su substitute user

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

---

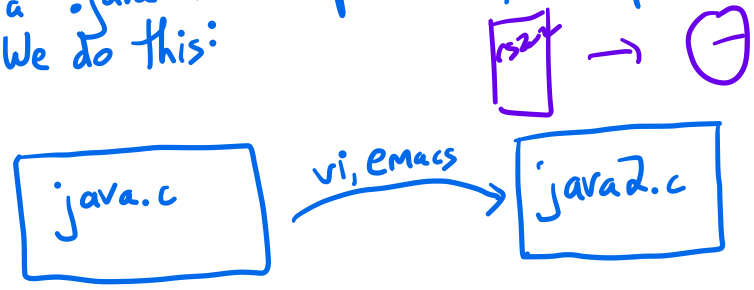
2. Trusting trust

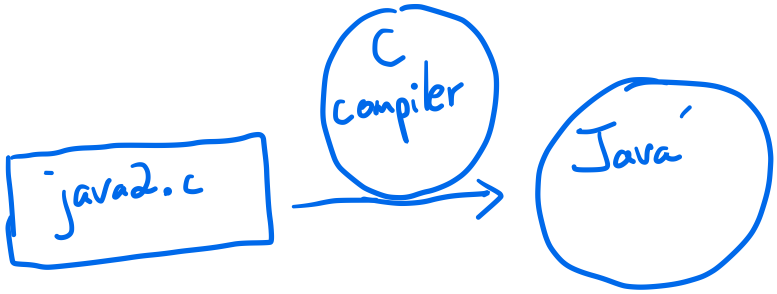
A. Background

B.1 Adding a feature to a language

Say we want the token "cs202" in a .java file to produce System.print("hi");

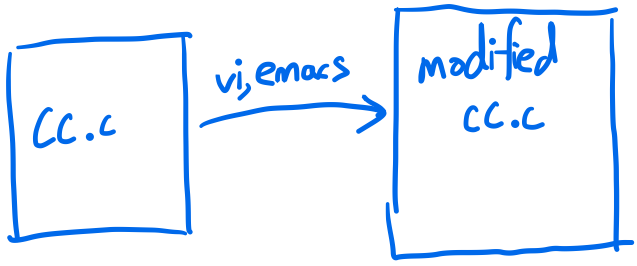
We do this:

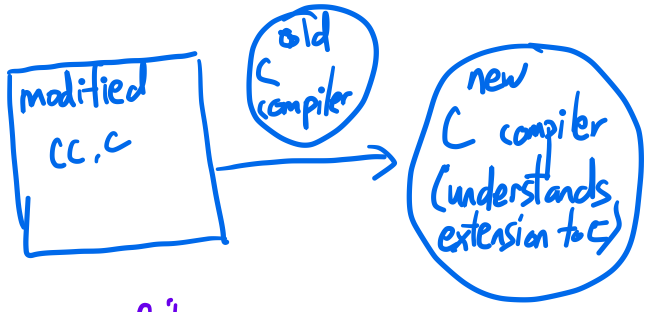


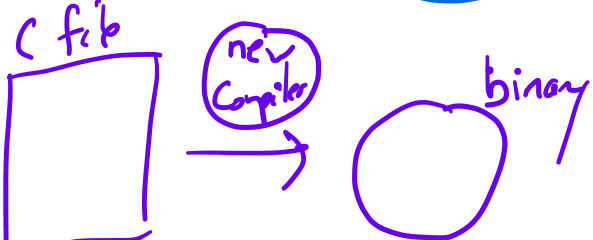


The compiler for Java' now recognizes "cs202" as a token in Java files.

