Review: Function privileges and lexical scope

The problem: how to look up non-local references in functions.

- **All functions are global**: Every non-local reference is a global variable. (Standard C/C++ rules.)
- **Nested functions**: Use static links to look up the name in the most recent instance of its defining scope.
- **2nd-class functions**: Use dynamic links to look up the name in some instance of its defining scope. It will definitely still be on the stack.
- **1st-class functions**: Local variables must be allocated on the heap using frames.

Example: Local data in non-local context

Let's define a stack in Scheme:

```
(define (make-stack)
  (define stack '())
  (lambda (arg)
    (if (eq? arg 'pop)
        (let ((popped (car stack))
              (stack (cdr stack))
              popped)
          (set! stack (cons arg stack))))))
```

The local variable stack must be persistent. How can we implement this?

Frames

A frame is a data structure that represents the referencing environment of some part of a program. It contains:

- A link to the parent frame. This will correspond to the enclosing scope, (or null for the global environment frame).
- A symbol table mapping names to values. (Notice: no stacks!)

Looking up a name means checking the current frame, and if the name is not there, recursively looking it up in the parent frame.

Function calls create new frames.
SPL Example for Frames

How would this program work using lexical scoping?

```plaintext
new x := 8;
new f := lambda n {
    write n + x;
};
{ new x := 10;
    f(2);
}
```

How do frames compare with activation records on the stack?

Can we use frames for dynamic scoping?

Closures

How are functions represented as values (i.e., first-class)?
With a closure!

Recall that a closure is a function definition plus its referencing environment.
In the frame model, we represent this as a pair of:
- The function definition (parameters and body)
- A link to the frame where the function was defined

Example with closures

Draw out the frames and closures in a Scheme program using our stacks:

```plaintext
(define (make-stack)
    (define stack '())
    (lambda (arg)
        (if (eq? arg 'pop)
            (let ((popped (car stack)))
                (set! stack (cdr stack))
                popped)
            (set! stack (cons arg stack))))
)

(define s (make-stack))
(s 5)
(s 20)
(s 'pop)
```
Class outcomes

You should know:
- How memory for local variables is allocated when in lexical scoping with first-class functions
- Why first class functions require different allocation rules
- What is meant by closure, referencing environment, and frame.

You should be able to:
- Draw the frames and closures in a program run using lexical or dynamic scoping