"Functional Languages"

- First-class functions — functions are values
- Partial application — pass some but not all args — get back closure
- Anonymous functions — functions can be written without being named (fun n→n) (lambda n: n)

List.map (fun n→n+1)

let rec compile_expression env e = ...

Falcon

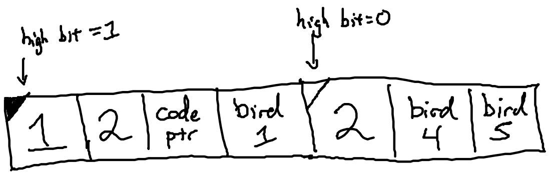
List.map (compile_expression env) exprs

```
def add a b =
  a+b
end
def twice f n =
  f (f n)
end
let inc = add 1 in
let q = (4,5) in
let z = istuple( inc) in
twice inc 4
```

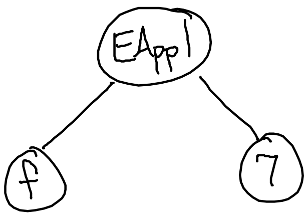
3rd	which fn will be called	closure
2nd	how many params	
1st	how many args so far	
4th	which args	

$2^{10} = 1024 > 1000 = 10^3$

$2^{60} > 10^{18}$



```
0x8000000000000001   0x0000000000000002   0x000 000 000 0040210   0x0000000000000002
0x0000000000000002   0x0000000000000008   0x000000000000000A
```

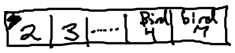


```
def g a b =
  a+b
end
 let f= g 4 in
 f 7
```

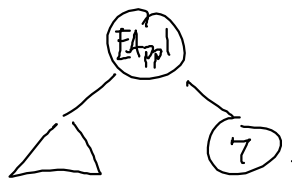
11

```
def g a b c
  a+b*c
end
let f =g 4 in
 f 7
```

closure:



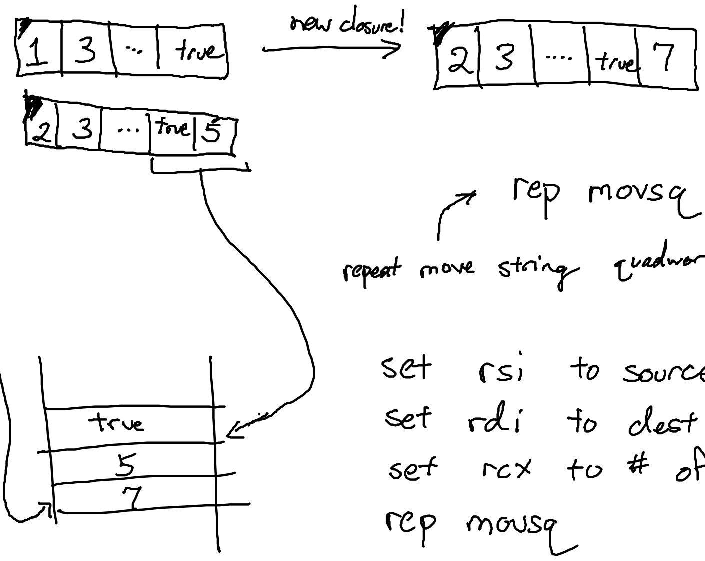
```
def g1 a b =
  a+b
end
def g2 a b c=
  a+b*c
end
........
(if cond then g1 else g2) 4 7
```



1. Look up closure contents by following f ptr
2. Check if $\#_{args} + 1 < \#_{params}$

   If true: not ready.
   a. make a copy of closure
   b. increment # args by 1
   c. put a new arg on end

   Otherwise GO TIME
   a. Save caller-saved regs
   b. Copy args into stack
   c. Call fn
   d. Clean up stack



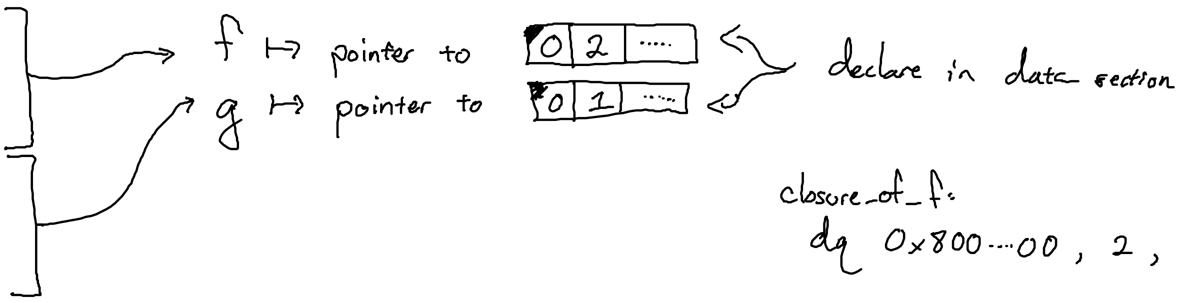
rep movsq

repeat move string quadword (64 bits)

set rsi to source address
set rdi to dest address
set rcx to # of 64-bit words
rep movsq

```
def f a b =
  g
end
def g c =
  c
end
f 3 4 5
```

```
def f a b =
    g
end
def g c =
    c
end
.....
```



```
closure_of_f:
  dq 0x800....00 , 2 , fn_f
```