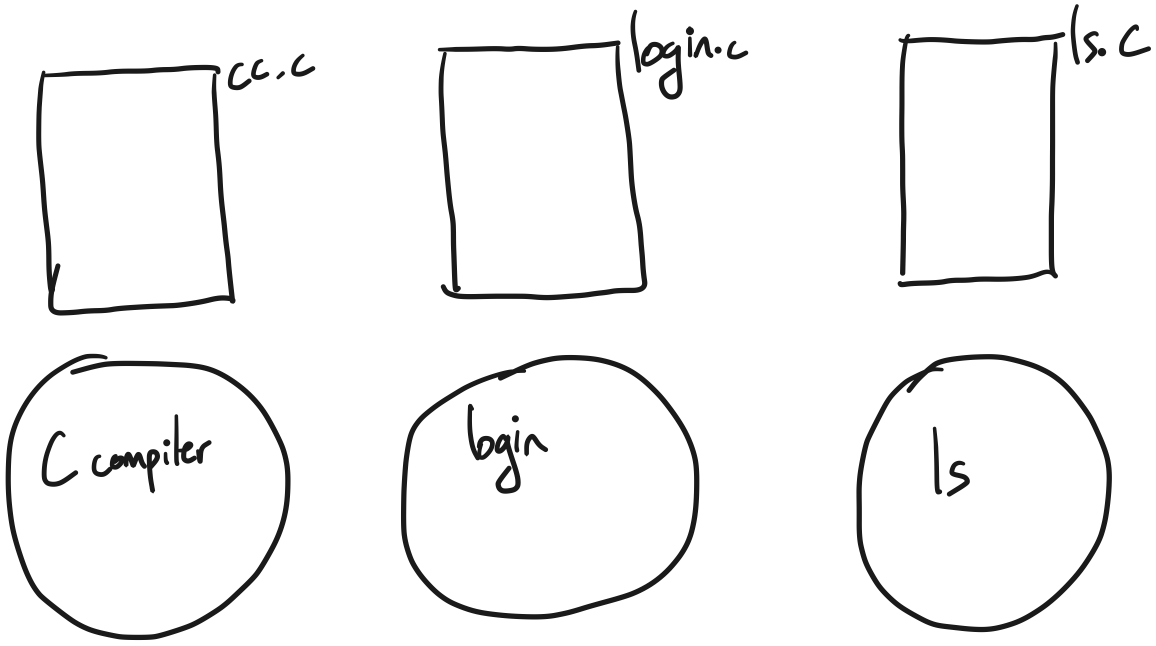
We can do the same thing to extend C:

