- [x] 1. Last time
- [x] 2. Trusting trust
  - [x] Background
  - [x] Adding a new feature to a language
  - [x] Context
  - [x] Goal: bug login [and keep it bugged]
  - [x] Self-reproducing programs
  - [x] Result
  - [x] Moral, discussion
- [x] 3. Further thoughts on trust

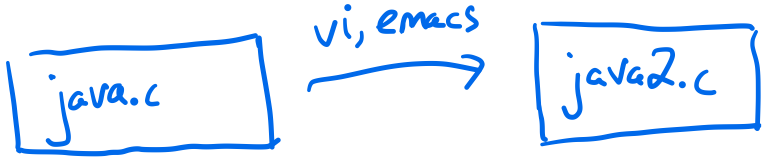
---

2. Trusting trust

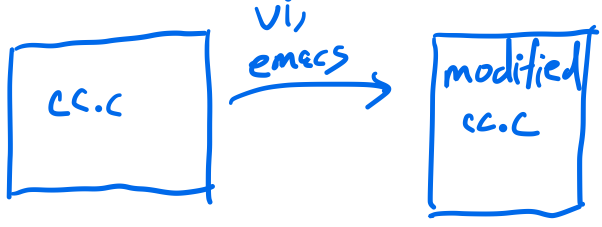
A. Background

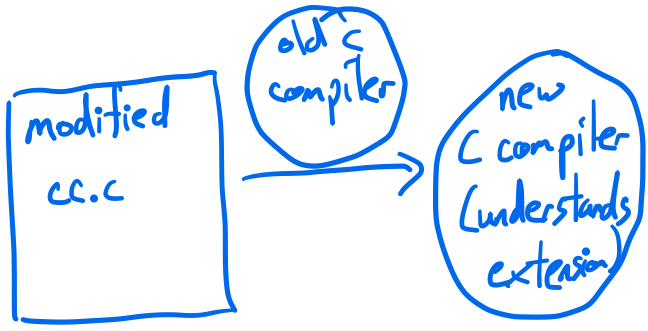
B.1 Adding a feature to a language

Say we want the token "cs202" in a .java file to produce System.print("hi"). We do this:



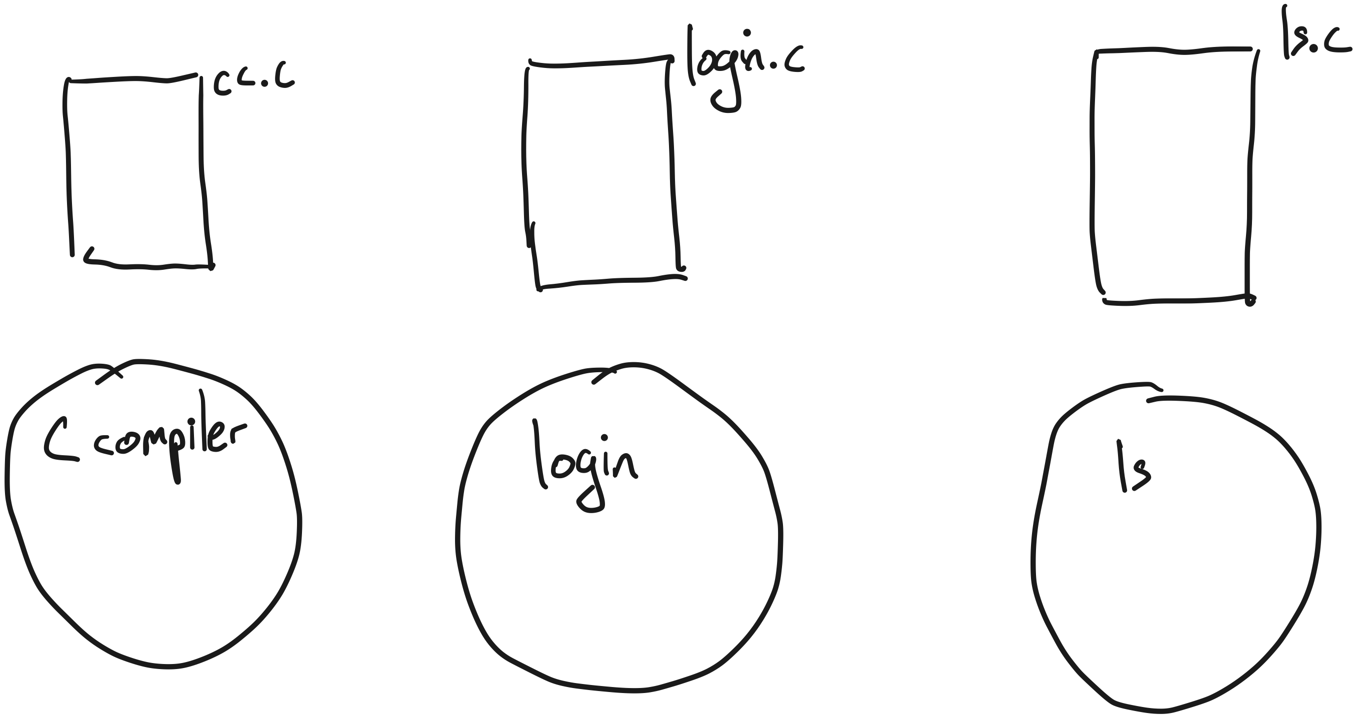
We can do the same thing to extend C:

