COMP 301 Fall'20

Project 3

Altay Atalay	64568
Ege Yelken	61742
Umur Demircioğlu	64609

Task Assignment and Bugs

Tasks	Member Responsible	Progress Status (0-5)	Bugs
Part A(arrays)	Ege Yelken	5	Everything works, no bugs.
Part B(stack)	Altay Atalay	5	Everything works, no bugs.
Part C(queue)	Umur Demircioglu	5	Everything works, no bugs.

**All members are living in the same apartment building, so we were actually together at each stage of the implementation; however, the general distribution of workload is as shown in the table above.*

Implementation

A) Array

1. Redefined the value types of the language such that the Expressed Values include ArrVal.
2. Defined **newarray()** that passes parameters *length* and *value*, initializes the empty array as '() and does the **cons** operation on the reference of the value and the initial array.
3. Defined **update-array()** that passes parameters *arr, index* and *value*, uses **list-ref** for referencing the element at the given *index* of given *arr* and sets its reference to the given *value*.
4. Defined **read-array()** that passes parameters *arr* and *index*, again uses **list-ref** for referencing the element at the given *index* of given *arr*, then dereferences it.
5. Added new expressions to the grammar and corresponding changes in the interpreter.

B) Stack

1. Stack keeps the values in an array using the functions described in part.
2. **newstack()** passes a dummy node for initializing an empty array and first holds the size of the stack, initially 0. (Second element holds the length of the array in case we need to update the size when the stack is full, even though it is not used in the implementation.)
3. **stack-push()** updates the last element as the given *value* of the array.
4. **stack-pop()** returns -1 if the stack is empty, otherwise returns the next element after the size values in the array, and updates the array by removing the returned value.
5. **stack-size()** reads the first element which holds the size of the stack.
6. **stack-top()** reads the next element after the size value.
7. **empty-stack?()** calls the stack-size() function and returns true if its output equals zero.
8. **print-stack()** displays 0 if the stack is empty, otherwise calls read-array() to iterate over elements in the stack starting from the second index. (First two holds size.)
9. Added new expressions to the grammar and corresponding changes in the interpreter.

C) Queue

1. Similarly to the Stack implementation, Queue keeps the values and the current size in an array, only difference is that it also holds the starting position.
2. **newqueue()** passes a dummy node for initializing an empty array and first two elements holds the size of the stack and the array, initially 0, and holds the position at index 2.
3. **queue-push()** updates the first element as the given *value* and increments the size.
4. **queue-pop()** returns -1 if the queue is empty, otherwise returns the starting element, decrements the size and updates the array so that the popped value is removed.
5. **queue-size()** reads the first element which holds the size of the queue.
6. **queue-top()** reads the element at index 2 which holds the starting position.
7. **empty-queue?()** returns true if the queue size is zero, false otherwise.
8. **print-queue()** displays 0 if the queue is empty, otherwise calls read-array() to iterate over elements in the queue starting from the second index.

9. Added new expressions to the grammar and corresponding changes in the interpreter.