LING185A, Assignment #1

Due date: Wed. 1/18/2017

Assumptions: Evaluation rules

We have looked closely at two kinds of evaluation steps: *lambda reduction* and *case reduction*.

- The general **recipe** for a lambda reduction step is

$$\texttt{(\textbackslash}v\texttt{ -> }e_1\texttt{)}\ e_2 \quad\Longrightarrow\quad [^{e_2}/_{v}]e_1$$

  where v is a variable and e_1 and e_2 are expressions.

- An **example** showing how to use the recipe for lambda reduction is

$$\texttt{(\textbackslash x -> (x + 3)) (f y)} \quad\Longrightarrow\quad [^{\texttt{(f y)}}/_{\texttt{x}}]\texttt{(x + 3)} \quad=\quad \texttt{((f y) + 3)}$$

  where the variable `x` is playing the role of v in the recipe, the expression `(x + 3)` is playing the role of e_1 in the recipe, and the expression `(f y)` is playing the role of e_2 in the recipe.

- The **recipe** for a case reduction step based on something of type `Shape` is

$$\begin{aligned}
\texttt{case Rock of }\{\texttt{Rock -> }e_1\texttt{; Scissors -> }e_2\texttt{; Paper -> }e_3\} &\quad\Longrightarrow\quad e_1\\
\texttt{case Scissors of }\{\texttt{Rock -> }e_1\texttt{; Scissors -> }e_2\texttt{; Paper -> }e_3\} &\quad\Longrightarrow\quad e_2\\
\texttt{case Paper of }\{\texttt{Rock -> }e_1\texttt{; Scissors -> }e_2\texttt{; Paper -> }e_3\} &\quad\Longrightarrow\quad e_3
\end{aligned}$$

  and the **recipe** for a case reduction step based on something of type `Result` is

$$\begin{aligned}
\texttt{case Draw of }\{\texttt{Draw -> }e_1\texttt{; Win }v\texttt{ -> }e_2\} &\quad\Longrightarrow\quad e_1\\
\texttt{case (Win }e\texttt{) of }\{\texttt{Draw -> }e_1\texttt{; Win }v\texttt{ -> }e_2\} &\quad\Longrightarrow\quad [^{e}/_{v}]e_2
\end{aligned}$$

  where v is a variable and e, e_1, e_2 and e_3 are expressions. (A completely general recipe for *any* type is very ugly to write — but I will trust that you can generalize appropriately from these.)

- An **example** showing how to use the recipe for case reduction based on a `Result` is

$$\texttt{case (Win Paper) of }\{\texttt{Draw -> 5; Win s -> f s}\} \quad\Longrightarrow\quad [^{\texttt{Paper}}/_{\texttt{s}}]\texttt{(f s)} \quad=\quad \texttt{(f Paper)}$$

  where the variable `s` is playing the role of v in the recipe, the expression `Paper` is playing the role of e in the recipe, and the expression `(f s)` is playing the role of e_2 in the recipe.

The recipes above use the notation $[^{e_2}/_{v}]e_1$ to mean "the result of replacing all free occurrences of the variable v in e_1 with e_2". There are two tricks to watch out for here.

- First, notice that only *free* occurrences of variables get replaced. For example,

$$\texttt{(\textbackslash x -> 2*x + (\textbackslash x -> 3+x) 1) 5} \quad\Longrightarrow\quad [^{5}/_{\texttt{x}}]\texttt{(2*x + (\textbackslash x -> 3+x) 1)} \quad=\quad \texttt{(2*5 + (\textbackslash x -> 3+x) 1)}$$

  since only the first occurrence of `x` is free in `(2*x + (\x -> 3+x) 1)`.

- Second — this is trickier and comes up relatively rarely, so don't get too caught up on it straight away — the substitution $[^{e_2}/_{v}]e_1$ is only allowed if there is no free variable in e_2 that also appears in e_1. Without this restriction, we would end up evaluating

  ```
  (\x -> (\y -> 2*x + y)) (y+1)
  ```

  to

  ```
  (\y -> 2*(y+1) + y)
  ```

  which is not what we would want. (Think about why.) (This is "Trap 2" on page 325 of the Keenan and Moss book.)

1 Practice evaluating things by hand

Show how the evaluation of the following expressions would proceed, one "step" at a time. Assume that the names `n`, `f` and `whatItBeats` have been defined via the following code (for example, in a Haskell file that we have loaded):

```
n = 1
f = \s -> case s of {Rock -> 334; Paper -> 138; Scissors -> 99}
whatItBeats = \s -> case s of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper}
```

For the purposes of this question, we'll take one step to be either (i) a lambda reduction step, (ii) a case reduction step, (iii) a substitution using one definition in the code above, or (iv) a single arithmetic addition or multiplication operation. Label each intermediate expression in your answer to indicate which kind of step is being taken.

Here are two examples:

	`(\x -> (3 + x) * 4) 2`	
$\Longrightarrow$	`(3 + 2) * 4`	lambda reduction
$\Longrightarrow$	`5 * 4`	arithmetic
$\Longrightarrow$	`20`	arithmetic

	`f ((\y -> case (2+y) of {1 -> Scissors; 2 -> Paper; 3 -> Rock}) 1)`	
$\Longrightarrow$	`f (case (2+1) of {1 -> Scissors; 2 -> Paper; 3 -> Rock})`	lambda reduction
$\Longrightarrow$	`f (case 3 of {1 -> Scissors; 2 -> Paper; 3 -> Rock})`	arithmetic
$\Longrightarrow$	`f Rock`	case reduction
$\Longrightarrow$	`(\s -> case s of {Rock -> 334; Paper -> 138; Scissors -> 99}) Rock`	substitution
$\Longrightarrow$	`case Rock of {Rock -> 334; Paper -> 138; Scissors -> 99}`	lambda reduction
$\Longrightarrow$	`334`	case reduction

Now, over to you ...

A. `((\x -> (\y -> y + (3 * x))) 4) 1`

B. `((\x -> (\y -> x + (3 * x))) 4) 1`

C. `((\x -> (\y -> y + (3 * y))) 4) 1`

D. `(\y -> y + ((\y -> 3*y) 4)) 5`

E. `(\y -> ((\y -> 3*y) 4) + y) 5`

F. `f ((\fn -> fn Rock) (\x -> whatItBeats x))`

G. `whatItBeats (case Paper of {Rock -> Paper; Paper -> Rock; Scissors -> Scissors})`

H. `(case (n+1) of {3 -> whatItBeats; 2 -> (\s -> Scissors)}) Paper`

I. `case (Win (whatItBeats Rock)) of {Draw -> n; Win x -> (n + f x)}`

A couple of things to note:

- You can check the final results using ghci, but you will be graded on getting all of the intermediate steps correct as well.
- In many cases there are multiple routes to the final result, depending on which part of the expression you choose to simplify first. You'll get the same result no matter which route you take, but some routes involve more work than others.