Haskell Assignment
Programming and Paradigms, Dr. Pierpaolo Dondio

Due Date: 21 December 2016 (15% of total exam marks)

Define the following Haskell functions. Always include the function signature.
(Note: the marks displayed sum up to 100)

1. is_square. [7]
Define a function is_square that takes an integer (positive) and returns True if the number is the square of an integer number. DO NOT use the SQRT function.

2. freq_letter_pc [7]
Define a function **freq_letter_pc**, that displays the percentage of each letter instead of the number of occurrences (do not count symbols other than [a..z]). Transform the text in small case first. Example:

```
freq_letter_pc ''this is text'' =
[(t,0.3),(s,0.2),(i,0.2),(h,0.1),(e,0.1),(x,0.1)]
```

3. db in Haskell [14]
A small database contains two tables, one is a list of cities, identified by a city_id, city_name, city_population and country_id. The other table is a list of countries (country_id, country_name). Table country is related by a 1 to many relation with table cities, country_id is the foreign key.
The following is the content of the table:

```
cities
=[(1,''Paris'',7000000,1),(2,''London'',8000000,2),(1,''Rome'',3000000,3),
(1,''Edinburgh'',500000,2),(1,''Florence'',500000,3),
(1,''Venice'',200000,3), (1,''Lyon'',1000000,1),(1,''Milan'',5000000,3),
(1,''Madrid'',6000000,4),
(1,''Barcelona'',5000000,4),]

countries = [(1,''UK), (2,''France''), (3,''Italy''), (4,''Spain'')]
```

Write the following Haskell functions:

a. get_city_above n – to get all the names of the cities with a population above n (=input)

```
get_city_above 6000000 = [''Paris'',''London'',''Madrid'']
```

b. get_city country_name – to get all the cities given a country_name (not country_ id! You need to have a function that, given the country_name gets the country_id)

```
get_city ''Italy'' = [''Rome'',''Milan'',''Florence'',''Venice'']
```

c. num_city– to list all the country and for each country the number of cities in each country.

```
num_city = [(''UK'',2),(''Italy'',4), (''Fance'',2),(''Spain'',2)]
```

4. **Caesar Cypher [17]**

A (simplified) Caesar cypher is one of the oldest and simplest forms of encryption. Given a positive integer number n and a string of text (for simplicity, we only work with sentences containing small letters and the space symbol), each letter is shifted by n. If, for instance, n=3, then “a” becomes “d”, “b” -> “e” , “c” -> “f” and so on… It is circular, therefore “x” becomes “a”, “y” -> “b” and “z” -> “c”. The space character is unchanged.

a. Write an Haskell function c_encrypt that takes a string of small letters and spaces , an integer positive number and return the same string encrypted with the Caesar cypher

```
c_encrypt “this is a text” 2 = “vjku ku c vgzv”
```

b. Write a function c_decrypt that takes a string of small letters and spaces , an integer positive number and return the same string decrypted using a Ceasar cypher

```
c_decrypt “vjku ku c vgzv” 2 = “this is a text”
```

Using the function getContents, create a version of the two programs running from command line that takes a file as an input (redirect the file to the Haskell program using < (windows) or the pipe | (unix) and display the encrypted (or decrypted) file on the screen (optionally you might redirect it to another file with the operator > (windows) or another pipe | (unix).

5. Euclidean distance between two lists [7]

Create a function eucl_dist to compute the euclidean distance between two vectors. Each vector is represented by a list of float number of unknown length. The two lists must have the same number of elements.

The distance between list x and y is defined as follows:

$$d(x,y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}$$

You can use the built-in Haskell function sqrt.

6. **Language Identification** [20]

The file lang.hs contains two lists of 26 float numbers each. The lists represent the frequency of each letter in English (list eng_freq) and Portuguese (pt_freq). For instance, the letter "b", the second letter in the alphabet, has a frequency of 1.492% in English and 1.04% in Portuguese.

Write a Haskell function get_lang that gets a text and print the message "The text is in English" if the text is written in English or "The text is in Portuguese" if the text is written in Portuguese (as a simplification, we only use text containing the 26 basic letters, without accents or other phonetic symbols). In order to identify the language, compute the frequency distribution of the letters in the text, store it into a list and check if the distribution is closer to eng_freq or pt_freq. You can use the function defined at exercise 2 and 6. Transform the text in small case first.

Using the function getContents, create a version of the program running from command line that takes a file as an input (redirect the file to the Haskell program using > (windows) or the pipe | (unix) and detect the language of the file.

HIGHER ORDER FUNCTIONS

7. Integral of a function [28]

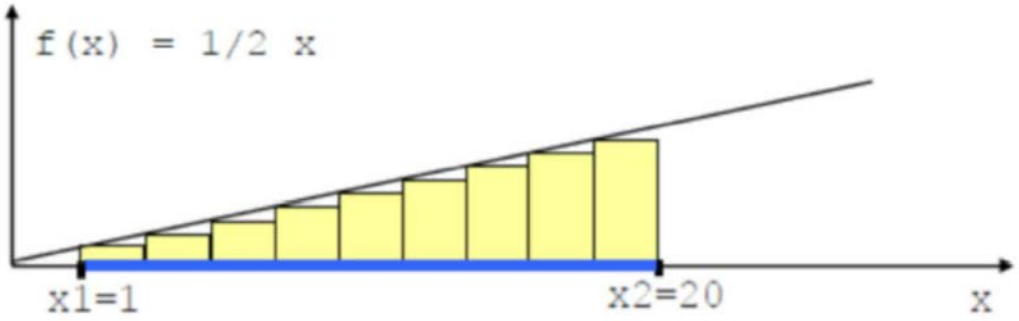
Define a function integral, that approximates the integral of a funciton f. The function integral has the following inputs:

f = a function from a float number to a float number

x1 , x2 = the two integration intervals (x1 < x2), float numbers as well

n = the number of intervals (positive integer >0)

The integral is an approximation of the real integral, and it is the sum of the yellow rectangles as shown in the example below, where we use the function f = ½ x, x1=1, x2=20 and n=9.
Note that the precise value of the integral is 100.



Example of function usage (referred to the above picture):

integral f 1 20 9 = 89.72222 (the sum of the area of the 9 rectangles with same width).

integral f 1 20 10000 = 99.75184 (the precision is higher when the number of intervals is higher - and the interval width is smaller. When the number of intervals goes to infinity, the integral value is the correct one).

Note that f (or any other function you want to use) must be defined. In our example f is defined as follows (but it can be any function returning a real number):

```
f x = 0.5*x
```