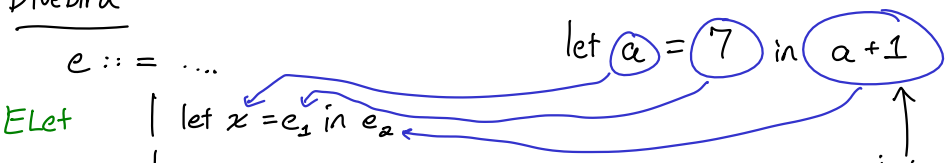
Bluebird

```
e ::= ...
ELet  | let x = e1 in e2
      | ...
EVar  | x
```




```
let a = 7 in w
```

```
let c = 4 in
let d = 5 in
c + d
```

(x = c, e_1 = 4, e_2 = let d = 5 in c + d)

```
let c =
  let d = 5 in
  after(d)
in
c + 1
```

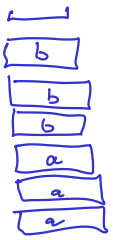
```
let a =
  let b = 5 in
  b + b
in
b   <- unbound
```

```
let c = 5 in
let d = 6 in
c + d
```

```
mov eax, 5
mov [esp-4], eax
mov eax, 6
mov [esp-8], eax
mov eax, [esp-4]
add eax, [esp-8]
```


```
let a =
  let b = 5 in
  after(b)
in
after(a)
```

```
mov eax, 5
mov [esp-4], eax
mov eax, [esp-4]
add eax, 1
mov [esp-4], eax
mov eax, [esp-4]
add eax, 1
```



↳ int StringMap.t * int ← how many bytes have I used? or where is the next free word? or ...

dictionary from string ↦ int

```
ELet(x, e1, e2) ->
    let instrs1 = compile_expression env e1 in
    let env' = environment_alloc_var env x in
    let loc = environment_lookup_var env' x in
    let store_instr = [mov [esp-loc], eax] in
    ...
```


```
EVar(x) ->
    let loc = environment_lookup_var env x in
    ...
```

```
let a = a + 1 in
a
```

```
ELet("a", EPlus(EVar "a", EInt 1),
         EVar("a"))
```

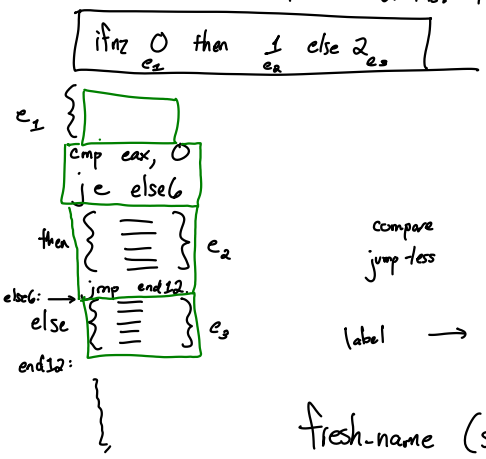
```
let a = 4 in
b
```

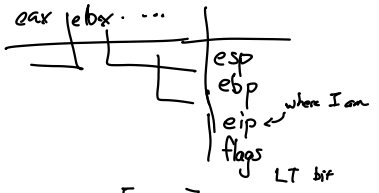
```
ELet("a", EInt 4, EVar "b")
```

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

ifnz e_1 then e_2 else e_3 : evaluate e_1
if produces non-zero, then run and return e_2
else run and return e_3

ifnz 1 then 4 else 6 ⟹ 4
ifnz 0 then 4 else 6 ⟹ 6
ifnz 2-2 then 4 else 6 ⟹ 6





compare → cmp eax, [esp-4]
jump-less → jl foo
mov eax, 0

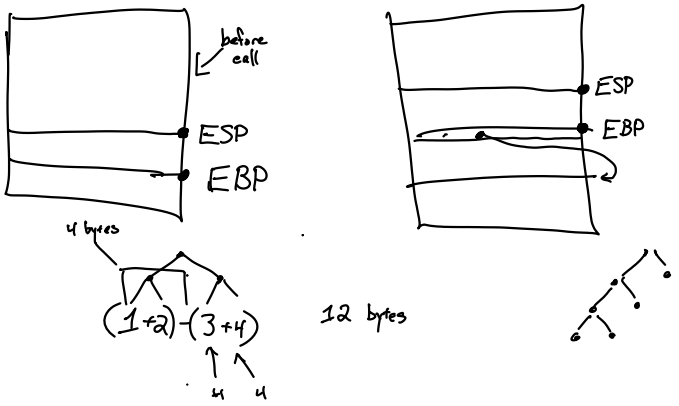
label → foo:

jne jump if not equal

jl jle je jne
jg jge jmp

fresh_name (s : string) : string

cmp eax, ebx



12 bytes