1. Review scope
2. Binary operators
3. Conditionals and branching

Auklet —compilation→ x86-64

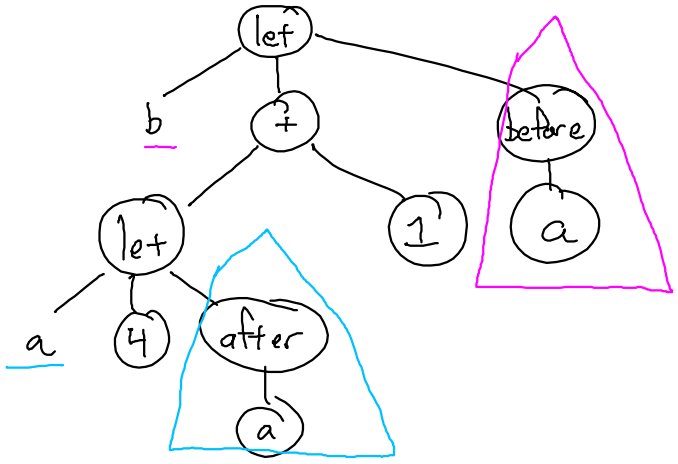
- integers
- variables & let
- binary operations +, −, *
- unary operations after, before

---

scope: part of the program in which the variable can be named/used

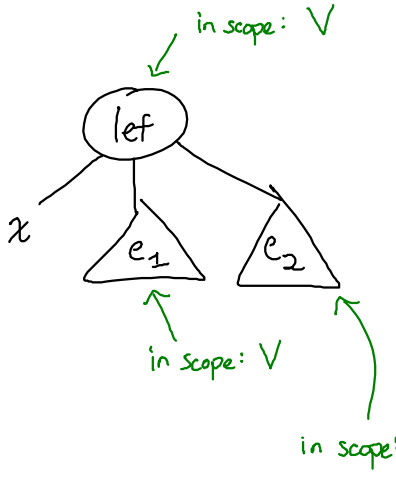
```
let b =
    (let a = 4 in after(a)) + 1
in
before (a)
```