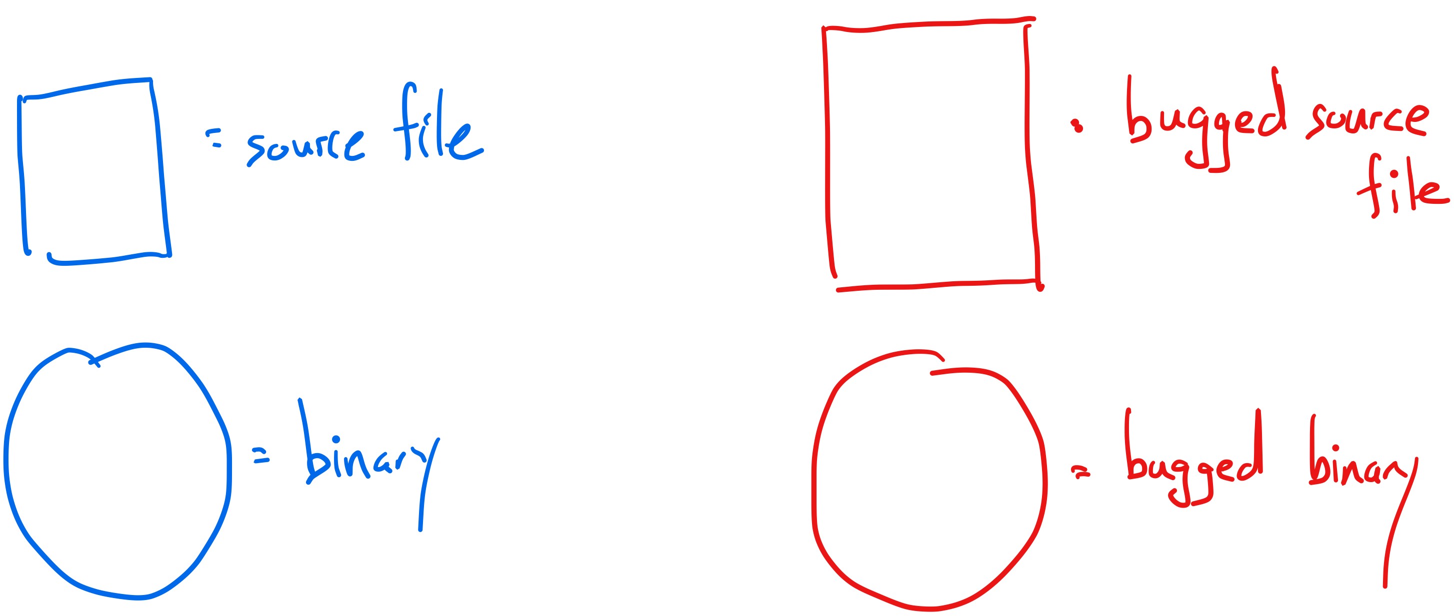




The compiler for Java now recognizes "cs202" as a token in Java files

C. Context

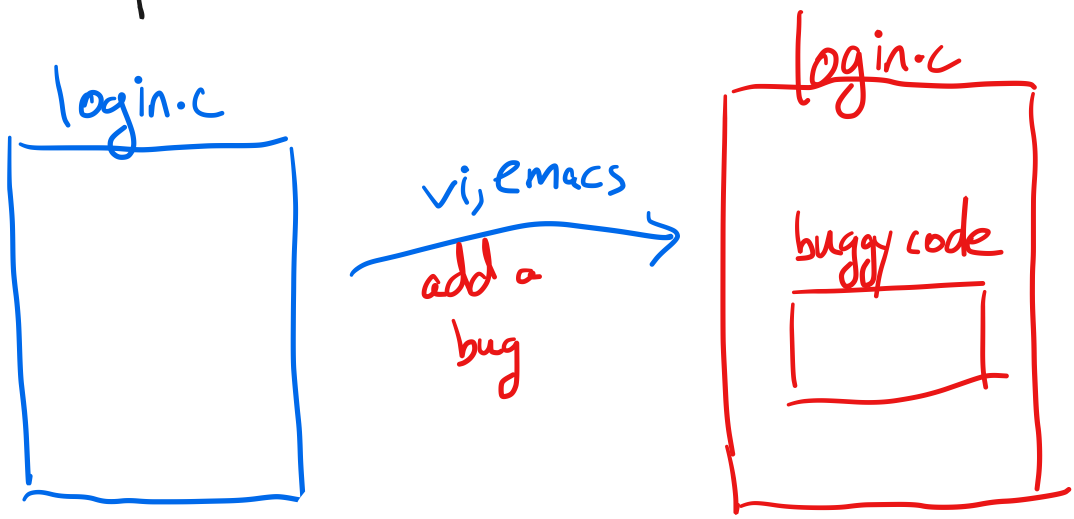


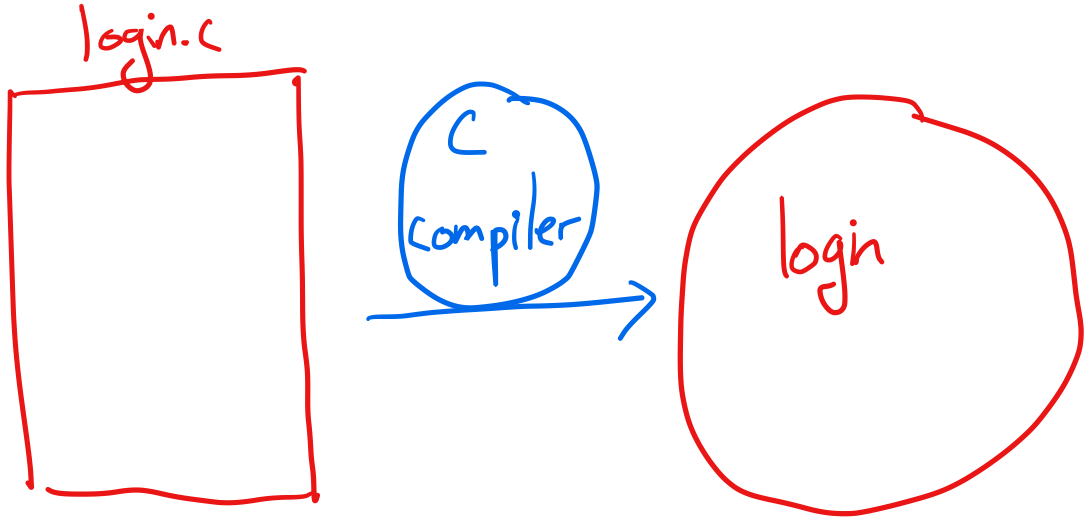


trace in any source files keep

D. Goal: bug login, leave no trace in any source files, keep it bugged across recompilations.

D.1 First attempt:

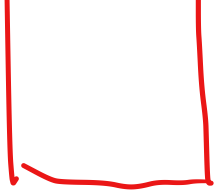


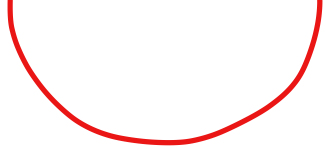


Distribute:

"Problem":
anyone looking

at login.c can see that it is bugged.

