Sairam INSTITUTIONS

Sri SAI RAM ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY
West Tambaram, Chennai - 44

YEAR	SEM
II	III

CS8391

DATA STRUCTURES

Common to CSE & IT

UNIT No. 2

LINEAR DATA STRUCTURES – STACKS, QUEUES

QUESTION BANK

Version: 1.XX

1. What are the types of queues?

- Linear Queues – The queue has two ends, the front end and the rear end. The rear end is where we insert elements and front end is where we delete elements. We can traverse in a linear queue in only one direction ie) from front to rear.
- Circular Queues – Another form of linear queue in which the last position is connected to the first position of the list. The circular queue is similar to linear queue has two ends, the front end and the rear end. The rear end is where we insert elements and front end is where we delete elements. We can traverse in a circular queue in only one direction ie) from front to rear.
- Double-Ended-Queue – Another form of queue in which insertions and deletions are made at both the front and rear ends of the queue.

2. List the applications of stacks

- Towers of Hanoi
- Reversing a string
- Balanced parenthesis
- Recursion using stack
- Evaluation of arithmetic expressions

3.. List the applications of queues

- Jobs submitted to printer
- Real life line
- Calls to large companies
- Access to limited resources in Universities
- Accessing files from file server

4. Define a stack

Stack is an ordered collection of elements in which insertions and deletions are restricted to one end. The end from which elements are added and/or removed is referred to as top of the stack. Stacks are also referred as piles, push-down lists and last-in-first- out (LIFO) lists.

5. List out the basic operations that can be performed on a stack

The basic operations that can be performed on a stack are

- Push operation
- Pop operation
- Peek operation
- Empty check
- Fully occupied check

6. State the different ways of representing expressions

The different ways of representing expressions are

- Infix Notation
- Prefix Notation
- Postfix Notation

7. State the rules to be followed during infix to postfix conversions

- Fully parenthesize the expression starting from left to right. During parenthesizing, the operators having higher precedence are first parenthesized
- Move the operators one by one to their right, such that each operator replaces their corresponding right parenthesis
- The part of the expression, which has been converted into postfix is to be treated as single operand

8. Mention the advantages of representing stacks using linked lists than arrays

- It is not necessary to specify the number of elements to be stored in a stack during its declaration, since memory is allocated dynamically at run time when an element is added to the stack
- Insertions and deletions can be handled easily and efficiently
- Linked list representation of stacks can grow and shrink in size without wasting memory space, depending upon the insertion and deletion that occurs in the list
- Multiple stacks can be represented efficiently using a chain for each stack

9. Mention the advantages of representing stacks using linked lists than arrays

a.It is not necessary to specify the number of elements to be stored in a stack during its declaration, since memory is allocated dynamically at run time when an element is added to the stack

b. Insertions and deletions can be handled easily and efficiently

c. Linked list representation of stacks can grow and shrink in size without wasting memory space, depending upon the insertion and deletion that occurs in the list

d. Multiple stacks can be represented efficiently using a chain for each stack

10. Define a queue

Queue is an ordered collection of elements in which insertions are restricted to one end called the rear end and deletions are restricted to other end called the front end. Queues are also referred as First-In-First-Out (FIFO) Lists.

11. Define a priority queue

Priority queue is a collection of elements, each containing a key referred as the priority for that element. Elements can be inserted in any order (i.e., of alternating priority), but are arranged in order of their priority value in the queue. The elements are deleted from the queue in the order of their priority (i.e., the elements with the highest priority is deleted first). The elements with the same priority are given equal importance and processed accordingly.

12. State the difference between queues and linked lists

The difference between queues and linked lists is that insertions and deletions may occur anywhere in the linked list, but in queues insertions can be made only in the rear end and deletions can be made only in the front end.

13.Define a Deque

Deque (Double-Ended Queue) is another form of a queue in which insertions and deletions are made at both the front and rear ends of the queue. There are two variations of a deque, namely, input restricted deque and output restricted deque.

