**Unit - III**

**Stacks and Queues**

**Learning Material**

**QUEUES**

**Queue** is a linear Data structure.

**Definition:** Queue is a collection of homogeneous data elements, where insertion and deletion operations are performed at *two extreme ends*.

* The insertion operation in Queue is termed as *ENQUEUE*.
* The deletion operation in Queue is termed as *DEQUEUE*.
* An element present in queue is termed as *ITEM*.
* The number of elements that a queue can accommodate is termed as *LENGTH* of the Queue.
* In the Queue the *ENQUEUE* (insertion) operation is performed at ***REAR*** end and *DEQUEUE* (deletion) operation is performed at ***FRONT*** end.
* Queue follows ***FIFO*** principle. i.e. First In First Out principle. i.e. a item First inserted into Queue, that item only First deleted from Queue, so queue follows FIFO principle.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| REAR  DEQUEUE  ENQUEUE  FRONT | . | . | - | - | - | . | . | . |  |

**Schematic Representation of Queue**

**Representation of Queue**

A Queue can be represented in two ways

1. Using arrays

2. Using Linked List

**1. Representation of Queue using arrays**

A one dimensional array Q[1-N] can be used to represent a queue.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** |  |  |  |  |  |  |  |  |  |  | **N-2** | **N-1** | **N** |
|  |  |  |  | **.** | **.** | **-** | **-** | **-** | **.** | **.** |  |  |  |  |  |

FRONT

REAR

**Array representation of Queue**

* In array representation of Queue, two pointers are used to indicate two ends of Queue.

The above representation states as follows:

1. Queue Empty condition

Front = = -1 and Rear = = -1

2. Queue Full condition

Rear = = N-1 where N is the size of the array we are taken

3. Queue contains only one element

Front = = Rear

4. Number of items in Queue is

Rear – Front + 1

*Queue overflow:* Trying to perform ENQUEUE (insertion) operation in full Queue is known as Queue overflow.

Queue overflow condition is Rear > = N-1

*Queue Underflow:* Trying to perform DEQUEUE (deletion) operation on empty Queue is known as Queue Underflow.

Queue Underflow condition is Front = = -1

**Operation on Queue**

1. ENQUEUE : To insert element in to Queue

2. DEQUEUE : To delete element from Queue

3. Status : To know present status of the Queue

**Algorithm Enqueue(item)**

**Input:** *item* is new item insert in to queue at rear end.

**Output:** Insertion of new item queue at rear end if queue is not full.

1. if(rear = = N-1)

a) print "queue is full, not possible for enqueue operation"

2. else

a) if(front = = -1 && rear = = -1) /\* Queue is Empty \*/

i) rear = rear+1

ii) Q[rear] = item

iii) front = 0

b) else

i) rear = rear+1

ii) Q[rear] = item

c) end if

3. end if

**End Enqueue**

While performing ENQUEUE operation two situations are occur.

1. if queue is empty, then newly inserting element becomes first element and last element in the queue. So Front and Rear points to first element in the list.

2. If Queue is not empty, then newly inserting element is inserted at Rear end.

**Algorithm Dequeue( )**

**Input:** Queue with some elements.

**Output:** Element is deleted from queue at front end if queue is not empty.

1. if(front = = -1 && rear = = -1)

a) print "Q is empty, not possible for dequeue operation"

2. else

a) if(front = = rear) /\* Q has only one element \*/

i) item = Q[front]

ii) front = -1

iii) rear = -1

b) else

i) item = Q[front]

ii) front = front+1

c) end if

d) print "deleted item is" item

3. end if

**End Dequeue**

While performing DEQUEUE operation two situations are occur.

1. If queue has only one element, then after deletion of that element Queue becomes empty. So Front and Rear becomes -1.

2. If Queue has more than one element, then first element is deleted at Front end.

**Algorithm Queue\_Status( )**

**Input:** Queue with some elements.

**Output:** Status of the queue. i.e. Q is empty or not, Q is full or not, Element at front end and rear end.

1. if(front = = -1 && rear = = -1)

a) print "Queue is empty"

2. else if (rear = = size-1 )

a) print "Queue is full"

3. else

a) if(front = = rear)

i) print "Queue has only one item"

b) else

i) print "element at front end is" Q[front]

ii) print "element at rear end is" Q[rear]

c) end if

4. end if

**End Queue\_Status**

**2. Representation of Queue using Linked List**

* Array representation of Queue has static memory allocation only.
* To overcome the static memory allocation problem, Queue can be represented using Linked List.

