Let’s add **first class functions**

```
e ::= ...
   | (defn (f x1... xn) e) ; definition
   | (f e1 ... en)         ; function call
```

```
(defn (incr x)
 (+ x 1))

(defn (f it)
 (it 5))

(f incr)
```

```
pub struct Defn {
   pub name: Option<String>,
   pub params: Vec<String>,
   pub body: Box<Expr>,
}

pub enum Expr {
   ...
   Fun(Defn),
   Call(String, Vec<Expr>),
}
```

Code Labels as Values

```
(defn (incr x) (+ x 1))

(defn (f it) (it 5))

(f incr)
```

```
;; definition of incr
fun_start_incr:
 push rbp
 mov rbp, rsp
 sub rsp, 8*100
fun_body_incr:
 mov rax, [rbp - 8*-2]   ; load x
 add rax, 2              ; add <1>
fun_exit_incr:
 mov rsp, rbp
 pop rbp
 ret
;; definition of f
fun_start_f:
 push rbp
 mov rbp, rsp
 sub rsp, 8*100
fun_body_f:
 mov rax, 10
 push rax
 call FIXME1
 add rsp, 8*1
fun_exit_f:
 mov rsp, rbp
 pop rbp
 ret
;; definition of main
our_code_starts_here:
 ; setup stack frame
 push rbp
 mov rbp, rsp
 sub rsp, 8*100
 ; body of `main`
 mov [rbp - 8], rdi  ; save `input`
 mov r11, rsi        ; save start of
heap
 push FIXME2
 call fun_start_f
 add rsp, 8*1
 ; teardown stack frame
 mov rsp, rbp
 pop rbp
 ret
```

Anonymous Functions

```
(let (f   (fn (it) (it 5)))
 (let (inc (fn (z)  (+ z 1)))
   (f foo)))
```

```
;; block for `(let f (fn ...))`
 jmp fun_finish_f
fun_start_f:
 push rbp
 mov rbp, rsp
 sub rsp, 8*101
fun_body_f:
 mov rax, 10           ;; push arg 5
 push rax
 mov rax, [rbp - 8*-2] ;; load `it`
 call rax              ;; call `it`
 add rsp, 8*1          ;; pop arg
fun_exit_f:
 mov rsp, rbp
 pop rbp
 ret
fun_finish_f:
 mov rax, fun_start_f  ;; save `f` as local#1
(f) in "main"
 mov [rbp - 8*2], rax

;; block for `(let inc (fn ...))`
 jmp fun_finish_anon_1
fun_start_anon_1:
 push rbp
 mov rbp, rsp
 sub rsp, 8*100
fun_body_anon_1:
 mov rax, [rbp - 8*-2] ;; load z
 add rax, 2            ;; add 1
fun_exit_anon_1:
 mov rsp, rbp
 pop rbp
 ret
fun_finish_anon_1:
 mov rax, fun_start_anon_1
 mov [rbp - 8*3], rax  ;; save `fn..` as
local#2 (inc) in "main"

;; block for `(f incr)`
mov rax, [rbp - 8*3]    ;; load `foo` into
rax
push rax                ;; push as arg
mov rax, [rbp - 8*2]    ;; load caller `f`
into rax
call rax
add rsp, 8*1
```

How to test function arity?

lam-arity.snek

```
(let (f (fn (it) (it 5)))
  (let (add (fn (x1 x2 x3 x4 x5)
             (+ x1 (+ x2 (+ x3 (+ x4 x5))))))
     (f add)))
```

lam-arity.s

```
;; block to define `f`
jmp fun_finish_anon_1
fun_start_anon_1:
  push rbp
  mov rbp, rsp
  sub rsp, 8*101
fun_body_anon_1:
  mov rax, 10
  push rax                ;; push arg <5>
  mov rax, [rbp - 8*-2]   ;; load `it`
  call rax                ;; call `it`
  add rsp, 8*1
fun_exit_anon_1:
  mov rsp, rbp
  pop rbp
  ret
fun_finish_anon_1:        ;; save `fn` as local-#1 `f`
  mov rax, fun_start_anon_1
  mov [rbp - 8*2], rax

;; block to define `add`
jmp fun_finish_anon_2
fun_start_anon_2:
 push rbp
 mov rbp, rsp
 sub rsp, 8*105
fun_body_anon_2:
 mov rax, [rbp - 8*-2]     x1
 mov rcx, [rbp - 8*-3]   + x2
 add rax, rcx
 mov rcx, [rbp - 8*-4]   + x3
 add rax, rcx
 mov rcx, [rbp - 8*-5]   + x4
 add rax, rcx
 mov rcx, [rbp - 8*-6]   + x5
 add rax, rcx
fun_exit_anon_2:
 mov rsp, rbp
 pop rbp
 ret
fun_finish_anon_2:
 mov rax, fun_start_anon_2
 mov [rbp - 8*3], rax

;; (f add)
mov rax, [rbp - 8*3]    ;; push `add` as arg
push rax
mov rax, [rbp - 8*2]    ;; load `f` into rax
call rax
add rsp, 8*1
```

