```
structure pointer_t       {ptr: pointer to node_t, count: unsigned integer}
structure node_t          {value: data_type, next: pointer_t}
structure queue_t         {Head: pointer_t, Tail: pointer_t}

initialize(Q: pointer to queue_t)
      node = new_node()                                    # Allocate a free node
      node->next.ptr = NULL                                # Make it the only node in the linked list
      Q->Head = Q->Tail = <node, 0>                        # Both Head and Tail point to it
```

Listing 1: Definitions and Initialization

The structures and initialization for the MSQ

```
enqueue(Q: pointer to queue_t, value: data type)
E1:   node = new_node()                                    # Allocate a new node from the free list
E2:   node->value = value                                  # Copy enqueued value into node
E3:   node->next.ptr = NULL                                # Set next pointer of node to NULL
E4:   loop                                                 # Keep trying until Enqueue is done
E5:       tail = Q->Tail                                   # Read Tail.ptr and Tail.count together
E6:       next = tail.ptr->next                            # Read next ptr and count fields together
E7:       if tail == Q->Tail                               # Are tail and next consistent?
E8:           if next.ptr == NULL                          # Was Tail pointing to the last node?
E9:               if CAS(&tail.ptr->next, next, <node, next.count+1>)   # Try link node at the end of the linked list
E10:                  break                                # Enqueue is done. Exit loop
E11:              endif
E12:          else                                         # Tail was not pointing to the last node
E13:              CAS(&Q->Tail, tail, <next.ptr, tail.count+1>)         # Try to swing Tail to the next node
E14:          endif
E15:      endif
E16:  endloop
E17:  CAS(&Q->Tail, tail, <node, tail.count+1>)            # Try swing Tail to inserted Node
```

Listing 2: Enqueue

The function enqueues a new node into the back of the MSQ

```
dequeue(Q: pointer to queue_t, pvalue: pointer to data type): boolean
D1:   loop                                                 # Keep trying until Dequeue is done
D2:       head = Q->Head                                   # Read Head
D3:       tail = Q->Tail                                   # Read Tail
D4:       next = head.ptr->next                            # Read Head.ptr->next
D5:       if head == Q->Head                               # Are head, tail, and next consistent?
D6:           if head.ptr == tail.ptr                      # Is queue empty or Tail falling behind?
D7:               if next.ptr == NULL                      # Is queue empty?
D8:                   return FALSE                         # Queue is empty, couldn't dequeue
D9:               endif
D10:              CAS(&Q->Tail, tail, <next.ptr, tail.count+1>)         # Tail is falling behind. Try to advance it
D11:          else                                         # No need to deal with Tail
                  # Read value before CAS, otherwise another dequeue might free the next node
D12:              *pvalue = next.ptr->value
D13:              if CAS(&Q->Head, head, <next.ptr, head.count+1>)      # Try to swing Head to the next node
D14:                  break                                # Dequeue is done. Exit loop
D15:              endif
D16:          endif
D17:      endif
D18:  endloop
D19:  free(head.ptr)                                       # It is safe now to free the old dummy node
D20:  return TRUE                                          # Queue was not empty, dequeue succeeded
```

Listing 3: Dequeue

The function dequeues a node from the front of the MSQ