Université d'Ottawa
Faculté de génie

École de science
d'informatique
et de génie électrique

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical
Engineering
and Computer Science

Assignment 2
CSI2120 Programming Paradigms

Winter 2018

Due on March 29th before 11:00 pm in Virtual Campus

6 marks

There are [10 points] in this assignment. The assignment is worth 6% of your final mark.

All code must be submitted in a scm file. Screenshots, files in a format of a word editor, pdfs, handwritten solutions, etc. will not be marked and receive an automatic 0.

Question 1. [1.5 points]

Define a global function that calculates the distance between two geographic locations specified by latitude and longitude in radians. You are allowed extra global helper functions.

```
(distanceGPS 45.421016 -75.690018 45.4222 -75.6824 )
 ⇨ 0.6089563918931657
```

As before, this distance can be found as follows (great circle interpolation):
Input: lat_1,lon_1 and lat_2, lon_2 in radians
Output: $distance$ in kilometers

$$d_{radians} = 2 * asin\left(sqrt\left(\left(sin\left(\frac{lat_1 - lat_2}{2}\right)\right)^2 + cos(lat_1) * cos(lat_2) * \left(sin\left(\frac{lon_1 - lon_2}{2}\right)\right)^2\right)\right)$$

$$distance = 6371.0 * d_{radians}$$

Note that the GPS location use angles in degree but the above formula expects angles in radians. The angles can be converted as usual:

$$angle_{radians} = pi * \frac{angle_{degrees}}{180}$$

Question 2. **[2 points]**

Write a function `absDiff` that calculates the absolute difference of the corresponding elements of two lists. The function also evaluates to a list.

```
(absDiff '(1 3 5 6) '(3 5 2 1))
 ⇨ (2 2 3 5)
```

The function must also work if the two input lists are not the same length. We ask you to create two versions of this function:

a) Assume that the missing elements in the shorter list have the value of 0. The result will be of length equal to the longer list.

```
(absDiffA '(1 3 5 6 2 5) '(3 5 2 1))
 ⇨ (2 2 3 5 2 5)

(absDiffA '(1 3 5 6) '(3 5 2 1 2 5))
 ⇨ (2 2 3 5 2 5)
```

b) Ignore the extra values of the longer list. The result will be of length equal to the shorter list.

```
(absDiffB '(1 3 5 6 2 5) '(3 5 2 1))
 ⇨ (2 2 3 5)

(absDiffB '(1 3 5 6) '(3 5 2 1 2 5))
 ⇨ (2 2 3 5)
```

Question 3. **[2.5 points]**

Write a function that organizes an input list as a list of lists or a list of pairs where repeated elements are only added once and instead a count is maintained, i.e., each element of the original list is the first element in a sub-list (or in a pair) with the count as the second element in the sub-list (or in the pair).

a) Create the predicate `duplicatePair` which evaluates to a list-of-pairs.

Examples

```
(duplicatePair '(1 a 5 6 2 b a 5 5))
 ⇨ ((1 . 1) (a . 2) (5 . 3) (6 . 1) (2 . 1) (b . 1))

(duplicatePair '(b a a a))
 ⇨ ((b . 1) (a . 3))
```

Create the predicate `duplicateList` which evaluates to a list-of-list.

Examples

```
(duplicateList '(1 a 5 6 2 b a 5 5))
 ⇨ ((1 1) (a 2) (5 3) (6 1) (2 1) (b 1))

(duplicateList '(b a a a))
 ⇨ ((b 1) (a 3))
```

b) Create the predicate `duplicateList` which evaluates to a list-of-list.

Examples

```
(duplicateList '(1 a 5 6 2 b a 5 5))
 ⇨ ((1 1) (a 2) (5 3) (6 1) (2 1) (b 1))

(duplicateList '(b a a a))
 ⇨ ((b 1) (a 3))
```

c) Create the predicate `duplicateListSorted` which evaluates to a list-of-list but sorts the resulting list by the decreasing count.

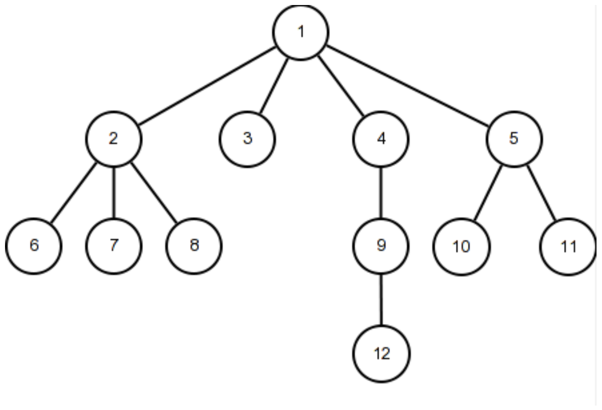
Examples

```
(duplicateListSorted '(1 a 5 6 2 b a 5 5))
 ⇨ ((5 3) (a 2) (1 1) (6 1) (2 1) (b 1))

(duplicateListSorted '(b a a a))
 ⇨ ((a 3) (b 1))
```

Question 4. **[4 points]**

Consider the tree below and its list representation in Scheme.



```
'(1
    (2 (6) (7) (8))
    (3)
    (4 (9 (12)))
    (5 (10) (11)))
```

Write a Scheme function to obtain the children of a node in decreasing order. The order of the nodes in the tree is arbitrary, i.e., you must sort the children. You can assume that the numbers stored in the nodes of the tree are unique.

Example:

```
(define atree '(10
               (2 (4 (9 (3))
                     (12 (1 (2)))
                     (16)))
               (5 (7)
                  (21))
               (6)))

(children 5 atree)
⇨ '(21 7)
```