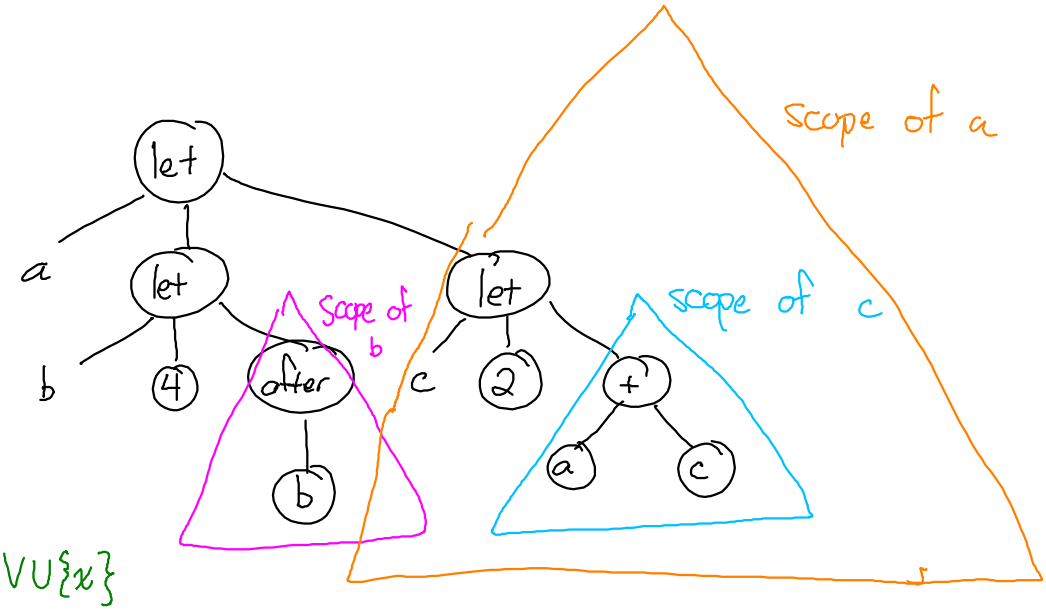
has no meaning

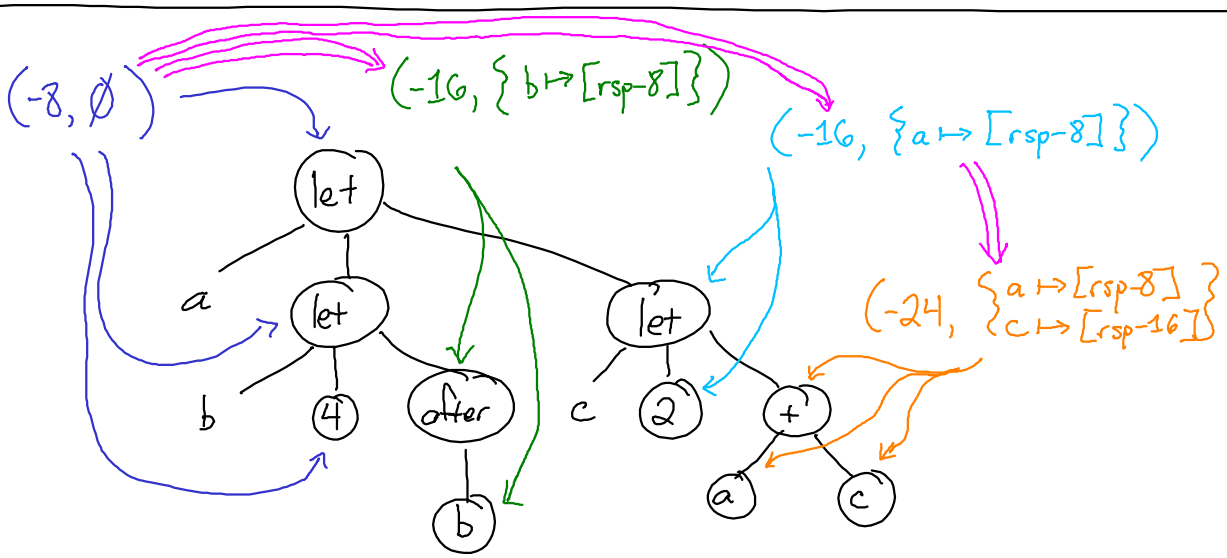


```
let a =
    let b = 4 in after(b)
in
let c = 2 in
a + c
```

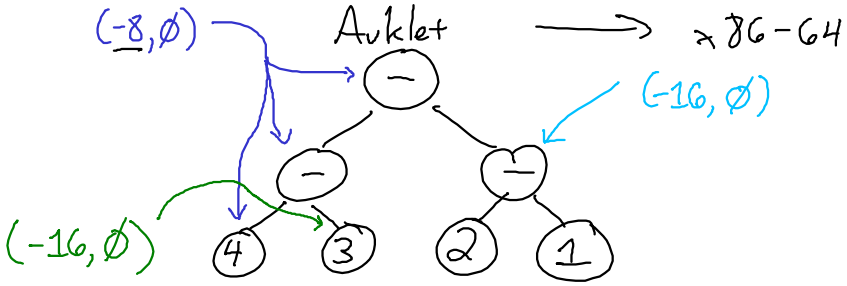
AST







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



calculate e_1

```
mov [rsp-A], rax
```

calculate e_2

```
mov [rsp-B], rax
mov rax, [rsp-A]
sub rax, [rsp-B]
```

```
; Comment!
mov rax, 4
mov [rsp-8], rax
mov rax, 3
mov [rsp-16], rax
mov rax, [rsp-8]
sub rax, [rsp-16]
```

Not Bluebird!

Albatross — C-bird

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

ifnz 4 then 3 else 2 ⟹ 3

ifnz 2-1-1 then 5 else after(2) ⟹ 3

label:

jmp label jz label jl label jg label jge label jnz label je label

cmp rax, rbx

ifnz 4 then 3 else 2

```
mov rax, 4
cmp rax, 0
je equal_6
mov rax 3
jmp end_9
equal_6:
mov rax, 2
end_9:
```

```
jmp compare_8
equal_6:
mov rax, 2
jmp end_9
not_equal_7:
mov rax 3
jmp end_9
Compare_8:
mov rax, 4
cmp rax, 0
je equal_6
jmp not_equal_7
end_9:
```