### 多个源文件组成的程序、预处理、翻译和链接

Eg：有两个文件 main.c、fun.c，main函数中调用了fun.c中的函数，此时编译两个文件为：

gcc main.c fun.c -o main

./main 🡪执行可执行文件

也可以为：

gcc -c fun.c -masm=intel //fun文件中存在汇编指令 🡪生成fun.o文件

gcc -c main.c 🡪生成main.o文件

gcc main.o fun.o 🡪生成a.out文件

./a.out 🡪执行可执行文件

### 函数库：

#### 静态库：

Eg: 现有三个文件fun1.c、fun2.c、fun3.c，main函数中调用了这三个文件中的函数，此时按照之前的编译仍然是可以的，即：

gcc main.c fun1.c fun2.c fun3.c

./a.out

也可以为：

gcc -c fun1.c fun2.c fun3.c 🡪生成fun1.o…文件

gcc main.c fun1.o fun2.o fun3.o

./a.out

或者利用**函数库**的方法：

gcc -c fun1.c fun2.c fun3.c 🡪生成 .o 文件

ar r libxxx.a fun1.o fun2.o fun3.o 🡪将 .o 文件生成库文件

gcc main libxxx.a 🡪生成可执行文件a.out

./a.out 🡪执行可执行文件

**注意：这里的用来生成库的文件是原函数功能文件通过 gcc -c 生成 .o 文件得来的，然后将所生成的 .o 文件通过命令 ar r libxxx.a fun1.o fun2.o fun3.o 生成 libxxx.a库文件（若库文件存在，则该命令的含义是将所添加的 .o 文件添加到库文件libxxx.a中，若库文件中存在某个已经添加的.o文件，则将之前文件中的函数覆盖掉）。**

#### 动态库（共享库）：

生成共享库：

gcc -shared -o libxxx.so fun1.o fun2.o fun3.o

源文件和共享库进行链接生成可执行程序

gcc main.c libxxx.so

当可执行程序需要调用函数时，程序会找到共享库的位置，然后将其所需要的函数从共享库中加载到内存中去，程序通过寻找指定路径的方式去寻找库文件（设置环境变量）。

export \_LD\_LIBRARY\_PATH = ./ //添加当前的路径到**环境变量**中

#### 静态库和动态库之间的区别：

1、静态库编译的时候会将所用到的函数编译到可执行程序中（可执行程序中只载入了它所需要的函数，而不是载入整个静态库中的函数）。而动态库编译生成的可执行文件中不包含他所用到的函数的代码（包含一个动态库的引用），程序体积相对于静态库来说比较小。

2、动态库体积小、不会造成内存浪费、更新/升级程序更加方便。

#### 头文件、预处理指令：

我们可以将一个函数库中的所有函数的原型声明集中在一起，保存为一个以 .h 结尾的文件，这个文件我们称之为头文件。

当我们用到函数库中的函数时，只需在文件开始添加对应的有文件

🡪#include “xxx.h” //# 🡪预处理指令的标识

//include 🡪预处理指令（文件包含指令）

//作用是将自己替换为它所指定的文件中的内容。