Assignment 2
CISC 260, Winter 2018
Topics: Haskell Lists & Tuples

Due Date: 9 a.m. on Wednesday, Feb. 14
For this assignment and every other assignment in CISC 260, I will program a 24-hour "grace period" into OnQ. This means you can submit your assignment up to 24 hours late if you need to, but we will deduct 10% from your mark. After the grade period is over we will not accept any more submissions. If there are special circumstances such as illness that prevent you from completing the assignment on time please inform the instructor via e-mail.

There are two parts to this assignment. The first includes a few problems using lists but not tuples. The second is a little more like a real-life application, using lists and tuples to represent a database.

What to hand in (please read carefully) : Hand in a single file, called `Assignment2.hs`. Its first line must be

```
module Assignment2 where
```

If you don't use this name and this header line your file won't work with our testing code and we will deduct a 10% administrative penalty. (Fixing everyone's file names and header lines adds up to a lot of work!)

Type Declarations: Type declarations are not required for this assignment; no points will be given for including them and no points will be deducted for leaving them out.

An important restriction: You MAY NOT import any Haskell libraries into your function. The only "import" you may use for this assignment is

```
import OlympicDatabase
```

as explained in Part 2. If you import additional files, whether they're part of the official Haskell library or files from another source, we will not mark your assignment.

Part 1: List Functions

A. doubleVowels: This function must take a string as a parameter and return a copy of that string with all lower-case vowels doubled. For the purpose of this problem, y is not a vowel. Just double all instances of "a", "e", "i", "o" and "u".

Examples:

```
*Assignment2> doubleVowels "cat"
"caat"
*Assignment2> doubleVowels "CAT"
"CAT"
*Assignment2> doubleVowels "Queen's"
"Quueeeen's"
*Assignment2> doubleVowels "Albert Einstein"
"Albeert Eiinsteeiin"
*Assignment2> doubleVowels ""
""
*Assignment2> doubleVowels "cpr"
"cpr"
```

B. pairFlip: This function must take a list as a parameter and reverse the order of each pair of values in the list. In other words, the result would contain:

1. the second value in the parameter list
2. the first value in the parameter list
3. the fourth value in the parameter list
4. the third value in the parameter list

and so on

If there are an odd number of values in the list, the last one is not changed (see the third example below).

Examples:

```
*Assignment2> pairFlip [1,2,3,4]
[2,1,4,3]
*Assignment2> pairFlip []
[]
*Assignment2> pairFlip [1.1,4.2,8.9,10.0,3.14]
[4.2,1.1,10.0,8.9,3.14]
*Assignment2> pairFlip [1,5,8,4,2,7]
[5,1,4,8,7,2]
*Assignment2> pairFlip [8,3,4,2,6]
[3,8,2,4,6]
*Assignment2> pairFlip "Kingston"
"iKgntsno"
```

C. **sublistSum**: This function must take two parameters: a list of integers and a single integer (the sum). It must return a list of all of the sublists of the list which add up to the sum. By a sublist, I mean a list of integers which all appear in the original list. They must be in the same order inside the sublist as they were did in the original list, but they don't have to have been consecutive in the original list.

Here are some examples to clarify what I mean by a sublist. [4,6,8] is a sublist of [2,**4,6,8**,10] and [**4,6,8**,10] and of [**4,6,8**] (itself). But [4,6,8] is not a sublist of [4,8,6,10] because the numbers 4,6 and 8 don't appear in the right order. An empty list is considered a trivial sublist of any other list.

Examples:

```
*Assignment2> sublistSum [1..10] 5
[[5],[2,3],[1,4]]
*Assignment2> sublistSum [1..10] 6
[[6],[2,4],[1,5],[1,2,3]]
*Assignment2> sublistSum [1..10] 7
[[7],[3,4],[2,5],[1,6],[1,2,4]]
*Assignment2> sublistSum [8,2,3,5,9] 15
[[8,2,5]]
*Assignment2> sublistSum [8,2,3,5,9] 4
[]
*Assignment2> sublistSum [8,2,3,5,9] 3
[[3]]
*Assignment2> sublistSum [1,2,3] 0
[[]]
```

Part 2: Olympics!

I was watching a pre-Olympic figure skating event recently and I heard the commentators casually mentioning lots of facts about past Olympic medals – for example, the fact that a particular country had only won 2 Olympic skating medals in the last 20 years, or that a certain skater had been on the podium during each of the last 3 Olympics. I was wondering if they had some kind of computer database for looking up these facts when they were needed rather than having them all memorized. And then, of course, since CISC 260 was already in progress, it occurred to me that one could implement such a database in Haskell! This part of the assignment lets you experiment with a simple version of such an Olympic medal database.

