Project 4
COMP301 Fall 2021

Deadline: Jan 9, 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

Note: You need to **self-enroll** to your Project 4 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 4 - Group #YourGroup.

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_ alikaratas17_ 0022222_ merciyes18.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 4 Discussion Forum* on Blackboard for all your questions.

The deadline for this project is Jan 9, 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	25
Part C	25
Report	-
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: (50 pts)

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

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`.
`print-array(arr)` prints the elements in the array `arr`.

Part B. In this part, you will implement Stack using arrays that you implemented in Part A. (25 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 array comprehension: (25 pts)

Array Comprehension is a quick way to assemble an array using another array. The syntax of array comprehension for this assignment is similar to list comprehension in python.
`[2*x for x in range(4)]` in python returns `[0, 2, 4, 6]`

```
Expression ::= [Expression for Identifier in Expression]
                    array-comprehension-exp (body, var, arr-exp)
```

FIGURE 1. Syntax for Array Comprehension 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 stack work?, How did you implement array comprehension? 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 array, you may assume print-array will only be used for arrays of integers.
(2) Stack does not have to be new defined data types, you can utilize the array implementation from Part A.
(3) For stack, values will be integers in the range [1, 10000].
(4) The number of push operations will not exceed 1000 for a single stack.
(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 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. The reason is we may create multiple stacks 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 = 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);
   print-array([-(i, 1) for i in x])
end
;;; [ -1 0 1 2 ]
```