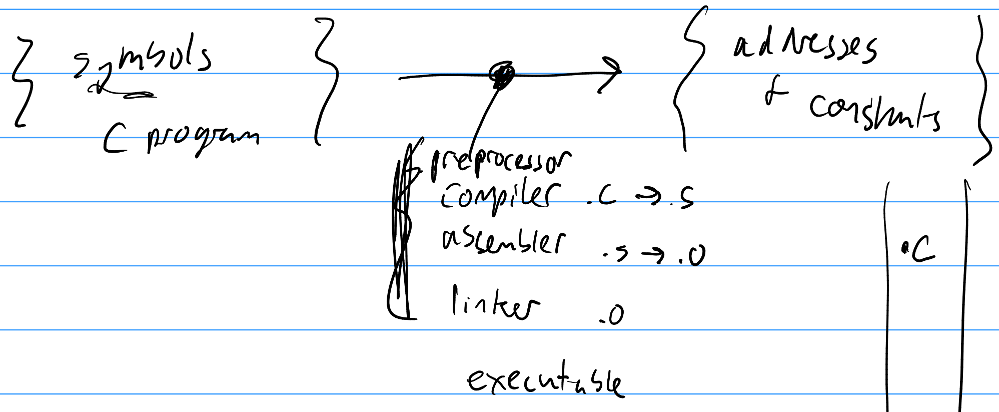
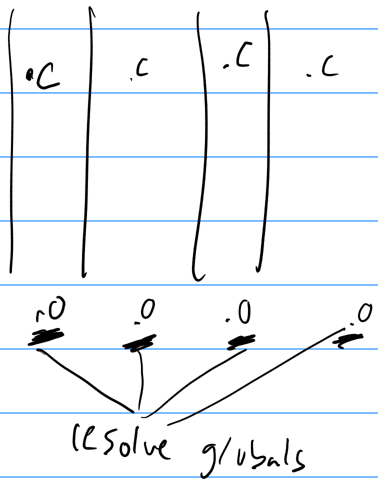
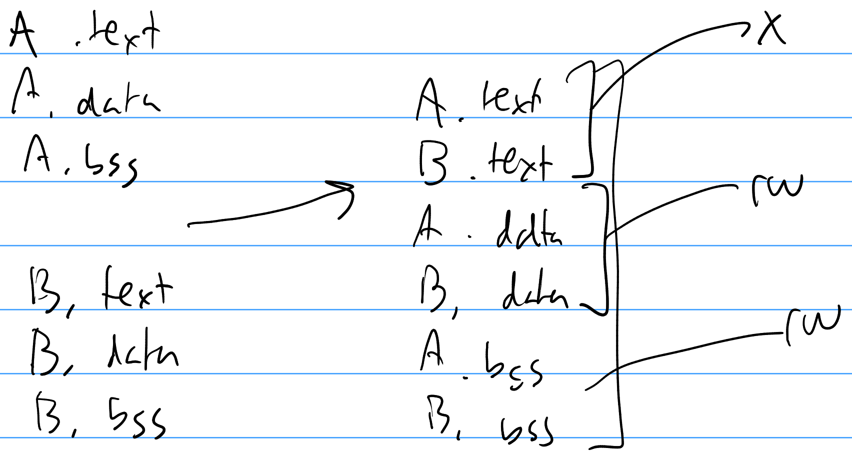
CS 3214 lecture #10 "linking"





ELF executable & linkable format

relocatable object file



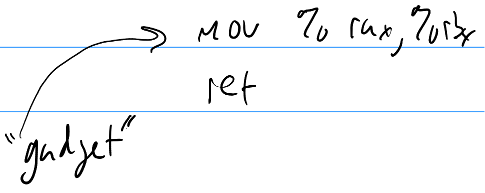
relocations

ASLR - address space layout randomization

fine-grained ASLR

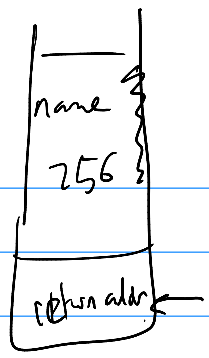
security: ROP - return oriented programming

NX



```
get_user_input( ) {
    char name[256];

    gets(name);
```



```
mov %r1, %sp
ret
```

CFI

what if multiple .o files define same symbol (at linker level)
"conflict"

local vs. global

static int x; → local

int x; → global

strong vs. weak global symbols

strong + strong	"multiply defined" error
strong + weak	no error, weak ignored
weak + weak	no error, one is ignored

Functions: (definition)

static void f() { } — static: local symbol; f(): strong global symbol

(extern) void f(); (declaration)

variables: int v; actually definition — weak global symbol

extern int v; — declaration

int v=0; ← strong global symbol

header files:

```
void f();             ✓
void f() { }          strong symbol conflict
static inline void f() { }   inlining
int v;
int v=42;             strong symbol conflict
```

```
static int v = 42;
static int v;        | no error but probably wrong

extern int v;
```

Best practices : vars
- avoid global vars
- do not define in hdr files
  - `extern int v;` in ~~one~~ hdr file
  - `int v = 0;` in <u>one</u> c file
- always use static
- `-Wl, --warn-common`

Best practices : fns
- use static
- proto decl in hdr file (o/w) `-Wmissing-prototypes`
  "implicit declaration"
- naming conventions <u>list</u>_add list.c
- small fns : inline