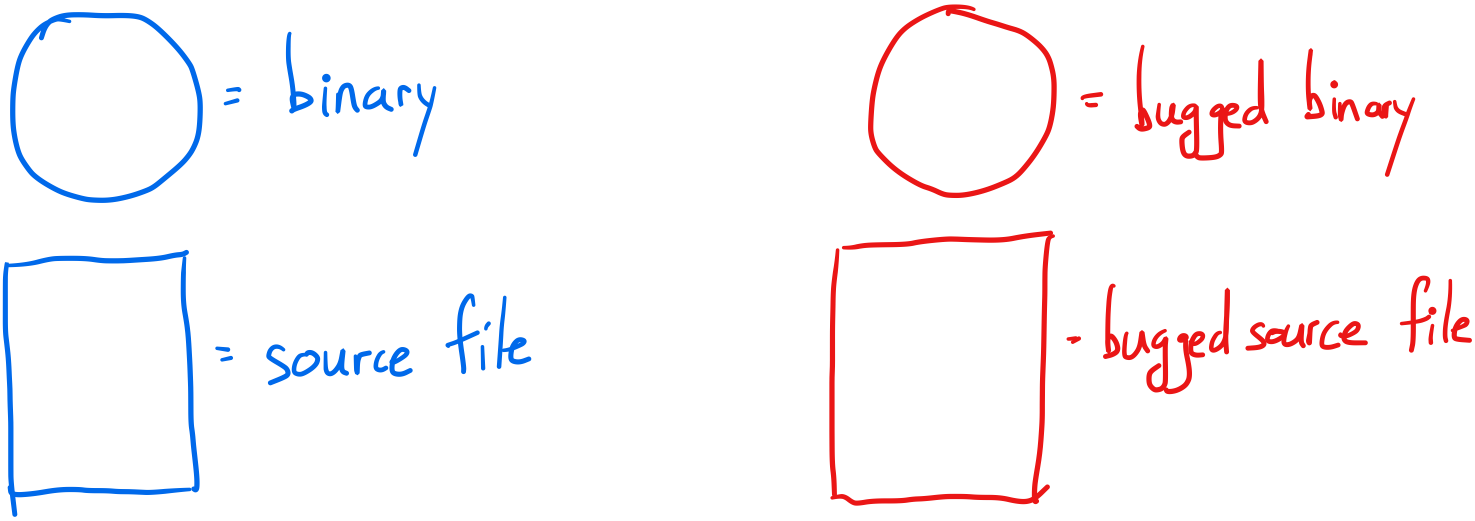






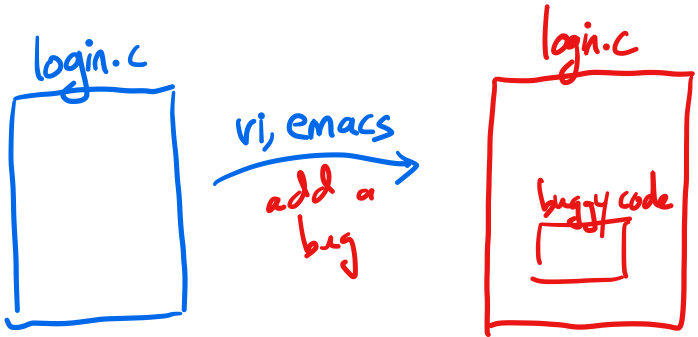
C. Context

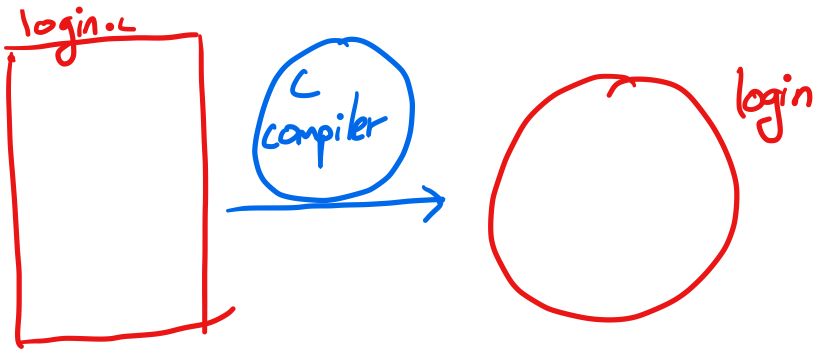




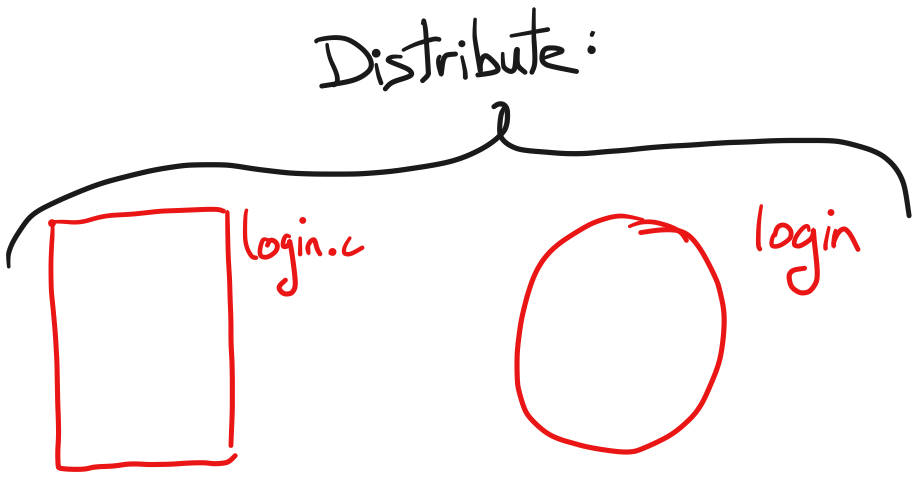
D. Goal: bug login, leave no trace in any source files, keep it bugged across recompilations.

D.1 First attempt:



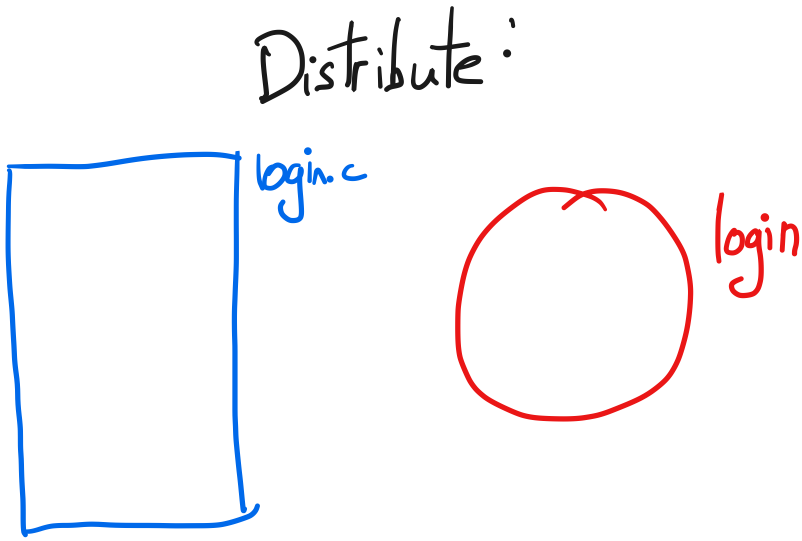


"Problem": anyone looking at login.c can see that it is bugged.