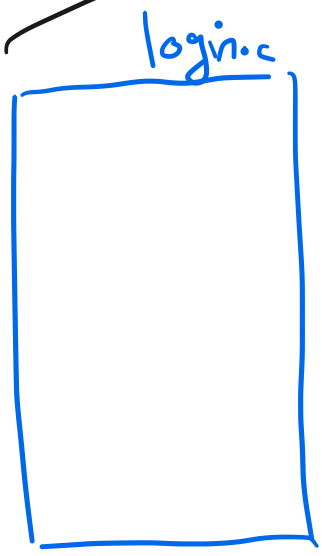




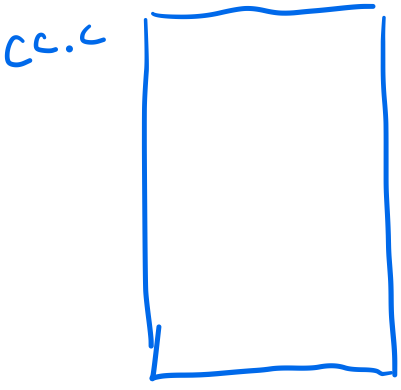
D.2 Second attempt:

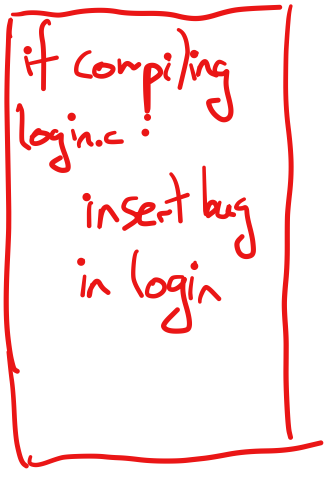
Distribute

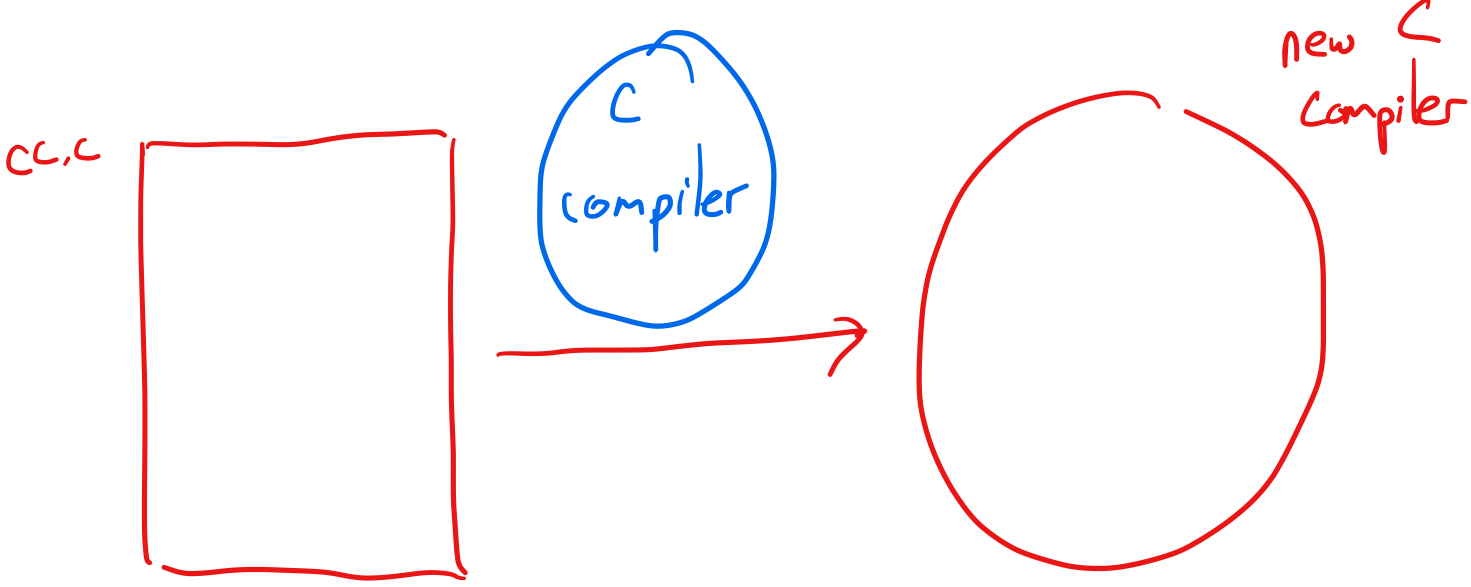
"Problem": recompile login.c and login becomes non-bugged again

