

Homework 9

Due Thursday, 11/13/2014

Please turn in your homework solutions online at <http://www.dci.pomona.edu/tools-bin/cs131upload.php> before the beginning of class on the due date.

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. (20 points) **Activation Records for Inline Blocks**

Please do problem 7.1 from Mitchell, page 191.

2. (10 points) **Time and Space Requirements**

Please do problem 7.3 from Mitchell, page 193. Keep in mind our discussion of tail calls.

3. (15 points) **Function Returns and Memory Management**

Please do problem 7.13 from Mitchell, page 199.

4. (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.

```
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.)

```

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 catch the result of `calcList` and then extract the value (or -1) from the monad to get an `Int` answer.

- (b) Implement Haskell functions 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?

5. (15 points) **Exceptions and Recursion**

Please do problem 8.4 from Mitchell, page 230.

6. (5 points) **Tail Recursion**

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

7. (10 points) **Tail Recursion and Exception Handling**

Please do problem 8.5 from Mitchell, page 230.