linking part
1. 编译器驱动程序
用户需要调用语言预处理器，编译器，汇编器和链接器
具体 先用语言预处理器和 C编译器 。将其翻译成为一个汇编语言文件 。然后运行汇编器，翻译成一个可重定位目标文件 。
最后运行链接器程序，将其转换为可执行文件

3.目标文件
可重定位目标文件 Relocatable object file (`.o` file)
可执行目标文件 Executable object file (`a.out` file)
共享目标文件 Shared object file (`.so` file)dynamic linking

5.符号解析 Symbol resolution
符号和符号表 three kinds Symbols
global symbols 在文件头部定义，且没有加static的变量，可以被别的文件通过加external的方式调用
external symbols 在别的文件中定义，在本文件中被引用。需要 加上external
local symbols
how linker determine the global symbol
strong Symbols 初始化后的全局变量
weak Symbols 未初始化后的全局变量
rule1 不可以同时存在多个强变量
rule2 如果有多个弱变量，随机选择一个 pick an arbitrary one
linker puzzles *pic 1

notice avoid too many global variable , try to use static global variables

几种老的链接方法 Old-fashioned Solution: Static Libraries

6.Executable and Linkable Format(ELF)

7.relocation
重定位其实非常简单，这一步所做的工作是把原先分开的代码和数据片段汇总成一个文件，会把原先在 `.o` 文件中的相对位置转换成在可执行程序的绝对位置，并且据此更新对应的引用符号。

shared libraries

link in three siutation
compile time / linking time /run load time
interpositioning

Linker Puzzles

`int x; p1() {}`	`p1() {}`	Link time error: two strong symbols (p1)
`int x; p1() {}`	`int x; p2() {}`	References to x will refer to the same uninitialized int. Is this what you really want?
`int x; int y; p1() {}`	`double x; p2() {}`	Writes to x in p2 might overwrite y! Evil!
`int x=7; int y=5; p1() {}`	`double x; p2() {}`	Writes to x in p2 will overwrite y! Nasty!
`int x=7; p1() {}`	`int x; p2() {}`	References to x will refer to the same initialized variable.

Nightmare scenario: two identical weak structs, compiled by different compilers with different alignment rules.