```
def f x =
  def g x =
    4
  end
  g
end

f 1 2
```

```
def sum x y =
  x + y
end
```

```
def inc y =
  1 + y
end
```

```
let inc = sum 1 in
(inc 2, inc 3)
```

TODAY

Algorithm for constructing & using closures

- Inductive step first
- Base case

Assembly tricks

Functional languages

* Partial application
* Higher-order functions (first-class functions)
* Anonymous functions (Tuesday)

Values

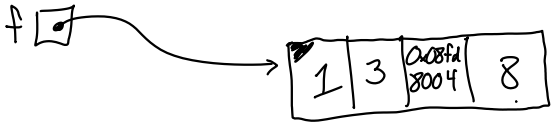
Integers NNNNNNN[nnn0]
Booleans [b111]FFFFFFFF
Tuples PPPPPPP [pp01] → size word | ... | .. .. | ...
Closures PPPPPPP [pp01] → size word | param count | fn addr | .....

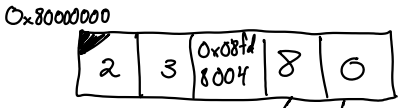
Closure Creation

```
let f = ..... in
f 0
```

f is a closure

what do I do now?

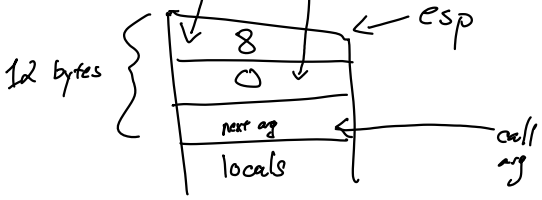




- If #args in current closure + 1 < #params fn needs
  - Copy all of old closure
  - Increase size in copy by 1
  - Append new arg
  - Result is ptr to copy

  } undersaturated

Else
- IT'S GO TIME
- Save any caller-saved regs you care about
- Allocate stack memory for params based on closure
- Copy args into param memory
- Call function based on addr in closure ("call eax")
- Remove args from stack
- Restore registers



E Appl of expr * expr

~~E Call of string * expr list~~

Errors

4 4 ⟹ 11

int + int
tuple [int]
closure anything

Initial Closures

* Not On The Heap
* For each declared fn, closure of 0 args in data section
    * example function f:

```
section .text
    ...
    ...
    function_f:
      ...
      ...
.....
section .data
align 4
heap_cursor:
    dd 0
closure_of_f:
    dd 0x80000000  3  function_f
       arg count   param count
```

Falcon Code

f

Assembly

mov eax, closure_of_f +1

"f" ↦ ArgMem By Label ....

"x" ↦ Arg Memory (Add By Reg Off(EBP, -4))

Rep Movsd

ecx — count

callee-saved:
- esi — source address
- edi — dest address

```
while ecx > 0:
    ecx ← ecx - 1
    [edi] ← [esi]
    esi ← esi + 4
    edi ← edi + 4
```

Copy ecx words from esi to edi

```
mov ecx, 4
mov esi, [ebp - 4]
mov edi, [heap_cursor]
rep. movsd
```