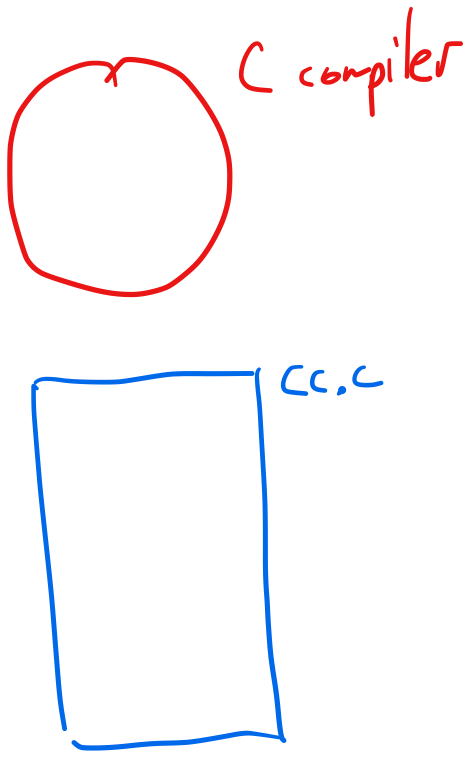


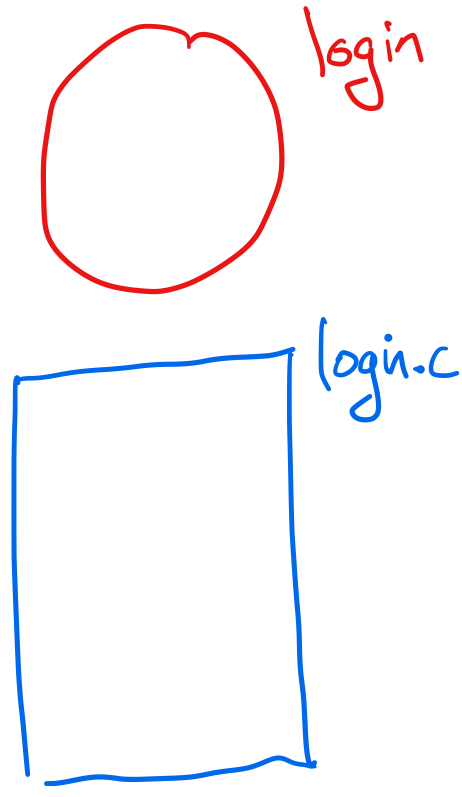
D.3. Third attempt: bug the C compiler!