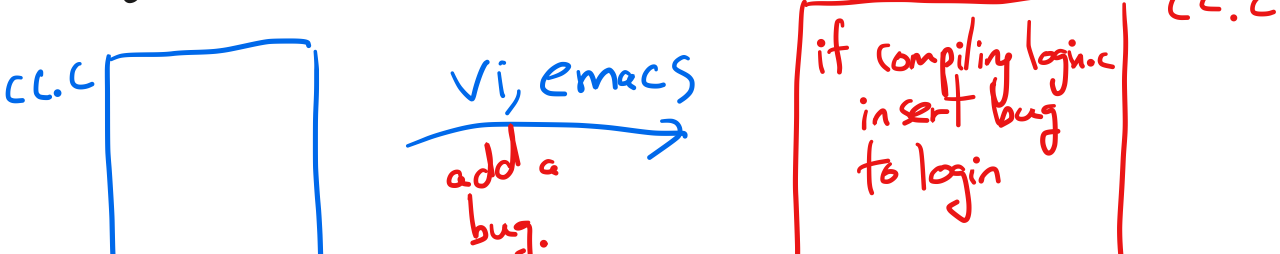


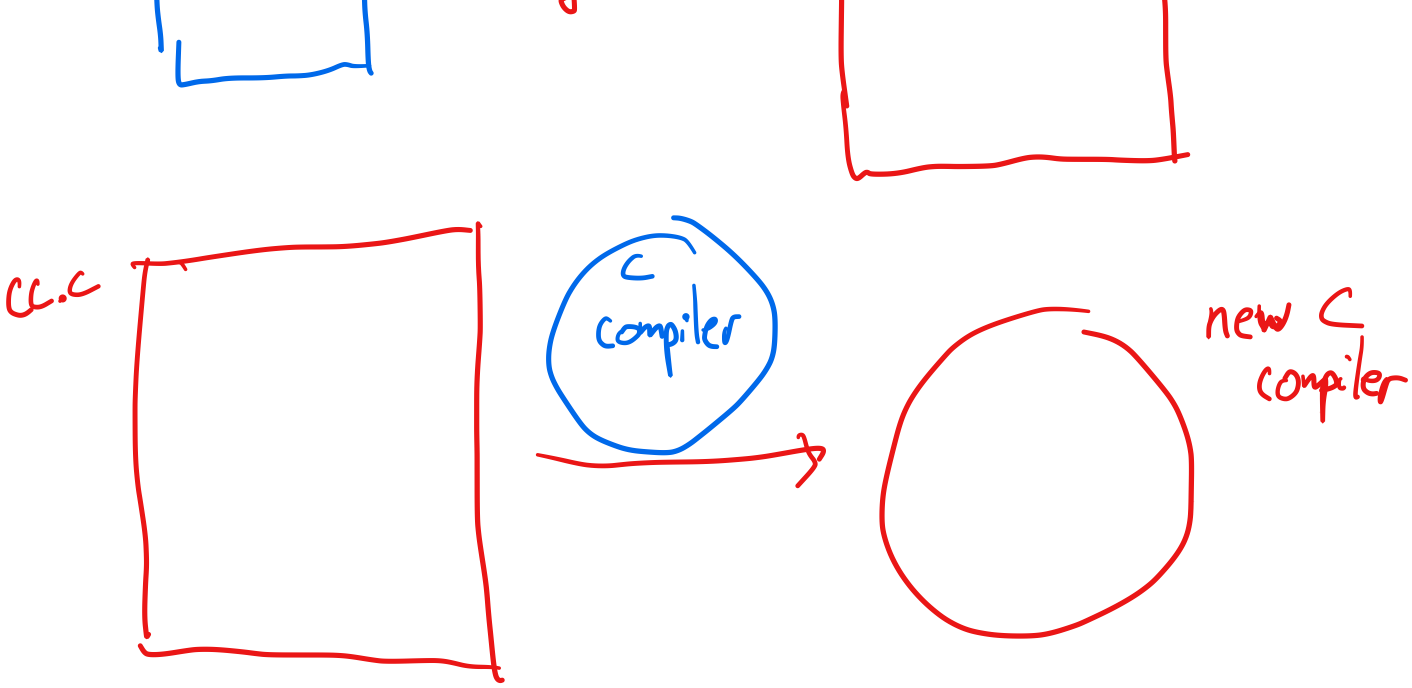
D.2. Second attempt:

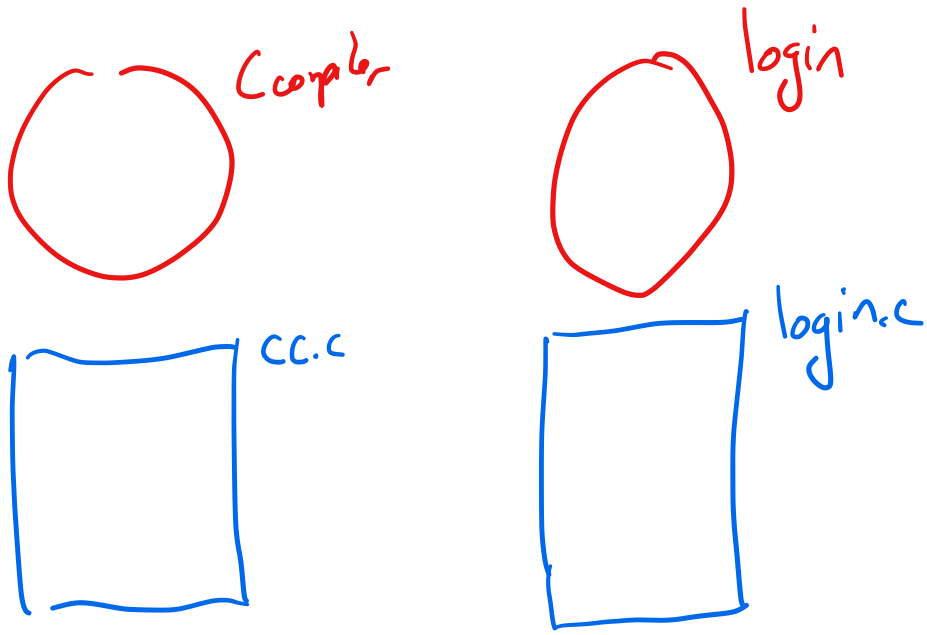
"Problem": recompile login.c and login becomes non-bugged again

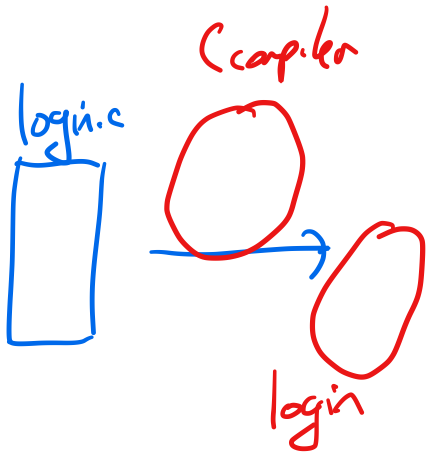


D.3. Third attempt: bug the C compiler!



