Assignment 6

Due on **March 6th, at the end of the day**. Please follow the submission instructions in the "notes for all labs" on Moodle.

- Read the "Notes for All Labs" document on Moodle. All assignments must be submitted as specified there. **This means .pdf**.
    - Please number your written assignment answers.
- Read Chapter 11, especially up through 11.4. This assignment mostly follows along with the material covered there. You may need to reference other chapters, such as Chapter 10.
- All of your code should be in a file called your_id_330_lab6.hs, where your_id is your Earlham user ID (dmbarbe16, or whatever).
- Explain anything about your code that is not obvious in comments.
- Make sure that your functions are named correctly and take their arguments in the specified order. Functions that do not meet these requirements will earn zero points.

Written Assignment 6.1: (2 Points)

What is a lambda abstraction? What are two reasons we might want to use one?

Written Assignment 6.2: (4 Points)

What is partial application? What are operator sections?

Written Assignment 6.3: (4 Points)

My friend says that every function in Haskell takes just one argument, but I know I've seen functions that look like they take more than one argument. (`max`, for instance.) What is my friend talking about?

Written Assignment 6.4: (4 Points)

In your **own words**, in about a paragraph, describe what the curry and uncurry functions in section 11.4 of your text do.

Coding Assignment 6.5: (4 Points)

Write your own version of `foldr1` called `myfoldr1`. It should not use `foldr1`.

Coding Assignment 6.6: (2 Points)

Have you wondered why `foldr1` has such an odd name? There is another version, called `foldr`. It is similar to `foldr1`, but it takes an extra argument in between the function name and the list. That argument is just a value, and it is the thing that should be returned by the function when it is used with an empty list. For example:

```
foldr (*) 12 []
```

would return 12.

Write your own version of `foldr` called `myfoldr`. It should not use `foldr` or `foldr1`.

Coding Assignment 6.7: (6 Points)

Write a function called `myfilterA`. It should take two arguments: A list and a unary function that returns a Boolean. (We sometimes call a function that returns a Boolean a *property*). Your function should return a new list that is like the old list, but should only include the values for which the property function is true. For example, my`filterA odd [2,3,4,5]` should return `[3,5]`. (Do not use the built-in `filter` function for this.)

Write a second function called my`filterB`. It should work exactly like my`filterA`, but it should use a different technique to do the filtering.

Coding Assignment 6.8: (4 Points)

Write a function called `titleMachine`. It should take one argument: a string, which represents a title that might be applied to a person's name. For example, we might call `titleMachine "Senator"` or `titleMachine "Doctor"`. Your `titleMachine` function should return a **function** that takes another string as an argument and returns the string with the title put in front of it, with a space in between them.

Coding Assignment 6.9: (4 Points)

Write a function called `binaryArgFlip`. It should take one argument: a binary function. It should return a new function that acts like the old one, but with the order of the arguments flipped. For example, `binaryArgFlip map` should return a new function that works just like map, but which takes map's arguments in the other order.

Written Assignment 6.10: (2 Points)

How might `binaryArgFlip` be useful when we're trying to build a partial application?

Coding Assignment 6.11: (6 Points)

Define a function of your choice. It should take a function as an argument and return a function. Its signature should be something like

```
totalLA :: (Integer -> Integer) -> (Integer -> Integer)
```

It should use lambda abstractions as part of what it does.

Now, define another function that does the same thing (or something related), but which uses partial application instead.

In a comment above each function, indicate that the functions are for part 6.11 and describe what they do.

Coding Assignment 6.12: (6 Points)

Read about the curry and uncurry functions defined in section 11.4 of your text. Define functions:

```
curry3 :: ((a,b,c) -> d) -> (a -> b -> c -> d)
uncurry3 :: (a -> b -> c -> d) -> ((a,b,c) -> d)
```

You may copy the definitions of curry and uncurry into your code and use them in your functions, if you think it will be helpful.

Discussion Prep 6.13: (4 Points)

Include **one to three** thoughts or questions you had about the Chapter 11 reading or related material. You may put an asterisk (*) next to the ones that you **most** want to discuss in class, if any.

Extra Credit Written Assignment 6.EC (2 Points)

Do some research on Moses Schönfinkel. In a paragraph or two, describe why he might be relevant to what we learned about in this chapter. Why do some people think that Currying should be called Schönfinkelization? Cite your sources. You will only receive credit for work that successfully ties him to this week's topics and that is written in your own words. Do not just paraphrase a few paragraphs from Wikipedia.

Preparation for the Future

Moving forward, the expectation will be that you feel comfortable with the topics covered in this assignment. If you do not, you should ask questions on Piazza and/or continue to practice with GHC. If you want more practice, try the various exercises in Chapter 11 of your textbook. (You do not have to submit these, but I would encourage you to do a few.) You could also try creating tests of various kinds for your functions.