Homework 5
Due Friday, 5/6/2011

Please turn in your programs in the dropbox on Blackboard and hand in written assignments and listings before the beginning of class on the due date.

1. (10 points) Activation records
   Draw the stack of activation records for the following Ada program (a) after the first call to procedure b; (b) after the second call to procedure b. Show the static (access) and dynamic (control) links for each activation record. (c) Indicate how x is found in procedure c. Note: “in” indicates that the parameter is call-by-value.

   ```ada
   procedure env is
       integer x = 12;
   procedure a is
       integer y = 2;
   procedure b(in z: integer) is
       procedure c is
           begin
               b(x);
           end c;
       begin
           c;
       end b
   begin
       b(x+y);
   end a;
   begin
       a;
   end env
   ```

2. (15 points) Function Calls and Memory Management

3. (15 points) Function Returns and Memory Management
   Please do problem 7.13 from Mitchell, page 199.

4. (5 points) Closures
   Please do problem 7.15 from Mitchell, page 201. (Note: in ANSI C, all functions must be declared at the top level. There is no procedure nesting.)

5. (15 points) State Monad
   In class we saw a simple Turtle program that used the State monad to keep track of the current state of the turtle. You are to modify that program to add the following four commands:
6. (10 points) **Custom Haskell Control Structures**

You will find it extremely helpful to read Sections 1 and 2 (pp. 1 – 16) of Simon Peyton Jones’ paper, “Tackling the Awkward Squad: monadic input/output, concurrency, exceptions, and foreign-language calls in Haskell” before tackling this problem. You will find a link from the syllabus page.

One of the claimed advantages of first-class actions is the ability to write custom control structures. This question will explore this by asking you to write some familiar control structures.

Below, we provide code for the `whileIO` control structure implemented in the IO monad. We also provide an example of its usage which prints the integers between 0 and 3 inclusive.

```haskell
import Control.Monad.ST
import Data.IORef

whileIO :: IO Bool -> IO () -> IO ()
whileIO b m = ifIO b
  (do {m; whileIO b m})
  (return ()

whileTest = do {v <- newIORef 0;
  whileIO (do{ x <- readIORef v;
                return (x<4)})
  (do{ x<-readIORef v;
           print x;
           writeIORef v (1+x) )))
```

(a) The specification for `whileIO` above is not complete without the `ifIO` custom control structure. Please implement the appropriate `ifIO` action. We provide the type signature and an example usage of the action below.

```haskell
ifIO :: IO Bool -> IO a -> IO a -> IO a

ifTest = ifIO (return (3<4)) (print "True") (print "False")
```

(b) `untilIO` is a very similar control structure to `whileIO`, except the loop condition test 1) executes after the loop body and 2) causes loop exit if true instead of false, as is the case for `whileIO`. Please implement `untilIO` with the provided type signature and also create `untilTest` to print the integers between 0 and 3 inclusive.

```haskell
untilIO:: IO () -> IO Bool -> IO ()
```
For this problem, please turn in the code given here along with the functions you write so that I will be able to grade it without cutting and pasting code.