COMP301 Project Report - 4

Berke Can Rizai - 69282
Bumin Aybars İnci - 69032

All the parts are working properly. Each implementation can be found in the code.

Workload Distribution

Worked cooperatively
Part A - Berke Can Rizai - Aybars Inci
Part B - Berke Can Rizai - Aybars Inci
Part C - Berke Can RIzai - Aybars Inci

Part A

The lang file is trivial, as we just looked at the grammar from the PDF and wrote it down.
In the new-array, we implemented a function called array-creation that takes the length and values fo put in it. This is a recursive function, and we cons up value and array-creation function with length - 1 so that we have length many values in the array and if the length is 0 we return ‘() empty.

In new-array, we take the number from the expression that is length and look up the value from the environment and cast array-creation to arr-val and return.

In update-array, we take arr, num and value and call the array-update function we have implemented. In array-update, we just modify the index of the array with value with setref!. Here we also take the first element of the array since it points to the beginning of the array and sum it with the desired index.

In the read-array, we have an array-read function that takes the array and index and since we need to look up the value the pointer is pointing, we deref it and return it.

For the print-array, we take the array and, with begin, run the procedures one by one.
We open a square bracket and add a space to it, then call the array-print.
Here, if the array is emptier than we close it with “]” and otherwise we print the value that is pointed by the start with deref (car array) and add a space with display function and call itself with rest of the array.

Part B

In stack, we use the bottom of the array for the basis of stack. For example, [0] is first filled when something is pushed.

Firstly, find-top function of stack needs to be explained. Here, it takes a stack and index because this is a recursive function. We always call it with 0 initially. If the current index is holding the empty-value that is -1 inside it, then we found the top of the stack, and we return the current index, and we look for it in the rest of the stack otherwise. Here, stack is just an array of -1s and values.

Newstack just creates an array with the defined max length that is 1000 and each entry has value -1. Returns it as arr-val.

Stackpush takes a stack and a value to push. It calls the array-update we have previously defined and gives it the stack, index of top that it gets from find-top and the value.

In stackpop, we first checked if stack is empty and returned -1 if so. And if not, we read the value from stack with array-read from the top, but we extracted 1 from index since it returns the length and store it in retval. This is because of 0 based index. Then, we update that element of stack with empty-value and return the retval.

Stack-size is as explained, uses the find-top. We assign the value of exp1 turned into arr with expval->arr to a variable named stack. Return num-val of what find-top returns starting from 0.

Stacktop is an easier version of the stackpop as we don’t need to modify the stack but just return the value from the top. We deduct 1 from top index and read that from the array with array-read and return it.

In emptystack, we just look if find-top returns 0 and if so, we return bool-val #t and #f otherwise. Find-top was the by far most useful function.

Printstack prints the whole stack, here we first take the arr value of the exp1. After that, we check if length is 0 and if so just print nothing and else check a condition if it is null however, we don’t expect this to happen in any case. It is just try-catch in a sense. Otherwise, if the current entry is -1 and if dereferenced value is -1, we stop with printing “” and if dereferenced is not -1 we print it.
If entry is not -1 meaning that there are more items, we display the num value of deference from start of stack and add a space than call itself with cdr of stack.

PART C

In array comprehension, we used a helper function “comp”. Given array’s elements mapped one by one using the “comp” function recursively. Map operation is done with respect to the given first expression and array-update helper function.