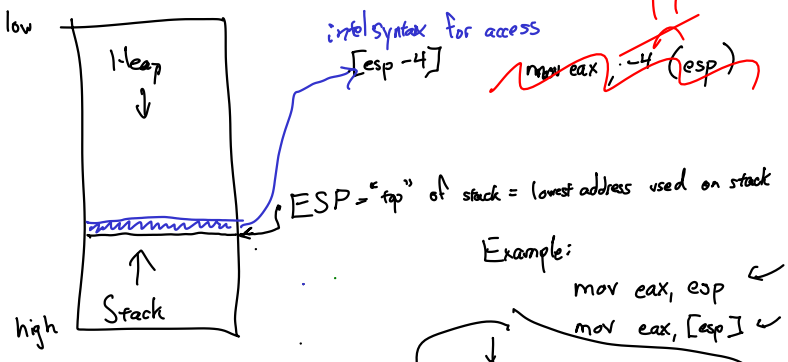
```
let rec compile_expression (e:expr) : instruction list =
  match e with
  | EInt n -> [AMov(ARegister EAX, AConstant n)]
  | EAfter e ->
      let instrs = compile_expression e in
  | EPlus (e1,e2) ->
      let instrs1 = compile_expression e1 in
      ... instrs2 = ... e2 in
```

produces instructions which finish by leaving answer in EAX

(1+2)-(3+4)

need to store — while I work on



Example:

```
mov eax, esp      ; copies ptr
mov eax, [esp]    ; copies value at end of ptr
```

(print 1; 2) + (print 4; 3)

not tuktuk

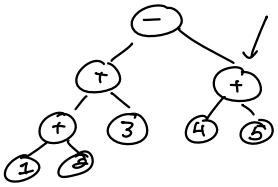
```
let instrs1 = ... in
let instrs2 = ... in
instrs1 @ [mov [esp-4], eax] @ instrs2 @ [mov [esp-8], eax] @
  [mov eax, [esp-4]
   add eax, [esp-8]
  ]

instrs1 @ [mov [esp-4], eax] @ instrs2 @ [sub eax, [esp-4]
                                         neg eax
                                        ]
```

Environment

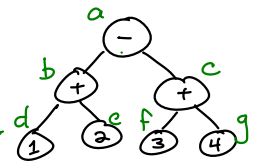
```
type environment = ... ;;
let empty_env : environment = ... ;;
let alloc_temp (env: environment) : environment * int =
  ...
;;
```

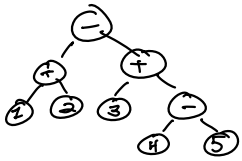
(environment: new state)

```
let compile_expression (e:expr) (env:environment) : instruction list =
  match e with
  ...
  | ESub(e1,e2) ->
      let instrs1 = compile_expression e1 env in
      let (env', offs1) = alloc_temp env in
      let instrs2 = ... e2 env' in
      let (env'', offs2) = alloc_temp env' in
```

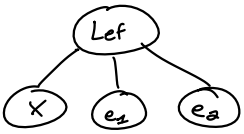


compile a env=4
- compile b env=4
  - compile d
  - alloc temp (4) env'=8 1
  - compile e
  - alloc temp (8) env''=12 2
- alloc temp (4) env'=8 3 ← sum : 1+2
- compile c env'=8
  - compile f
  - alloc temp (8) env'=12 3 ← literal 3
  - compile g
  - alloc temp (12) env''=16 4
- alloc temp (8) env''=12 7





let x = e_1 in e_2



let x = 4 in (x+2)

let x = (let y=2 in y-y) in x+1

let x=3 in let y=2 in x+y

let x = let y = 4 in y in y

let x = ... in

x → map from name to location ; next free offset

let x=4 in

({x↦-4}, 8)

StringMap