CS119 – Module 1: Functions and Expressions

Purpose: We will be using the programming language Haskell throughout this course and writing programs in a functional language will force you to think in new ways. To start out, we need to consider expressions and defining functions which return the value of an expression. The purpose of this lab is to get you thinking in terms of expressions rather than in terms of statements as you have previously done.

Skills: After completion of this module you should be able to

1. Use ghci REPL to evaluate expressions
2. Write, save and load Haskell functions

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

- Expressions that evaluate to various types like Integer, Float, Language or Image
- `if` expressions
- Functions and their signatures

Activity: With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 45 minutes.

1. Run the NoMachine application on your laptop or on one of the lab computers and connect to phoenix. Once you are logged into phoenix, open your home directory that you should see on your desktop.

   In any directory you can open a terminal by doing a right click and selecting the Open Terminal option. Open a terminal window in your home directory.

   Copy the lab1 folder from my account by using the command in the terminal window:

   ```
   cp -r ~jillz/cs119/lab1 .
   ```

   Note: The period at the end of the command indicates that the destination for the copy command is the current directory.

2. Open a terminal in the lab1 folder. An alternative is to use your previous terminal window and change the directory by using the command: `cd lab1`

   In your terminal we want to run the Haskell compiler with the command `ghci` and then when you get the ghci prompt use the command `:load Example1.hs` to compile this code. You will know that this succeeded because the ghci prompt will change to `Example1>`

3. Try entering each of the following expressions and explain the result for each.:

   ```
   2 + 3
   sqrt 16 + 1
   word "haskell"
   sent "this is a sentence"
   firstItem (word "haskell")
   butLast (sent "this is a sentence")
   (word "haskell") +++ (word "!")
   (sent "haskell programs") +++ (word "rock")
   member (word "cat") (sent "I love my cat")
   member (word "dog") (sent "I love my cat")
   ```

4. Try entering each of the following expressions and explain the result for each:
   Note: You need to close a quilt window before you get a prompt to type another expression.

   ```
   draw testBB
   draw (quarterTurnRight testBB)
   draw (stack testBB testBB)
   ```

5. **Predict** the value of each of the expressions before trying them out and then verify your predictions.

   ```
   butFirst (word "cat") +++ lastItem(word "meows")
   count ((word "abc") +++ (word "de"))
   draw (stack (quarterTurnRight testBB) testBB)
   draw (quarterTurnRight (stack testBB testBB))
   ```

6. Take a look at the function `addS` in the `Example1.hs` file. Each function has a *signature* which explains the *type* of the function. What does the signature `Language -> Language` mean for this function?

7. In ghci REPL (that is where we have been typing the expressions), type an expression which uses the `addS` function on the word ”cat”.

8. Take a look at the function `thirdPerson` in the `Example1.hs` file. **Predict** what the expression `thirdPerson (word "play")` will do and then verify your prediction. Explain what this function is doing.

9. Take a look at the function `plural` in the `Example1.hs` file. This uses an `if` expression. **Predict** what this function will do on the words ”cat” and ”fly” and then verify your predictions.

10. An `if` expression differs from an `if` statement in that it has to always return a value and the value always has to be of the same type.
    For each of the following explain why or why not they are `if` expressions:

    (a)
    ```
    if 1 < 2 then
       word "zero"
    else
       word "one"
    ```
    (b)
    ```
    if 1 < 2 then
       0
    ```
    (c)
    ```
    if 1 < 2 then
       0
    else
       (word "zero")
    ```

11. Now we will type a new function into `Example1.hs`. Each time you add a function or make a change, you will need to perform the load command in ghci to compile the changes. Complete the following function which deletes the second word from a sentence *s*. Then reload and try out your function in REPL.

    ```
    delete2 :: Language -> Language
    delete2 s = (firstItem s) +++ (_____________________________)
    ```

    Hint: Consider the `butFirst` function.

    An example of how to use the function in REPL:

    ```
    > delete2 (sent "this is a test")
    [this a test]
    ```

12. Complete the function `min3` which is given three words (all of different lengths) and returns the word with the smallest length. Reload and try out your function in REPL.

    ```
    min3 :: Language -> Language -> Language -> Language
    min3 w1 w2 w3 = if (count w1) < (count w2) then
                           if _____________________________
                           then w1
                           else _____________________________
                       else if _____________________________
                           then _____________________________
                           else _____________________________
    ```

    An example of how to use the function in REPL:

    ```
    > min3 (word "lions") (word "tigers") (word "cats")
    cats
    ```

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 `Example1.hs`.

Assignment 1:
Write the following function using only those functions provided for you in the Words module.

```
insertAnd :: Language -> Language
insertAnd s = ______________________________
```

which inserts the word "and" as the next to last word of the sentence.

For example,

```
> insertAnd (sent "John Bill Wayne Fred")
[John Bill Wayne and Fred]
```

Criteria for Success: Your function works for any parameter which is a sentence with at least two words.

Assignment 2:
We know how the provided function `plural` works on the words "fly" and "cat". But if you try the function on "toy" or "box" you won't get the correct results. Modify the function so that it works correctly for words ending in a vowel followed by the letter "y", as well as words ending in the letter "x".
Make sure that you are using correct `if` expressions by nesting the `if` expressions. A sequence of expressions, one after another, or an `if` without an `else` would NOT be correct.
Hint: You may check that a letter is a vowel by seeing if it is a `member` of the word "aeiou"

Criteria for Success: Your function generates the correct plural for words that end in y regardless of whether a vowel or non-vowel precedes the y. Your function should also work correctly for words that end in an x as well as other words that don't end in x or y.

Assignment 3:
Define the quilt functions `halfTurn` and `quarterTurnLeft` which rotate a quilt image appropriately. These functions should be written in the Example1.hs file. Be sure to **maintain abstraction by not looking at the Quilt module!** That means that you should use the quilt functions that we already know to define your new quilt functions

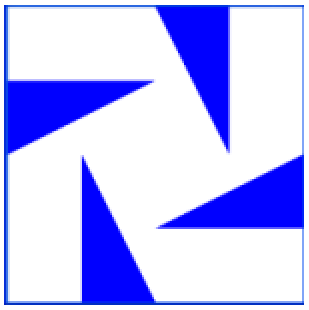
Criteria for Success: Your functions correctly rotate an image, and do so by using our previously defined Quilt functions.

Assignment 4:
Define the quilt function `sideByside` which combines two quilt images next to each other, with the first quilt of the left and the second quilt on the right. Make sure that the two images stay in their original orientation. This function should therefore take two parameters which can be different quilt images.

Criteria for Success: Your function joins any quilts that are the same height and leaves them in the proper orientation.

Assignment 5:
Define a function `pinwheel` where

`draw (pinwheel testBB)` will look like:



Test out pinwheel on some other quilt images as well.

Criteria for Success: Your function pinwheels any quilt image.

If you look at the bottom of the Quilt module (I know I told you not to look before!) you will see the definitions of the basic blocks. They are formed by defining polygons to be drawn. Just for fun, you can create some basic blocks of your own and test them out.

The icon for Firefox is on the top of the toolbar. Use this browser to open up Canvas and submit your `Example1.hs` file for grading.