# Homework 1: Queues Answers

1. A keyboard buffer on a computer’s operating system is implemented as a circular queue.

(a) Explain why a circular queue is an appropriate data structure choice. [2]

* Size can be preset to suit objectives of minimum memory or maximum capacity
* Space can be reused when items are removed from the queue

(b) A particular keyboard buffer consists of five cells in a circular queue. The queue **kBuffer** is initialised by setting a variable **size** (containing the number of items in the array) to 0, pointers **front** to 0 and **rear** to -1. A variable **maxSize** holds the maximum size of the queue.

(i) Complete the table to show the results after the following operations. [4]

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **kBuffer** | | | | |  |  |  |  |
|  | **[0]** | **[1]** | **[2]** | **[3]** | **[4]** |  | **size** | **front** | **rear** |
| Initial state |  |  |  |  |  |  | 0 | 0 | -1 |
| Enqueue S | S |  |  |  |  |  | 1 | 0 | 0 |
| Enqueue W | S | W |  |  |  |  | 2 | 0 | 1 |
| Dequeue | S | W |  |  |  |  | 1 | 1 | 1 |
| Enqueue E | S | W | E |  |  |  | 2 | 1 | 2 |

* 1 mark for each correct row

(ii) Complete the table to show the results after the following operations. [3]

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **kBuffer** | | | | |  |  |  |  |
|  | **[0]** | **[1]** | **[2]** | **[3]** | **[4]** |  | **size** | **front** | **rear** |
| Current state | J | U | X | L | M |  | 3 | 1 | 3 |
| Enqueue T | J | U | X | L | T |  | 4 | 1 | 4 |
| Enqueue R | R | U | X | L | T |  | 5 | 1 | 0 |
| Dequeue | R | U | X | L | T |  | 4 | 2 | 0 |

* 1 mark for each correct row

(c) Code for the keyboard buffer operations needs to be written.

Use the variables defined in part (b): kBuffer, maxSize, size, front, and rear.

1. Write the pseudocode for the isFull() operation, including function header. [2]

SUB isFull

IF (size = maxSize) THEN

RETURN true

ELSE

RETURN false

ENDIF

ENDSUB

Alternative solution:

SUB isFull

RETURN (size = maxSize)

ENDSUB

This will return True if size = maxSize, False otherwise

* 1 mark = workable solution
* 1 mark = RETURN statement

(ii) Write the pseudocode for the deQueue operation. [4]

SUB deQueue

IF (size = 0) THEN

item = Null

ELSE

item 🡨 kBuffer[front]

front 🡨 (front + 1) MOD maxSize

size 🡨 size - 1

ENDIF

RETURN (item)

ENDSUB

* 1 mark = checking and handling empty queue
* 1 mark = locating correct return item using front pointer
* 1 mark = adjusting front pointer AFTER removing item with correct use of MOD or alternative IF statement to set index to 0
* 1 mark = adjusting queue size

(c) (i) Describe, with the aid of an example, the operation of a priority of a priority queue from the user’s point of view. [2]

Priority:

* Gives preference to more important items
* Items are inserted in order of priority, joining the queue behind items of the same priority.
* Items are always removed from the front of the queue
* Example: Hospital accident and emergency, or other suitable example

(ii) Explain how the principles of data abstraction and encapsulation can be used to hide the details of implementation of a priority queue. [3]

Priority queue may be implemented as a circular queue using a static array, or as a dynamic data structure such as a list. Programmer using the ADT just has to know how to call routines to add an item with a given priority, remove an item from the queue, test for full and empty queue. Details of how these operations are done are hidden.

Total 20 marks