**X**

**X**

header

N1

N2

N4

N3

**Linked List Representation of Queue**

Front

Rear

* In Linked List Representation of Queue, **Front** always points to **First** node in the Linked List and **Rear** always points to **Last** node in the Linked List.

The Linked List representation of Queue stated as follows.

1. Empty Queue condition is

Front = = NULL and Rear = = NULL or header🡪link = = NULL

2. Queue full condition is not available in Linked List representation of Queue, because in Linked List representation memory is allocated dynamically.

3. Queue has only one element

Front = = Rear

**Operation on Linked List Representation of Queue**

1. ENQUEUE : To insert element in to Queue

2. DEQUEUE : To delete element from Queue

3. Status : To know present status of the Queue

**Algorithm Enqueue \_LL(item)**

**Input:** item is new item to be insert.

**Output**: new item i.e new node is inserted at rear end.

1. new = getnewnode()

2. if(new = = NULL)

a) print "required node is not available in memory"

3. else

a) if(front = = NULL && rear = = NULL) /\* Q is EMPTY \*/

i) header🡪link = new

ii) new🡪link = NULL

iii) front = new

iv) rear = new

v) new🡪data = item

b) else /\* Q is not EMPTY \*/

i) rear🡪link = new /\* 1 \*/

ii) new🡪link = NULL /\* 2 \*/

iii) rear = new /\* 3 \*/

iv) new🡪data = item

c) end if

4. end if

**End\_Enqueue\_LL**

While performing ENQUEUE operation two situations are occur.

1. if queue is empty, then newly inserting element becomes first node and last node in the queue. So Front and Rear points to first node in the list.

2. If Queue is not empty, then newly inserting node is inserted at last.

**X**

**X**

header

N1

N2

N3

new

**Before ENQUEUE**

**X**

**X**

header

N1

N2

N3

new

**After ENQUEUE**

Rear

Rear

NULL

Front

Front

Rear

1. Previous last node link part is replaced with address of new node.
2. Link part of new node is replaced with NULL, because new nodes becomes the last node.
3. Rear is points to last node in the list. i.e. newly inserted node in the list.

**Algorithm Dequeue\_LL( )**

**Input:** Queue with some elements

**Output:** Element is deleted at front end, if queue is not empty.

1. if(front = = NULL && rear = = NULL)

a) print "queue is empty, not possible to perform dequeue operation"

2. else

a) if(front = = rear) /\* Q has only one element \*/

i) header🡪link = NULL

ii) item = front🡪data

iii) front = NULL

iv) rear = NULL

b) else /\* Q has more than one element \*/

i) header🡪link = front🡪link /\* 1 \*/

ii) item = front🡪data

iii) free(front)

iv) front = header🡪link /\* 2 \*/

c) end if

d) print "deleted element is item"

3. end if

**End\_Dequeue\_LL**

While performing DEQUEUE operation two situations are occur.

1. If queue has only one element, then after deletion of that element Queue becomes empty. So Front and Rear points to NULL.

2. If Queue has more than one element, then first node is deleted at Front end.

**X**

header

N2

N3

N4

**X**

**X**

header

N1

N2

N3

**Before DEQUEUE**

**X**

N4

Front

Rear

Front

Front

Rear

**After DEQUEUE**

1. Link part of the header node is replaced with address of second node. i.e. address of second node is available in link part of first node.

2. Front is set to first node in the list.

**Algorithm Queue\_Status\_LL**

**Input:** Queue with some elements

**Output:** Status of the queue. i.e. Q is empty or not, Q is full or not, Element at front end and rear end.

1. if(front = = NULL && rear = = NULL)

a) print "Q is empty"

2. else if(front = = rear)

a) print "Q has only one item"

3. else

a) print "element at front end is" front🡪data

b) print "element at rear end is" rear🡪data

4. end if

**End Queue\_Status\_LL**

**Various Queue Structures**

1. Circular Queues
2. DEQue
3. Priority Queue
4. **Circular Queues**

* Physically a circular array is same as ordinary array, say a[1-N], but logically it implements that a[1] comes after a[N] or a[N] comes after a[1].
* The following figure shows the physical and logical representation for circular array.