14. What is the need for Priority queue?

In a multiuser environment, the operating system scheduler must decide which of the several processes to run only for a fixed period of time. One algorithm uses queue. Jobs are initially placed at the end of

the queue. The scheduler will repeatedly take the first job on the queue, run it until either it finishes or its time limit is up and place it at the end of the queue if it does not finish. This strategy is not appropriate, because very short jobs will soon to take a long time because of the wait involved in the run.Generally, it is important that short jobs finish as fast as possible, so these jobs should have precedence over jobs that have already been running. Further more, some jobs that are not short are still very important and should have precedence. This particular application seems to require a special kind of queue, known as priority queue. Priority queue is also called as Heap or Binary Heap.

15. State the operations on stack. Define them and give the diagrammatic representation.

Push

Pop

Top

Push: Push is performed by inserting at the top of stack.

Pop: pop deletes the most recently inserted element.

Top: top operation examines the element at the top of the stack and returns its value.

16. Write the routine for push operation.

Void Push(elementtype X,Stack S) Stack S

Pop(S) Push(X,S) Top(S)

{

Ptrtonode tmpcell; tmpcell=malloc(sizeof(struct Node)); If(tmpcell==NULL)

Fatalerror("out of space!!!");

Else{

Tmpcell->Element=x; tmpcell->Next=s->Next; S->Next=tmpcell;

}}

17. What is the purpose of top and pop?

Top operation examines the element in the top of the list and returns its value. Pop operation deletes the element at the top of the stack and decrements the top of the stack pointer by one.

18. State the disadvantages of linked list implementation of stack.

1. Calls to malloc and free functions are expensive.

2. Using pointers is expensive.

19. Convert into postfix and evaluate the following expression.

(a+b*c)/d

A=2 b=4 c=6 d=2

Ans: Post fix: abc*+d/ Evaluation:

2 4 6 * +2/

=13

20. List the operations of queue.

Two operations

1. Enqueue-inserts an element at the end of the list called the rear.

2. Dequeue-delets and returns the element at the start of the list called as the front.

21. Write the routines for dequeue operation in queue.

Void Dequeue(Queue Q)

{ If(isempty(Q)) Error(“Empty Queue”); Else

Q->front++;

}

22. Give the applications of priority queues.here are three applications of priority queues

1. External sorting.

2. Greedy algorithm implementation.

3. Discrete even simulation.

4. Operating systems.

23. . Define double ended queue(R)

A deque, also known as a double-ended queue, is an ordered collection of items similar to the queue. It has two ends, a front and a rear, and the items remain positioned in the collection.

24. Define a priority queue (R)

Priority queue is a collection of elements, each containing a key referred as the priority for that element. Elements can be inserted in any order (i.e., of alternating priority), but are arranged in order of their priority value in the queue. The elements are deleted from the queue in the order of their priority (i.e., the elements with the highest priority is deleted first). The elements with the same priority are given equal importance and processed accordingly.

25.What is the need for Priority queue?(U)

In a multiuser environment, the operating system scheduler must decide which of the several processes to run only for a fixed period of time. One algorithm uses queue. Jobs are initially placed at the end of the queue. The scheduler will repeatedly take the first job on the queue, run it until either it finishes or its time limit is up and place it at the end of the queue if it does not finish.

This strategy is not appropriate, because very short jobs will soon to take a long time because of the wait involved in the run. Generally, it is important that short jobs finish as fast as possible, so these jobs should have precedence over jobs that have already been running. Further more, some jobs that are not short are still very important and should have precedence. This particular application seems to require a special kind of queue, known as priority queue. Priority queue is also called as Heap or Binary Heap.

26.State the difference between queues and linked lists(U)

The difference between queues and linked lists is that insertions and deletions may occur anywhere in the linked list, but in queues insertions can be made only in the rear end and deletions can be made only in the front end

27.what are the four cases for inserting and deleting the elements in DEQUE ?(U)

