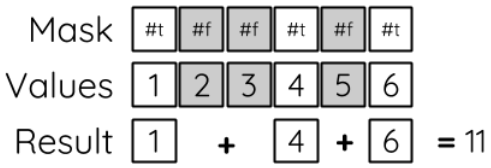
Programming languages (TC-2006)

Homework 08

In this homework, you will practice what you know from the Racket language. Please consider that the purpose of this homework is to allow you to practice and identify strengths and weaknesses. Then, implement these functions as requested and avoid using any built-in functions that already do what you are requested to implement.

1 `maskedsum` (10%)

Write a function in Racket that receives two lists of the same length, one of numbers and the other one of booleans (there is no need for additional verifications). The function must sum all the elements that correspond to true values in the list of booleans. For example, invoking the function with the arguments `'(1 2 3 4 5 6)` and `'(#t #f #f #t #f #t)` will return 11 (the sum of the numbers 1, 4 and 6; the ones located at positions where the mask is `#t`). A graphical representation of this example is depicted as follows:



2 `shift` (10%)

Implement a function in Racket that shifts a list of numbers. Right-shifting (moving the elements to the right) inserts n zeros before the elements already defined in the list (where n is provided as an argument), while left-shifting (moving the elements to the left) inserts n zeros after the elements already defined in the list. This time, a positive n will indicate to right-shift the list and a negative n will indicate to left-shift it.

Some examples of the expected behaviour of the function are shown below:

- `(shift '(3 5 0 0 2) 3)` $\Rightarrow$ `'(0 0 0 3 5 0 0 2)`
- `(shift '(19 4 50 10 2) -2)` $\Rightarrow$ `'(19 4 50 10 2 0 0)`

3 `list2matrix` (10%)

Implement a function in Racket that converts a list into a a matrix. For representing the matrices, use a *by-row* representation. The function must receive three parameters: the list to convert and the number of rows and columns in the resulting matrix. In case the number of rows and columns given as arguments does not allow the matrix to be produced, the function returns the same list that was received as input. You might find the functions `take` and `drop` useful for implementing this function[1].

[1] Please take into consideration that these functions do not seem to work in some online Racket interpreters.

4 `myFilter` (10%)

Write a **recursive** function in Racket that mimics the behavior of the function `filter`. Your function must receive two arguments: a predicate and a list, and the result must be a list containing all the elements in the list provided as argument that make the predicate `true`. For this function you are not allowed to use higher-order functions or any built-in function that provides the requested behavior.

5 `ackermann` (10%)

The Ackermann function (named after Wilhelm Ackermann), is one popular recursive function that returns values that grow rapidly, even for small inputs. Although there are various versions of the function, a common one is defined as follows for nonnegative integers m and n:

$$
\begin{aligned}
A(0, n) &= n + 1 \\
A(m, 0) &= A(m - 1, 1) \\
A(m.n) &= A(m - 1, A(m, n - 1))
\end{aligned}
$$

Implement this version of the Ackermann function in Racket.

6 `swap` (15%)

Implement a function in Racket that swaps two elements in a list (the ones located at positions i and j, provided as arguments). For simplicity, assume that the input is always in the proper format and the values i and j always satisfy that $i < j$ and that they are within the boundaries of the list to modify.

7 `evaluate` (15%)

Write a function in Racket that evaluates polynomials in the form $a_n x^n + a_{n-1}^{x-1} + \ldots + a_2 x^2 + a_1 x + a_0$, where $a_0, \ldots, a_n$ are constants and x is a variable.

8 `select` (20%)

Imagine that you are given a data structure similar to a table. The main difference with a table is that, in this data structure, the registers have an arbitrary number of elements (including none). In each register, the data is stored in pairs `(key, value)`, which are internally represented as lists of two elements. Please note that the data inside the registers does not follow a specific order.

Write a function `select` that receives a data structure as the one described above and one key. The function must return a list containing all the values in the data structure that correspond to the key provided as argument. Additionally, the first element in the returned list must be the key itself.

For example, the following code defines a global variable `table` that contains three registers with different keys and values:

```
(define table '(
   ((name "Charles") (age 24))
   ((age 23) (name "Mary") (lastName "Danvers") (gender "female"))
   ((name "Caroline") (lastName "Ortiz") (age 19))
   )
```

)

Then, calling `(select table 'age)` must return `'(age 24 23 19)`.

Please note that, if the key is not found in the data structure, the resulting list must only contain the key. For example, calling `(select table 'phone)` will return `'(phone)`.

Deliverables

Prepare an RKT file that contains the functions requested and submit it to Canvas. **Please, do not submit other formats but RKT**. To prepare your RKT file, use the code template distributed along with this document. The template contains some test cases for each function to help you verify that your codes work as requested.

I promise to apply my knowledge, strive for its development, and not use unauthorized or illegal means to complete this activity, following the Tecnológico de Monterrey Student Code of Honor.