Format of an Olympic Medal Database: I decided that one reasonable way to represent an Olympic medal database in Haskell was as a list of tuples, with each tuple representing a single medal and its winner. Each tuple would have 5 parts:

1. The name of the event (a string)
2. The year (an integer)
3. The kind of medal (an integer, 1=gold, 2=silver, 3=bronze)
4. The name of the winner (a string)
5. The country that the winner was competing for (a string)

For example, the following tuple represents the fact that Patrick Chan won the silver medal for Men's Figure Skating in the 2014 Olympics and that he was competing for Canada.

```
("Men's Figure Skating", 2014, 2, "Patrick Chan", "Canada")
```

For events like ice dancing, in which a pair of people compete as a team, I used a single "name" to represent the team – for example:

```
("Ice Dancing", 2014, 1, "Davis & White", "USA")
```

To keep things from becoming more complicated, none of my facts represent sports like hockey with larger teams.

For this assignment, an Olympic medal database consists of a list of this kind of tuple. With the assignment I will supply a sample database including Olympic medals in a few sports in recent years. You do NOT have to create or modify a database for this assignment; you just have to write some functions that would answer questions about a database.

You may not assume that a database will be in any kind of logical order. For example, the three medal winners for a particular event may not be in three consecutive elements of the list. However you may assume that it will be a list of tuples in the format described on the last page. You may not assume that every possible medal will be listed. There may be medals missing and there may be duplicates (because of ties as mentioned earlier, or just because of data entry error).

For an example, in 2014 there was a tie for the top spot in women's downhill skiing, so the facts for that event look like this:

```
("Women's Downhill Skiing", 2014, 1, "Dominique Gisin", "Switzerland"),
("Women's Downhill Skiing", 2014, 1, "Tina Maze", "Slovenia"),
("Women's Downhill Skiing", 2014, 3, "Lara Gut", "Switzerland")
```

with two gold medals and no silver medal.

Near the description of this assignment on OnQ, there is a file called `OlympicDatabase.hs`, which contains a database for a few events in the last four Olympic games. You should download this to use as example data and import OlympicDatabase into your module:

```
module Assignment2 where
import OlympicDatabase
```

(and remember that you may NOT use any additional `imports`!)

It must define a value called `results`, a list of medals in the format described on the previous page.

Feel free to modify your copy of OlympicDatabase.hs to create more test cases or delete some, or even to create a completely different testing file. When we mark your `Assignment2` module we will use a different version of `OlympicDatabase.hs`. Like the version provided on OnQ, it will define a list called `results`, in the same format as the `results` list in the file we're providing for you, but the list will contain a different collection of facts. So you must be careful not to make any assumptions from the data in OlympicDatabase.hs. Your functions must work with any database that's in the right format.

Problem 4: `runnerUp`

Write a function called `runnerUp` that takes 3 parameters:

1. The name of an event (a String)
2. a year (Integer)
3. a medal list (in the format described on the last page)

and returns a String. This function must find out who was the "runner up" (silver medal winner) in the event in the year specified. Don't worry about the possibilities of ties or duplicates for this problem. Just return the first person in the medal list who won a silver medal for the specified event in the specified year and return that person's name as a result. If there is no fact in the medal list that describes a silver medal for that event and that year, return the empty string (`""`).

For example, using the `results` list defined in `OlympicDatabase.hs`:

```
*Assignment2> runnerUp "Women's Slalom" 2014 results
"Marlies Schild"
*Assignment2> runnerUp "Ice Dancing" 2014 results
"Virtue & Moir"
*Assignment2> runnerUp "Hockey" 2012 results
""
```

Problem 5: `medals`

Write a function called `medals` that takes 2 parameters:

1. the name of an athlete (a String)
2. a medal list (in the format described on the last page)

This function must return a list of the medals this athlete has earned. The elements of the list must be tuples of the form (`event`, `year`). The color of the medal doesn't matter for this problem. If the athlete did not earn any medals, the list must be empty.

Here are some examples, where `medals` is defined in the OlympicDatabase.hs file provided on OnQ.

```
*Assignment2> medals "Maria Riesch" results
[("Women's Slalom",2010)]
*Assignment2> medals "Usain Bolt" results
[("Men's 100 Metre Race",2016),("Men's 200 Metre Race",2016),("Men's
100 Metre Race",2012),("Men's 200 Metre Race",2012)]
*Assignment2> medals "Charlie Brown" results
[]
```

Problem 6: `goldCount`

Write a function called `goldCount` that takes 2 parameters:

1. the name of a country (a String)
2. a year (Integer)
3. a medal list (in the format described on the last page)

This function must return an Integer: the number of gold medals the country earned in the year specified, according to the facts in the medal list.

For example:

```
*Assignment2> goldCount "Canada" 2010 results
1
*Assignment2> goldCount "France" 2010 results
0
*Assignment2> goldCount "Russia" 2014 results
2
```

Marking Scheme:

There are 6 functions required for this assignment. Each of them will be marked out of 4 points, for a total of 24 points.

If you submit your assignment during the 24 hour "grace period" after the due date we will deduct 10% from your mark (in other words, we'll multiply it by 0.9).

If you don't use the header and "import" lines as specified in this document, we will also deduct 10%. This sounds picky, but if we have to fix those things in all the submissions in order to use our testing module it will cost us a substantial amoutn of time.

If you submit late AND don't use the header and "import" lines we will deduct 20% (i.e. multiply your raw mark by 0.8).