1. Insertion At Rear End [same as Linear Queue]

2. Insertion At Front End

3. Deletion At Front End [same as Linear Queue]

4. Deletion At Rear End

PART B

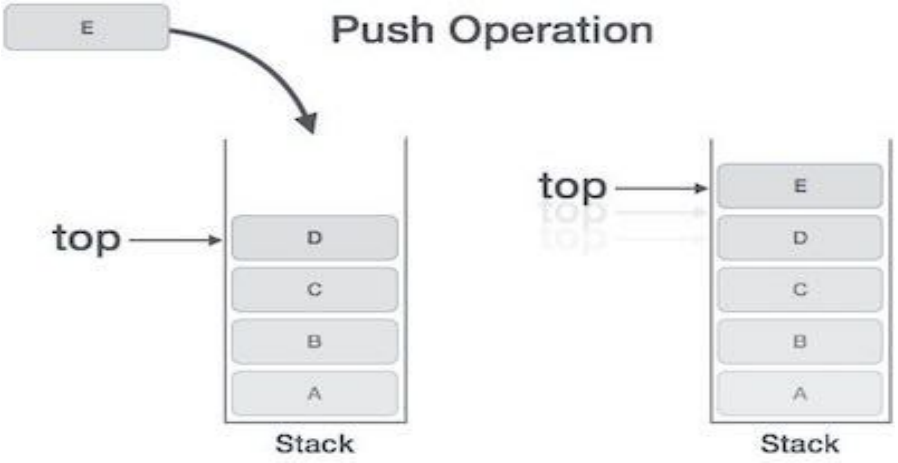
1.Describe about stack ADT using array in detail.

Definition

A stack is linear data structures, a container of elements that are inserted and removed according to the last-in first-out (LIFO) principle. A stack is an ordered list of elements of the same data type.

for example – a deck of cards or a pile of plates, etc.

Stack Representation



Basic Operations

Stack operations may involve initializing the stack, using it and then de-initializing it. Apart from these basic stuffs, a stack is used for the following two primary operations −

- push() − Pushing (storing) an element on the stack.
- pop() − Removing (accessing) an element from the stack.

When data is PUSHed onto stack.

To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks −

- peek() − get the top data element of the stack, without removing it.
- isFull() − check if stack is full.
- isEmpty() − check if stack is empty.

At all times, we maintain a pointer to the last PUSHed data on the stack. As this pointer always represents the top of the stack, hence named top. The top pointer provides the top value of the stack without actually removing it.

Stack Using Array

A stack data structure can be implemented using a one-dimensional array. But stack implemented using array stores only a fixed number of data values. This implementation is very simple. Just define a one dimensional array of specific size and insert or delete the values into that array by using LIFO principle with the help of a variable called 'top'. Initially, the top is set to -1. Whenever we want to insert a value into the stack, increment the top value by one and then insert. Whenever we want to delete a value from the stack, then delete the top value and decrement the top value by one.

Stack Operations using Array

A stack can be implemented using an array as follows...

Before implementing actual operations, first follow the below steps to create an empty stack.

Step 1 - Include all the header files which are used in the program and define a constant 'SIZE' with specific value.

Step 2 - Declare all the functions used in stack implementation.

Step 3 - Create a one dimensional array with fixed size (int stack[SIZE])

Step 4 - Define a integer variable 'top' and initialize with '-1'. (int top = -1)

Step 5 - In the main method, display a menu with a list of operations and make suitable function calls to perform operations selected by the user on the stack.

push(value) - Inserting value into the stack

In a stack, push() is a function used to insert an element into the stack. In a stack, the new element is always inserted at top position. Push function takes one integer value as a parameter and inserts that value into the stack. We can use the following steps to push an element on to the stack...

Step 1 - Check whether stack is FULL. (top == SIZE-1)

Step 2 - If it is FULL, then display "Stack is FULL!!! Insertion is not possible!!!" and terminate the function.

Step 3 - If it is NOT FULL, then increment top value by one (top++) and set stack[top] to value (stack[top] = value).

pop() - Delete a value from the Stack

In a stack, pop() is a function used to delete an element from the stack. In a stack, the element is always deleted from the top position. Pop function does not take any value as a parameter. We can use the following steps to pop an element from the stack...

Step 1 - Check whether stack is EMPTY. (top == -1)

Step 2 - If it is EMPTY, then display "Stack is EMPTY!!! Deletion is not possible!!!" and terminate the function.

Step 3 - If it is NOT EMPTY, then delete stack[top] and decrement top value by one (top--).

display() - Displays the elements of a Stack

We can use the following steps to display the elements of a stack...

Step 1 - Check whether stack is EMPTY. (top == -1)

Step 2 - If it is EMPTY, then display "Stack is EMPTY!!!" and terminate the function.

Step 3 - If it is NOT EMPTY, then define a variable 'i' and initialize it with top. Display stack[i] value and decrement i value by one (i--).

