# Project 4

# COMP301 Fall 2021

İrem Demir, 69563

Özlem Şerifoğulları, 69361

**Part A:**

For starting to add arrays to our language, the steps below are followed.

*lang.scm:*

1. New grammar for new expressions is defined with respect to rules given in the pdf.
   a. newarray, update-array, read-array, print-array

*data-structures.scm*

2. New expressed value "arr-val" is added as a new value type inside the expval datatype.
3. Extractor for arr-val (expval->arr) is also added among extractors.
4. To be able to add the arr-val (step-2), new datatype arrval is also defined. Since arrval consists of list-of references the definition is made in this way.
5. Helper functions is also defined.
   a. newarray-helper takes two parameters: length and value. It creates a reference with value using newref and constantly cons these values until length is reached.
   b. update-array-helper takes an array index and value. Updates the value in the given index using ref methods to get the address and setting the new value in there.
   c. read-array-helper takes an array and index and returns the element in the index of the given array using deref function.
   d. print-array-helper takes an array and prints it. Firstly it prints "[" and then by using print-array-helper-rec exhausts all elements then finally prints a "]" symbol.

*interp.scm*

6. new value-of cases of expressions are defined. After making necessary transformations for each expression in the new expressions: newarray-exp, update-array-exp, read-array-exp, print-array-exp, helper functions defined above are called to handle

**Part B:**

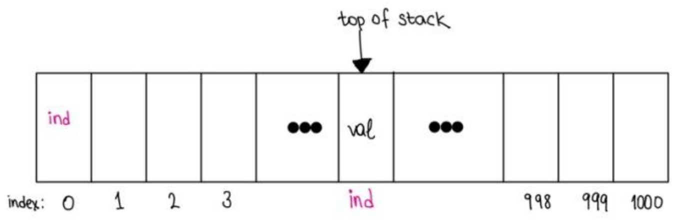
For starting to add stack to our language, the steps below are followed.

*lang.scm:*

1. New grammar for new expressions is defined with respect to rules given in the pdf.
    a. newstack, stack-push, stack-pop, stack-size, stack-top, empty-stack?, print-stack

<u>*interp.scm*</u>

2. For stacks, we decided to implement them in a way that size of stack/ index of top element will be stored in the 0th index. To achieve this, we decided to when creating an stack we will create an array of size 1000+1 which will make us to store 1000 elements which is given as maximum elements and one for size. Moreover, last element added will be added the end of the array and when we need to pop/top/push we will do it from the last index. So last in first out will be accomplished successfully.



3. new-stack-exp: as we decided it creates an array of length max-stack-size and initializes it with empty-stack-value which is conventionally 0. Since it is both out of boundary that stack values can have and when stack is initialized it has no elements->0. It does the given procedure by calling  newarray-helper function.
4. stack-size-exp: takes an expression calculates value-of that and using extractor for array we defined for part-a passes an arrayval into stack-size-helper function. Stack-size-helper function takes the stack and basically uses read-array-helper and reads value in the 0th index which is the place we decided to store size.
5. stack-push-exp: takes two expression one for stack one for value to push. It calculates value-of the expressions in the given environment and call stack-push helper. Stack-push-helper gets the index of the last element using stack-size-helper and updates the next index with given value and increases stack's 0th index/size by 1.
6. stack-pop-exp: takes an expression for stack. It calculates value-of the expressions in the given environment and call stack-pop-helper. Stack-pop-helper gets the index of the last element using stack-size-helper and gets the value at the last index. Returns value at the last index and updates its place by 0 and decreases size counter by 1. Lastly, it checks if the stack-size-helper returns 0 which means an empty stack it returns -1.
7. stack-top-exp: takes an expression for stack. It calculates value-of the expressions in the given environment and call stack-top-helper. Stack-top-helper gets the index of the last element using stack-size-helper and gets the value at the last index. Returns it.

8. empty-stack?-exp: takes an expression for stack. It calculates value-of the expressions in the given environment and call empty-stack?-helper. Empty-stack?-helper checks the size of the stack using stack-size-helper and if it is 0 it returns true, false otherwise.
9. print-stack-exp: After getting the array it immediately calls print-stack-helper function. Print-stack-helper calls print-stack-helper-recursive which has a stack, stack-size and index as parameter. This function prints the stacks in the following format "Stack-size: " + (stack-size that stored in index 0) + "stack:" + (stack elements until reaches the top element)

## Part C:

For starting to add array comprehension to our language, the steps below are followed.

*lang.scm:*

1. New grammar is defined with respect to rules given in the pdf. array-comprehension-exp takes 2 expressions and 1 identifier.

*interp.scm*

2. For implementing array comprehension first, we needed to understand what it does exactly and how can we apply it. The identifier in the expression works as variable. We needed to apply first given expression using that variable which will have values of elements of the given array (third expression). So, we needed to evaluate first expression for given variable in an environment that will have the variable and value pair for each value in array. To obtain that, we decided to define a helper function and recursive version of it to iterate through all elements of the array.
3. Arr-comp-helper takes body, variable, array and environment and calls arr-comp-helper-rec with additional size and index that will help us to iterate until we will go over each element in the array. It extends environment for each variable and value pair and evaluates the body in that current environment and updates the value in the arrary.

## Parts that work properly:

Our implementation passes all test cases. Moreover, we wrote few additional test cases to test some edge cases like if the pop returns the value, or empty stack returns -1 etc. Also we printed example array and stack. To sum up, we believe that our code work properly for all aspects.

## Workload:

Part A:  İrem Demir
Part B: İrem Demir
Part C: Özlem Şerifoğulları