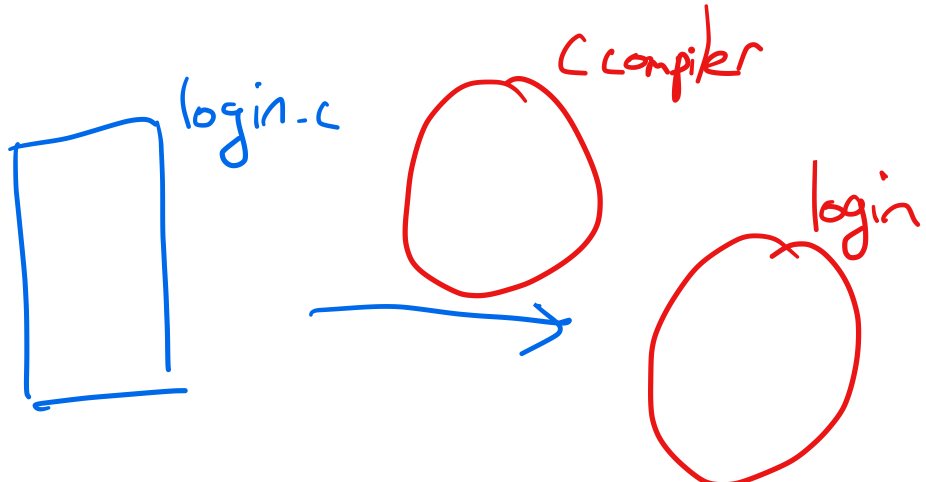





cc.c
C
compiler
new C
compiler

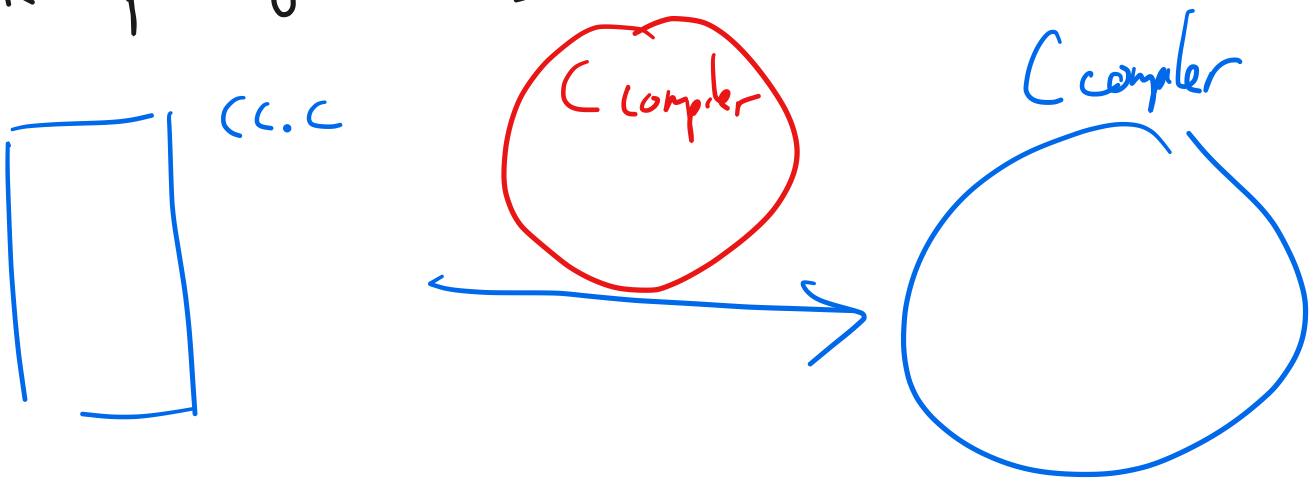

C compiler
cc.c


login
login.c

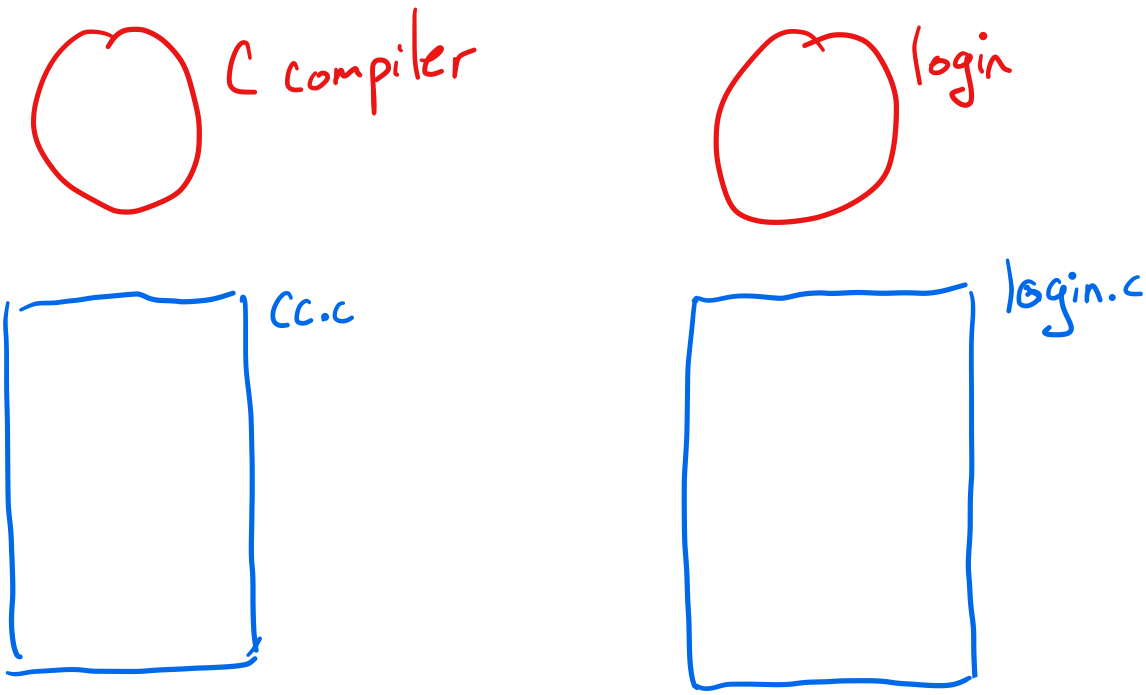

login.c
C compiler
login

Close!! No trace in source

- Recompile login ⇒ bugged login



Goal: make this picture stable:



E. Self-reproducing programs

[exec stuff]
X = "Output 'X'. Output '='. Output quote mark.
Output X. Output quote mark. Output X."

