CS119 – Module 6: Higher Order Functions

Purpose: Abstracting common patterns on computation into a function is a powerful tool in computer science, since it makes it so we don't have to duplicate a lot of work over and over again. We will examine three such abstractions in the functions `every`, `keep`, and `accumulate` and we will be using these three functions to solve many different problems.

Knowledge: This module will help you become familiar with the following content knowledge:

- Higher order functions `evey`, `keep`, and `accumulate`

Activity: With your group perform the following tasks and answer the questions. You will need to copy the `lab6` directory. You will be reporting your answers back to the class in 30 minutes.

1. Take a look at the function `firstLetters` contained in `Example6.hs`.

   ```
   firstLetters::Language->Language
   firstLetters s =
      if (empty s)
       then s
       else (firstItem (firstItem s)) +++ (firstLetters (butFirst s))
   ```

   Try this function on the sentence `sent "sentence of spam"`

2. Take a look at the function `squareSent`.

   ```
   squareSent::Language->Language
   squareSent s =
      if (empty s)
       then s
       else (squareWord (firstItem s)) +++ (squareSent (butFirst s)) where
       squareWord w = if (wordIsNum w)
          then intToWord (square (wordToInt w))
          else w
   ```

   Try this function on the sentence `sent "1 2 3"`

3. Describe how these two functions are similar.

4. Haskell allows a function to be passed as a parameter to another function. This allows us to "abstract" the similarities of these two functions into a *higher order function* which we will call `every`

```
every::(Language->Language) -> Language -> Language
every f s =
   if (empty s)
    then s
    else (f (firstItem s)) +++ (every f (butFirst s))
```

Predict the result of typing the following in the console. The `let` allows us to define a function directly in the console.

```
let f w = word "spam"
every f (sent"one two three")
```

5. The function `keep` is another higher order function. It takes a predicate (a function that returns true or false) as a parameter.

```
keep::(Language->Bool) -> Language -> Language
keep test s =
   if (empty s)
    then s
    else if test (firstItem s)
    then (firstItem s) +++ (keep test (butFirst s))
    else keep test (butFirst s)
```

Predict the result of:

```
let f w = w == (word "spam")
keep f (sent"one spam two spam three")
```

6. The function `accumulate` is another higher order function. It takes a combining function (a function that combines two values) as a parameter.

```
accumulate::(Language->Language->Language) -> Language -> Language
accumulate combine s =
   if (count s)==1
    then firstItem s
    else combine (firstItem s) (accumulate combine (butFirst s))
```

Predict the result of:

```
accumulate (+++) (sent "one spam two spam three")
```

Activity: With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 40 minutes.

Which higher-order function (every, keep, or accumulate) would you use to perform the following tasks? Test each of them out in the file `Activity6.hs`.

1. Create a sentence which contains the count of the letters for each word.

```
numberLetters:: Language -> Language
numberLetters w = intToWord (count w)

countLetters :: Language -> Language
countLetters s = _____ numberLetters s
```

   For example,

```
> countLetters (sent "this is a test")
[4 2 1 4]
```

2. Remove all but the letter 'a' in a given word

```
isAnA :: Language -> Bool
isAnA letter = letter == (word "a")

removeAllButA :: Language -> Language
removeAllButA w = _____ isAnA w
```

   For example,

```
> removeAllButA (word "banana")
 aaa
```

3. Remove all but the letter 'a' in each word of a sentence

```
removeAllButASent :: Language -> Language
removeAllButASent s = _____ removeAllButA s
```

   For example,

```
> removeAllButASent (sent "eat a large banana")
 [a a a aaa]
```

4. Take a sentence which we know contains only numbers and returns the sum of those numbers.

```
addNum :: Language -> Language -> Language
addNum x y = intToWord (wordToInt x + wordToInt y)

sumNums :: Language -> Language
sumNums s = _____ addNum s

> sumNums (sent "4 2 1 4")
 11
```

5. Combine the answers above to count the number of a's in a sentence.

```
countASent :: Language -> Language
countASent s = (_____________ ( _____________ (_____________ s)))
```

For example,

```
> countASent (sent "eat a large banana"))))
 6
```

Complete the following assignments to be submitted for grading. Each should be done individually but you can consult with a classmate to discuss your strategies or if you get an error message that you do not understand.
Write all of your functions in the file `Example6.hs`.In all of these assignments, you will not have to use recursion since all the recursion is built into the higher order functions.

Assignment 1:
Use `every` to write a function `exaggerate` that takes a sentence and doubles all the numbers in the sentence and replaces the word "good" with the word "great and the word "bad" with the word "terrible":

```
> exaggerate (sent "I ate 3 good hotdogs")
[I ate 6 great hotdogs]
```

Hint: You need to write the function that will be applied to each word in the sentence. This function checks for the special cases and returns the appropriate results. Otherwise it should just return the word unchanged.

Criteria for Success: You have written a function that when passed as a parameter into `every` gives you the body of your function `exaggerate`, behaving like the given example.

Assignment 2:
Use `keep` to write a function `firstLast` that keeps only the words in a sentence whose first and last letters are the same:

```
> firstLast (sent "california ohio nebraska alabama maryland")
[ohio alabama]
```

Criteria for Success: You have written a function that when passed as a parameter into `keep` gives you the body of your function `firstLast`, behaving like the given example.

Assignment 3:
Use `accumulate` to write a function `hyphenate` that hyphenates all the words of a sentence together.

```
> hyphenate (sent "one thousand forty five")
one-thousand-forty-five
```

Hint: The combiner function used in `accumulate` must take two words and combine them to return a single word.

Criteria for Success: You have written a function that when passed as a parameter into `accumulate` gives you the body of your function `hyphenate`, behaving like the given example.

Assignment 4:
Use `every`, `keep` and `accumulate` to write a function `acronym`.

```
> acronym (sent "reduced instruction set computer"
risc

> acronym (sent "foundations of computer science")
fcs
```

Small connecting words like "of" are not part of the acronym. You may use the function `realWord` to determine if a word is irrelevant or not.

Criteria for Success: You have used a combination of the higher-order functions to define `acronym`, behaving like the given examples.

Submit your `Example6.hs` file in Canvas for grading.