

9-1/

Symbol resolution

- where is the defn of a symbol

</>

```
void foo();
```

strong

weak

a.c

```
int main() {
```

```
foo();
```

fun definition

fun prototype

}

y (initialized global)

x (uninitialized)

</>

</>

```
int x;
```

</> typedef

b.c

```
double y = 3.14;
```

#define

struct ...

</>

</>

</>

c.c

```
int x; //&nbxi; (ld a.o b.o c.o) => exe
```

```
double y; //inty;
```

Rule 1: Only 1 strong defn is allowed,

```
void foo() {
```

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

```
x = x + y;
```

}

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>

```
#ifndef EXAMPLE_SERVER
```

```
#define EXAMPLE_DEF(var, val) var = val;
```

```
#else #define EXAMPLE_DEF(var, va) extern var; #endif
```

```
EXAMPLE_DEF(int x, 3)
```

</e.h>

```
<example.c> #define EXAMPLE_SERVER 1
```

```
#include "example.h"
```

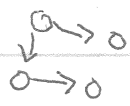
</e.c>

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

math.h (or stdio.h) ~ define hundreds of fns

inside of stdio.a: (100kb)

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



ld example.o stdio.a -o example.bin
 printf ↘ 2kb

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

ld arg0 ... argn

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

2. For arg_i from 0 to n -

if arg_i is a .o — E' = E ∪ {arg_i}
 D' = D ∪ defn(arg_i)
 U' = U ∪ referenced(arg_i) - defn(arg_i)

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

3. If U isn't empty, then fail and print U

4. Combine into executable/output

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

19-3/

Relocation (mapping defs 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

12/21/21

[Faint, illegible text, possibly bleed-through from the reverse side of the page. The text is mostly illegible due to low contrast and blurring.]