Rear

Front

j

i

n

1

2

n-1

**Circular Queue (Logical)**

1

j

i

n

Front

Rear

**Circular Array (Physical)**

**Logical and physical view of a Circular Queue**

* Here both *Front* and *Rear* pointers are move in clockwise direction. This is controlled by the ***MOD*** operation.
* For e.g. if the current pointer is at i, then shift next location will be

(i mod LENTH) +1, 1<= i <= Length

Circular Queue empty condition is

Front = = 0 && Rear = = 0

Circular Queue is full

Front = = (Rear % Length) + 1

**Algorithm CQ\_Enqueue(item)**

**Input:** item is new element insert in to Circular queue at rear end.

**Output:** Insertion of new item in Circular queue at rear end if queue is not full.

1. next = (REAR % N) + 1

2. if( FRONT == next)

a) print(“Circular queue is full, enqueue operation is not possible”)

3. else

i) if(FRONT==0 and REAR==0) /\* CQ is Empty \*/

a) REAR=(REAR % N) +1

b) CQ[REAR]=item

c) FRONT=1

ii) else

a) REAR=(REAR% N) + 1

b) CQ[REAR]=item

iii) end if

4. endif

**End CQ\_Enqueue**

**Algorithm CQ\_Dequeue( )**

**Input:** Circular Queue with some elements.

**Output:** Element is deleted from circular queue at front end if circular queue is not empty.

1. if(FRONT= =0 and REAR= =0)

a) print(“Circular Queue is empty, dequeue operation is not possible”)

2. else

i) if(FRONT = =REAR) /\* Q has only one element \*/

a) item=CQ[FRONT]

b) FRONT=0

c) REAR=0

ii) else

a) item=CQ[FRONT]

b) FRONT=(FRONT % N)+1

iii) End if

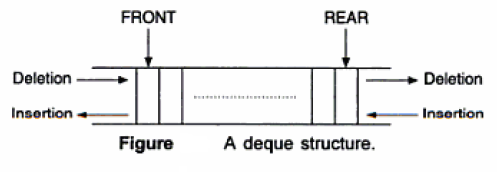
iv) print(“deleted item is ‘item’ “)

3. end if

**End CQ\_Dequeue**

**2. Deque:**

* It is one of the Queue variant. It is also called as ‘deck’.
* In this both insertion and deletion operations are performed at both the ends of structure.
* The term deque is originated from double ended queue.



* Deque structure is general representation of both queue and stack. That means the deque can be used as a queue as well as a stack.
* We can implement deque by using double linked list or by circular array (as in circular queue).
* Operations on deque:

1. Push\_DQ(ITEM): To insert ITEM at the FRONT end of a deque.
2. Pop\_DQ(): To remove the FRONT ITEM of a deque.
3. Inject(ITEM): To insert ITEM at the REAR end of a deque.
4. Eject(): To remove the REAR ITEM of a deque.

* Operations of deque based on circular array of size LENGTH. Let the array be DQ[1……LENGTH].

**Algorithm for Push\_DQ(ITEM):**

**Input:** ITEM to be inserted at the FRONT

**Output:** deque with newly inserted element ITEM if it is empty.

**Data structures:** circular array representation of deque.

If(FRONT ==1) then //If FRONT is at extreme left

Ahead=LENGTH;

Else //If deque is empty

If((REAR==0) or (FRONT==0)) then

Ahead=1;

REAR=1;

Else

If(FRONT==LENGTH) //If FRONT is at extreme right

Ahead=1;

Else

Ahead=FRONT-1 //If FRONT is at an intermediate position

If(Ahead==REAR) then

Print “Deque is full”;

Exit;

Else

FRONT=Ahead;

DQ[FRONT]=ITEM; //PUSH the ITEM

Stop;

**Algorithm for Pop\_DQ():**

**Input:** A DQ with elements. Two pointers FRONT and REAR are known.

**Output:** The deleted element is ITEM if the DQ is not empty.

**Data structures:** circular array representation of deque.

/\* this algorithm is same as the algorithm DECQUEUE \*/

If(FRONT ==0) then

Print “deque is empty”