Exec stuff
Output 'X'. Output '='. Output quote mark. Output X.
Output quote mark. Output X.

X = "Output 'X'. Output '='. Output quote mark.
Output X. Output quote mark. Output X."

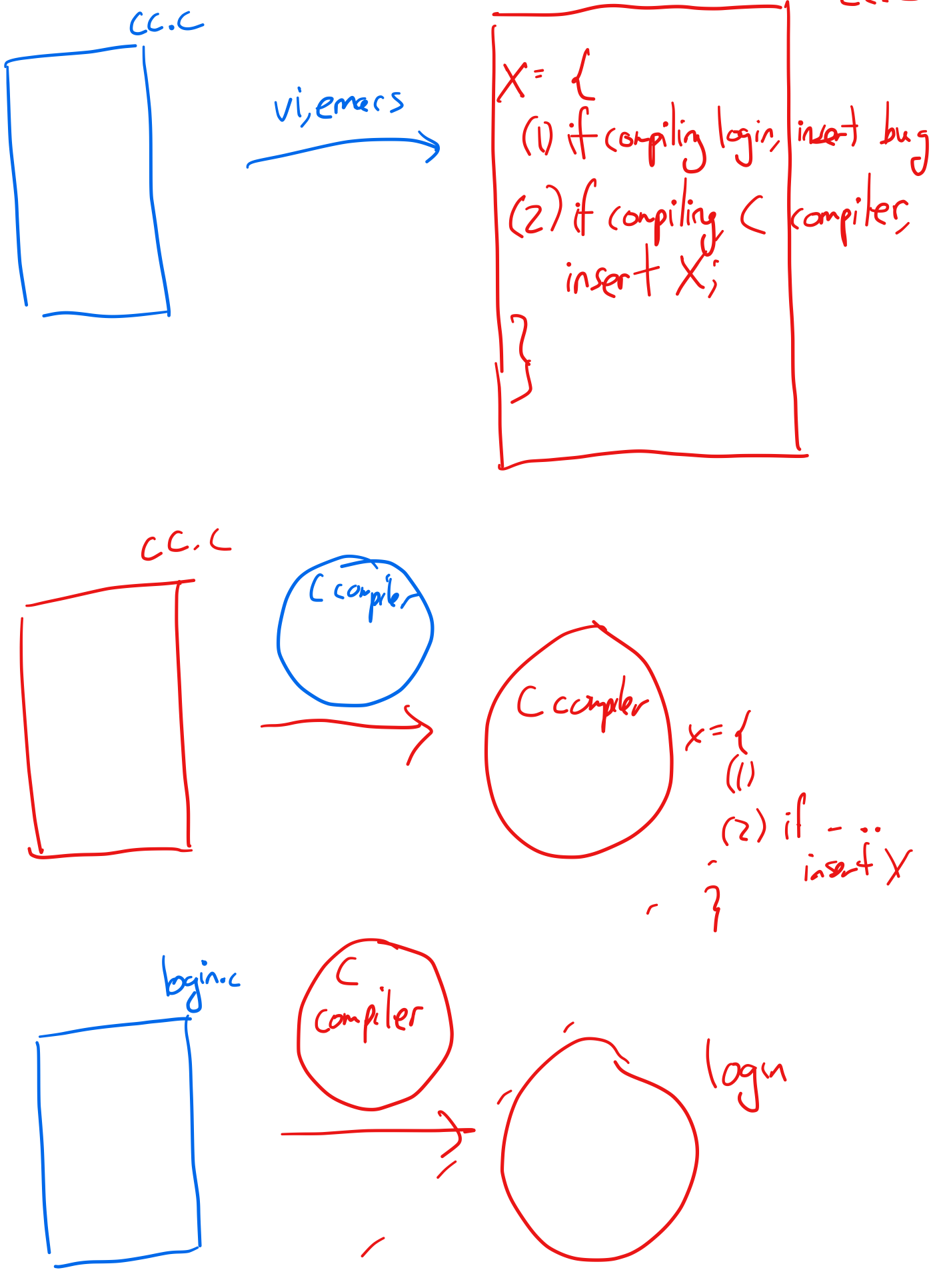
Output 'X'. Output '='. Output quote mark.
Output X. Output quote mark. Output X.

Print this followed by its quotation:
"Print this followed by its quotation."