Step 3 - Repeat above step until i value becomes '0'.

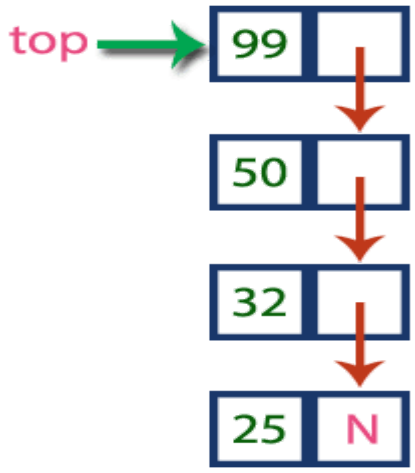
2.Explain the linked list implementation of stack ADT in detail?

The major problem with the stack implemented using an array is, it works only for a fixed number of data values. That means the amount of data must be specified at the beginning of the

implementation itself. Stack implemented using an array is not suitable, when we don't know the size of data which we are going to use. A stack data structure can be implemented by using a linked list data structure. The stack implemented using linked lists can work for an unlimited number of values. That means, a stack implemented using linked lists works for the variable size of data. So, there is no need to fix the size at the beginning of the implementation. The Stack implemented using linked list can organize as many data values as we want.

In linked list implementation of a stack, every new element is inserted as 'top' element. That means every newly inserted element is pointed by 'top'. Whenever we want to remove an element from the stack, simply remove the node which is pointed by 'top' by moving 'top' to its previous node in the list. The next field of the first element must be always NULL.

Example



In the above example, the last inserted node is 99 and the first inserted node is 25. The order of elements inserted is 25, 32,50 and 99.

Stack Operations using Linked List

To implement a stack using a linked list, we need to set the following things before implementing actual operations.

Step 1 - Include all the header files which are used in the program. And declare all the user defined functions.

Step 2 - Define a 'Node' structure with two members data and next.

Step 3 - Define a Node pointer 'top' and set it to NULL.

Step 4 - Implement the main method by displaying the Menu with a list of operations and make suitable function calls in the main method.

push(value) - Inserting an element into the Stack

We can use the following steps to insert a new node into the stack...

- Step 1 - Create a newNode with given value.
- Step 2 - Check whether stack is Empty (top == NULL)
- Step 3 - If it is Empty, then set newNode → next = NULL.
- Step 4 - If it is Not Empty, then set newNode → next = top.
- Step 5 - Finally, set top = newNode.

pop() - Deleting an Element from a Stack

We can use the following steps to delete a node from the stack...

- Step 1 - Check whether stack is Empty (top == NULL).
- Step 2 - If it is Empty, then display "Stack is Empty!!! Deletion is not possible!!!" and terminate the function
- Step 3 - If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'.
- Step 4 - Then set 'top = top → next'.
- Step 5 - Finally, delete 'temp'. (free(temp)).

display() - Displaying stack of elements

We can use the following steps to display the elements (nodes) of a stack...

- Step 1 - Check whether stack is Empty (top == NULL).
- Step 2 - If it is Empty, then display 'Stack is Empty!!!' and terminate the function.
- Step 3 - If it is Not Empty, then define a Node pointer 'temp' and initialize with top.

- Step 4 - Display 'temp → data --->' and move it to the next node. Repeat the same until temp reaches to the first node in the stack. (temp → next != NULL).
- Step 5 - Finally! Display 'temp → data ---> NULL'.

3.Write the algorithm for converting infix expression to postfix (polish) expression?

INFIX EXPRESSIONS

If an operator is preceded and succeeded by an operand then such an expression is termed infix expression.

It follows the scheme of <operand><operator><operand>

E.g., A+B.

POSTFIX EXPRESSIONS

If an operator is succeeded by both the operand then such an expression is termed postfix expression.

It follows the scheme of <operand><operand><operator>

E.g., AB+

Advantage of Postfix Expression over Infix Expression

An infix expression is difficult for the machine to know and keep track of precedence of operators. On the other hand, a postfix expression itself determines the precedence of operators (as the placement of operators in a postfix expression depends upon its precedence). Therefore, for the machine it is easier to carry out a postfix expression than an infix expression.

