CS119 – Lab 5
Due Date: March 8

Purpose: Another way of creating an abstraction is having a function return another function through partial application. This will allow your code to be more flexible and reusable. We will see an example of that in this lab.

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

- How to use use partial application to create a function that returns another function

Task: Follow the steps in this lab carefully to complete the assignments. Copy the lab5 folder and complete the following assignments by writing functions in `Lab5.hs`.

Have you ever wondered how a sales representative knows when they have typed in an incorrect credit card number or a scanner knows when it fails to read a UPC barcode correctly? These are examples of *self-verifying numbers*. They are designed to have a certain property that will fail if a simple error occurs, like changing a digit. A self-verifying number with the rightmost digit d_1, second digit d_2, etc. will satisfy the following property for a specific function f and divisor m:

$$f(1, d_1) + f(2, d_2) + f(3, d_3) + ...\text{is divisible by } m \tag{1}$$

We will write a function `makeVerifier` that takes as arguments f and m and returns a function that will perform a verification test for a given number. In other words `makeVerifier` is a function factory that makes the verifying functions for us. Before we do this, let's consider a particular function that this factory is supposed to produce. Suppose we want to check that the sum of the digits is divisible by 17. That is, $f(i, d_i) = d_i$ and m is 17. The code in `Lab5.hs` gives you the functions to get the last digit of a number (by using the remainder of division by 10) and everything but the last digit (by using integer division). We can use these to create a function `sumOfDigits` (also provided). Make sure you understand this code and try it out.

Assignment 1:
Write a function `divisibleSum :: Integer -> Integer -> Bool` which takes a divisor m and a number num and determines whether the **sum of the digits** of num is divisible by m.

Criteria for Success: Test your function with various m and num values and verify that your function correctly determines whether **sum of the digits** of num is divisible by m. For example, the sum of the digits of 123 is 6 so would be divisible by 3 but not divisible by 4.

Consider defining:

```
divisibleSumBy17 = divisibleSum 17
```

We have used partial application to get a new function. The expression `divisibleSumBy17 num` will now verify that the sum of the digits of *num* is divisible by 17. We can now generalize this approach to achieve our function factory `makeVerifier`.

Assignment 2:
Write a function `digitSum` which takes a function f and a number *num* and computes $f(1, d_1) + f(2, d_2) + f(3, d_3) + ...$ for all the digits in *num*.

Criteria for Success: One example of using this function is `digitSum (*) 123` which should compute `1*3 + 2*2 + 1*3` which is 10.

Assignment 3:
Put the pieces together to write the function

```
makeVerifier :: (Integer->Integer->Integer) -> Integer -> (Integer->Bool)
makeVerifier f m = ...
```

Criteria for Success: A simple example of a self verifying number is an ISBN number. The ISBN numbers use the function $f(i, d_i) = i * d_i$ and a divisor of 11. We can use our function factory to create a function which checks ISBN numbers:

```
checkISBN = makeVerifier (*) 11
```

For example, take the legal ISBN 0-201-53082-1 since 1*1 + 2*2 + 3*8 + 4*0 + 5*3 + 6*5 + 7*1 + 8*0 + 9*2 + 10*0 = 99 = 0 (mod 11)
You should now be able to verify this with `checkISBN 0201530821`
Change a digit and verify that it detects that you no longer have a legal ISBN.

Assignment 4:

UPC barcodes use a divisor of 10 and the function $f(i, d_i) = d_i$ when i is odd and $3d_i$ when i is even. Build a verifier `checkUPC`.

Criteria for Success: The UPC number consists of all the digits: the one to the left of the bars, the ones underneath the bars, and one on the right. Check out your function by using the valid UPC code above. Change a digit and verify that it detects that the code is no longer valid.

Assignment 5:
Credit card numbers also have a divisor of 10 and use the function $f(i, d_i) = d_i$ when i is odd. But when i is even, the function returns and $2d_i$ when $d_i < 5$ and $2d_i + 1$ otherwise. Build a verifier for checking credit card numbers.

Criteria for Success: Try out your function with the account number 79927398713. Change a digit and verify that it detects that the code is no longer valid.

Submit your `Lab5.hs` file in Canvas for grading.