CSE 231: Closures

Ranjit Jhala *May 12, 2026*

Closures

In a few steps, we will extend our language with the ability to treat functions as first-class values.

1. Labels
2. Anonymous functions
3. Arity
4. Free Variables
5. Recursion

Part 1: Functions as Values

QUIZ: Semantics

What should the following code evaluate to?

```
(fun (f it)
  (it 5))

(fun (inc x)
  (+ x 1))

(f inc)
```

Abstract Syntax

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

pub enum Expr { ...
    Fun(Defn),
}
```

Parser

No need for `Prog`! But need to change the parser, to produce a single `Expr` instead of a `Prog`.

```
fn prog(defns: Vec<Defn>, e: Expr) -> Expr {
  ...
}
```

Evaluator

The type of `Val` is extended to include `Fun`

```
pub enum Val { ...
    Fun(Defn),
}
```

QUIZ: Extend `eval` to remove `funs`

```
fn eval(&self, e: &Expr, env: &mut Env) -> Val {
  match e {
    ...
    Expr::Defn(defn) =>
      _____________________________,
    Expr::Call1(f, e1) => {
      let v1 = eval(e1, env)?;

      _____________________________________________

      _____________________________________________

      _____________________________________________

      _____________________________________________

      _____________________________________________

    }
    ...
  }
}
```

QUIZ: Compiler: Fun Call

```
Fun(defn) => self.compile_defn(defn),

Call1(f, e1) =>

  _____________________________________________

  _____________________________________________

  _____________________________________________

  _____________________________________________
```

Compiler: Fun Definition

```
fn compile_defn(&mut self, defn: &expr::Defn) ->
String {
  ____________________________________
  ____________________________________
  ____________________________________
  ____________________________________
  ____________________________________
  ____________________________________
  ____________________________________
  ____________________________________
  ____________________________________
  ____________________________________
  )
}
```

QUIZ: Stack Allocation

What is `max_vars` for a `Defn`?

Part 2: Anonymous functions

Since `fun` are also `let`-bound values, we can just use `let` to name them. So we can write:

QUIZ: Semantics

What should the following code evaluate to?

```
; anon function that takes `it` and returns `it 5`
(let (f   (fn (it) (it 5)))

; anon function that takes `z` and returns `(+ z 1)`
(let (inc (fn (z) (+ z 1)))

  (f inc)

))
```

Abstract Syntax

We just need to support **nameless** definitions, so we can reuse `Defn` with an optional name.

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

Update the parser to support `fn` as well as `fun`...

Evaluation, Compilation is unchanged!