**1.用tcc将下面的程序编译为.Obj文件**

int f(void)

{

return 1;

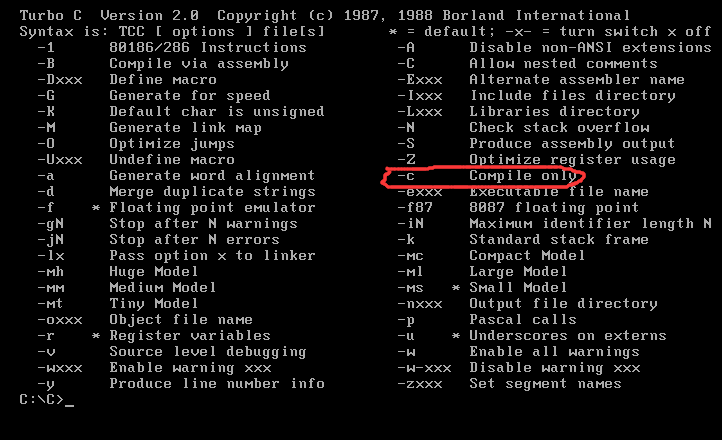
}

TCC在默认的编译连接一个C语言的源程序a.c的时候分为以下两步：

（1）.TCC将源程序文件编译成a.obj。

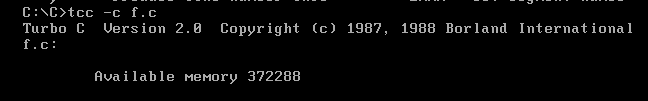
（2）.TCC调用TLINK将c0s.obj，cs.lib，emu.lib，maths.lib中的a.obj中的程序要用到的代码与a.obj的代连接到一起生成.exe文件。

那如何生成OBJ呢？ 根据提示 tcc有很多选项如下：

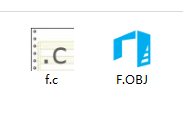


如上我们看到tcc 一个使用选项-c应该可以直接生成.obj 文件

Compile only 翻译过来为 只编译。



可以发现好像成功了，查看文件夹：

只生成OBJ文件，没有生成exe文件

**2.tcc a.c的方法编译链接下面的程序**

main()

{

f();

}

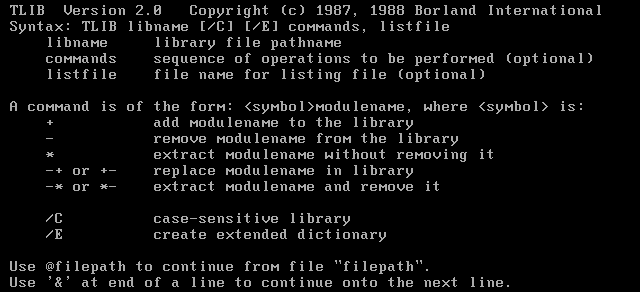
显示的信息为:



在a.c中f函数未定义。说明未定义的函数在使用过程中会报错。

**3.TLIB.exe研究**

从书中看到，TC2.0给我们提供了一个工具tlib.exe，可以用tlib.exe将一个.obj文件中的代码加到一个.lib文件中。

对于TLIB.exe的使用 直接运行TLIB也会出现许多选项，如下：****

从上可以看出 TLIB使用格式 TLIB libname[/C][/E] commands,listfile

也可以看出对TLIB使用格式中的参数进行了解释

lib文件 是obj文件的集合,可以使用tlib.exe 从lib文件中取出obj文件,向lib文件加入obj文件,等操作

+ 是向lib文件加入obj文件。

- 是从lib文件中移除obj文件。

\* 是从lib文件中取出obj文件，不删除。

-+ or +- 是把lib文件中的obj文件更换成指定的obj文件。

-\* or \*- 是从lib文件中取出obj文件并从lib文件中删除。

/C:大小写敏感标志。该选项不常用，此参数为可选项。

/E:建立扩展字典。建立扩展字典可以加速大的库文件的连接过程，此参数同样为可选项。

函数名和文件名都不能和cs.lib中文件发生重复。

函数名与文件名一样 +- 可以

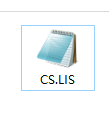
函数名与文件名不一样 ，先- 后+

listfile :lib文件中obj文件列表（可选项）

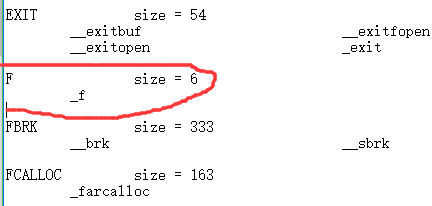
此参数说明了命令运行后，生成的对应lib文件的列表文件名。它记录了当前lib文件内obj文件列表。如果你新加入了一个obj文件，就能从生成的lst文件中找到对应的obj文件的名称。