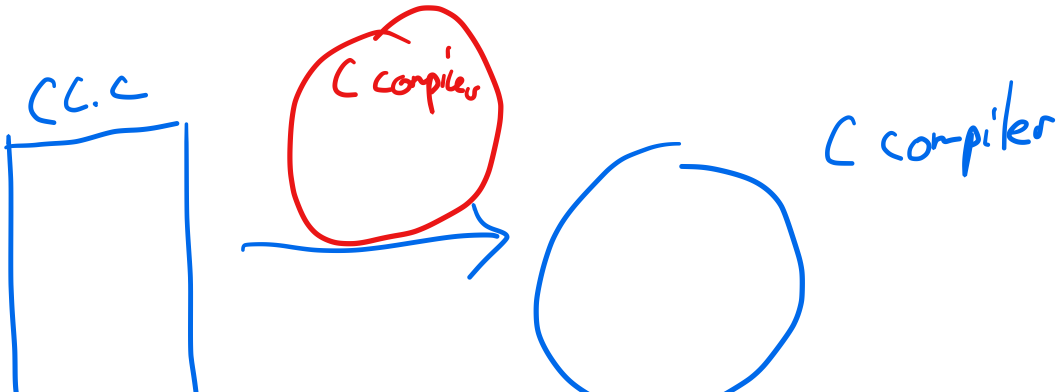


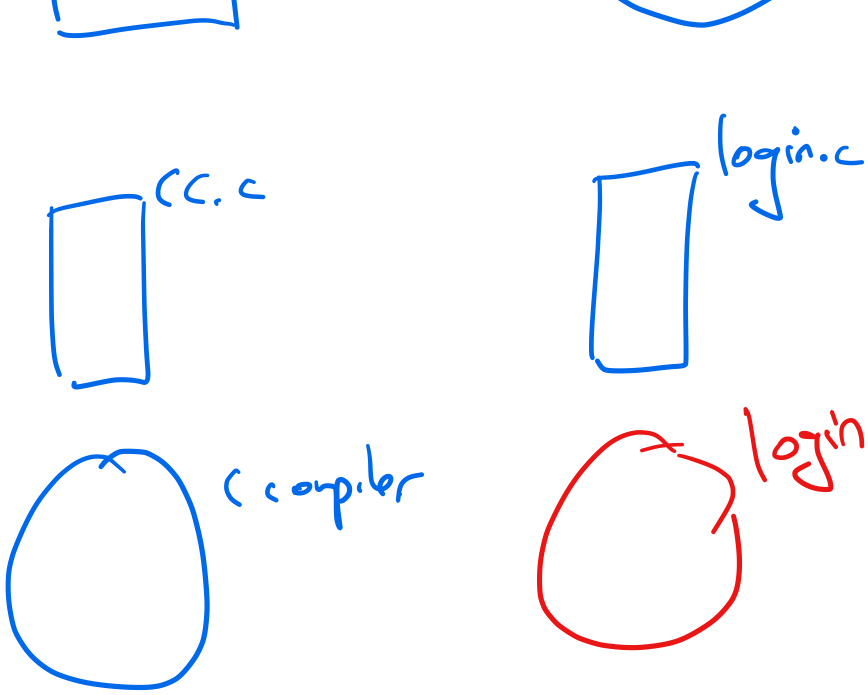




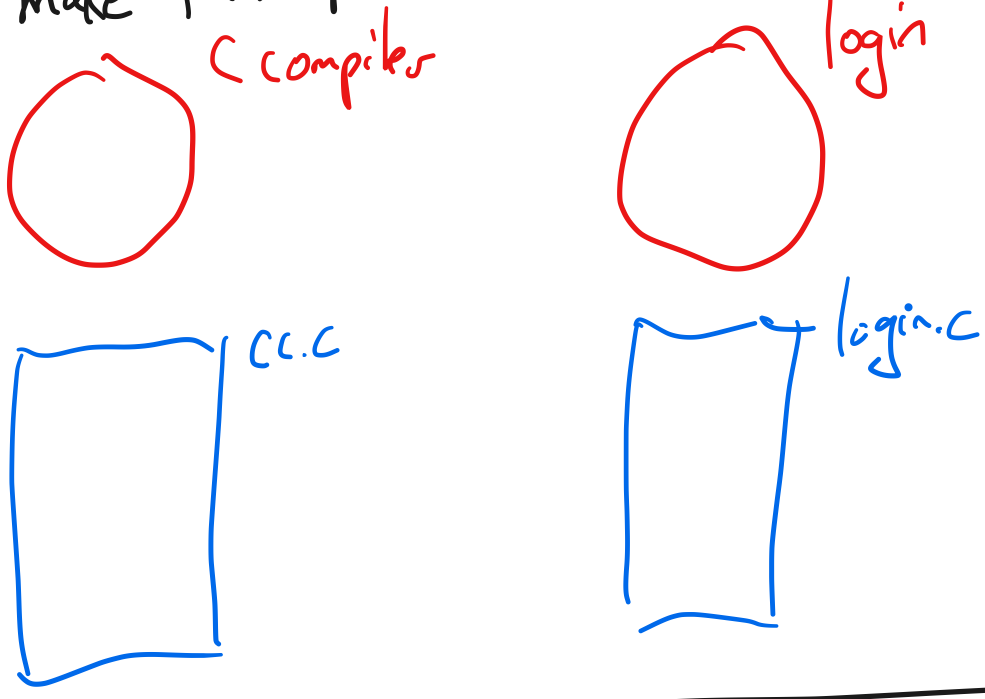
Close!! No trace in source.

