Comp 301 Project 5 Report

Team:

Batuhan Arat

Hüseyin Şenol

Metin Arda Oral

Workload Distribution

The workload was strictly and equally distributed. We had three meetings to understand the topics and share ideas. The implementation was mostly carried out together, and any remaining tasks were evenly distributed among team members.

Part A

In Part A, we were tasked with creating a vector datatype in EREF. We defined a vector as a list of references, where each element pointed to a specific value in memory. Utilizing operations such as `newref`, `setref`, and `deref`, we successfully constructed a vector.We have defined a helper method in data-structure file which we used to create vector. This method name is make-vector. This is not scheme built in function, that is a function that we define.

Part B

In Part B, we were instructed to create a queue data structure. As suggested in the project description, we implemented a queue using a vector but with some modifications to accommodate queue operations. Since iteration was not allowed, we had to devise a way to access elements. To accomplish this, we initialized our queue with two additional elements. These two extra elements served as metadata for our queue: the first element pointed to the front index, and the second element always pointed to the rear index. These are not global pointers but are specific to each queue. The front index assisted us in dequeueing, and the rear index assisted in enqueueing. However, we didn't actually remove elements from the queue during dequeueing. Instead, we updated the front index which effectively removed the element from our queue representation. Both the rear and front indices needed to be updated after each operation. The length of the queue could be determined by subtracting the front index from the rear index, and the queue's emptiness could be checked by comparing if the rear index and front index were equal.

Part C

In Part C, we implemented a function to perform element-wise multiplication of two vectors. This function takes two expressions representing the vectors (`vec-exp1` and `vec-exp2`) as inputs. It evaluates each expression in the given environment and extracts the vector of references (`vec1-refs` and `vec2-refs`). If the vectors have different lengths, the function raises an error. It then dereferences all the elements of each vector, converting the references into actual values (`vec1-values` and `vec2-values`). A new vector of references (`result-refs`) is created to hold the results of the multiplication. This vector is initially populated with zeroes. The function then performs the element-wise multiplication and updates the `result-refs` vector with the products. Finally, the function returns the result vector, wrapped in an `expval` data type using the `vec-val` constructor.