# Worksheet 2 Queues

### Task 1 – Desserts: Simple Array Queue

In a restaurant, the dessert chef must make the desserts in the order that they are requested by the waiting staff.

Complete the table to show the missing items based on the information in each column.

Note:

* Array **q** starts at index 0
* **front** points to first item in the queue, initialised to 0
* **rear** points to last item in queue, initialised to -1
* Desserts = **Y**akgwa, **M**ochi, **T**rifle, **G**elato, **S**achertorte, **B**aklava

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Request** | **Operation** | **Contents of array q** | **size** | **rear** | **front** | **Returns** |
| is the queue empty? | isEmpty() | [‘ ‘,’ ‘,’ ‘,’ ‘,’ ‘] | 0 | -1 | 0 |  |
| add Y, add M | enQueue(Y), enQueue(M) | [Y,M,’ ‘,’ ‘,’ ‘] | 2 | 1 | 0 | Nothing |
| Add T | enQueue(T) | [Y, M, T, ,] | 3 | 2 | 0 | Nothing |
| remove one item | deQueue(Y) | [, M, T, ,] | 2 | 2 | 1 | Nothing |
| is the queue full? | isFull() | [, M, T, ,] | 2 | 2 | 1 | False |
| add G, add S | enQueue(G)  enQueue(S) | [, M, T, G, S] | 4 | 4 | 1 | Nothing |
| how many elements in the queue? | print(size) | [, M, T, G, S] | 4 | 4 | 1 | 4 |
| remove one item | deQueue(M) | [, , T, G, S] | 3 | 4 | 2 | Nothing |
| Is the queue full? | isFull() | [, , T, G, S] | 3 | 4 | 2 | True |

How do you tell if the queue is full?

If all the positions in the queue are taken up.

Are there any problems with this implementation of the **queue** ADT as a static array?

Yes, when you dequeue an item, the position the element that was dequeued will no longer be available as the position after it it becomes the front.

### Task 2 – Print server: Circular Queue

A print server keeps all submitted jobs in a circular queue.

Complete the table for a server to show the missing print jobs based on the information in the first column.

Note:

* The array is indexed from 0..4
* **Front** points to the next item to remove from the queue, initialised to 0
* **Rear** points to last item in queue, initialised to -1
* Print job ID = J<number>; for example J38, J21

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | **Queue** | | | | |
|  | **front** | **rear** | **size** | **[0]** | **[1]** | **[2]** | **[3]** | **[4]** |
| Initialise | 0 | -1 | 0 |  |  |  |  |  |
| Add J45 | 0 | 0 | 1 | J45 |  |  |  |  |
| Add J38 | 0 | 1 | 2 | J45 | J38 |  |  |  |
| Add J92 | 0 | 2 | 3 | J45 | J38 | J92 |  |  |
| Remove 1 | 1 | 2 | 2 |  | J38 | J92 |  |  |
| Remove 1 | 2 | 2 | 1 |  |  | J92 |  |  |
| Add J44 | 2 | 3 | 2 |  |  | J92 | J44 |  |
| Add J55 | 2 | 4 | 3 |  |  | J92 | J44 | J55 |
| Add J66 | 2 | 0 | 4 | J66 |  | J92 | J44 | J55 |
| Add J77 | 2 | 1 | 5 | J66 | J77 | J92 | J44 | J55 |
| Remove 1 | 3 | 1 | 4 | J66 | J77 |  | J44 | J55 |
| Add J04 | 3 | 2 | 5 | J66 | J77 | J04 | J44 | J55 |
| Remove 1 | 4 | 2 | 4 | J66 | J77 | J04 |  | J55 |
| Remove 1 | 0 | 2 | 3 | J66 | J77 | J04 |  |  |

How many are there in the queue at the end?

3

Where is the front of the queue?

0

### Task – Accident and Emergency: Priority Queue

An accident and emergency room triage system rates each new patient according to a 3 point system. One is the highest priority and 3 is the lowest priority. A priority queue is maintained of patients in the order that they are to be seen. (You can assume that the queue will always be long enough to accommodate all patients, and that the first to be seen will always be at q[0].)

Complete the table for an accident and emergency department to show the missing items based on the information in the first column.

Note:

* Queue implemented as dynamic data structure, starting at q[0]
* Pointers to front and rear of queue are not needed
* Same priority items are added at the end of all equivalents
* Patient ID = <priority><last initial><first initial>; for example 3DA, 2HG, 1NB

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Queue q** | | | | |
|  | **[0]** | **[1]** | **[2]** | **[3]** | **[4]** |
| Add 2HG | 2HG |  |  |  |  |
| Add 3DA | 2HG | 3DA |  |  |  |
| Add 1NB | 1NB | 2HG | 3DA |  |  |
| Add 2NF | 1NB | 2HG | 2NF | 3DA |  |
| Remove 1 | 2HG | 2NF | 3DA |  |  |
| Remove 1 | 2NF | 3DA |  |  |  |
| Add 3FC | 2NF | 3DA | 3FC |  |  |
| Add 2AB | 2NF | 2AB | 3DA | 3FC |  |
| Add 1WT | 1WT | 2NF | 2AB | 3DA | 3FC |
| Remove 1 | 2NF | 2AB | 3DA | 3FC |  |
| Add 2CS | 2NF | 2AB | 2CS | 3DA | 3FC |
| Remove 1 | 2AB | 2CS | 3DA | 3FC |  |
| Add 3DS | 2AB | 2CS | 3DA | 3FC | 3DS |

This queue is implemented using a **dynamic data structure** such as a **list** in Python.

The queue will grow and shrink according to the number of items it contains.