Exit;

Else

ITEM=DQ[FRONT]

If(FRONT==REAR) then //If DQ consists single element

FRONT=0;

REAR=0;

Else

FRONT=(FRONT mod LENGTH)+1;

End if;

End if;

Stop;

**Algorithm for Inject(ITEM):**

**Input:** ITEM to be inserted at the FRONT

**Output:** deque with newly inserted element ITEM if it is empty.

**Data structures:** circular array representation of deque.

/\* this algorithm is same as the algorithm ENCQUEUE \*/

If (FRONT==0) then //when DQ is empty

FRONT=1;

REAR=1;

DQ[REAR]=ITEM;

Else // DQ is not empty

Next=(REAR mod LENGTH)+1;

If(Next != FRONT) then //if DQ is not full

REAR=Next;

DQ[REAR]=ITEM;

Else

Print “DQ is FULL”

End if;

End if;

Stop;

**Algorithm for Inject(ITEM):**

**Input:** A DQ with elements. Two pointers FRONT and REAR are known.

**Output:** The deleted element is ITEM if the DQ is not empty.

**Data structures:** circular array representation of deque.

If(FRONT ==0) then

Print “deque is empty”

Exit;

Else

If(FRONT==REAR) then //If DQ consists single element

ITEM=DQ[REAR]

FRONT=0;

REAR=0; //DQ becomes empty

Else

If (REAR == 1) then //REAR is at extreme left

ITEM=DQ[REAR];

REAR=LENGTH;

Else

If (REAR == LENGTH) then //REAR is at extreme right

ITEM=DQ[REAR];

REAR=1;

**Types of Deque**

* There are two variations of Deque known as

1. Input restricted Deque
2. Output restricted Deque
3. **Input restricted Deque**

* Here Deque allows insertion at one end (say REAR end) only, but allows deletion at both ends.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Deletion  Insertion  Deletion  FRONT  REAR |  |  |  |  |  |  |  |  |

Input restricted Deque

1. **Output restricted Deque**

* Here Deque allows deletion at one end (say FRONT end) only, but allows insertion at both ends.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Deletion  Insertion  Insertion  FRONT  REAR |  |  |  |  |  |  |  |  |

Output restricted DEQue

**Assignment-Cum-Tutorial Questions**

**A. Questions testing the remembering / understanding level of students**

***I) Objective Questions***

1. 1) ......... form of access is used to add and remove nodes from a queue.  
   a) LIFO, Last In First Out b) FIFO, First In First Out  
   c) Both a and b d) None of these
2. A linear list of elements in which deletion can be done from one end (front) and insertion can take place only at the other end (rear) is known as a\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
3. Queue follows \_\_\_\_\_\_\_\_\_\_\_ principle [ ]

a) FIFO b) LIFO c) Both A &B d) None of the above

1. Insertion and Deletion operation in Queue is known as ? [ ]

a) Push and Pop b) Enqueue and Dequeue c) Insert and Delete d) None

1. If the elements “A”, “B”, “C” and “D” are placed in a queue and are deleted one at a time, in what order will they be removed?

a) ABCD b) DCBA c) DCAB d) None

1. In linked list implementation of a queue, where does a new element be inserted? [ ]

a) At the head of linked list b) At the tail of the linked list

c) At the centre position in the linked list d) None of the above

1. A normal queue, if implemented using an array of size MAX\_SIZE, gets full when[ ]

a) rear = MAX\_SIZE b) front = (rear+1)mod MAX\_SIZE

c) front = rear+1 d) rear = front

1. In a linked queue, only a maximum of 100 elements can be added.[True/False] [ ]
2. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_are appropriate data structures to process a list of employees having a contract for a seniority (based on date of joining) system for promotions.
3. What is the minimum number of stacks of size n required to implement a queue of size n?  
   a) One b) Two c) Three d) Four [ ]

***II) Descriptive Questions***

* + - 1. Write insertion operation on Queue using array.
      2. Write Deletion operation on Queue using array.
      3. Write insertion operation on Queue using SLL.
      4. Write Deletion operation on Queue using SLL.
      5. What is circular queue? Explain the operations of circular Queues.
      6. Explain how a queue can be implemented using Stacks with an example.

