Date 30/09/23
Page No. 10

ASSIGNMENT-2

Q1) Define Queue? How it differs from Stack.

Answer) Queue is a linear data structure in which elements can be inserted only from one side of the list called rear, and the elements can be deleted only from the side called the front. The queue data structure follows the FIFO (First In First Out) principle i.e., the element inserted at first in the list, is the first element to be removed from the list.
The insertion of an element in a queue is called an enqueue operation and the deletion of the element is called a dequeue operations.

It differs from stack through the following:

i) Enqueue (Add) elements at the back of the queue, and dequeue (remove) elements from the front of the queue.
ii) Follows FIFO principle. (First In First Out)
iii) Insertion and deletion in stacks takes place only from one end of the list called the top on other side in queue insertion takes place at the rear of the list and the deletion takes place from the front of the list.

Q3) Write an algorithm to insert a node in doubly linked list.

Answer) Input:
- Doubly Linked List (DLL)
- Node to be inserted (newNode)

Output:
- Doubly linked list with the newNode inserted.

Steps:

① Create a new node (a newNode) with the given data.

② Check if the doubly Linked List (DLL) is empty.

③ - If it is empty, set the newNode as both the head and the tail of the list.
- Update newNode's previous and next pointers to NULL since its the only node in the list.

③ If the DLL is not empty:
- Determine the position where you want to insert newNode.
- For insertion at the beginning (before the current head).
- Set newNode's next pointer to the current head of the DLL.
- Set the current head's previous pointer to newNode.
- Update the DLL's head to newNode.
- Set newNode's previous pointer to NULL.
- For insertion at the end (after the current tail).
- Set newNode's previous pointer to the current tail of the DLL.
- Set the current tail's next pointer to newNode.
- Update the DLL's tail to newNode.
- Set newNode's next pointer to NULL.
- For insertion at a specific position (between two nodes, before or after).
- Traverse the list to find the node before which you want to insert the newNode.
- Update pointers to connect the newNode properly.
- Set newNode's next pointer to the next node.
- Set newNode's previous pointer to the previous node.
- Update the next node's previous pointer to newNode.
- Update the previous node's next pointer to newNode.

④ The newNode has been successfully inserted into the DLL at the desired position.

Q4) How to perform factorial calculation using stack? Explain.

Answer) Algorithm: Calculate Factorial using a stack.

Input:
- An integer n for which you want to calculate the factorial.

Output:
- The factorial of n.

Steps:

① Create an empty stack to store intermediate results.
② Push the value 1 onto the stack. (1 is the base case for factorial)
③ Initialize a variable "result" to 1. This variable will store the final factorial value.
④ Use a loop to calculate the factorial.
- Start a loop from 2 to n (inclusive).
- For each iteration, do the following:
- Peek the top element of the stack (let's call it "top").
- Multiply "top" by the current iteration value (i) and store the result in "top".
- Push "top" onto the stack.
- After the loop, the stack will contain a sequence of values representing the factorial calculation from 1 to n.
⑤ Pop the top element from the stack and store it in the "result" variable. This value is the factorial of n.
⑥ The "result" variable now contains the factorial of n.
⑦ Return the "result" as the output.

Q5) Write an algorithm to delete an item from circular queue.

Answer) Algorithm: Delete an item from a circular queue.

Input:
- Circular Queue (cq) with a size limit (N).
- Position / Index of the item to be deleted.

Output:
- Updated circular queue after deletion.

Steps:
① Check if the circular queue (cq) is empty.
- If the queue is empty, print an error message ("Queue is empty. Deletion not possible.") and exit the algorithm.

② Check if the provided index is valid.
- If index < 0 or index >= N, print an error message ("Invalid index. Deletion not possible.") and exit the algorithm.

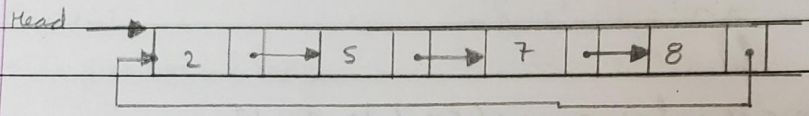
③ If the queue is not empty and the index is valid.
- Decrement the size of the circular queue by 1.

