Homework 8
Due Thursday, 4/4/2019

Please turn in your homework solutions online at https://submit.cs.pomona.edu/2019sp/cs131. As usual, include Haskell programs in a separate file named Hmwk8.hs and with a first line

module Hmwk8 where

Put both the pdf and hs files in a zipped folder before turning them in.

Several questions on this homework involve ML programs. However, they behave identically to equivalent Haskell programs. If you have any problem understanding them, please let me know.

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.

procedure env is
  integer x = 12;
procedure a is
  integer y = 2;
procedure b(z) 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. (10 points) Exceptions (I understand that some of you don’t know ML, but the programs below behave identically to Haskell – aside from exceptions. Let me know if you have any problems understanding the ML code below.)

Consider the following functions, written in ML:
exception Excpt of int;
fun twice(f, x) = f(f(x)) handle Excpt(x) => x;
fun pred(x) = if x = 0 then raise Excpt(x) else x-1;
fun dumb(x) = raise Excpt(x);
fun smart(x) = (1 + pred(x)) handle Excpt(x) => 1;

What is the result of evaluating each of the following expressions?
(a) twice(pred, 1);
(b) twice(dumb, 1);
(c) twice(smart, 0);

In each case, be sure to describe which exception gets raised and where.

4. (20 points) Activation Records for Inline Blocks
Please do problem 7.1 from Mitchell, page 191.

5. (10 points) Time and Space Requirements

6. (15 points) Function Returns and Memory Management
Please do problem 7.13 from Mitchell, page 199.

7. (10 points) Haskell Exceptions via monads
The function stringToNum defined below uses two auxiliary functions to convert a string of digits into a non-negative integer.

```haskell
import Data.Char

charToNum c = ord c - ord '0'
calcList ([], n) = n
calcList (fst:rest, n) = calcList(rest, 10 * n + charToNum fst)

stringToNum s = calcList(s, 0)
```

For instance, stringToNum "3405" returns the integer 3405.

Unfortunately, calcList returns a spurious result if the string contains any non-digits. For instance, stringToNum "3a05" returns 7905, while stringToNum "405" returns -15595. This occurs because charToNum will convert any character, not just digits. We can attempt to fix this by having charToNum raise an exception if it is applied to a non-digit.

(a) Revise the definition of charToNum to raise an exception, and then modify the function stringToNum so that it handles the exception, returning -1 if there is a non-digit in the string. Here is the Haskell code to throw and catch exceptions (via the Exn monad), as well as the code for charToNum and calcList. The only thing missing is the code for stringToNum.
(Note that I changed the name of “catch” to “catchIt” to avoid a name conflict with a different function in the standard prelude.) Please include the given code for Exn as well as your solution in your hs file.

```haskell
import Char

data Exn a = Oops String | Answer a deriving (Show)

instance Monad Exn where
  return a = Answer a -- recall that return :: a -> Exn a

  -- recall that (>>=) :: M a -> (a -> M b) -> M b
  (Oops s) >>= f = Oops s
  (Answer a) >>= f = f a

throw :: String -> Exn a
throw = Oops

catchIt :: Exn a -> (String -> Exn a) -> Exn a
catchIt (Oops l ) h = h l
catchIt (Answer r) _ = Answer r

charToNum :: Char -> Exn Int
charToNum c = if (c < '0' || c > '9') then throw "non-digit"
  else return (ord c - ord '0')

calcList:: ([Char],Int) -> Exn Int
calcList ([],n) = return n
calcList (fst:rest,n) =
  do nextDigit <- charToNum fst
  calcList(rest,10 * n + nextDigit)

stringToNum :: [Char] -> Int
stringToNum s = ...
```

Note that because calcList returns a value of type Exn Int, stringToNum (which should be written with helping functions) will need to extract the value (or -1) from the monad to get an Int answer.

(b) Implement a Haskell function `stringToNum2` to provide the same behavior (including returning -1 if the string includes a non-digit) as in the first part, but without using exceptions. While you may change any function, try to preserve as much of the structure of the original program as possible.

(c) Which implementation do you prefer? Why?

8. (5 points) **Tail Recursion**

Please define a tail recursive function `sumsquares(n)` in Haskell that can compute the sum of the first n squares: \(1^2 + 2^2 + \ldots + n^2\).