CS3214 lecture #10

linking and loading

UB - undefined behavior

→ compiler can optimize things they couldn't otherwise

Rust unsafe {

how to tradeoff compiler/analysis power with programmer responsibility

what can you do in C?

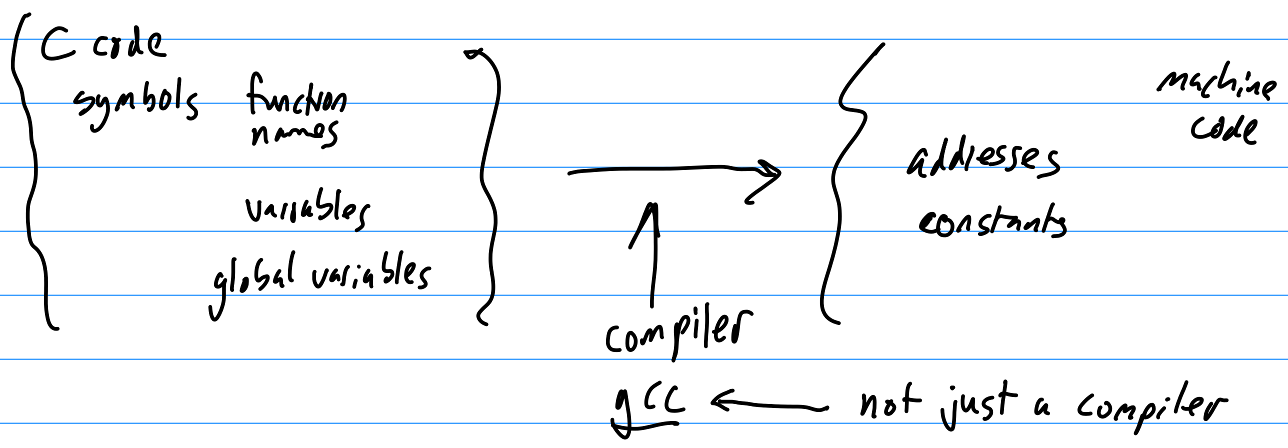
→ -Werror -Wall

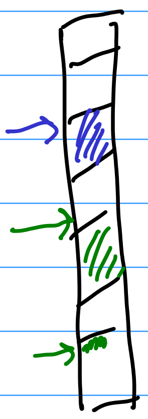
- valgrind memcheck tool ← can't find bugs that optimizer removes
- clang experimental
  - fcatch-undefined-behavior

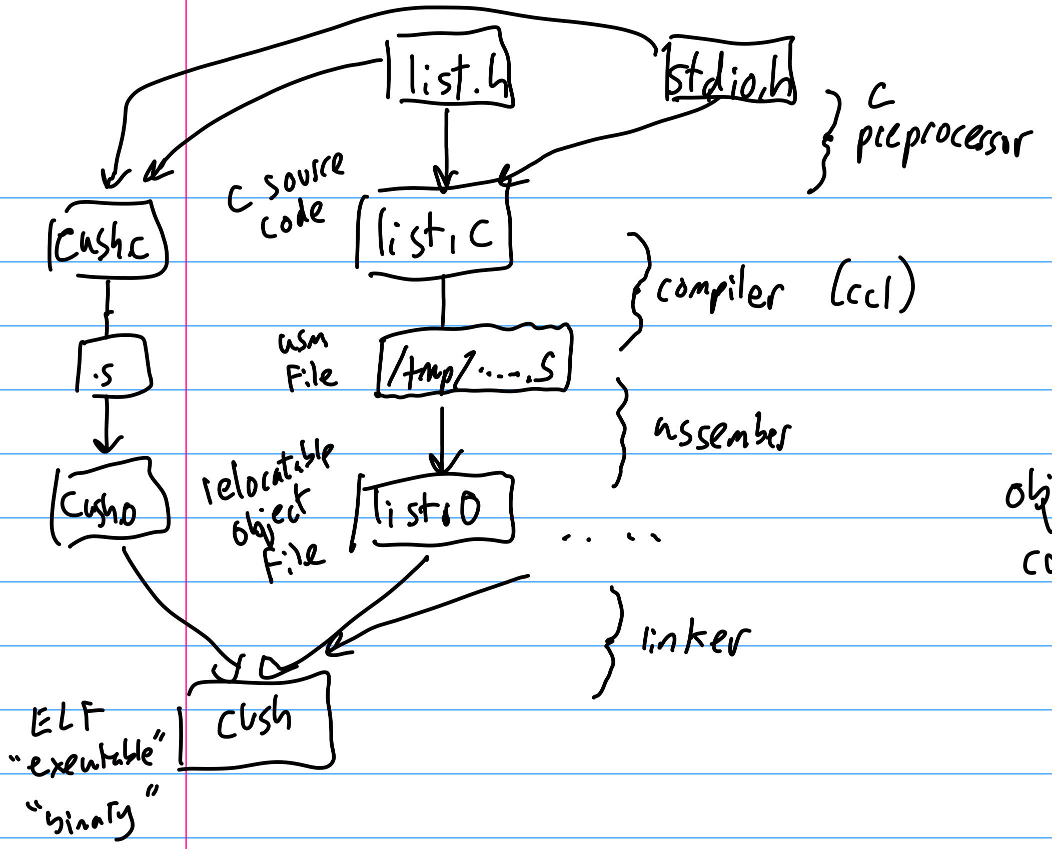
Process - how to manage lifecycle

fork/exec new programs ← ELF executable + linkable format

how do they get built and executed







#include

#define GNU_SOURCE

#ifdef #else

#ifeq

preprocessor directives

object files are created independently

```
struct foo{
  int a;
  int b;
}
```

f→6

Preprocessor

textual insertion #define / #include...

Compiler .c → .s

- resolves symbolic names — local variables ✓ / field names in structs ✓

Assembler .s → .o

some labels for relative branches ✓

→ still unresolved symbols!

externs : global variables / functions

Linker : .o → executable

- creates an in-memory layout of process code & data
- resolve references & fill in placeholders

ELF - Mach-O , PE , a.out — extensible

- provides all info to link — compiler → linker → loader
- debugging info / tool info

ELF

header

program header — readelf

- .text
- .data
- .bss
- .symtab
- .rel.text
- .rel.data
- .debug

section headers