CS119 – Module 2: Linear Recursion

Purpose: Recursion is a powerful tool for for solving algorithmic problems by having a function repeatedly call itself. We start with a version of recursion called Linear recursion where the function makes a single call to itself within the returned expression.

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

- The substitution model
- Linear recursion

Activity: With your group perform the following tasks and answer the questions. You need to use copy and use the lab2 directory for this module. You will be reporting your answers back to the class in 1 hour.

1. We will start with the algorithm given in the notes for making a paper chain. Examine the discussion of this algorithm at http://phoenix.goucher.edu/~jillz/cs119/chain.pdf and note the recursion in the algorithm. Explain in your own words what is meant by "winding", "base case", and "unwinding" in this example.

2. We can always see what a function is doing by using the substitution model. Take a look at the following function and complete the substitutions to determine the value of the expression `f 3`

```
f::Int -> Int
f n = if n==0
        then 1
        else 2 * n + f (n-1)
```

   At each step in the substitution model you replace the function call with the appropriate expression from the function definition. So for example we would substitute `2*3 + f 2` for `f 3` since the value of n is 3.

```
f 3
2*3 + f 2
2*3 + (_____________________________)
2*3 + (2 *2 + (_____________________________))
2*3 + 2*2 + 2*1 +  _____________________________
_____________________________
```

   Do you see the winding and unwinding?
   You can test if you are correct by typing and executing the function.

3. Use the substitution model to determine the value of the expression `g (word "abc")`

```
g::Language -> Language
g w = if (empty w)
          then w
          else (lastItem w) +++ g (butLast w)

  g "abc"
  'c' + g "ab"
  'c' + (_____________________________)
  'c'+ ('b' + (_____________________________))
  _____________________________
  _____________
```

   You can test if you are correct by typing and executing the function.

4. All linear recursive functions have a base case tested by the `if` and the recursive case which uses the recursive call to the function. What would be an appropriate base case and base value for each of the following problems? As you think about these problems it helps to think about what parameter will be getting smaller at each step in the recursion and at what point you want to stop the recursion.

   (a) The function `power base exp` which computes the base raised to the power. For example `power 2 3` would be 2^3 or 8.

   (b) The function `length w` which computes the length of a word (the same as the `count` function;).

5. For the recursive case you need an expression which involves the recursive call. Complete the following linear recursive functions:

```
power:: Int -> Int -> Int
power base exp = if exp == 0
                    then 1
                    else base * ____________________
```

```
length:: Language -> Int
length w = if (empty w)
                then 0
                else 1 + ____________________
```

6. Complete the linear recursive function which sums the digits of an integer n. For example `digitSum 173` would return 11

```
lastDigit:: Int -> Int
lastDigit n = n `rem` 10

butLastDigit:: Int -> Int
butLastDigit n = n `div` 10

digitSum:: Int -> Int
digitSum n = if n==0
           then  ____________________
           else (____________________) + digitSum ( ____________________)
```

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 `Example2.hs`.

Assignment 1:
Look at the function `downUp`. We want this function to behave as follows:

```
> downUp (word "cake")
[cake ake ke e ke ake cake]

>downUp (word "a")
[a]
```

If you try this however, you will notice that there is an error in the definition of `downUp`. Fix the error. You will want to be careful in considering the base case for the recursion. You may also want to use the function `wordToSent` which converts a word into a sentence.

Criteria for Success: The function uses linear recursion and behaves properly for the examples given above.

Assignment 2:
We want a function `explode` which behaves as follows:

```
> explode (word "dynamite")
[d y n a m i t e]
```

The function takes a word and returns a **sentence** containing the single letters of this word. You do not have to worry about adding spaces since a sentence already has spaces between the words.

Criteria for Success: The function uses linear recursion for the task and returns the correct type, which is a **sentence rather than a word**.

Assignment 3:
We want a function `countdups` which takes a sentence and returns the number of words in the sentence that are immediately followed by the same word:

```
> countdups (sent "y a b b a d a b b a d o o")
3

>countdups (sent "yeah yeah yeah")
2
```

Write a linear recursive function `countdups`. Think carefully about the signature for this function and when you want to stop the recursion.

Criteria for Success: The function uses linear recursion and behaves properly for the examples given above.

Assignment 4:
Define a function `stackCopies :: Int -> Image -> Image` such that `stackCopies n q` will produce a stack of n copies of the quilt image q.

Hint: Your base case will have to be when there is one copy of the image q.

Criteria for Success: The function correctly stacks **any** quilt for an n value greater than 0 using linear recursion.

Assignment 5:
Define a function `quilt :: Int -> Int -> Image -> Image` in which `quilt w h q` will produce a quilt of width w and height h composed of multiple copies of quilt image q.

Hint: Put `stackCopies` of the image together using `sideBySide` with linear recursion.

Criteria for Success: For positive values of w and h a quilt is generated with the proper number of rows and columns.

Just for fun, you can design your own quilts.

Submit your `Example2.hs` file in Canvas for grading.