```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                             ;
;                                    QUEUE                                    ;
;                                Queue Routines                               ;
;                                   EE/CS 51                                  ;
;                                 Archan Luhar                                ;
;                                 TA: Joe Greef                               ;
;                                                                             ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; QueueInit
;
; Description:      This function is used to create a queue of a given length
;                   and given element size at a given address.
;
; Operation:        This function writes the meta data of the queue in the first
;                   byte and three words of the queue: the size of each element,
;                   the max number of elements, the index of the head (0), and
;                   the count of elements in the queue also initialized to 0.
;                   The start of the queue elements would be the eigth byte.
;
; Arguments:        AX - the length, max number of elements in the queue.
;                   SI - the location at which to initialize the the queue.
;                   BL - size of each element (0: byes, 1: words)
;
; Return Value:     The address of the byte after the end of the queue is in AX.
;
; Local Variables:  SI (increment to write metadata)
;
; Shared Variables: None.
; Global Variables: None.
;
; Input:            None.
; Output:           None.
;
; Error Handling:   None.
;
; Algorithms:       None.
;
; Data Structures:  Cyclic array
;
; Registers Used:   AX (return value)
;
; Stack Depth:      0
;
; Author:           Archan Luhar
; Last Modified:    10/28/2013
;
;
; Pseudo Code
; -----------
;   queue.size = size ? 2 : 1   ; set queue's size - word if nonzero, byte if 0
;   queue.length = length       ; set queue's length
;   queue.head = 0              ; set queue's head index
;   queue.count = 0             ; set queue's count of number of elements
```

```
56  ;
57  ;   queueSize = length * queue.size
58  ;   metadataSize = 7 bytes
59  ;   afterQueuePtr = queuePtr + metadataSize + queueSize
60  ;   return afterQueuePtr
61
62
63
64  ; QueueEmpty
65  ;
66  ; Description:      This function is used to see if a given queue is empty.
67  ;
68  ; Operation:        This function simply looks at the word five bytes into
69  ;                   the metadata which stores the count of elements in queue.
70  ;                   Then it returns true if it is zero, else it returns false.
71  ;
72  ; Arguments:        SI - the address of the queue.
73  ;
74  ; Return Value:     ZF - 1 if empty, else 0.
75  ;
76  ; Local Variables:  None.
77  ;
78  ; Shared Variables: None.
79  ; Global Variables: None.
80  ;
81  ; Input:            None.
82  ; Output:           None.
83  ;
84  ; Error Handling:   None.
85  ;
86  ; Algorithms:       None.
87  ;
88  ; Data Structures:  Cyclic array
89  ;
90  ; Registers Used:   ZF
91  ;
92  ; Stack Depth:      0
93  ;
94  ; Author:           Archan Luhar
95  ; Last Modified:    10/28/2013
96  ;
97  ;
98  ; Pseudo Code
99  ; -----------
100 ;   return count == 0
101
102
103
104 ; QueueFull
105 ;
106 ; Description:      This function is used to see if a given queue is full.
107 ;
108 ; Operation:        This function simply looks at the word five bytes into
109 ;                   the metadata. This word stores the num of elements in queue.
110 ;                   If it equals the word stored at 1 byte into the metadata,
```

```
111   ;                   the length of the queue, then it returns true, else false.
112   ;
113   ; Arguments:        SI - the address of the queue.
114   ;
115   ; Return Value:     ZF - 1 if full, else 0.
116   ;
117   ; Local Variables:  None.
118   ;
119   ; Shared Variables: None.
120   ; Global Variables: None.
121   ;
122   ; Input:            None.
123   ; Output:           None.
124   ;
125   ; Error Handling:   None.
126   ;
127   ; Algorithms:       None.
128   ;
129   ; Data Structures:  Cyclic array
130   ;
131   ; Registers Used:   ZF
132   ;
133   ; Stack Depth:      0
134   ;
135   ; Author:           Archan Luhar
136   ; Last Modified:    10/28/2013
137   ;
138   ;
139   ; Pseudo Code
140   ; -----------
141   ;   return queue.count == queue.length
142
143
144
145   ; Dequeue
146   ;
147   ; Description:      This function returns the value at the head of the queue.
148   ;                   It is a blocking function that waits until there is a value
149   ;                   if initially the queue is empty.
150   ;
151   ; Operation:        This function loops, waiting, until the queue is not empty.
152   ;                   Then, it stores the head in AL if element size is byte.
153   ;                   Else, element size is word so it stores the head in AX.
154   ;                   It then decrements the count.
155   ;                   And also it sets the head to (head + 1) mod (length - 1).
156   ;                   The location to read the value would be
157   ;
158   ; Arguments:        SI - the address of the queue.
159   ;
160   ; Return Value:     AX if element size is word, else AL - the head of queue.
161   ;
162   ; Local Variables:  None.
163   ;
164   ; Shared Variables: None.
165   ; Global Variables: None.
```

```
166   ;
167   ; Input:            None.
168   ; Output:           None.
169   ;
170   ; Error Handling:   None.
171   ;
172   ; Algorithms:       None.
173   ;
174   ; Data Structures:  Cyclic array
175   ;
176   ; Registers Used:   AX if element size is word, else AL.
177   ;
178   ; Stack Depth:      0
179   ;
180   ; Author:           Archan Luhar
181   ; Last Modified:    10/28/2013
182   ;
183   ;
184   ; Pseudo Code
185   ; -----------
186   ;   while (queue.count == 0):    ; queue is empty
187   ;       continue loop
188   ;
189   ;   returnVal = queue.queue[queue.headIndex * queue.size]
190   ;   queue.headIndex = (queue.headIndex + 1) mod (queue.length - 1)
191   ;   queue.count--
192   ;   return returnVal
193
194
195
196   ; Enqueue
197   ;
198   ; Description:      This function pushes to the end of a given queue a given
199   ;                   value.
200   ;                   It is a blocking function that waits until the queue is
201   ;                   not full to enqueue the value.
202   ;
203   ; Operation:        This function loops, waiting, until the queue is not full.
204   ;                   Then it increments the count.
205   ;                   The tail index is just (head index + count) mod (length - 1)
206   ;                   If element size is byte, it stores argument from AL at tail.
207   ;                   Elese element size is word so it stores argument from AX
208   ;                   at tail.
209   ;                   The location to store would be start of queue elements +
210   ;                   tail index * element size.
211   ;
212   ; Arguments:        SI - the address of the queue.
213   ;                   AX if element size is word, else AL - value to enqueue
214   ;
215   ; Return Value:     None.
216   ;
217   ; Local Variables:  None.
218   ;
219   ; Shared Variables: None.
220   ; Global Variables: None.
```

```
221   ;
222   ; Input:            None.
223   ; Output:           None.
224   ;
225   ; Error Handling:   None.
226   ;
227   ; Algorithms:       None.
228   ;
229   ; Data Structures:  Cyclic array
230   ;
231   ; Registers Used:   None.
232   ;
233   ; Stack Depth:      0
234   ;
235   ; Author:           Archan Luhar
236   ; Last Modified:    10/28/2013
237   ;
238   ;
239   ; Pseudo Code
240   ; -----------
241   ;   while (queue.count == queue.length):    ; queue is empty
242   ;       continue loop
243   ;   queue.count++
244   ;   tailIndex = (queue.headIndex + queue.count) mod (queue.length - 1)
245   ;   queue.queue[tailIndex * queue.size] = value
```