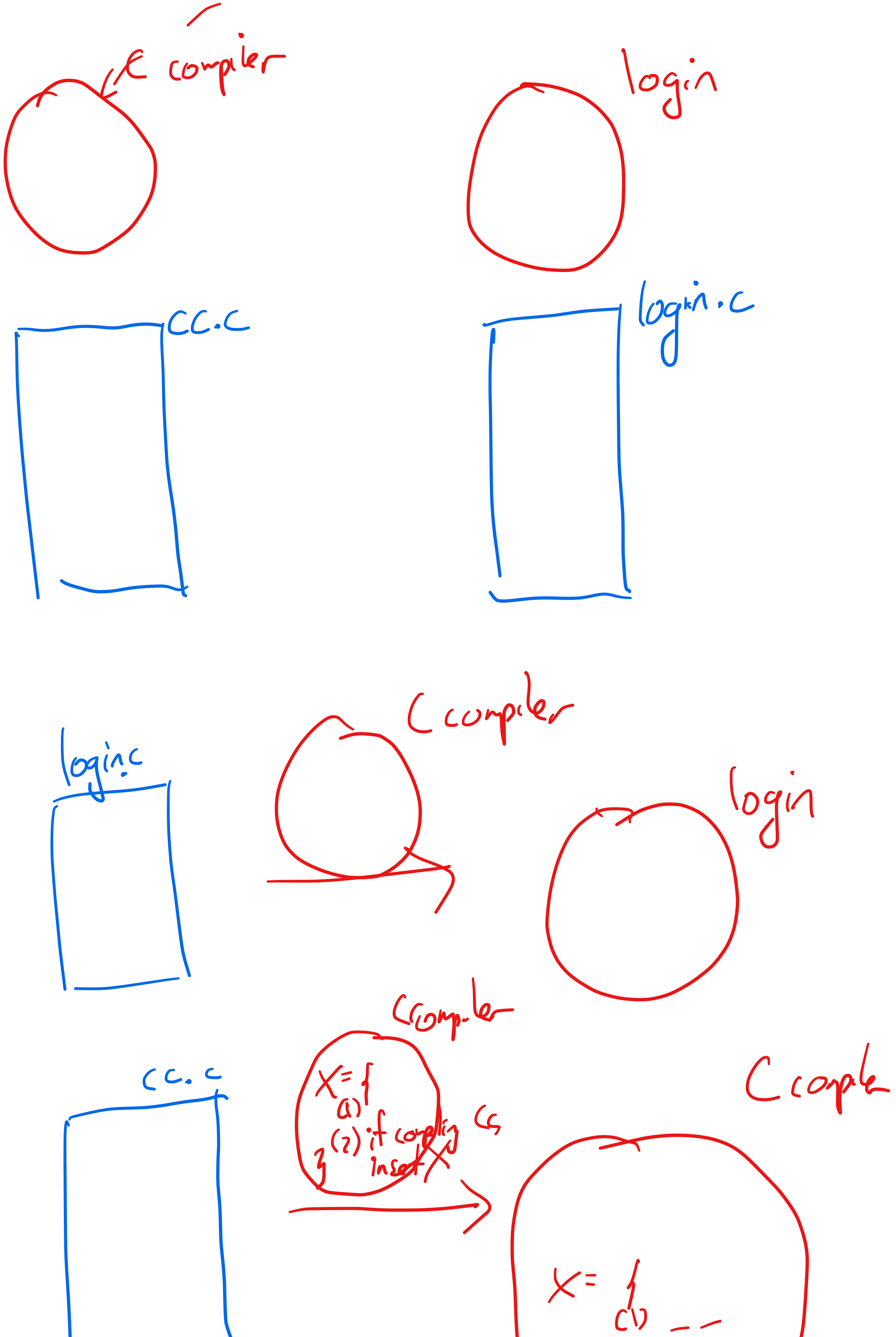
quine

```
((lambda (x) `(,x ',x))
 '(lambda (x) `(,x ',x)))
```

F.1


cc.c
vi, emacs
cc.c
x = {
(1) if compiling login, insert bug
(2) if compiling C compiler,
insert X;
}
cc.c
C compiler
C compiler
x = {
(1)
(2) if ...
insert X
}
login.c
C compiler
login

F.2


C compiler
login
cc.c
login.c
login.c
C compiler
login
C compiler
cc.c
x = {
(1)
(2) if compiling
insert X
C compiler
x = {
(1)

---

cc -S cc.c ; as cc.s