④ Update the circular queue.
- If the deleted item is the only item in the queue (i.e., cq.front = cq.rear).
- Set ~~the value~~ cq.front and cq.rear to -1 to mark the queue as empty.
- If there are multiple items in the queue:
- Shift elements in the circular queue to fill the gap left by the deleted item.
- Update cq.front and cq.rear as needed.

⑤ The item has been successfully deleted from the circular queue.

Q6) Explain circular linked list. Write an algorithm to insert a node into circular linked list.

Answer) The circular linked list is a linked list where all nodes are connected to form a circle. In a circular linked list, the first node and the last node are connected to each other which forms a circle. There is no NULL at the end.



To insert a node into a circular linked list, you need to follow these steps:

① Create a new node with the given data.
② Check if the circular linked list is empty. If it is empty, set the new node as the head and make it point to itself.
③ If the circular linked list is not empty, find the last node in the list (the node whose 'next' pointer points to the head).
④ Update the 'next' pointer of the new node to ~~the~~ point to the current head node's 'next'.
⑤ Update the 'next' pointer of the current head node to point to the new node.
⑥ Update the head of the circular linked list to be the new node.

Q7) Explain double ended queue and its operations.

Answer) Double ended queue is a generalized version of queue data structure that allows the user to insert and delete at both ends.

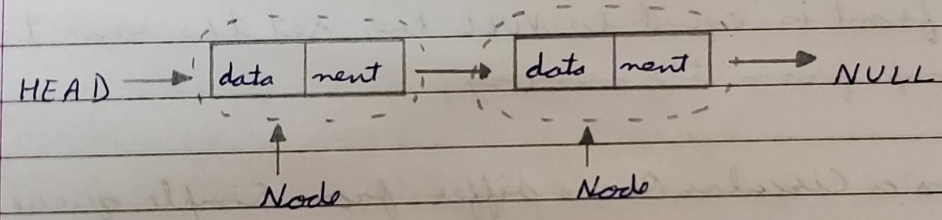
Operations:

1) Push_Front(): Inserts the element at the beginning.
2) Push_Back(): Adds element at the back.
3) Pop_Front(): Removes first element from the queue.

4) Pop_Back(): Removes the last element from the queue.
5) Front(): Gets the front element of the queue.
6) Back(): Gets the last element from the queue.
7) Empty(): Check whether the queue is empty or not.
8) Size(): Determines the number of elements in the queue.

Q8) Explain linked list. How many types of linked lists exist?

Answer) Linked list is a linear data structure, in which elements are not stored at a contiguous location, rather they are linked using pointers. Linked list forms a series of connected nodes, where each node stores the data and the address of the next node.



Types of Linked List:
1) Singly linked list
2) Doubly linked list
3) Circular linked list

Q9) Explain Queue implementation using Linked List?

Answer) Approach:
We maintain two pointers, front and rear. The front points to the first item of the queue and rear points to the last item.

- Enqueue(): This operation adds a new node after the rear and moves the rear to the next node.
- Dequeue(): This operation removes the front node and moves the front to the next node.

- Create a class QNode with data members integer data and QNode* next.
  - A parameterized constructor that takes an integer 'n' value as a parameter and sets data equal to n and next as NULL.
- Create a class Queue with data members QNode front and rear.
- Enqueue operation with parameter 'n':
  - Initialize QNode* temp with data = n.
  - If the rear is set to NULL then set the front and rear to temp and return (Base case).
  - Else set rear next to temp and then move rear to temp.
- Dequeue operation
  - If the front is set to NULL return (Base case).
  - Initialize QNode temp with front and set front to its next.
  - If the front is equal to NULL then set the rear to NULL.
- Delete temp from the memory.

Q10) How does a Circular Queue differ from Simple queue?

