MA 151: Homework #2

due Tuesday September 19

Written problems

In each of these, simplify the expressions step-by-step to get the final value. If there is an error, say exactly what the problem is. You should show enough detail to make it clear that you know what is going on. In all cases, you should be able to check your answer by typing the expressions into GHCi.

Assume that the following definition has been loaded into GHCi: `f x = 2 * (x + 1)`
Now evaluate:

1. `if ((f 4) > 3) then (f 2) else 0`
2. `tail (head [[2,3,4]])`
3. `drop 5 (take 20 [1..])`
4. `take 3 (repeat (take 3 (repeat 4)))`
5. `take 5 ('h':(repeat 'i'))`

Programming problems

- Write a function called `isDivBy` which takes two integers and tells whether the first is divisible by the second. For example `isDivBy 15 5` is `True`, and `isDivBy 9 4` is `False`.
- Write a function called `isEven` which takes an integer n and returns the Boolean `True` if n is even and `False` otherwise. Try to do this without using an `if` block.

  Also create a function called `isOdd` which takes an integer n and returns True if n is odd and False otherwise.

  (The above two functions are already built into Haskell as `even` and `odd`.)
- Write a function called `collatz` which takes an integer x and does this formula:

$$\text{collatz}(x) = \begin{cases} 3x+1 & \text{if } x \text{ is odd,} \\ x/2 & \text{if } x \text{ is even.} \end{cases}$$

  You'll need to use `div` rather than `/` to make sure that everything stays as an integer. (This function is a big deal in math today- look up "Collatz conjecture" if you're interested.)
- Write a function called `bro` that takes a string and returns `"Dude!"` if the string begins with letter before 'g' in alphabetical order, and returns `"Sweet!"` if the string begins with 'h' through 'r', and returns `"Bummer"` otherwise. You can assume that the string parameter is always lower-cased. (You can test alphabetical order with characters using `<` and `>`.) (To get 3 different outcomes, do an `if` inside an `if`.)

- Write a function called `upToDouble` which takes an integer x and returns the list from x up to $2x$. So `upToDouble 5` is `[5,6,7,8,9,10]`.
- Write a function called `mirrorRangeNoZero` which takes an integer x and gives the list of numbers from $-x$ to x, not including 0. So `mirrorRangeNoZero 5` is `[-5,-4,-3,-2,-1,1,2,3,4,5]`.
- Write a function called `evenRange` which takes two integers and gives the list of numbers from one to the other, but only includes evens. So `evenRange 5 13` is `[6,8,10,12]`.
- Write a function called `chopper` which takes a list and returns the same list with the first and last 3 elements deleted. (If there are fewer than 7 elements, the answer should be empty.) For example:

  `chopper "calculator"` is `"cula"`
- Define a function called `takeTwice` which takes one `Int` parameter n and a list l and returns a list of length $2n$ consisting of the first n elements of l followed by the first n elements of l again. For example:

  `takeTwice 3 ['a','b','c','d','e']` is `['a','b','c','a','b','c']`

  You should probably use `take` inside your definition.