ALGORITHM TO CONVERT INFIX TO POSTFIX EXPRESSIONS

Let, X is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression Y.

1. Push “(“onto Stack, and add “)” to the end of X.
2. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
3. If an operand is encountered, add it to Y.
4. If a left parenthesis is encountered, push it onto Stack.

5. If an operator is encountered ,then:
    1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.
    2. Add operator to Stack.

    [End of If]
6. If a right parenthesis is encountered ,then:
    1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.
    2. Remove the left Parenthesis.

    [End of If]

    [End of If]
7. END.

EXAMPLE 1

	RPN	Stack	Input Expression
1			A+(B*(C-D)/E)
2	A		+(B*(C-D)/E)
3	A	+	(B*(C-D)/E)
4	A	(+	B*(C-D)/E)
5	AB	(+	*(C-D)/E)
6	AB	* (+	(C-D)/E)
7	AB	(* (+	C-D)/E)
8	ABC	(* (+	-D)/E)
9	ABC	(* (+	D)/E)
10	ABCD	(* (+	)/E)
11	ABCD-	* (+	/E)
12	ABCD-*	/ (+	E)
13	ABCD-*E	/ (+	)
14	ABCD-*E/	+	
15	ABCD-*E/+		

EXAMPLE 2

Infix Expression: A+ (B*C-(D/E^F)*G)*H, where ^ is an exponential operator.

Symbol	Scanned	STACK	Postfix Expression	Description
1.		(		Start
2.	A	(	A	
3.	+	(+	A	
4.	(	(+(	A	
5.	B	(+(	AB	
6.	*	(+(*	AB	
7.	C	(+(*	ABC	
8.	-	(+(-	ABC*	'*' is at higher precedence than '-'
9.	(	(+(-(	ABC*	
10.	D	(+(-(	ABC*D	
11.	/	(+(-(/	ABC*D	
12.	E	(+(-(/	ABC*DE	
13.	^	(+(-(/^	ABC*DE	
14.	F	(+(-(/^	ABC*DEF	
15.	)	(+(-	ABC*DEF^/	Pop from top on Stack, that's why '^' Come first
16.	*	(+(-*	ABC*DEF^/	
17.	G	(+(-*	ABC*DEF^/G	
18.	)	(+	ABC*DEF^/G*-	Pop from top on Stack, that's why '^' Come first
19.	*	(+*	ABC*DEF^/G*-	
20.	H	(+*	ABC*DEF^/G*-H	
21.	)	Empty	ABC*DEF^/G*-H*+	END

Resultant Postfix Expression: ABC*DEF^/G*-H*+

4.Write the algorithm for evaluating postfix expression?

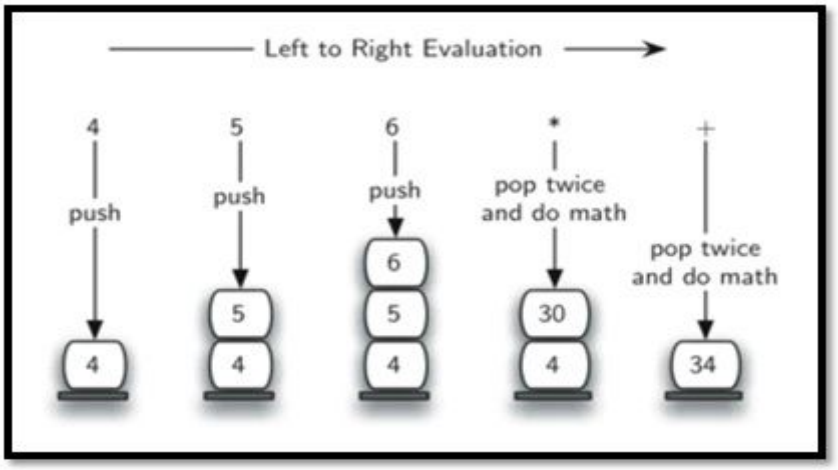
ALGORITHM TO EVALUATE POSTFIX EXPRESSION

1. Read the postfix expression from left to right.
2. If an operand is encountered then push the element in the stack.
3. If an operator is encountered then pop the two operands from the stack and then evaluate it.
4. Push back the result of the evaluation onto the stack.
5. Repeat it till the end of the expression.

