# c语言综合研究四

## 已解决问题

tcc编译链接问题整合。

**基础应用：**

**tlib.exe的使用：**

（1）/C:大小写敏感标志；

（2）/E:建立扩展字典,加速大的库文件的连接过程；

（3）动作符。有5种：“＋"、“－"、“\*"、“－\*"或“\*－"、“－＋"或“＋－"。“＋"是把指定的文件加到指定的库中； “－"从库中删除指定的模块；“\*"将相应的模块从库中抽取并写到指定的文件中，原库不变；“－\*"或“\*－"是将库中指定的模块拷贝到指定的文件中，然后把该模块从库中删除；“－＋"或“＋－"是将指定的模块用指定的文件或模块代替。

（4）查看列表。用tlib \*.lib ,\*.lst可以查看\*.lib文件中的列表。查看\*.lst文件。

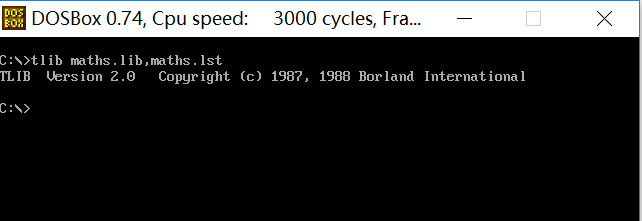
**Tlink.exe的使用：**

在编译时tcc.exe会直接调用tlink.exe连接，但如果手动连接的话就必须把连接需要的文件写上，如emu.lib等。分别制定用小模式连接\*\_1.obj，指出生成的目标文件是\*\_1.exe和需要用到的库文件有cs.lib、emu.lib、maths.lib。

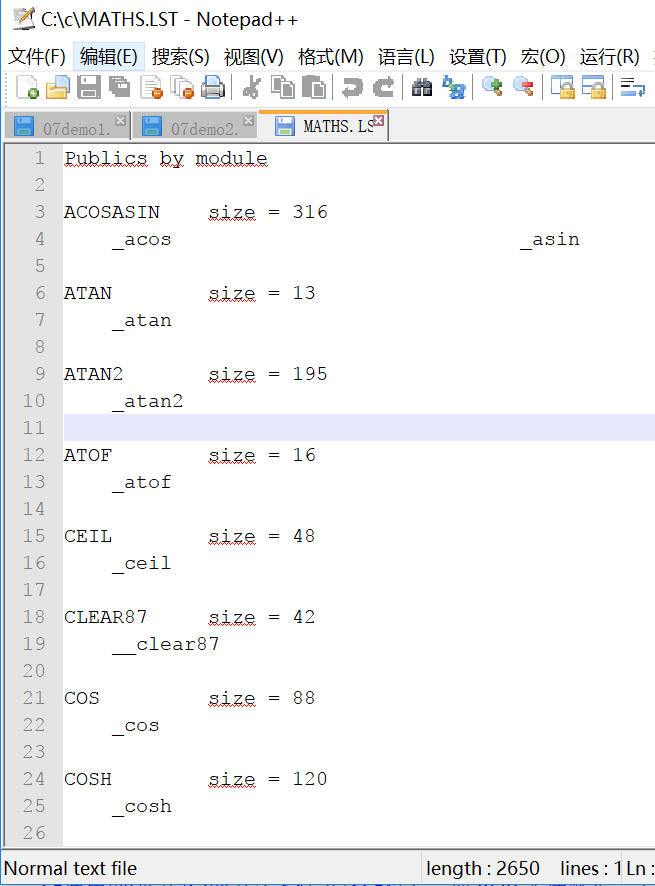
**相关问题：**

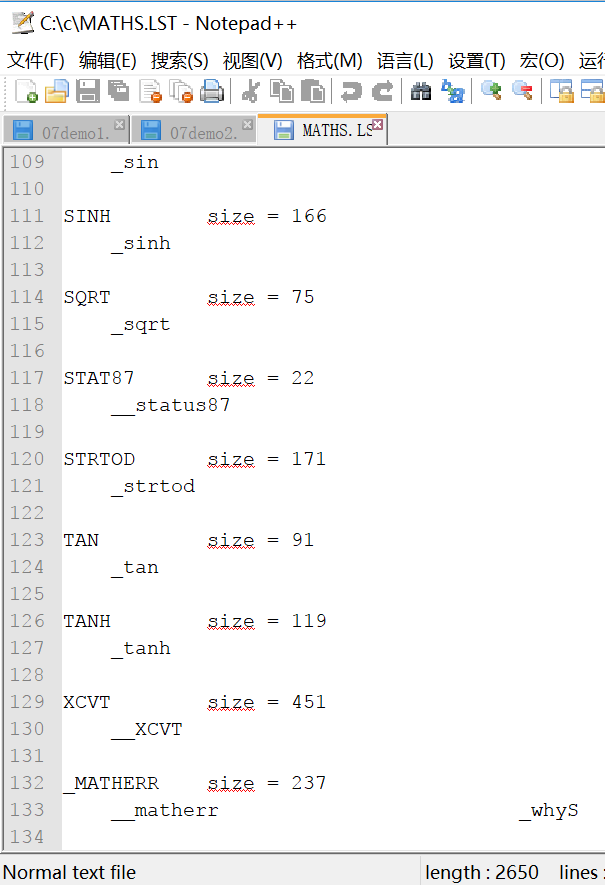
1. **库的组织的方式。相关性文件进行打包处理，比如maths.lib。（高内聚低耦合）。**