接下来我们将（1）生成的.obj文件加入到c:\c\cs.lib中

用查看文件夹：

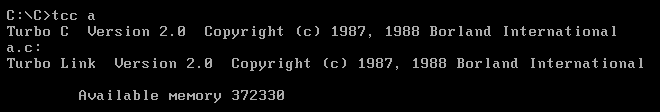


记事本打开cs.lis 可以看到：



表明已经添加成功了！

再进行tcc a.c 如下：



没有出现错误。然后进行debug调试得到如下：



进入main函数里调用call 0554 ，说明f函数偏移地址在0554处。



mov ax,0001

jmp 0559 这两条指令刚好对应 f函数里的return 1；证明此处放的就是f函数的代码。

问题：a.c中并没有写函数f，a.exe中的函数f的代码是什么时候加入的呢？

对于这个问题我们应该想到链接的过程。

（1）.TCC将源程序文件编译成a.obj。

（2）.TCC调用TLINK将c0s.obj，cs.lib，emu.lib，maths.lib中的a.obj中的程序要用到的代码与a.obj的代连接到一起生成.exe文件。

我们把f函数已经加载到cs.lib中，说明在调用TLINK时就把函数f代码加入a.exe中。

**4.将下面的程序编译为f.obj，将f.obj加入c:\c\cs.lib。**

程序f1.c如下：

int f1(int a,int b)

{

int c;

c=a+b;

return c;

}

int f2(int a,int b)

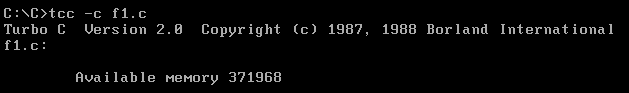
{

int c;

c=a-b;

return c;

}





将f1.obj加入到cs.lib中。

int f3(int a,int b)

{

return a+b+1;

}

程序b.c

int func(int,int);

int a,b;

main()

{

a=f1(1,2);

b=f2(20,10);

a=func(a,b);

printf("%d\n",a);

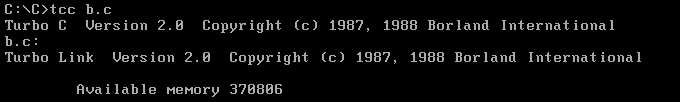
}

int func(int a,int b)

{

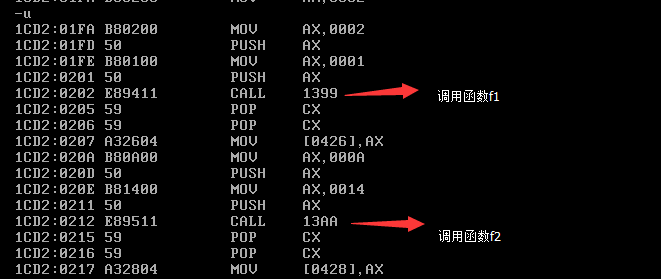
return a\*b;

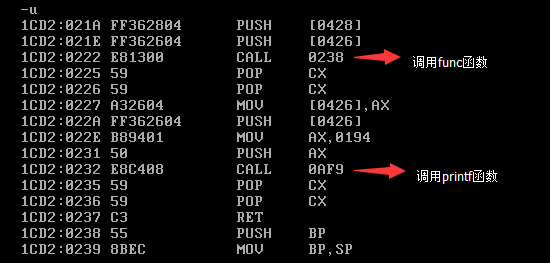
}



b.c编译链接成功。用debug加载b.exe 找到所有函数代码如下：

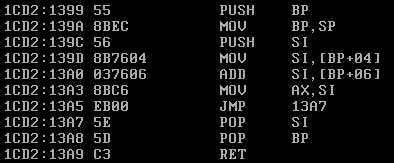
主函数：



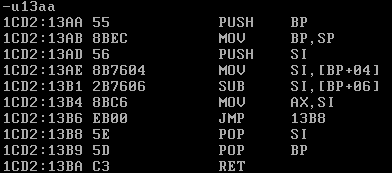


从主函数得知其他函数地址：

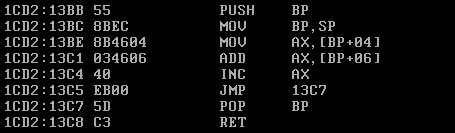
函数f1对应的汇编代码如下:



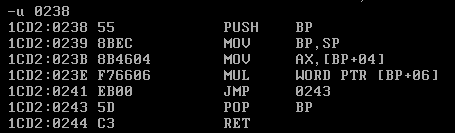
函数f2对应的汇编代码如下：



我们也会发现紧跟其后的就是f3函数



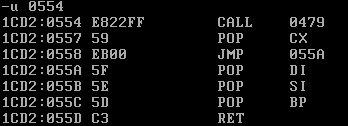
函数func对应的汇编代码：



从上可以看出b.exe中有f1,f2,f3以及printf函数代码，那这些代码什么时候加入的呢？

这个问题和上面问题类似，f1,f2,f3以及printf函数代码都是加载在cs.lib文件中，所以链接时候加入到b.exe文件中。

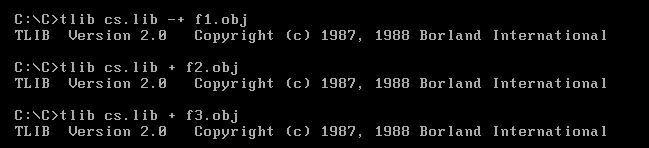
b.exe中很明显有了f3代码，但是f3代码没有调用也加载进去了？难道只要在cs.lib文件里的obj文件在链接的时候都会加载进去的? 经过查阅资料发现不是这样的，那又怎么回事呢，然后我想看看我之前的那个f函数代码有没有加到b.exe中



很明显不是f函数代码，那意思是没有加进来? 是不是写到一块的原因？

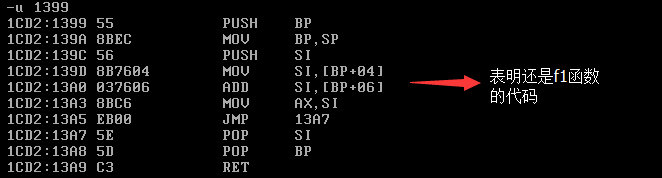
我把三个函数分开成f1.c f2.c f3.c,一个c程序对应一个函数，分别生成.obj加载到cs.lib再进行调试。

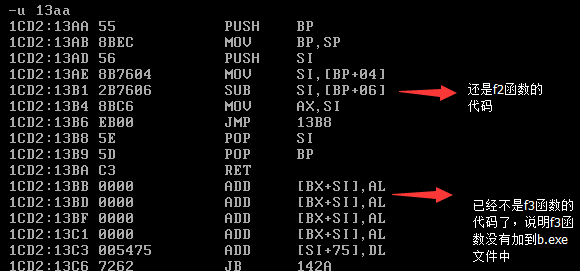




以上图片表明三个函数.obj已分别加入到cs.lib文件中。

然后tcc b.c





通过以上的验证过程得出一个结论：分别编译，分别加入cs.lib这个方案是可以实现用到哪个函数，装入哪个函数的代码的。

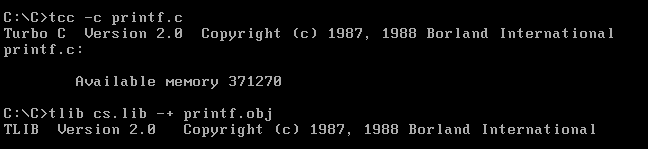
**5.用tlib.exe 将cs.lib中的printf函数代码变成下面程序的代码**

printf()

{

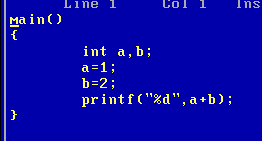
puts("Do you want to use printf? No printf here.");

}

****

以上图片表明已经将cs.lib文件中的printf函数.obj换成自己写的printf函数.obj。

**shao.c程序如下：**

****

运行得：

****

总结：

Tlib 可以将用户的代码加入到cs.lib文件中，这样使程序设计更灵活，分别加到cs.lib中，可以实现没有使用则不会将函数代码加入到exe文件中，执行效率更快，exe文件大小更小，更精简。