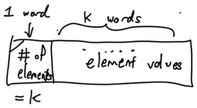
# Falcon

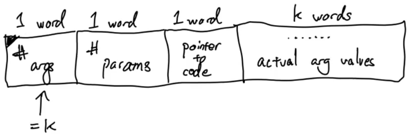
- First-class functions
- Partial application

closure on heap.

```
<expr> ::= ....
         | <expr> <expr>
```



closure



arg: value given to a function during a call

param: variable used by a function to store an arg

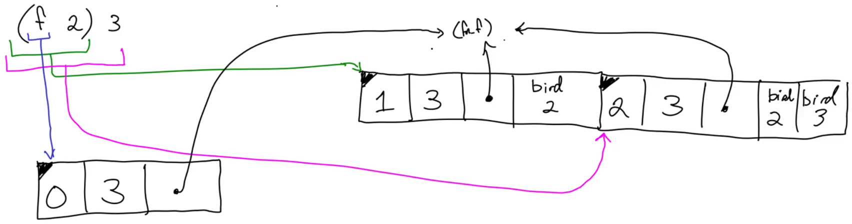
```
def f(x,y)
  x+y
end
f(3, 1+4)
```

params

args

let f= if ... then g else h in

```
def f x y z =
  x+y*z
end
```



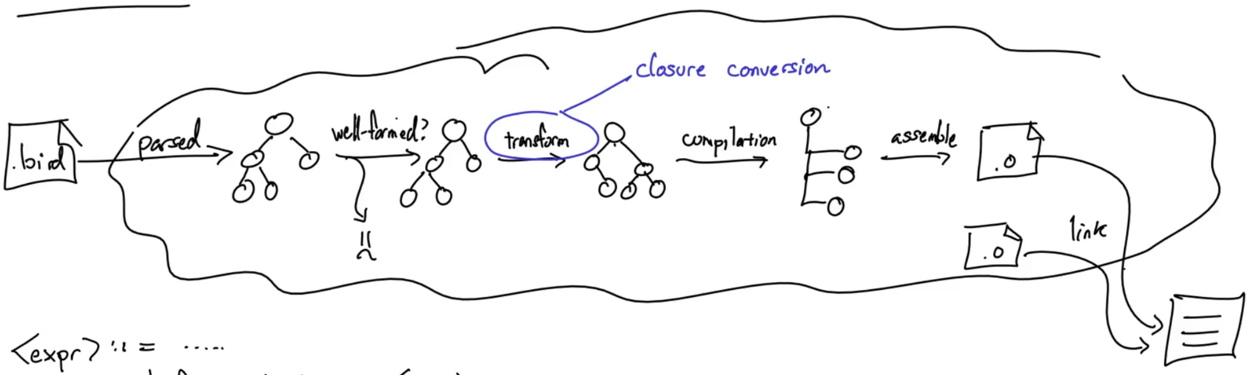
How to handle base case closures (0 arg values):

```
section .data   align 8
  closure_of_f:
    dq 0x8000000000000000, 3, fn_f

section .text

    ⋮

  mov rax, closure_of_f+1
```

f

let x=2 in
let x=3 in
x .

# Finch = Falcon + anonymous functions.



```
<expr> ::= ....
         | fun <ident> -> <expr>
```

## Finch

```
def twice f n =
    f (f n)
end
twice (fun x->x*5) 4
```

↳ (fun x->x*5) ((fun x->x*5) 4)

↳ (fun x->x*5) 20

↳ 100

## Falcon

```
def twice f n
   f (f n)
end
def $0 x =
   x * 5
end
twice $0 4
```

```
let rec closure_convert (e : expr) : expr * declaration list
```

## Finch

```
def twice f n =
   f (f n)
end
let w = 5 in
twice (fun x -> x*w) 4
```

w is "free" in this expression

## Falcon

```
def twice f n =
    f (f n)
end
def $0 w x =
    x*w
end
let w = 5 in
twice ($0 w) 4
```