Answer)

Linear Queue	Circular Queue
① Arranges the data in a linear pattern.	① Arranges the data in a circular order where rear end is connected with front end.
② The insertion and deletion operations are fixed i.e. done at the rear and front end respectively.	② Insertion and deletion are not fixed and it can be done in any position.
③ Linear queue requires more memory space.	③ It requires a less memory space.
④ Application: People standing for the bus, car lined on a bridge.	④ Application: Computer controlled traffic signal.

Q11) Explain Dequeue. Write algorithm to implement dequeue.

Answer) A dequeue can be visualized as a linear structure with two ends: a front end and a rear end. Elements can be added or removed from both ends.

Insert elements at the rear end of dequeue:

a) First we check dequeue is FULL or not.

b) If Rear = Size - 1
then reinitialize Rear = 0;
Else increment Rear by '1'
and push current key into Arr [rear] = Key
Front remain same.

Insert elements at the front end of dequeue:

a) First we check dequeue is full or not.

b) If front == 0 // initial position, move front to points last index of the array
front = size - 1
Else decremented front by '1' and push
current key into Arr[Front] = Key
Rear remain same

Delete element from rear end of dequeue:

a) First check dequeue is empty or not.

b) If dequeue has only one element
front = -1; rear = -1;
Else if rears points to the first index of array it's means we have to move rear to points last index [now first inserted element at front end become rear end]
rear = size - 1;
Else // decrease rear by '1' rear = rear - 1;

Delete element from Front end of dequeue:

a) First check dequeue is Empty or not.

b) If dequeue has only one element
front = -1, rear = -1;
Else if front points to the last index of the array it's means we have no more elements in array so we move front to points first index of array front = 0;
Else //Increment front by '1'
front = front + 1;

Q12) Explain priority queue.

Answer) A priority queue is a type of queue that arranges elements based on their priority values. Elements with higher priority values are typically retrieved before elements with lower priority values.
In a priority queue, each element has a priority value associated with it. When you add an element to the queue, it is inserted in a position based on its priority value. For example, if you add an element with a high priority value to a priority queue. It may be inserted near the front of the queue, while an element with a low priority value may be inserted near the back.

Q13) Write concept of Header Linked List.

Answer) A header node is a special node that is found at the beginning of the list. A list that contain this type of node, is called the header-linked list. This type of list is useful when information other than that found in each node is needed.

Q14) Write difference between Array and Queue.

Answer)

Array	Queue
① In the array the elements belong to indexes, i.e. if you want to get into the fourth element you have to write arr[3].	① Queues are based on the FIFO principle, i.e., the element inserted at the first, is the first element to come out of the list.
② Insertion and deletion in array can be done at any index in the array.	② Insertion and deletion in queues takes place only from rear and front respectively.
③ Array has a fixed size.	③ Queue has a dynamic and fixed size.
④ Array contains elements of same data type.	④ Queue can contain elements of different data type.
⑤ Different types: 1D, 2D, etc.	⑤ Different types: Circular queue, priority queue, doubly ended queue.

Q2) Explain Array. Describe the storage structure of Array. Also explain various types of array in detail.

Answer) An array is a fundamental data structure used to store a collection of elements or data types under a single variable name. These elements are stored in contiguous memory locations, and each element is accessed by its index or position within the array.

Storage structure of an array:

① Elements: Arrays can hold a fixed number of elements, and each element is of the same data type.

② Indexing: Elements in an array are identified by their index position, starting from zero.

③ Contiguous Memory: Elements in an array are stored in contiguous memory locations. This means that the elements are stored one after the other in memory, with no gaps between them.

④ Fixed size: Arrays have a fixed size, which is determined when the array is declared. Once the size is set, it cannot be changed during runtime.

Arrays can be classified in different types:

① One dimensional array: An array with only one row of data elements is known as one dimensional array. It is stored in ascending memory locations.

② Two dimensional array: An array consisting of multiple rows and columns of data elements is called a two dimensional array. It is also known as matrix.

③ Multiple dimensional array: Multi-dimensional arrays are arrays of arrays. Multi-dimensional array are not bounded to 2 indexes or 2 dimensions as they can include as many indices as per need.