Tlib maths.lib,maths.lst:



用notpad++打开生成的maths.lst文件：





可以看到，maths.lib文件中，存在很多数学运算相关的函数，比如“tan”、“sqrt”等。这样既可以看到利用tlib看到maths.lib文件中的大致内容，同时也可以观察出其是多个相关文件的打包。

1. （**学长的）链接器创建空文件，外部标号标记出来，识别ASCII码符号，在相**

**应的库中查找相应的符号，进行链接。举例：call 下一条指令；tlib /E 建立索引。此处还可以来联系一下之前的\*.map文件。**

一般的大型工程都会在生成可执行文件的同时让链接器生成一个map文件，从而大致查看一下可执行文件中符号的内存布局以及从哪里引入可执行文件。这个通常对于小型工程是作用不大，因为代码就那么多，随便grep一下就知道符号定义位置了。但是对于一些大型工程或者涉及了比较多的第三方库、或者涉及了比较多的功能模块的时候，就需要知道这些符号是在哪里定义，或者说如果一个符号引用了但是没有知道函数定义，此时也需要找到这个符号是哪个模块引入的，为什么需要，所以需要一些通用的(形式化)的方法来搜索这些符号，而map文件就是一个比较好的切入点。

1. **针对静态库、动态库占用内存资源问题。对比两者的实现。**

静态动态的关键区别（学长的）：加载的时间不同。

还有什么其他异同：针对静态库和动态库中存放的文件。静态库是多个\*.obj文件的集合，有的同学查出资料说动态库是可执行程序文件的集合。

动态库、静态库都是模块。

动态库是调用的时候加载，因为如果一运行就加载，那动态库和静态库还有什么分别呢？

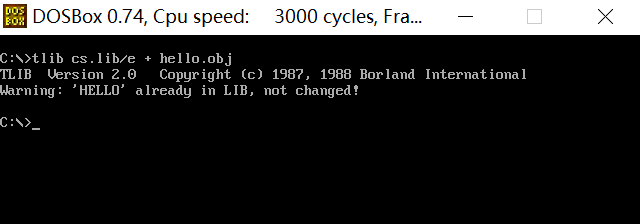
库: 指由标准常用函数编译而成的文件，旨在提高常用函数的可重用性，减轻开发人员负担。常用的sdtio.h，math.h等。库便是C函数库的冰山一角。  
  1.静态库:指编译链接阶段将整个库复制到可执行文件  
   1.1优点：静态链接的程序不依赖外界库支持，具有良好的可移植性。  
   1.2缺点: 每次库更新都需要重新编译程序，即使更新很小或只是局部。  
   1.3缺点：每个静态链接的程序都有一份库文件，存储时增加了硬盘空间消耗，运行时则增加了内存消耗。  
  2.动态库:指直到运行时才将库链接到可执行程序  
   1.1优点: 动态链接方式的程序不需要包含库（编辑链接时节省时间），占用的空间小很多。  
   1.2优点: 运行时系统内存只需提供一个共享库给所有程序动态链接，内存消耗减少。  
   1.3缺点: 需要系统中动态库支持才可运行，可能有动态库不兼容问题

1. **疑问 ：在一个现有的库文件中添加\*.obj文件，有的出现了变小的现象，有的没有变小。**

原本的cs.lib属性：



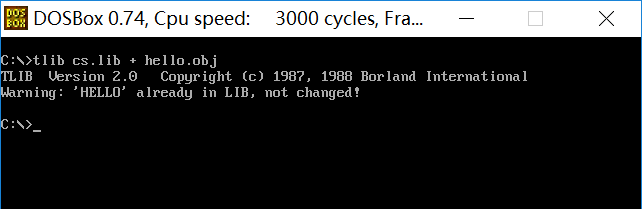
tlib cs.lib/e + hello.obj



执行之后：



tlib cs.lib + hello.obj



执行之后：



区别就是“/e”,索引。

1. **Debug--》调试，是一种运行状态。（与会同学提出）**

## 未解决问题

1. **（学长的）写一个程序，以静态链接库的方式模拟动态链接库的功能。**

程序功能：  
运行程序，  
1、用户输入自己要调用的函数名和参数。  
2、程序将被调用到的函数加载进内存。（即此函数在程序运行中才开始进入内存）  
3、该函数被运行，返回值显示在屏幕上。

设计：

基础：

第一部分：

建立和管理自己的静态lib库的程序libmanage.exe.

