Today

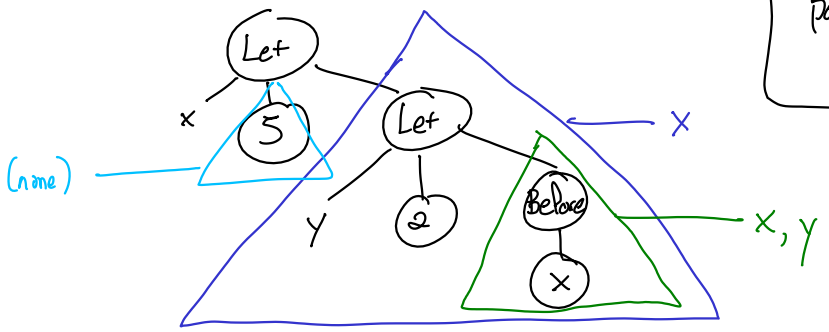
Auklet: integers, unary operations, binary operations, let

Bluebird: booleans, binary representations, conditionals

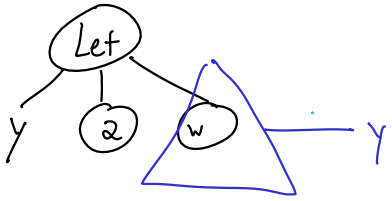
Environments

variable scope:
portion of program in which a variable can be used

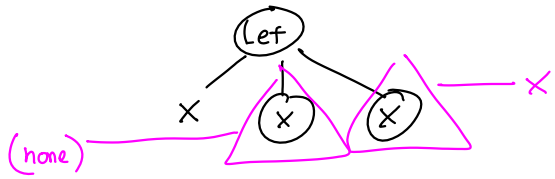
```
let x=5 in
let y=2 in
before(x)
```



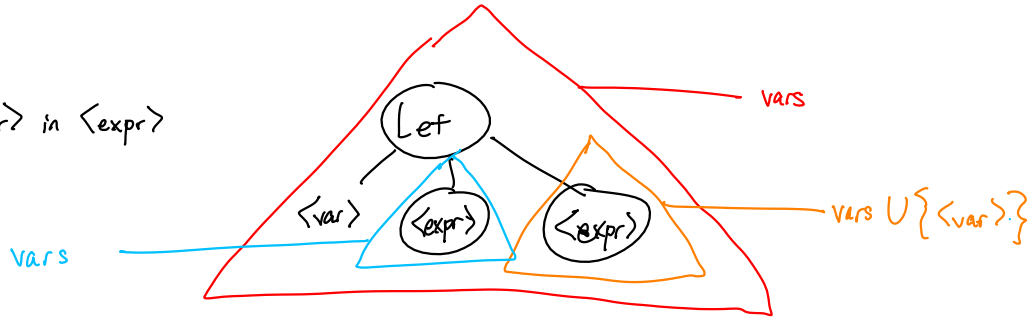

```
let y=2 in
w
```



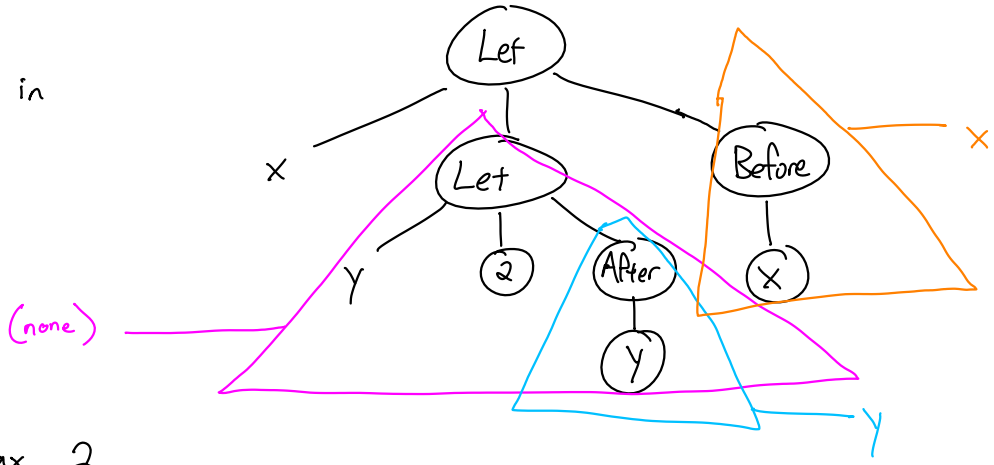

```
let x=x in x
```



let <var>=<expr> in <expr>



```
let x =
  let y = 2 in
  after(y)
in
before(x)
```




```
mov rax, 2
mov [rsp-8], rax
mov rax, [rsp-8]
add rax, 1
mov [rsp-8], rax
mov rax, [rsp-8]
sub rax, 1
```

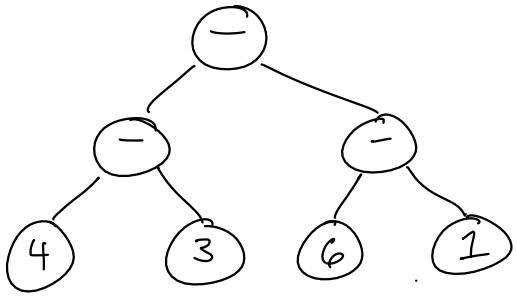
$\{y \mapsto [rsp-8]\}$

$\{x \mapsto [rsp-8]\}$

evaluate a let

1. evaluating 1st expr
2. allocate & store
3. evaluating 2nd expr

$(4 - 3) - (2 - 1)$



In general, for binary operators:

1. Compute left side
2. Store in newly allocated temp var
3. In the env where that var is allocated, compute right side ← keep the memory that holds the answer from left side safe
4. Store in another temp var
5. Load from 1st temp var
6. Do work

allocate named "x"

$(-16, \{w \mapsto [rsp-8]\})$

$\Downarrow$

$(-24, \{w \mapsto [rsp-8], x \mapsto [rsp-16]\})$

allocate temp

$(-16, \{w \mapsto [rsp-8]\})$

$\Downarrow$

$(-24, \{w \mapsto [rsp-8]\})$

$[rsp-16]$

Extension

<expr> ::= ...
| ifnz <expr> then <expr> else <expr>

Label: mylabel:

Comparison: cmp rax, 0

Jumps: jmp mylabel

"not equal" → jne mylabel

jl jle je

jg jge

How to compile?

ifnz after(2) then 6 else 9

```
mov rax, 2
add rax, 1
cmp rax, 0
jne then
mov rax, 9
jmp end
then:
mov rax, 6
end:
```

mov rax, 9 / jmp end: else

then: mov rax, 6: then