EXAMPLE

Expression: 456*+

Step	Input Symbol	Operation	Stack	Calculation
1.	4	Push	4	
2.	5	Push	4,5	
3.	6	Push	4,5,6	
4.	*	Pop(2 elements) & Evaluate	4	5*6=30
5.		Push result(30)	4,30	
6.	+	Pop(2 elements) & Evaluate	Empty	4+30=34
7.		Push result(34)	34	
8.		No-more elements(pop)	Empty	34(Result)



5.Describe about Queue ADT using array in detail.

Queue is a linear data structure in which the insertion and deletion operations are performed at two different ends. In a queue data structure, adding and removing elements are performed at two different positions. The insertion is performed at one end and deletion is performed at another end. In a queue data structure, the insertion operation is performed at a position which is known as 'rear' and the deletion operation is performed at a position which is known as 'front'. In queue data structure, the insertion and deletion operations are performed based on FIFO (First In First Out).

The following operations are performed on a queue data structure:

1. enQueue(value) - (To insert an element into the queue)
2. deQueue() - (To delete an element from the queue)
3. display() - (To display the elements of the queue)

Queue data structure can be implemented in two ways. They are as follows...

1. Using Array
2. Using Linked List

Queue Operations using Array

Queue data structure using array can be implemented as follows...

Before we implement actual operations, first follow the below steps to create an empty queue.

- Step 1 - Include all the header files which are used in the program and define a constant 'SIZE' with specific value.
- Step 2 - Declare all the user defined functions which are used in queue implementation.
- Step 3 - Create a one dimensional array with above defined SIZE (int queue[SIZE])
- Step 4 - Define two integer variables 'front' and 'rear' and initialize both with '-1'. (int front = -1, rear = -1)
- Step 5 - Then implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on queue.

enQueue(value) - Inserting value into the queue

In a queue data structure, enQueue() is a function used to insert a new element into the queue. In a queue, the new element is always inserted at rear position. The enQueue() function takes one integer value as a parameter and inserts that value into the queue. We can use the following steps to insert an element into the queue...

- Step 1 - Check whether queue is FULL. (rear == SIZE-1)
- Step 2 - If it is FULL, then display "Queue is FULL!!! Insertion is not possible!!!" and terminate the function.
- Step 3 - If it is NOT FULL, then increment rear value by one (rear++) and set queue[rear] = value.

deQueue() - Deleting a value from the Queue

In a queue data structure, deQueue() is a function used to delete an element from the queue. In a queue, the element is always deleted from front position. The deQueue() function does not take any value as parameter. We can use the following steps to delete an element from the queue...

- Step 1 - Check whether queue is EMPTY. (front == rear)
- Step 2 - If it is EMPTY, then display "Queue is EMPTY!!! Deletion is not possible!!!" and terminate the function.
- Step 3 - If it is NOT EMPTY, then increment the front value by one (front ++). Then display queue[front] as deleted element. Then check whether both front and rear are equal (front == rear), if it TRUE, then set both front and rear to '-1' (front = rear = -1).

display() - Displays the elements of a Queue

We can use the following steps to display the elements of a queue...

- Step 1 - Check whether queue is EMPTY. (front == rear)
- Step 2 - If it is EMPTY, then display "Queue is EMPTY!!!" and terminate the function.
- Step 3 - If it is NOT EMPTY, then define an integer variable 'i' and set 'i = front+1'.
- Step 4 - Display 'queue[i]' value and increment 'i' value by one (i++). Repeat the same until 'i' value reaches to rear (i <= rear)

6.Explain the linked list implementation of Queue ADT in detail?

Queue Using Linked List:

The major problem with the queue implemented using an array is, It will work for an only fixed number of data values. That means, the amount of data must be specified at the beginning itself. Queue using an array is not suitable when we don't know the size of data which we are going to use. A queue data structure can be implemented using a linked list data structure. The queue which is implemented using a linked list can work for an unlimited number of values. That means, queue using linked list can work for the variable size of data (No need to fix the size at the beginning of the implementation). The Queue implemented using linked list can organize as many data values as we want.

Example

In linked list implementation of a queue, the last inserted node is always pointed by 'rear' and the first node is always pointed by 'front'.

Operations

To implement queue using linked list, we need to set the following things before implementing actual operations.

