Project 4
COMP301 Fall 2021

Deadline: May 27, 2022 - 23:59 (GMT+3 : Istanbul Time)

In this project, you will work in groups of two. To create your group, use the Google Sheet file in the following link:

Link to Google Sheets for Choosing Group Members

This project contains a boilerplate provided to you use `Project4DataStructures` for the project. Submit a report containing your Racket files for the coding questions to Blackboard as a zip. Include a brief explanation of your approach to problems and your team's workload breakdown in a PDF file. Name your submission files as
p4_member1IDno_member1username_member2IDno_member2username.zip
Example: *p4_0011111_begumsen19_0022222_yakarken18.zip*.

Important Notice: If your submitted code is not working properly, i.e. throws error or fails in all test cases, your submission will be graded as 0 directly. Please comment out parts that cause to throw error and indicate both which parts work and which parts do not work in your report explicitly.

Testing: You are provided some test cases under `tests.scm`. Please, check them to understand how your implementation should work. You can run all tests by running `top.scm`. We will test your program with additional cases but your submission should pass all provided test cases.

Please use *Discussion Forum* on Blackboard for all your questions.

The deadline for this project is May 27, 2022 - 23:59 (GMT+3 : Istanbul Time). **Read your task requirements carefully. Good luck!**

Table 1. Grade Breakdown for Project 4

Question	Grade Possible
Part A	50
Part B	50
Part C	bonus 2 pts (overall course grade)
Report	-
Total	100

Project Definition: In this project, you will implement the most common data structures such as array, queue to EREF. Please, read each part carefully, and pay attention to *Assumptions and Constraints* section.

Part A. In this part, you will add arrays to EREF. Introduce new operators newarray, update-array, read-array ,length-array , and swap-array with the following definitions: (50 pts)

```
newarray:  ExpVal x ExpVal -> ArrVal
update-array:  ArrVal x ExpVal x ExpVal -> Unspecified
read-array:  ArrVal x ExpVal -> ExpVal
length-array:  ArrVal -> ExpVal
swap-array:  ArrVal x ExpVal x ExpVal-> Unspecified
copy-array:  ArrVal -> ArrVal
```

This leads us to define value types of EREF as:

```
ArrVal = (Ref(ExpVal))*
ExpVal = Int + Bool + Proc + ArrVal + Ref(ExpVal)
DenVal = ExpVal
```

Operators of array is defined as follows;

`newarray(length, value)` initializes an array of size `length` with the value `value`.
`update-array(arr, index, value)` updates the value of the array `arr` at index index by value value.
`read-array(arr, index)` returns the element of the array `arr` at index `index`.
`length-array(arr)` returns the length of the array `arr`.
`swap-array(arr, index, index)` swaps the values of the indexes in the array `arr`.
`copy-array(arr)` initializes an new array with the same values of the given array `arr`. (Creates a deep copy of the given array.)

Part B. In this part, you will implement a Queue using arrays that you implemented in Part A.

Queue is a data structure that serves as a collection of elements, where the elements are reached in a FIFO (First In First Out) manner. In other words, whenever dequeue is called, the element that is added earliest is removed and returned from the queue. You will implement the following operators of Queue with the given grammar:

`newqueue()` returns an empty queue.
`enqueue(q, val)` adds the element `val` to the queue `q`.
`dequeue(q)` removes the first element of the queue `q` and returns its value.
`queue-size(q)` returns the number of elements in the `q`.
`peek(q)` returns the value of the first element in the queue `q` without removal.
`empty-queue?(q)` returns true if there is no element inside the queue `q` and false otherwise.
`print-queue(q)` prints the elements in the queue `q`.

Part C (bonus). In this part, you will implement map function for arrays:

Map function allows to assemble an array using values in another array mapped with a given function.

```
Expression ::= map((Identifier -> Expression) in Expression)
                    map-exp (var, body, arr-exp)
```

Example:

```
let x = newarray(4, 2) in
begin
   update-array(x, 0, 0);
   update-array(x, 1, 1);
   update-array(x, 2, 2);
   update-array(x, 3, 3);
   map((y -> -(y,5)) in x)
end
;;; [ -5 -4 -3 -2 ]
```

FIGURE 1. Syntax for Map Expression

Report. Your report should include the following:

(1) Workload distribution of group members.
(2) Parts that work properly, and that do not work properly.
(3) Your approach to implementations: How does your queue work?, How did you implement map? etc.

Include your report as PDF format in your submission folder.

Assumptions and Constraints. Read the following assumptions and constraints carefully. You may not consider the edge cases related to the assumptions.

(1) For queue, you may assume print-queue will only be used for queues of integers.
(2) Queue does not have to be new defined data types, you can utilize the array implementation from Part A.
(3) For queue, values will be integers in the range [1, 10000].
(4) The number of enqueue operations will not exceed 1000 for a single queue.
(5) It is guaranteed that the correct type of parameters will be passed to the operators. For example, in enqueue(q), q always be a queue.
(6) If queue is empty, dequeue operation must return -1.
(7) You CANNOT define global variables to keep track of the size or top element of a queue. The reason is we may create multiple queue and each of them may have different sizes and front elements.
(8) If you consider using list of references for array and queue, find an efficient way to reach to an elements, you are NOT allowed to iterate over list.
(9) Please consider that we will test your code with some additional test cases, which will not be shared publicly. Thus passing all tests does not guarantee full points.