1. RAX between labels

① # args 'rax' wants

② # args we have at callsite ✓ = 1

r11

<label>	#args

① create "vec"

create "vec"

inc

xxx0	num
111	true
011	false
001	vec
101	fun

lam-free0.snek

```
(let* ((five 5)
       (f   (fn (it) (it five)))
       (inc (fn (z)  (+ z 1))))
   (f foo))
```

rbp+16

Free (non-local) Variables?

```
;; block to define `five` as local#1
mov rax, 10
mov [rbp - 8*2], rax

;; block to define `f`
jmp fun_finish_anon_1
fun_start_anon_1:
push rbp
mov rbp, rsp
sub rsp, 8*101
fun_body_anon_1:
mov rax, ?FIVE         ;; FIXME: what is `five`?
push rax               ;; push arg <5>
mov rax, [rbp - 8*-2]  ;; load `it`
;; CHECK FUNCTION
;; CHECK ARITY
sub rax, 5             ;; remove TAG
mov rax, [rax]         ;; load actual label of `it` into rax
call rax               ;; call `it`
add rsp, 8*1
fun_exit_anon_1:
mov rsp, rbp
pop rbp
ret
fun_finish_anon_1:
;; allocate tuple for fun_start_anon_1
mov rax, fun_start_anon_1
mov [r11], rax         ;; save label
mov rax, 1
mov [r11 + 8], rax     ;; save arity = 1
mov rax, r11           ;; save tuple address
add r11, 16            ;; bump allocation pointer (16-byte aligned)
add rax, 5             ;; tag rax as "function"
mov [rbp - 8*3], rax   ;; save `fn` as local-#2 `f`

;; block to define `inc`
jmp fun_finish_anon_2
fun_start_anon_2:
 push rbp
 mov rbp, rsp
 sub rsp, 8*105
fun_body_anon_2:
 mov rax, [rbp - 8*-2]
 add rax, 2
fun_exit_anon_2:
 mov rsp, rbp
 pop rbp
 ret
fun_finish_anon_2:
;; allocate tuple for fun_start_anon_2
 mov rax, fun_start_anon_2
 mov [r11], rax         ;; save label
 mov rax, 1
 mov [r11 + 8], rax     ;; save arity = 1
 mov rax, r11           ;; save tuple address
 add r11, 16            ;; bump allocation pointer
 add rax, 5             ;; tag rax as "function"
 mov [rbp - 8*4], rax   ;; save `fn` as local#3 `inc`

;; (f inc)
mov rax, [rbp - 8*4]    ;; push `inc` as arg
push rax
mov rax, [rbp - 8*3]    ;; load `f` tuple into rax
;; CHECK function TAG
;; CHECK arity
sub rax, 5
mov rax, [rax]          ;; load actual label
call rax
add rsp, 8*1
```

```
fn free_vars(e: &Expr) -> HashSet<String> {
   match e {
       Expr::Num(_) | Expr::Input | Expr::True | Expr::False
         =>

       Expr::Var(x)
         =>

       Expr::Fun(defn)
         =>

       Expr::Add1(e)
       | Expr::Sub1(e)
       | Expr::Neg(e)
       | Expr::Set(_, e)
       | Expr::Loop(e)
       | Expr::Break(e)
       | Expr::Print(e)
       | Expr::Get(e, _)
         =>

       Expr::Let(x, e1, e2) =>

       Expr::Eq(e1, e2)
       | Expr::Le(e1, e2)
       | Expr::Plus(e1, e2)
       | Expr::Mult(e1, e2)
       | Expr::Vec(e1, e2) =>


       Expr::If(e1, e2, e3) =>


       Expr::Block(es) =>


       Expr::Call(f, es) =>


   }
}
```