|  |  |  |  |
| --- | --- | --- | --- |
| رقم المقعد | الفصل | رقم الجلوس | اسم الطالب |
| **30** | **3** | **33160** | **كيرلس سامي شفيق شنوده** |

Ordered Delivery Packet

# Overview

The code consists of three main functions and operations in addition to one constructor for a class the main three functions as follows: -

Responsible for adding any packets to the Queue created, as it inserts the data in the index similar to the value of the data Ex: -   
Array [7] = 7  
Array [8] = 8

Responsible for re-arrange the remaining data or number inside the queue and print it, the arrangement technique is set to be circular as for example 0 is placed after 7

Responsible for searching for certain number as it receives this number and search for it inside the queue and empty its place if it was found and increment the number received by one and return it back

## Enqueue function

As stated, enqueue function insert data into the queue with respect to its index equal to the data itself, first of all the function make sure that the desired destination in the queue is empty, if not the enqueue return 1 indicating the current data or packet ID will be dropped

Array [0] Array [1] Array [2] Array [3] Array [0] Array [1] Array [2] Array [3]

|  |  |  |  |
| --- | --- | --- | --- |
| -1 | -1 | -1 | -1 |

|  |  |  |  |
| --- | --- | --- | --- |
| -1 | -1 | 2 | -1 |

Insert 2 at Array [2] Return 0 as Array [2] was empty (= -1)

Insert 2 at Array [2] and Return 1 as Array [2] was NOT empty and 2 must be dropped

## BufferSearch function

BufferSearch function search for the number it is called by if it was found it the number will be printed on the screen and the node that contain this number will be emptied, the number that was passed to the function will be incremented by one.

Array [0] Array [1] Array [2] Array [3]

|  |  |  |  |
| --- | --- | --- | --- |
| -1 | 1 | 2 | 3 |

Search for “3”, if found print 3 and empty its node

Array [0] Array [1] Array [2] Array [3]

|  |  |  |  |
| --- | --- | --- | --- |
| -1 | 1 | 2 | -1 |

Increment 3 by one with respect to wrapping up formula  
represented in the code snippets and loop again for all nodes  
in the array to search for the new value or number.

### Code snippets for BufferSearch

**int** **buffersearch** (**int** start)

{

**for** (**int** i = 0; i < max; i++)

{

**if**(nodes[i] == start)

{

cout<<nodes[i]<<" ";

nodes[i] = -1;

start = (start + 1) % (max);

}

}

**if** (start == 0) // wrapping up function to make sure the buffersearch function will wrap up

{

**for** (**int** i = 0; i < max; i++)

{

**if**(nodes[i] == start)

{

cout<<nodes[i]<<" ";

nodes[i] = -1;

start = (start + 1) % (max);

}

}

}

**return** start;

};

## PrintArray Function

This function responsible for re-arrange the queue according to circular sequence and print all the data, the function start by increasing any value with the max seqNumber if this value exceeds the last expected number plus the window size, then it swap every two nodes inside the queue if the previous node is greater than the next node

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | -1 | -1 | -1 | -1 | 7 |

Array [0] Array [1] Array [2] Array [3] Array [4] Array [5] Array [6] Array [7]

If (Node is not empty AND Array [index number] + max SeqNumber <= last expected number + window size) Array[indexnumber] = Array [indexnumber] + maxSeqNumber The array will looks like the follows

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 9 | 10 | -1 | -1 | -1 | -1 | 7 |

Array [0] Array [1] Array [2] Array [3] Array [4] Array [5] Array [6] Array [7]

Using swap techniques between every node and its next ones if and only if the previous node is greater than the next one

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| -1 | -1 | -1 | -1 | 7 | 8 | 9 | 10 |

Array [0] Array [1] Array [2] Array [3] Array [4] Array [5] Array [6] Array [7]

### Code Snippets

**for** (**int** i = 0; i < max; i++){

**if** (nodes[i] != -1)

cout<<nodes[i] % (max + 1)<< " ";

}

}

};

**void** **printarray**(**int** start, **int** window)

{

**for** (**int** i = 0 ; i < max ; i++){

**if** (nodes[i] != -1 && nodes[i] + max <= start + window)

nodes[i] += max + 1;

}

**for** (**int** i = 0; i < max; i++) {

**for** (**int** j = 1; j < (max - i); j++)

{

**if** (nodes[j - 1] > nodes[j])

{

**int** temp = nodes[j - 1];

nodes[j - 1] = nodes[j];

nodes[j] = temp;

}

}

}

# Main Flows



