编译过程

将下面代码保存为 Hello.c：

```
#include <stdio.h>int main(void){printf("Hello World!\n")❶;return 0;}
```

❶ printf() 函数

执行命令 cc Hello.c[26]，得到一个可执行文件 a.out，执行它 ./a.out

可以看到，C的源代码(Hello.c)是纯文本，不能够直接执行。可执行代码是计算机的本机语言或机器语言表示的代码，这种语言是由数字代码表示的详细指令组成，不同的计算机具有不同的机器语言。

编译器是一个程序，其工作是将源代码转换为可执行代码。

- 编译器用来将 C语言 转换成特定的机器语言。
- 编译器还从C的库中向最终程序加入代码。[27]
- 编译器还检查源代码是否为有效的C语言程序。如果编译器发现错误，将报告错误，而且不生成可执行文件

编译器分三步完成这个工作：

预处理	调用预处理器 cpp 对源代码文件中的文件包含(include)、预编译语句(如宏定义 define 等) 进行分析
编译	调用编译器 cc 将源代码转换为中间代码
链接	调用链接器 ld 将中间代码与其它代码结合起来生成可执行文件

- 这种方法使用程序便于模块化。分别编译各个模块，然后使用链接器将编译过的模块结合起来。这样，如果需要改变一个模块，则不必重新编译所有其它模块。

可执行文件包含目标文件、库例程和启动代码

编译器将源代码转换为机器语言代码(中间代码)，将结果放置在目标文件(*.o)中。虽然目标文件包含机器代码，但该文件还不能运行，它还不是一个完整的程序。

启动代码(start-up code)相当于程序和操作系统之间的接口。[28]

库例程为函数的实现。几乎所有C程序都利用标准C库中所包含的例程，目标代码文件不包含这一函数的代码，它只包含调用函数的指令。实际代码存储在一个称为"库"的文件中。库文件中包含许多函数的目标代码

链接器的作用是将这3个元素(目标代码、系统的标准启动代码和库代码[29])结合在一起，并将它们存放在可执行文件中。

---

[26] 在 Linux 系统中，编译器为gcc，cc为它的链接

[27] 库中包含许多标准例程供您使用，例如printf()。更准确的说，是一个被称为链接器(linker)的程序将库例程引入的，但在多数系统上，编译器为您运行链接器。

[28] 硬件相同的情况下，在 DOS 或 Linux 下可以使用同样的目标代码，但 DOC 与 Linux 要使用不同的启动代码，因为这两种系统处理程序的方式是不同的。

[29] 程序有两种方法来使用这些库函数，如果静态连接一个程序，这些函数就会被复制到可执行程序中,这就是lib*.a函数库的作用
如果你动态的连接一个程序（默认），那么当程序运行时需要库中的代码，它就会调用lib*.so中的内容。