**B. Question testing the ability of students in applying the concepts.**

***I) Multiple Choice Questions:***

1) A \_\_\_\_\_\_\_\_\_\_\_\_\_\_ is a data structure that organizes data similar to a line in the super market, where the first one in the line is the first one out. [ ]

(A) Stack (B) Queue (C) Both A & B (D) None

2) The initial configuration of a queue is a, b, c, d (a is at the **front** end). To get the configuration d, c, b, a one needs a minimum of \_\_\_\_\_\_\_\_\_\_\_ [ ]

(A) 2 deletions and 3 insertions (B) 4 deletions and 3 insertions

(C) 3 deletions and 3 insertions (D) 3 deletions and 4 insertions

3) \_\_\_\_\_\_\_\_\_\_\_\_\_\_ is one of the application of queue. [ ]

(A) Infix to Postfix Conversion (B) Evaluation of Postfix expression

(C) Recursion (D) Process Scheduling

4) Consider a linear queue of size 5, Assume the following operations are done on the queue: Enqueue(10), Enqueue(20), Enqueue(30), Enqueue(40), Enqueue(50), Enqueue(60).

The elements of the queue are. [ ]

(A) 10, 20, 30, 40, 50 (B) 10, 20, 30, 40, 50, 60

(C) 10, 20, 30, 40, 60 (D) 60, 50, 40, 30, 20, 10

5) Consider a linear queue of size 5, Assume the following operations are done on the queue: Enqueue(100), Enqueue(200), Enqueue(300), Enqueue(400), Enqueue(500), Dequeue(), Dequeue( ), Dequeue( ), Enqueue(600). The elements of the queue are: [ ]

(A) 100, 200, 300, 400, 50 (B) 100, 200, 300, 400, 500, 600

(C) 400, 500 (D) 400, 500, 60

6) A circular queue is implemented using an array of size 10. The array index starts with 0, front is 6, and rear is 9. The insertion of next element takes place at the array index. [ ]

(A) 0 (B) 7 (C) 9 (D) 10

7) \_\_\_\_\_\_\_\_\_\_ is used to overcome the problem of linear queue. [ ]

(A) Linked List (B) Circular Queue (C) Both A & B (D) None

8) Let the following circular queue can accommodate maximum 5 elements with the following data

front = 2 rear = 4 (assume queue ranges from 0 to 4)

queue = \_\_\_; L, M, N, \_\_\_, \_\_\_

What will happen after Enqueue(O) operation takes place? [ ]

(A) front = 2 rear = 5 (B) front = 3 rear = 5

(C) front = 2 rear = 0 (D) front = 2 rear = 4

9) Let the following circular queue can accommodate maximum 5 elements with the following data

front = 2 rear = 4 (assume queue ranges from 0 to 4)

queue = \_\_\_; L, M, N, \_\_\_, \_\_\_

What will happen after Enqueue(O), Dequeue()operation takes place? [ ]

(A) front = 2 rear = 5 (B) front = 3 rear = 5

(C) front = 2 rear = 0 (D) front = 3 rear = 0

10) Let the following circular queue can accommodate maximum **six** elements with the following data

front = 2 rear = 4 queue = \_\_\_; L, M, N, \_\_\_, \_\_\_

What will happen after ENQUEUE(O) operation takes place? [ ]

(A) front = 2 rear = 5 queue = \_\_\_; L, M, N, O, \_\_\_

(B) front = 3 rear = 5 queue = L, M, N, O, \_\_\_

(C) front = 3 rear = 4 queue = \_\_\_; L, M, N, O, \_\_\_

(D) front = 2 rear = 4 queue = L, M, N, O, \_\_\_

***II) Problems:***

Write C functions to

1. Enqueue() on Array representation of queue of size 10
2. Dequeue() on Array representation of queue of size 10
3. Traverse() on Array representation of queue of size 10
4. Enqueue() on linked representation of queue
5. Dequeue() on linked representation of queue
6. Traverse() on linked representation of queue
7. Enqueue() on circular queue of size 10
8. Dequeue() on circular queue of size 10