- Step 1 - Include all the header files which are used in the program. And declare all the user defined functions.
- Step 2 - Define a 'Node' structure with two members data and next.
- Step 3 - Define two Node pointers 'front' and 'rear' and set both to NULL.
- Step 4 - Implement the main method by displaying Menu of list of operations and make suitable function calls in the main method to perform user selected operation.

enQueue(value) - Inserting an element into the Queue

- Step 1 - Create a newNode with given value and set 'newNode → next' to NULL.
- Step 2 - Check whether queue is Empty (rear == NULL)
- Step 3 - If it is Empty then, set front = newNode and rear = newNode.
- Step 4 - If it is Not Empty then, set rear → next = newNode and rear = newNode.

deQueue() - Deleting an Element from Queue

- Step 1 - Check whether queue is Empty (front == NULL).
- Step 2 - If it is Empty, then display "Queue is Empty!!! Deletion is not possible!!!" and terminate from the function

- Step 3 - If it is Not Empty then, define a Node pointer 'temp' and set it to 'front'.
- Step 4 - Then set 'front = front → next' and delete 'temp' (free(temp)).

display() - Displaying the elements of Queue

- Step 1 - Check whether queue is Empty (front == NULL).
- Step 2 - If it is Empty then, display 'Queue is Empty!!!' and terminate the function.
- Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with front.
- Step 4 - Display 'temp → data --->' and move it to the next node. Repeat the same until 'temp' reaches to 'rear' (temp → next != NULL).

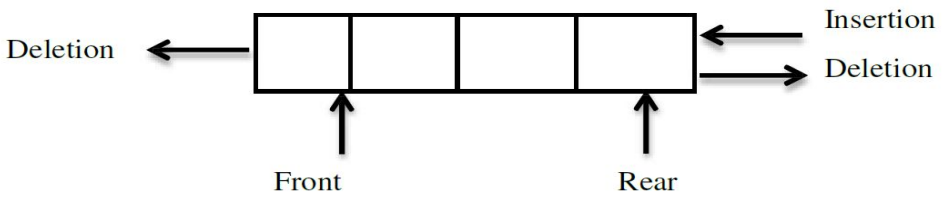
Step 5 - Finally! Display 'temp → data ---> NULL'.

7. What is a DeQueue? Explain its operation with example?(R)

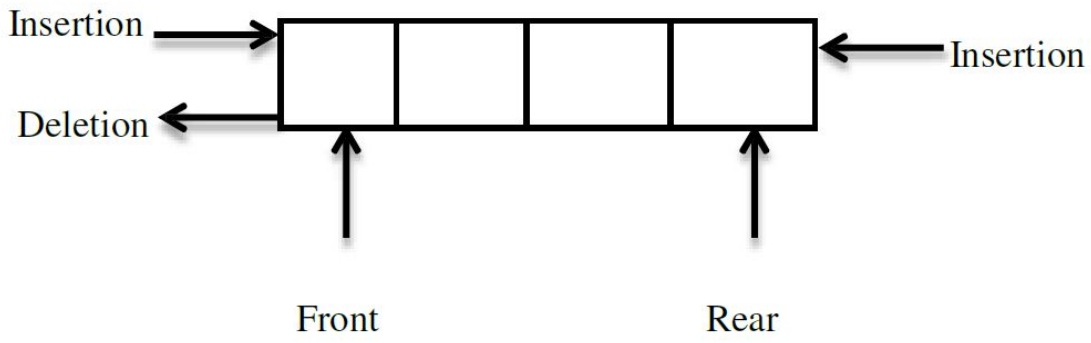
Double ended queue is a more generalized form of queue data structure which allows insertion and removal of elements from both the ends, i.e , front and back. Double Ended Queue also called as deque (pronounced as 'deck' or 'dequeue') is a list in which the elements can be inserted or deleted at either end in constant time. It is also known as a head-tail linked list because elements can be added to or removed from either the front (head) or the back (tail) end. However, no element can be added and deleted from the middle. In the computer's memory, a deque is implemented using either a circular array or a circular doubly linked list.

There are two variants of a double-ended queue. They include:

· Input restricted deque: In this dequeue,insertions can be done only at one of the ends,while deletions can be done from both ends.



