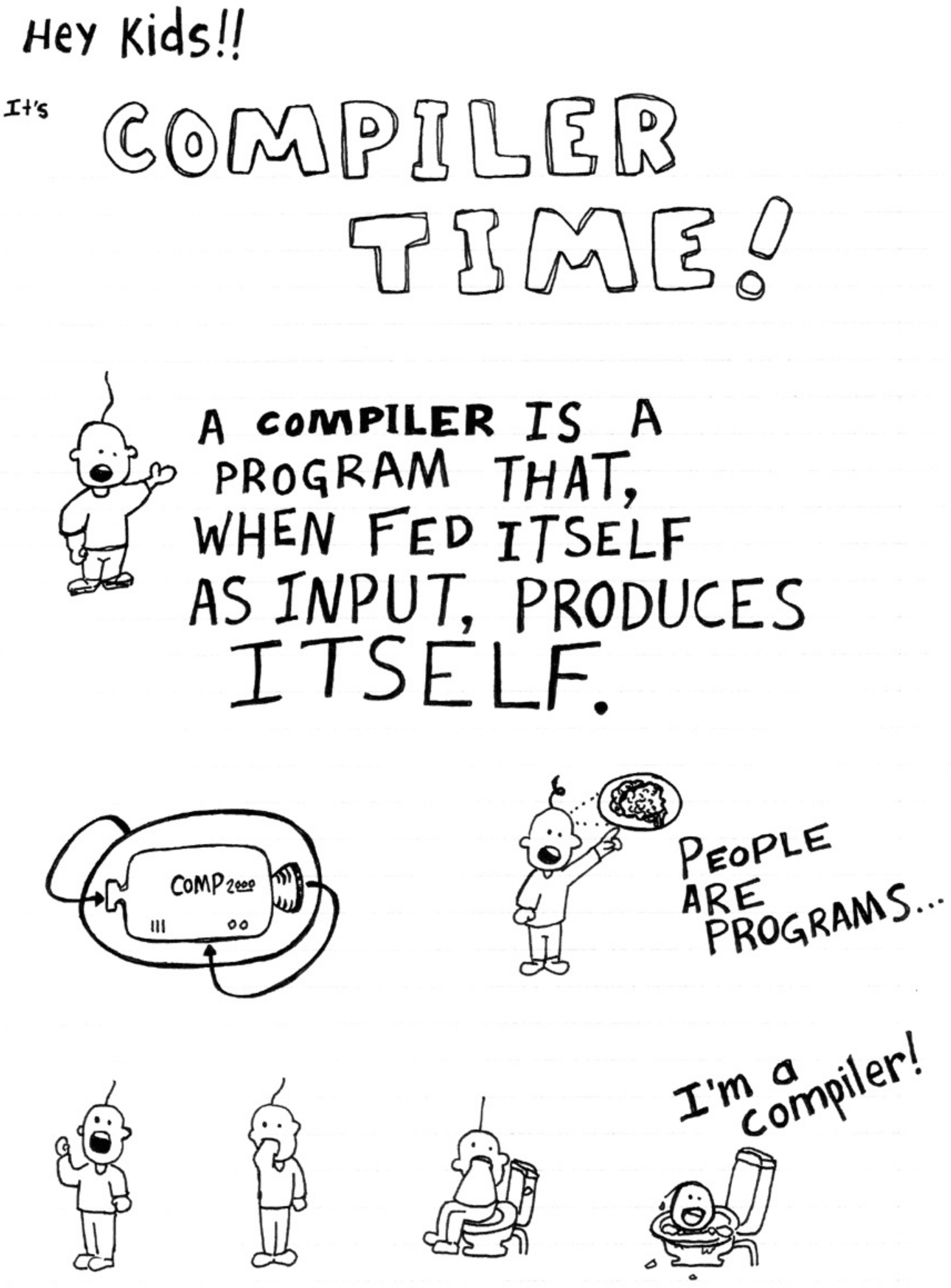
Meeting 4: [redacted]: P0



Announcements

- Homework 1 due this week: Friday at 6:00pm
- I am traveling next week. Guest lecturer: Benno Stein

Submission

- Sign-up for interview
- Code
  - Submit zip on COG (as many times as you like)
  - Upload zip to moodle
- Test
  - Submit (push) tests to https://github.com/csci4555-f17/pyyc-tests-contrib
- Survey: any non-empty answer to the following will receive full credit for this part of the lab.

