CSE 341, Winter 2009, Assignment 1
Haskell Warmup
Due: Monday Jan 12, 10:00pm

14 points total (2 points each for Questions 1–4; 3 points each for Questions 5 and 6)

You can use up to 2 late days for this assignment. (The reason for this is so that we can post sample solutions within a reasonable time — if we wait for 6 days that's getting long.)

For each top-level Haskell function you define, include a type declaration. For example, your `centigrade_fahrenheit` function for Question 1 should start with:

```
centigrade_fahrenheit :: Double->Double
```

1. Write a function `centigrade_fahrenheit` that takes Double representing a temperature in Centigrade and converts it to Fahrenheit. Remember to include the type declaration. (`Double` is a built-in Haskell type representing a double-precision floating point number.) If you write this function without a type declaration and let Haskell infer the type, it will actually come up with a more general type – but we're going to ease into Haskell's type system and just declare the function to take a Double and return a Double.

2. Write a function `average3` that takes three numbers (all Doubles) and that returns their average. (To be precise, since in Haskell all functions are curried, `average3` doesn't actually take three parameters; instead, it takes one number and returns a function that takes another number, which returns yet another function that takes the third number, which finally returns the average.) Again, remember to include the type declaration.

3. Write a *recursive* function `double_plus` that takes a list of integers, and returns a new list of integers of the same length, with each element in the new list computed using the formula $2n+1$. For example, `double_plus [1,3,10]` should evaluate to `[3,7,21]`, while `double_plus []` should evaluate to `[]`. Also try your function on an infinite list, for example `double_plus [1..]` or `double_plus [1,3..]`.

4. Write another version of the `double_plus` function, called `map_double_plus`, that uses the built-in `map` function in Haskell. `map_double_plus` should not be recursive. Don't define a named helper function to compute $2n+1$ — use an anonymous function.

5. Write a `sort` function that takes a list of integers and returns a new list of those integers, sorted. Use insertion sort. There is a quicksort function definition in the lecture notes, and a sort function in the Haskell library — don't use these for this question. (We're practicing writing recursive functions in Haskell.) You can use a helper function for this question if needed.

6. A palindrome is a sentence or phrase that is the same forwards and backwards, ignoring spaces, punctuation, and upper vs. lower case. Some palindromes are "Madam, I'm Adam", "Yreka Bakery", and "Doc, note, I dissent, a fast never prevents a fatness. I diet on cod." Write a Haskell function `palindrome` that takes a string as an argument and that returns True if the string is a palindrome and otherwise False. Hint: make use of the Haskell library for this problem; you shouldn't need to write a recursive function at all for your solution. In particular, consider using functions such as `map`, `filter`, `reverse`, and assorted functions in the `Char` module. If you use `Char`, include an `import Char` statement in your program.

Turnin: Turn in your Haskell program, including some well-chosen test cases. Your program should be tastefully commented (i.e. put in a comment before each function definition saying what it does). Style counts! In particular, think about where you can use pattern matching and higher order functions to good effect to simplify your program; and avoid unnecessary repeated computations.

To include test cases, for example for the `palindrome` function, just define them like this as part of your program:

```
-- palindrome function test cases
p1 = palindrome "Yo! Banana Boy!"
p2 = palindrome "Yo! Carrot Girl!"
```

Then `p1` will have the value True and `p2` the value False, assuming your function is working correctly. You don't need to include a separate script showing them running – the TAs can run them. (Shortly, we will see how to declare these in a more useful way, as unit tests. But you don't need to do this for the warmup.)