Project 3
COMP301 Fall 2020

Deadline: Dec 19, 2020 - 23:59 (GMT+3 : Istanbul Time)

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

Link to Google Sheets for Choosing Group Members

Note: You need to **self-enroll** to your Project 3 group on BlackBoard (please only enroll to the same group number as your group in the Sheets), please make sure that you are enrolled to Project 3 - Group #YourGroup.

This project contains a boilerplate provided to you use `Project3DataStructures` 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
p3_member1IDno_member1username_member2IDno_member2username.zip
Example: *p3_0011221_mozcelik17_0011222_hsasmaz16.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 *Project 3 Discussion Forum* on Blackboard for all your questions.

The deadline for this project is Dec 19, 2020 - 23:59 (GMT+3 : Istanbul Time). **Read your task requirements carefully. Good luck!**

TABLE 1. Grade Breakdown for Project 3

Question	Grade Possible
Part A	20
Part B	35
Part C	35
Report	10
Total	100

Project Definition: In this project, you will implement the most common data structures such as array, stack and 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, and read-array with the following definitions: (20 pts)

```
newarray:  Int * Int -> ArrVal
update-array:  ArrVal * Int * ExpVal -> Unspecified
read-array:  ArrVal * Int -> ExpVal
```

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`.

Part B. In this part, you will implement Stack with using arrays that you implemented in Part A. (35 pts)

Stack is a data structure that serves as a collection of elements, where the elements are reached in a LIFO (Last In First Out) manner. In other words, when an element is added to the stack, it is added on top of all elements, and when an element is popped from the stack, the topmost element in this data structure will be extracted from the stack. You will implement the following operators of Stack with the given grammar:

`newstack()` returns an empty stack.
`stack-push(stk, val)` adds the element `val` to the stack `stk`.
`stack-pop(stk)` removes the topmost element of the stack `stk` and returns its value.
`stack-size(stk)` returns the number of elements in the stack `stk`.
`stack-top(stk)` returns the value of the topmost element in the stack `stk` without removal.
`empty-stack?(stk)` returns true if there is no element inside the stack `stk` and false otherwise.
`print-stack(stk)` prints the elements in the stack `stk`.

Part C. In this part, you will implement Queue with using arrays that you implemented in Part A. (35 pts)

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. A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first. When an element is popped from the queue, the first element that is pushed in this data structure will be extracted from the queue.

You will implement the following operators of Queue with given definitions:

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

Report. Your report should include the following: (10 pts)

(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 stack/queue work?

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) Stack and Queue do not have to be new defined data types, you can utilize the array implementation from Part A.
(2) For stack and queue, you may assume that values are integers.
(3) For stack and queue, values will be in range [1, 10000].
(4) The number of push operations will not exceed 1000 for a single stack/queue.
(5) It is guaranteed that the correct type of parameters will be passed to the operators. For example, in stack-pop(stk), stk always be a stack.
(6) If stack/queue is empty, pop operation must return -1.
(7) You CANNOT define global variables to keep track of the size or top element of a stack/queue. The reason is we may create multiple stacks/queues and each of them may have different sizes and top elements.

Sample Programs. Here are some sample programs for you.

```
let x = newstack() in begin stack-push(x, 20); stack-push(x, 30);
         stack-push(x, 40); stack-pop(x); print-stack(x) end

;;; 20 30

let x = newstack() in begin stack-push(x, 20); stack-push(x, 30);
         stack-push(x, 40); stack-pop(x); empty-stack?(x) end

;;; (bool-val #f)

let x = newqueue() in begin queue-push(x, 20); queue-push(x, 30);
         queue-push(x, 40); queue-pop(x); print-queue(x) end

;;; 30 40

let x = newqueue() in begin queue-push(x, 20); queue-push(x, 30);
         queue-push(x, 40); queue-pop(x); queue-size(x) end

;;; (num-val 2)
```