- Recompile login ⟹ bugged login





God: make this picture stable:



E. Self-reproducing programs

X = "Output 'X'. Output '='. Output quote mark. Output X. Output quote mark. Output X. [execute whatever]

[execute whatever] Output 'X'. Output '='. Output quote mark. Output X. Output quote mark. Output X.

Output quote mark.

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.

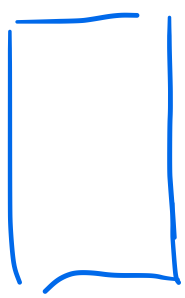
Print this followed by its quotation: =
"Print this followed by its quotation.

---

quine

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

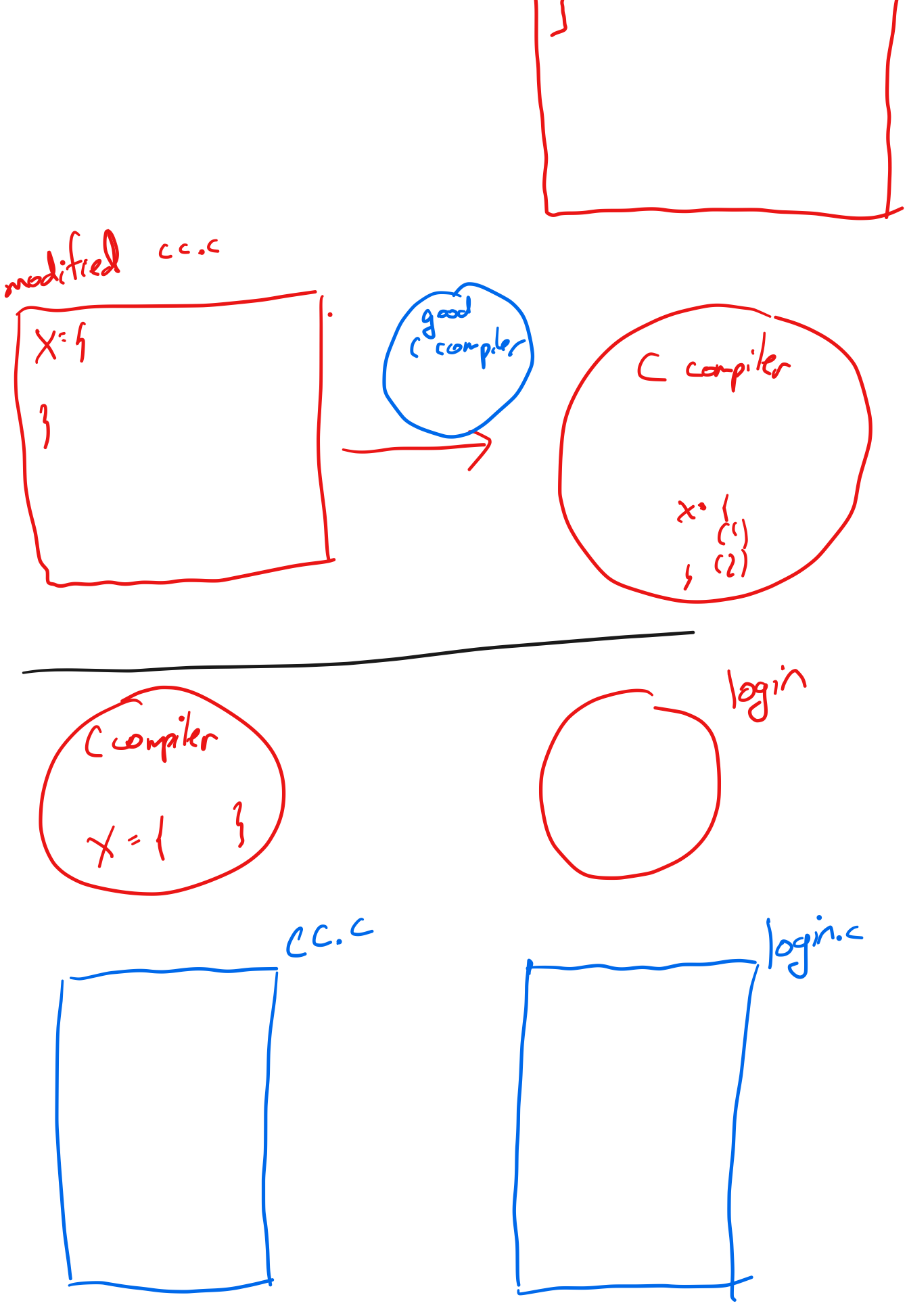
F. Result

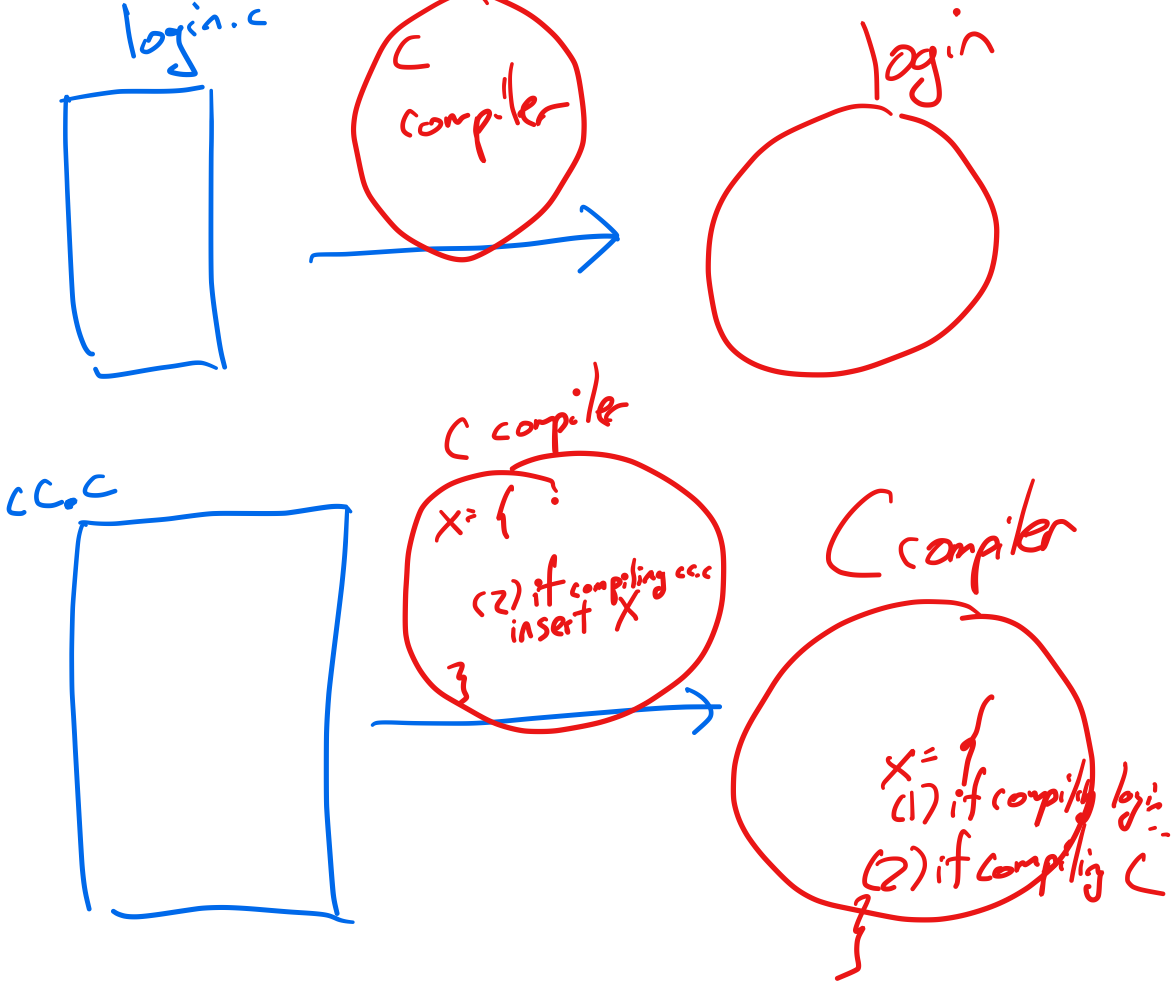


vi, emacs →

cc.c (modified)

X = {
(1) if compiling login,
insert V bug
(2) if compiling C compiler,
insert X.
}


modified cc.c
X = {
}
good C compiler
C compiler
X = {
(1)
; (2)
}
C compiler
X = 1 }
login
cc.c
login.c



```
$ cc x.c -o foo
  a.out
  foo
$ cc -S x.c
  x.s
x.s  --as-->  a.out
```

```
$ cc -S cc.c
$ as cc.s -o cc
  cc
```

c \ 0