a要求：建立并添加lib库的功能，执行，libmanage mylib.lib add

(自己的tlib),

main函数接参数，二级指针。

函数新建文件，读写文件。File等open等。

b要求：

数据处理的两个

建立索引，mylib.index文件，通过索引来找到自己的库函数，就是函数的起始地址和函数的长度。（一个函数一个obj）

第二部分

（可以用system函数）

调用部分：

提示用户输入调用的函数名称

提示用户输入提供给函数的参数。

a:接收调用的函数（scanf）。

b:把函数通过索引从mylib.lib中取出自己要调用的函数放到linshi.obj

C：写个程序，把调用的file类函数。为了实现动态，不可能有obj，建立临时的linshi2.c文件中

D:用system函数调用tcc。生成了.exe

E:再用system函数调用这个.exe，返回值要显示在用户的屏幕上。

扩展要求：

现在的一个lib很多obj库很粗糙，可以用“二分法”算法,先排序，再查找，提高搜索效率。

# c语言综合研究五

## 已解决问题

**1.（学长的）指令“ \*c = \*c + 1;”c为int\*类型，这条指令读写了几次内存？**

1>.读c的内容；

2>.读\*c的内容；

\*3>.读c的内容；

4>.写\*c。

这里注意一下：\*c在算过一次地址后要再算一遍。

1. **（学长的）指令“c = &ch;”c为int\*类型，ch为int类型，这条指令读写了几次内存？**

ch的地址并没有存放在内存中，所以访问&ch并不需要访问内存）写一次。

相关汇编代码：

Mov ax,offset ch

Mov [c],ax

1. **（学长的）Push [bx]**

运行这一句，访问了读指令、算地址、取一次数据、入栈4次指令。

**4.（学长的）看一下研究5中的第3个和第4个for循环语句，比较两者不同。**联系C语言机制。

反汇编比较：两者段地址，偏移地址相关指令执行顺序不同。

可以注意一下运算符优先级。

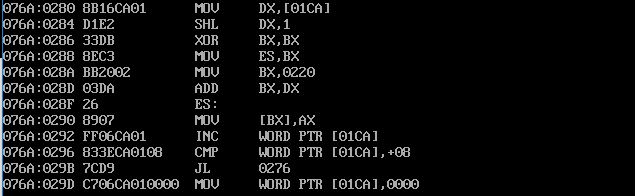
第3个for循环：

for(n = 0;n < 8;n++)

{

((int far\*)0x220)[n] = \*(a + n);

}



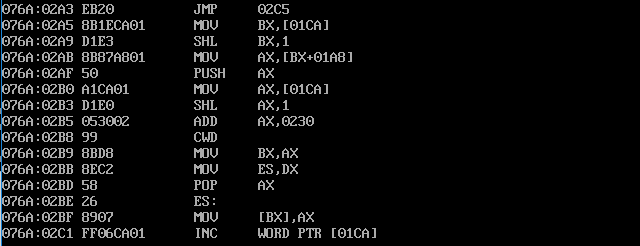
第4个for循环

for(n = 0;n < 8;n++)

{

\*(int far\*)(0x230 + n\*2) = \*(&a[0] + n);

}





## 未解决问题

## 1.（拓展）一级指针、二级指针、指针数组、数组指针等。

含义；

如何给他们赋值；

他们之间如何互相赋值；

怎样用是确实无警告无错误的。

还有“+”运算等。

**研究感想（心得体会）**

对于强类型语言c语言中，当然不止是c语言，这些语言中都有很多的描述性语言，但是有时候当一件拥有大量陌生名词的学习事物充斥在初学者身边时，很多时候，其实是排斥的。而且有很多概念如果拿以前已经学习吸收的知识组装成新知识的理解框架之后，相比于直接抛出这个陌生的概念，学习效果可能也会很不错。“=”右边的表达式中，放在变量、数字之前的，目前来看大部分看作“强制类型转换”来理解，特例如果有可以举出来，这样对于大家理解指针函数、函数指针、指向指针的指针等较复杂概念与一般化的（int）、（char）等大家熟悉的强制类型转化性质进行统一和类比，更容易消化吸收。“b =（int） a”、“ b =（int \*） a”、“ b =（int \*\*） a”等等，甚至将“b =& a”，也就是“b =（&）a”，也归入“强制转换类型”，好像也说的通。还有“b =\* a”，也就是“b =（\*）a”，从这个角度，我自己觉得也是。然后如果p是一个指向字符的指针，“b = p[3]”，这个呢？好像有点不像了，可是之前说的等价式：“b = \*(p + 3)”,也就是“b = （\*）(p + 3)”，这样好像和前面我认为的同样属于强制类型转换应用的“b =（\*）a”有点像呀，只不过这里（p+3）,抽象化，两者都是指针，属性不变呀，所以，这种也算吗？嗯，有点不确定。学长，您说呢？还有其他的什么情况可以归到这里呢？还需要再想想，从长计议，好好总结。