件之中，只能有一个定义声明。

按照这个模型，我们可以在first.h中定义如下TPYE G\_test=1；那么就确定在first中的是定义声明，在其他的所有声明都是引用声明。

**2、省略存储类型说明**

**在这个模型中，所有引用声明要显示的包括存储类extern，而每个外部变量的唯一定义声明中省略存储类说明符。**

这个与我们对函数的处理方法类似，不再举例说明。

    这里还有一个需要说明，本来与本文并不十分相关，但前一段有个朋友遇到此问题，相信很多人都会遇到，那就是数组全局变量。

他遇到的问题如下：

在声明定义时，定义数组如下：

int G\_glob[100];

在另一个文件中引用声明如下：

int \* G\_glob;

在vc中，是可以编译通过的，这种情况大家都比较模糊并且需要注意，数组与指针类似，但并不等于说对数组的声明起变量就是指针。上面所说的的程序在运行时发现了问题，在引用声明的那个文件中，使用这个指针时总是提示内存访问错误，原来我们的连接程序并不把指针与数组等同，连接时，也不把他们当做同一个定义，而是认为是不相关的两个定义，当然会出现错误。正确的使用方法是在引用声明中声明如下：

int G\_glob[100];

并且最好再加上一个extern，更加明了。

extern int G\_glob[100];

    另外需要说明的是，在引用声明中由于不需要涉及到内存分配，可以简化如下，这样在需要对全局变量的长度进行修改时，不用把所有的引用声明也全部修改了。

extern int G\_glob[];

    C语言是现今为止在底层核心编程中，使用最广泛的语言，以前是，以后也不会有太大改变，虽然现在java,.net等语言和工具对c有了一定冲击，但我们看到在计算机最为核心的地方，其他语言是无论如何也代替不了的，而这个领域也正是我们对计算机痴迷的程序员所向往的。

https://blog.csdn.net/st19890625/article/details/38977029