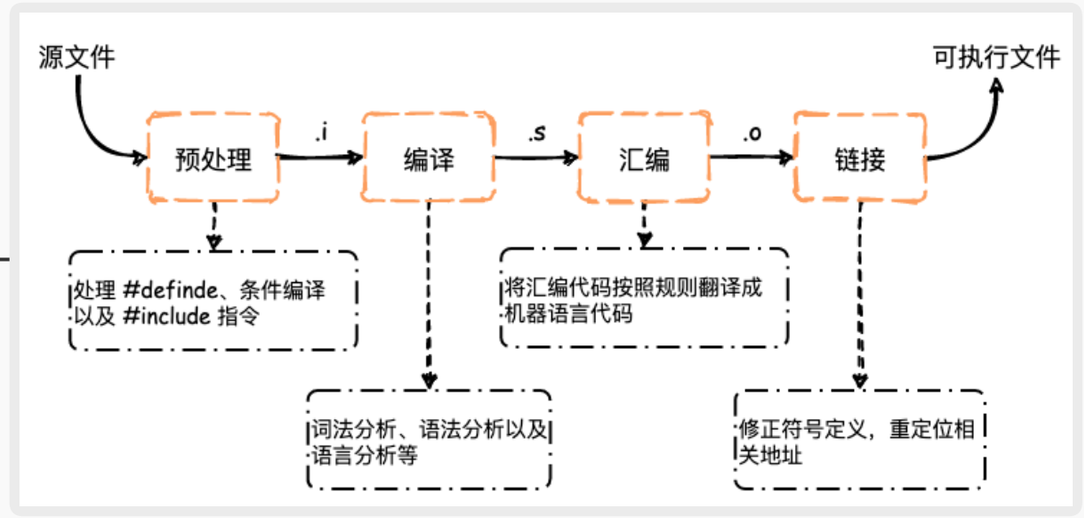
# 编译与链接

## 基本流程




## 预编译（.i）

**预编译又称为预处理，主要是对源文件进行一些预处理。比如说展开所有的宏定义，也就是使用宏实际的定义来替换调用宏名称，把包含的头文件插入到对应的位置**

主要过程

- 将所有的 #define 删除，并且展开所有的宏定义
- 处理条件编译指令，例如 "#if"、"#idnef" 等
- 处理 "include" 指令，将被包含的文件插入到对应的位置，这个过程是递归展开的

```
#include <stdio.h>

#define MAX_BUFFER 1024

int main() {
    printf("MAX_BUFFER: %d\n", MAX_BUFFER);
}
```

例如在上面的代码中我们使用了 #define MAX_BUFFER 1024，那么在代码中就可以使用 MAX_BUFFER 来代替硬编码 1024 了。在预处理的过程中，MAX_BUFFER 将会被替换成我们定义的 1024。并且我们"引入"了 stdio.h 这个头文件，那么预处理的过程也会将该头文件中的所有内容插入到该指令的位置

我们可以使用 gcc -E source.c 来查看只经过预处理后的文件内容，就会发现产生的内容基本上就是 stdio.h 这个头文件里面儿定义的变量和函数声明

gcc -E source.c

## 编译（.s）

**编译过程就是将预处理完之后的文件进行一系列的词法分析、语法分析和语言分析后产生的汇编代码文件，也就是我们经常看到的 pushq %rbp、movl $0, %eax。这是一种汇编语言，主要是和寄存器、内存打交道**

编译过程是程序构建最为复杂的一部分，同时也是最为重要的一部分。但是，这恰恰也是我们"最不需要关心"的一部分，因为在绝大部分时刻，我们放在链接阶段的注意力要远远多余在编译阶段的注意力

编译针对的是单个文件，在编译期间，有多少个源文件，就会产生多少个汇编代码文件

gcc -S source.c

## 汇编（.o）

编译产生的汇编代码文件我们通常还能够阅读，但是一旦完成汇编阶段，汇编代码文件将会转变为机器可以执行的指令

汇编的过程相对编译过程来讲就简单了许多，本质上就是根据汇编指令和机器指令的对照表进行一一翻译

**源代码经过预编译、编译和汇编将会产生目标文件（Object File），多个目标文件即可通过链接器链接到一起，形成真正的可执行文件**

在编写 C++ 面向对象相关代码，有时候我们不清楚编译器有没有为我们合成默认的构造函数或者是拷贝构造函数的话，那么我们就可以通过查看目标文件来确定

我们来看下 Linux 下汇编产生的目标文件的格式

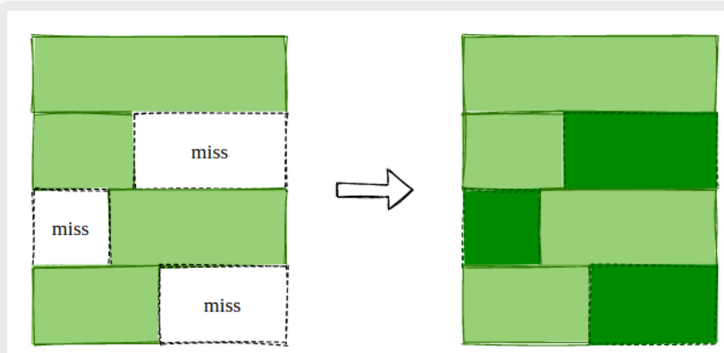
- gcc -c source.c<br>file source.o
  - 结果: source.o: ELF 64-bit LSB relocatable
- **ELF 就是 Executable and Linkable Format 的缩写，可执行与可链接文件**

gcc -c source.c

## 链接（.out）

链接过程我们前面已经提到了，其作用就是将多个目标文件链接到一起，组合成最终的可执行文件

在前面的预编译中，我们仅仅只是将 stdio.h 这个头文件里面儿的内容插入到了源文件中，编译和汇编的过程也仅仅是对预编译产生的 .i 文件进行相应的处理，那么 printf 这个函数地址是什么？或者说，这个函数的实现在什么地方？




链接的过程就是在打补丁，缺什么补什么

链接的本质就是使用 ld 将各个目标文件整合到一起，那我们来看看上面生成的 .o 文件还差什么

- gcc -c source.c<br>ld source.o
  - ld: warning: cannot find entry symbol _start; defaulting to 00000000004000b0<br>source.o: In function ` main':<br>source.c:(.text+0x16): undefined reference to ` printf'
  - 链接器告诉我们找不到 _start 这个符号，并且在 main 函数里面儿 printf 是未定义的

在编译 source.c 文件时，编译器并不知道 printf() 这个函数的地址是什么，定义是什么，因此编译器只能在这个地方"插一个眼"，等到链接的时候再由链接器去将这些指令的目标地址修正

**地址修正的过程就叫做重定位，编译器插的那个眼就叫做重定位入口，链接器要做的事情就是找到所有的重定位入口，然后进行重定位**