Project 3
COMP301 Spring 2023

Ayda Kanıl, akanil17, 64641
Yakup Enes Güven, yguven17, 64045

Most of the project was done by me (Yakup). Since there are other lessons and assignments, this is the max I can do individually from the group assignment. I request your consideration with this in mind.

Part A.
(1)
```
>  (define init-env
    (lambda ()
     (extend-env
      'x (num-val 1)
      (extend-env
       'y (num-val 5)
       (extend-env
        'z (num-val 10)
        (empty-env))))))
```
(2)
```
>  (define init-env
    (lambda ()
     (extend-env
      'x (num-val 1)
      (extend-env
       'y (num-val 5)
       (extend-env
        'z (num-val 10)
        [])))))

>  (define init-env
    (lambda ()
     (extend-env
      'x (num-val 1)
      (extend-env
       'y (num-val 5)
        [z=10]))))

>  (define init-env
    (lambda ()
     (extend-env
      'x (num-val 1)
       [y=5]
        [z=10])))

>  (define init-env
    (lambda ()
      [x=1]
       [y=5]
        [z=10]))
```

Part B.

- The expression queue-exp denotes an empty queue represented as a list.
- The expression queue-push-exp does not have a denoted value, it modifies the queue in place by adding an element to the beginning of the list.
- The expression queue-pop-exp returns the modified queue after removing the element at the front. If the queue is already empty, it returns an empty queue with a warning message.
- The expression queue-peek-exp returns the element at the front of the queue. If the queue is empty, it returns the number 2000 to denote a failed operation and prints a warning message.
- The expression queue-push-multi-exp does not have a denoted value, it modifies the queue in place by adding multiple elements to the end of the list in the order they were provided.
- The expression queue-pop-multi-exp returns the modified queue after removing n elements from the front. If n is greater than the size of the queue, it removes all elements and prints a warning message.
- The expression queue-merge-exp returns a single queue representing the merge operation of two input queues. It does not modify the original queues, but instead, it creates a new queue by popping elements from the second queue and pushing them to the end of the first queue.

Part C.

1. queue-exp :

```
;; Create a new empty queue
(newqueue-exp ()
              (expval->queue (make-queue)))
```

2. queue-push-exp :

```
;; Push an element to the back of the queue
(queue-push-exp (queue elem)
                (let ((q (expval->queue (value-of queue env)))
                      (val (value-of elem env)))
                  (enqueue val q)
                  (void-val)))
```

3. queue-pop-exp :

```
;; Pop an element from the front of the queue
(queue-pop-exp (queue)
               (let ((q (expval->queue (value-of queue env))))
                 (if (queue-empty? q)
                     (begin
                       (display "Warning: Queue is empty\n")
                       (newqueue-exp))
                     (queue-val (dequeue q)))))
```

4. queue-peek-exp :

```
;; Peek at the element at the front of the queue
(queue-peek-exp (queue)
                (let ((q (expval->queue (value-of queue env))))
                  (if (queue-empty? q)
                      (begin
                        (display "Warning: Queue is empty\n")
                        (num-exp 2000))
                      (num-exp (front q)))))
```

5. queue-push-multi-exp :

```
;; Push multiple elements to the back of the queue
(queue-push-multi-exp (queue elems)
                      (let ((q (expval->queue (value-of queue env)))
                            (vals (eval-list elems env)))
                        (for-each (lambda (val) (enqueue val q)) vals)
                        (void-val)))
```

6. queue-pop-multi-exp :

```
;; Pop multiple elements from the front of the queue
(queue-pop-multi-exp (queue n)
                     (let* ((q (expval->queue (value-of queue env)))
                            (count (value-of n env)))
                       (if (> count (queue-size q))
                           (begin
                             (display "Warning: Queue is smaller than n\n")
                             (newqueue-exp))
                           (queue-val (make-queue (list->queue (take (queue->list q) count)))))))
```

7. queue-merge-exp:

```
;; Merge two queues
(queue-merge-exp (queue1 queue2)
                 (let ((q1 (expval->queue (value-of queue1 env)))
                       (q2 (expval->queue (value-of queue2 env))))
                   (queue-val (make-queue (append (queue->list q1) (queue->list q2))))))
```

Helpers:

```
(define (expval->queue expval)
  (if (queue? (expval-val expval))
      (expval-val expval)
      (error "Expected a queue, but got" expval)))

(define (queue-push-exp queue exp)
  (let ((q (expval->queue (value-of queue env)))
        (elem (value-of exp env)))
    (set! q (cons elem q))
    (make-expval 'void '())))

(define (make-queue)
  '())

(define (queue-empty? queue)
  (null? queue))

(define (front queue)
  (if (null? queue)
      (error "Queue is empty")
      (car queue)))

(define (dequeue queue)
  (if (null? queue)
      (error "Queue is empty")
      (begin
        (set! queue (cdr queue))
        queue)))
```

Lang.rkt:

```
(expression
   ("newqueue(" ")")
   newqueue-exp)
(expression
   ("queue-push(" expression ", " expression ")")
   queue-push-exp)
(expression
   ("queue-pop(" expression ")")
   queue-pop-exp)
(expression
   ("queue-peek(" expression ")")
   queue-peek-exp)
(expression
   ("queue-push-multi(" expression ", " expression ")")
   queue-push-multi-exp)
(expression
   ("queue-pop-multi(" expression ", " expression ")")
   queue-pop-multi-exp)
(expression
   ("queue-merge(" expression ", " expression ")")
   queue-merge-exp)
```

Part D.

To implement queues natively in MyProc without delegating to Scheme, MyProc would need to have:

1. The ability to allocate memory dynamically, either with a built-in function or via expressions that allow for memory allocation.
2. The ability to define and manipulate pointers, allowing for the creation and traversal of linked lists that implement the queue.
3. The ability to perform basic arithmetic and comparison operations, allowing for the manipulation and comparison of queue elements.
4. The ability to implement control flow structures such as loops and conditionals, allowing for the implementation of queue operations such as pushing, popping, and peeking.

With these basic features, it would be possible to implement a native queue data structure within MyProc.