A) We defined the data type of the vector as vec number and reference. We then defined the grammar for all the functions for part one and then started with the implementation in interp.

NewVec: new vec calls the helper which recusivley calls itself until it creates the vector with length len

UpdateVector: update vector calls setref with index + the vector reference and gives it the val.

ReadVector: calls deref with the index + the vector reference.

LengthVector: returns just the numval of the lenght of the vector we created.

Swapvector: calls deref for the with index1 + reference with the vector reference and assigns it to val1. It does the same for index2 and calls it val2. It then setrefs val2 to (index1 + ref) and setrefs val1 to (index2 + ref)

CopyVector: we create a vecref by doing newref to the deref of the reference. We then call the helper copy-vector-helper which takes (len - 1) and (ref + 1) which recursivley creates a newref from the deref of the ref and calls itself until len is <= 0 and lastly in the main method we create a vec-val with the len we have and the vecref we created.

B) **newstack:** we created a new vector with the length and one more slot which was used to store metadata about the **current size**/**next index** of the stack

Push: we used **update-vector-exp** to update two values, the value at the **next index** and we increment the value in the metadata slot to keep track of the size. We check if the stack is full before this operation and raise an error if it's full.

Pop: we used **update-vector-exp** to update two values, the value at the address before the **next index** which we change to 0, and we decrement the value in the metadata slot to keep track of the size. But before that we check if the stack is empty using **stack-empty?-exp** and return -1 if it returns true

Stack-size: we just return the value in the metadata slot

Peek: we use **read-vector-exp** at the address before the **next index** and return its value

Stack-empty?: we check if the value in the metadata slot is 0, return true if it is and false otherwise

Print-stack: we call a helper procedure with a reference to the first address in the vector and the address before the next index, then we keep iterating and printing values until the current index becomes less than 0 then we return

C) For this part we implemented **vec-mult** by getting both first references of both of the vectors then we check if they are of the same size, if they are not we raise an error, but if they are we create a new reference(ref3) that multiplies the first values of the first references in each vec then we call a helper that takes the next references of each vector, the length and a counter initialized with 0, this only creates new references, with values that are the result of the multiplication of the values of the references at the corresponding index in each vector, and it returns a dummy value at the end, then when it returns to **value-of** we return a new **vec-val** with the first reference(ref3) that we created and the length

All of the parts work perfectly

Workload:
We all worked on the project together in a couple of sittings. We solved the questions together on one computer. Ahmed is the one that wrote the code.