Important Reminders

- Sign-up for project interviews on the moodle with your partners. Each person should sign up for a slot with the understanding that the two people at the same time are partners.
- Everyone submits a copy of HW1
- Your set of tests should make a significant attempt at testing the subset of the language of interest (P0 for HW1) to uncover bugs in your compilers.

Pair Programming and Project Interviews

There are two main reasons for having a partner.

1. Because the internal architecture of every compiler will end up looking very different, I would like all students to have a "partner" where they know each other's code and can easily ask detailed questions to each other.

2. We will conduct interviews about your projects in pairs.

Interviews

- 40 minutes per pair (each student should sign up)
- Time is limited: come ready with your compiler on your laptop ready to go
- Distance students join by Zoom
- No additional prep needed: just be comfortable with all the details of your compiler and

the concepts behind it

Questions

1. PO — use before defined?
   - other "bad" PO programs?

2. Only discard?

./pyyc foo.py → foo.s

$\underbrace{\text{foo.py} \rightarrow \text{foo.s}}_{\text{equivalent semantics}}$

input()

3+3

---

3. grammar for x86 ~~&~~ PO
4. unoptimal x86? — allocating registers
5. error checking
6. TDD
7. workflow of running your ~~compiler~~

~~architecture~~ [illegible]

⑧ arch internal representation

select : FlatAst × Env → x86IR

flatten : P0AST → FlatAST

$\uparrow$

P0Ast

flatten_expr : P0Expr →

FlatAST ≈ TAC

$\uparrow$

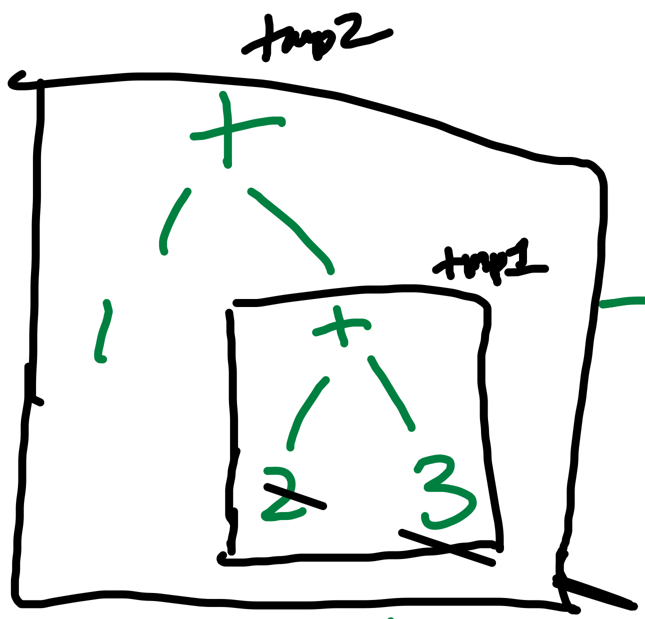
Pythonic

x86

$\uparrow$

movl

addl



"1 + (2+3)"

$\longrightarrow$

tmp1 = 2+3
tmp2 = 1+tmp1

tmp2

flatten_expr: P0Expr → Atomic × List[Assign]

"1 + 2+3" ≈ "(1+2)+3"

a∈Atomic ::= x | n

Add(a, a)

flatexpr e ::= a + a | - a | input() variables x

flat stmt s ::= x = e | print a | e numbers n

Discard(e)

atomics a ::= n | x

flat prog g ::= ε | s g

List[Flat Stmt]

flatten_stmt : $\underline{\text{DO Stmt}}$ → ~~Flat Stmt~~

flat_prog : List[DO Stmt] → List[Flat Stmt]

a ::= %eax | ... | -4(%ebp) | $n

x86 instructions i ::= movl a, a | . . . .

x86 programs p ::= · | i p