CSSE 304   Day 17 Summary (items 1-7 are reviews of Day 16)

1. How do variable bindings and lexical scoping work in an actual interpreter? What data structures are needed?

2. Logically, an *environment* is a table of variable names (symbols) and their associated values.
   There is a dynamic **global environment**, `define` and `set!` alter it.

3. **Local environments**: Application of a `lambda`-created procedure, or execution of a `let`, `let*`, or `letrec` creates a *local environment* that holds the local variables and associated values.

4. A procedure (also known as a **closure**) is created when a lambda-expression is evaluated.

5. A closure consists of three parts. See the diagram above.

6. Note that a lambda expression is *not* a procedure. What is it?

var	val	
var	val	Pointer to enclosing environment
var	val	

list of formal argument names	code (body the procedu
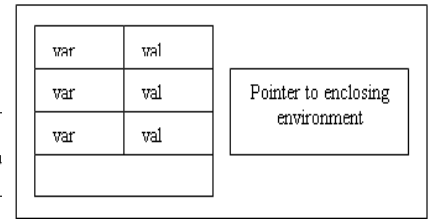
7. **Procedure application:**
   a. The expressions for the procedure and its arguments are evaluated.
   b. A new local environment is created.
      i. Each variable from the procedure's formal parameter list is bound to the corresponding value from the actual argument list.
      ii. The new environment's "pointer to an enclosing environment" is set to point to the local environment that is the third part of the closure.
   c. The body of the procedure is evaluated, using this new local environment. If a variable is not found in local environment or something it points to, look in the global environment.

8. **Nested Lambda example:**
```
((lambda (x)
   ((lambda (y)
      (+ x y))
    15))
  20)
```

9. **Another example:**
```
>(define fact
       (lambda (n)
          (fact2 n 1)))
>(define fact2
       (lambda (n acc)
      (if (zero? n)
          acc
          (fact2 (- n 1) (* n acc)))))
>(fact 3)
```

10. **Evaluate let* expressions**
    Expand the let* to nested lets and then evaluate.

11. Evaluate letrec expressions

Create a new local environment, similar to a let environment, except that:

- The "saved environment" pointers of all closures that are bound to the letrec variables point to the new environment, not the enclosing environment.
- Evaluate the body of the letrec in this new environment.

12. letrec example

```
(define odd?
  (letrec ([odd? (lambda (n)
                    (if (zero? n)
                        #f
                        (even? (- n 1))))]
           [even? (lambda (m)
                     (if (zero? m)
                         #t
                         (odd? (- m 1))))])
    (lambda (x)      ; should use eta-reduction here, but
       (odd? x))))   ; diagram is more interesting if we don't!
>(odd? 2)
```

13. Final example

```
>(define f
   (lambda (x)
     (let ([a (lambda (y z) (+ x y z))])
       (lambda (b)
          (a (+ 5 b) x)))))
 >((f 3) 4)
```