Project 3   
Array/Stack/Queue Implementation in EREF

Mehmet Enes Erciyes, 68906  
Adar Bayan, ...  
Mert Gülsün, ...

Contents

[Question 1) 2](#_Toc59318264)

[Question 2) 2](#_Toc59318265)

[Question 3) 2](#_Toc59318266)

[Array: 2](#_Toc59318267)

[Stack: 2](#_Toc59318268)

# Question 1)

While completing this project, our workload distribution was as follows:

* Implementing and array operations => TOGETHER (Enes, Mert, Adar)
* Implementing Stack => Enes
* Implementing Queue => Adar and Mert
* Writing report => TOGETHER (everybody responsible for writing about his implementation)

# Question 2)

Currently, our implementation passes all the tests provided and other tests we have written ourselves.

# Question 3)

### Array:

Arrays are, as suggested by the grammar specification, implemented as Scheme lists of .

* newarray(length, value): new-array-exp is added to the variants of expressions. The main functionality is in datastructures.scm file. It creates a list of that has the value
* update-array(arr, index, value): Uses list-ref to access the references at the and sets it by using
* read-array(arr, index): Uses list-ref and deref to access and read the value from array.

Our references in our arrays point to values to allow all kind of elements to be stored in an array.

### Stack:

For implementing stacks, we used the arrays we built in the A part. Our main idea was creating an array with a size more than the possible insertion number and keeping a counter to keep track of where to put the next elements. To achieve this, we created an array with size 1001 (max insertions: 1000 + 1 counter). We set the first element of array to be our counter and set the rest of the elements to

As you see, we are benefitting from the two assumptions presented in the project description: maximum number of push operations and that the pushed values are integers.

* newstack(): Our new stack operation creates a new array with size 1001 using . Sets the first element to and the rest to
* stack-push(stk, val): Stack-push first extract the stack array, the value, and the counter. Then, it increments the by one and with value at location .
* stack-pop(stk) : Stack-pop extract the stack array. Then, it checks if the array is empty with . If it is, returns . If it is not, it decrements by one and set the value at counter to be and returns the popped value.
* stack-size(stk): Stack size directly returns
* stack-top(stk): Does exactly what stack-pop does except that the counter is not decremented, and value is not updated, just returned.
* empty-stack?(stk): Returns
* print-stack(stk): Prints the stack until it comes across a . Signals the top of the stack.

### Queue:

For implementing queue, we used the arrays we built in the A part. Our main idea was creating an array with a size more than the possible insertion number and keeping a front and rear index to keep track of where to put the next elements. To achieve this, we created an array with size 1002 (max insertions: 1001 + 1 counter). We set the first element of array to be our rear , the second element as front (num-val 2) and set the rest of the elements to

As you see, we are benefitting from the two assumptions presented in the project description: maximum number of push operations and that the pushed values are integers.

* newqueue(): Our new queue operation creates a new array with size 1002 using . Sets the first element to , the second element as rear ( num-val 2) and the rest to
* queue-push(que,val): Queue-push first extract the queue array, the value, front, and the rear. Then, it increments the by one and with value at location .
* queue-pop(que) : Queue-pop extract the queue array. Then, it checks if the array is empty with . If it is, returns . If it is not, it increments by one and set the value at counter to be and returns the popped value.
* queue-size(que): Queue size directly returns
* queue-top(que): Does exactly what queue-pop does except that the counter is not decremented, and value is not updated, just returned.
* empty-queue?(que): Returns
* print-queue(que): Prints the queue until it comes across a . Signals the top of the stack.