Output restricted deque: In this dequeue,deletions can be done only at one of the ends,while insertions can be done on both ends.



There are four basic operations in usage of Deque

1. Insertion at rear end - Enqueue Right
2. Insertion at front end - Enqueue Left
3. Deletion at front end - Dequeue Left
4. Deletion at rear end - Dequeue Right

➔ For input restricted dequeue the operations are Enqueue Right, Dequeue Left, Dequeue Right.

➔ For output restricted dequeue the operations are Enqueue Left, Enqueue Right, Dequeue Left.

8.Write the routine to insert an element in front for dequeue.(U)

```
void Insert_Front ( int X, DEQUE DQ )
{
        if( Front = = 0 )
                Error("Element present in Front!!!!! Insertion not possible");
        else if(Front = = -1)
```

```
        {
                Front = Front + 1;
                Rear = Rear + 1;
                DQ[Front] = X;
        }
        else
        {
                Front = Front - 1;
                DQ[Front] = X;
        }
}
```

9.What is Circular Queue?and its Implementation.(U)

A circular queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle.

- Step 1 - Include all the header files which are used in the program and define a constant 'SIZE' with specific value.
- Step 2 - Declare all user defined functions used in circular queue implementation.
- Step 3 - Create a one dimensional array with above defined SIZE (int cQueue[SIZE])
- Step 4 - Define two integer variables 'front' and 'rear' and initialize both with '-1'. (int front = -1, rear = -1)
- Step 5 - Implement the main method by displaying a menu of operations list and make suitable function calls to perform operations selected by the user on a circular queue.

10.Write a Routine for Insert and Deletion in circular queue(K)

```
void insert(int item)
        {   if((front == 0 && rear == MAX-1) || (front == rear+1))
                {printf("Queue Overflow n");
```

```
                return;
                }
        if(front == -1)
        {
                front = 0;
                rear = 0;
                }
        else
        { if(rear == MAX-1)
                rear = 0;
                else
                rear = rear+1;
        }
cqueue_arr[rear] = item ;
}
```

Delection() Routine

```
void deletion()
        {
                if(front == -1)
                {
                printf("Queue Underflow");
```

```
        return;
        }
    printf("Element deleted from queue is : %dn",cqueue_arr[front]);
        if(front == rear)
        {
        front = -1;
        rear=-1;
        }
        else
        {
        if(front == MAX-1)
            front = 0;
        else
            front = front+1;
    }
    }
```

11.Explain in detail display() routine in circular queue (K)

display() - Displays the elements of a Circular Queue

Following steps are used to display the elements of a circular queue...

- Step 1 - Check whether the queue is EMPTY. (front == -1)

- Step 2 - If it is EMPTY, then display "Queue is EMPTY!!!" and terminate the function.
- Step 3 - If it is NOT EMPTY, then define an integer variable 'i' and set 'i = front'.
- Step 4 - Check whether 'front <= rear', if it is TRUE, then display 'queue[i]' value and increment 'i' value by one (i++). Repeat the same until 'i <= rear' becomes FALSE.
- Step 5 - If 'front <= rear' is FALSE, then display 'queue[i]' value and increment 'i' value by one (i++). Repeat the same until i <= SIZE - 1' becomes FALSE.
- Step 6 - Set i to 0.
- Step 7 - Again display 'cQueue[i]' value and increment i value by one (i++). Repeat the same until 'i <= rear' becomes FALSE.

Display() Routine

```
void display()
{
    int front_pos = front,rear_pos = rear;
        if(front == -1)
        {
                printf("Queue is empty");
                return;
        }
    printf("Queue elements :n");
        if( front_pos <= rear_pos )
                while(front_pos <= rear_pos)
                {
```

```
                printf("%d ",cqueue_arr[front_pos]);
                front_pos++;
            }
        else
        {   while(front_pos <= MAX-1)
            {
                printf("%d ",cqueue_arr[front_pos]);
                front_pos++;
            }
            front_pos = 0;
            while(front_pos <= rear_pos)
            {
                printf("%d ",cqueue_arr[front_pos]);
                front_pos++;
            }
        }
    printf("n");
    }
```

Explanation

The circular queue displayed. Check if the circular queue is empty here as well. Then, print the elements of the circular queue according to the position of the variables front and rear.