# Homework 2: Queues

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.

With Circular queues you won’t run out of positions to add new items to the rear of the queue so the queue can continue for as long as you want it to. It is an efficient method for queues as you don’t have to waste a lot of processing for moving each item up a position, by simply assigning the rear pointer to the first empty space and so on, recycling the positions that were held by previous front items.

[2]

(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 |  | W |  |  |  |  | 1 | 1 | 1 |
| Enqueue E |  | W | E |  |  |  | 2 | 1 | 2 |

(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 |  |  |  |  |  |  |  |  |  |
| Enqueue R |  |  |  |  |  |  |  |  |  |
| Dequeue |  |  |  |  |  |  |  |  |  |

(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.

function isFull()

if size == maxSize then #checks if the size has reached the limit yet

print(“full”)

return True #returns true in the case that the queue is full

else

return False #returns false in the case the queue is not yet full

endif

endfunction

[2]

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

procedure deQueue()

if isEmpty() then

print(“nothing to dequeue”)

else

kBuffer[front] = None

size -= 1

front = (front + 1) MOD maxSize

endif

endprocedure

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

Elements in a priority queue are added based on their priority level. For example, in an emergency room, you obviously should prioritise someone who is in danger of dying over someone who has a papercut. In which case when the person in danger comes in, even though the person with a papercut was in the queue before them, the person at risk gets a higher priority so they are moved in front of the person with a papercut making them closer to the front.

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

Abstraction can be used to not show people how the algorithm for the queue works so people can’t think prove it is biased as there is not enough information to say that the queue is a priority based queue. For example, with the previous example, if someone with a papercut sees that they have been moved back in the queue, they will know that it is priority based and it will seem unfair to them. By using abstraction, you can simply use the queue to check what place to add the new patient who is at a higher risk, without removing the papercut patient so they don’t know they have been moved down. Encapsulation can be used to bundle the data so that it is hard to find the specific scripts that are used to implement the queue.

Total 20 marks