MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Spring 2006

Recitation 2
More Scheme: λ and if

Scheme

1. **Special Forms**

   (a) *if* - `(if` *test consequent alternative*`)`
   If the value of the test is not false (#f), evaluate and return the value of the consequent, otherwise evaluate the alternative.

   (b) *lambda* - `(lambda` *parameters body*`)`
   Creates a procedure with the given parameters and body. Parameters is a list of names of variables. Body is one or more scheme expressions. When the procedure is applied, the body expressions are evaluated in order and the value of the last one is returned.

   (c) *cond* - `(cond (`*test consequent*`) (`*test consequent*`) ... (else` *alternative*`))`
   Alternative to if when there are more than two cases. The value returned is the consequent where the first test evaluates to true (anything but #f). If no tests are true, evaluate and return the alternative, if any. The alternative `else` is optional.

Problems

1. Evaluate the following expressions (assuming `x` is bound to 3).

   (a) `(if #t (+ 1 1) 17) --> 2`

   (b) `(if #f #f 42) --> #f`

   (c) `(if (> x 0) x (- x)) --> 3`

   (d) `(if 0 1 2) --> 1`

   (e) `(if x 7 (7)) --> 7`

2. Evaluate the following expressions:

   (a) `(lambda (x) x) --> proc`

   (b) `((lambda (x) x) 17) --> 17`

   (c) `((lambda (x y) x) 42 17) --> 42`

   (d) `((lambda (x y) y) (/ 1 0) 3) --> error`

   (e) `((lambda (x y) (x y 3)) (lambda (a b) (+ a b)) 14) --> 17`

   (f) `(define y (lambda (x) (+ x 1))) --> undefined`

   (g) `(y x) --> 4`

3. Alyssa P. Hacker is working on a proposal to amend the MIT policy on pets in dorms—she wants to permit dogs in addition to cats. To include with her proposal, Alyssa is working out how many toys all of the dogs will need to make sure there's enough to go around.

   For now, Alyssa is working out how many tennis balls all of the dogs will need. Different sized dogs need different amounts, since the larger ones can carry multiple toys at the same time.

   Initially, only four breeds will be allowed on campus: chihuahuas only need one tennis ball, border collies can handle two, golden retrievers can carry three [1], and a newfoundland can carry 4.

   Alyssa realizes that puppies don't need as many, and guesses that half as many will work (round up), so a young chihuaha or border collie needs one, and a golden retriever or newfoundland puppy needs two.

   Alyssa chooses to represent the puppy breeds by the numbers 1,2,3,4 respectively, and the adults by 5,6,7,8.

   (a) Define a procedure named `adult-breed` which when given a puppy breed returns the corresponding adult.

   ```
   (define adult-breed (lambda (x) (+ 4 x)))
   ```

   (b) Write a procedure `puppy-breed` which when given an adult breed returns the correct puppy breed.

   ```
   (define puppy-breed (lambda (x) (- x 4)))
   ```

   (c) Write a procedure named `puppy?` which when given a breed returns true if it is a puppy and false otherwise.

   ```
   (define puppy?
     (lambda (breed)
             (if (< breed 5) #t #f)))
   ```

   (d) Write a procedure named `breed-toys` that takes a breed and returns the number of tennis balls required for that dog.

[1] Yes, I've really seen this done. The other values are made up though

```
(define breed-toys
  (lambda (breed)
          (if (puppy? breed)
              breed
      (* 2 (puppy-breed breed)))))
```

(e) Instead of provisioning for individual dogs, Alyssa wants to order toys by hall, so we'll call the resulting breeds the live on in any given hall a pack. We'll represent a pack as a set of digits corresponding to the breeds present. So, 817, represents a newfoundland, a chihuaha puppy, and an adult golden retriever. Write a procedure called `empty-pack` that takes no arguments and returns an empty pack.

```
(define empty-pack
   (lambda () 0))
```

(f) Write a procedure called `add-to-pack` that takes a pack and a breed and returns a new pack with the both the old and new members. For example, `(add-to-pack 15 2) -> 152`.

```
(define add-to-pack
   (lambda (pack breed)
           (+ (* 10 pack) breed)))
```

(g) Write a procedure named `pack-size` which takes a pack and returns the number of dogs in that pack. For example, `(pack-size 237) -> 3`.

```
(define pack-size
   (lambda (pack)
     (if (= pack 0)
         0
 (+ 1 (pack-size (quotient pack 10))))))
```

(h) Write a procedure named `pack-toys` which takes a pack and retuns the total number of tennis balls required for the whole pack. You might find `quotient` (integer division) useful.

```
(define pack-toys
   (lambda (pack)
     (if (= pack 0)
         0
 (+ (breed-toys (remainder pack 10))
            (pack-toys (quotient pack 10))))))
```

(i) Alyssa needs to plan for the future. Write a procedure called `grown-pack-toys` which takes a pack and returns the number of toys that will be needed once all it's members have grown up.

```
(define grown-pack-toys
   (lambda (pack)
```

```
    (if (= pack 0)
        0
(+ (breed-toys (if (puppy? (remainder pack 10))
                           (adult-breed (remainder pack 10))
   (remainder pack 10)))
           (grown-pack-toys (quotient pack 10))))))
```