Symbol resolution
— where is the defn of a symbol

<> a.c
```
void foo();
int main() {
    foo();
}
```
</>

<> b.c
```
int x;
double y = 3.1;
```
</>

<> c.c
```
int x; //double x;
double y; //int y;
void foo() {
  x = x+y;
}
```
</>

strong ↓ fun definition, y (initialized global)

weak ↓ fun prototype, x (uninitialized)

<> typedef
#define
struct ....
</>

(ld a.o b.o c.o) => exe

Rule 1: Only 1 strong defn is allowed.

Rule 2: If there's 1 strong and many weak, use the strong.

Rule 3: If 0 strong and some weak, choose one.

Good C programming:
- Make strong variable defs → initialize globals
- Use 'static' for non-exports
- Clients should use 'extern'
- Write type once for server & client
  - define the variable in a public header

<example.h>
```
#ifdef EXAMPLE_SERVER
  #define EXAMPLE_DEF(var,val) var = val;
#else  #define EXAMPLE_DEF(var,val) extern var; #endif
EXAMPLE_DEF(int x, 3)
```
</e.h>

<example.c>
```
#define EXAMPLE_SERVER 1
#include "example.h"
```
</e.c>

9-2/ object files = .o [all-or-nothing]
executable = .bin, .exe [must have 'main', need everything]
archives = .a

math.h (or stdio.h) — define hundreds of funs

inside of stdio.a: (102kb)

printf.o (1kb), scanf.o (1kb), bread.o, ...

ld example.o stdio.a -o example.bin → 2kb
(printf)

ld stdio.a example.o → fail (no defn of printf)

ld arg0 ... argn

1. Maintain three sets (E = objects to be merged
U = unresolved symbols
D = resolved defns)

2. For argi from 0 to n —
if argi is a .o — E' = E ∪ {argi}
D' = D ∪ defn(argi)
U' = U ∪ referenced(argi) − defn(argi)

if argi is a .a — save U as U0
↳ obj0 ... objm check if obji defines anything in U
(defn(obji) ∩ U ≠ ∅) do this for obji:
if U ≠ U0, do it all again

3. If U isn't empty, then fail and print U
4. Combine E into executable/output

ld myprog.o `pkg-config gtk --libs`
1.a 2.a x.a 3.a

Relocation (mapping defns to addresses)

Inside of example.o

reference to printf at byte 18
definition of main at byte 36

⇓

main will be at byte 900
printf's ref will be rewritten @ 18 to 1000

The main job is to find & rewrite relocations

```
printf("Hello\n");
```

cc0 ⇒ call _printf (_printf not defined)

as ⇒ 0x62 0x71 0x80 | 0x00 0x00
20 — call — addr

add entry in .rel.text

```
struct { int offset;   // offset of addr
         int symbol:24; // ref (printf)
             type:8; }; // how to compile it
```

(32, printf(99), call) // ⇒ 2: relative

absolute

R_386_PC32

R_386_32