BSc 4th Semester

Page No
Date

Unit - 2 (Stack and Queues)

(#) Stacks

- A stack is a linear data structure. A stack is a list of element in which an element may be inserted or deleted only at one end

- Stack principal is FILO (first in last Out) which element inserted first in the stack will be deleted last from the stack.

- To implement the stack, it is required to maintain the pointer to the top of the stack.
  We can access the element only on the top of the stack.

(#) Operation on Stack

The two basic operation associated with stacks are :-

→ Push
→ Pop

* Time Complexity
  Time complexity is a type of complexity that describe the time required to execute an algorithm

→ Push

Push operation is used to add new element in to the stack. At the time of addition first cheack the stack is full or not. If the stack is full it generate an error message "stack overflow".

→ Pop

Pop operation is used to delete element from stack. At the time of deletion first check the stack is empty or not. If the stack is empty it generates an error message "stack underflow"

* All insertion and deletion take place at the same end, so the last element added to the stack will be the first element removed from the stack.

Representation or Implementation of Stack

It can be represented in two ways :-

- Stack using array
- Stack using Linked list

→ Stack using array

Let us consider stack with 6 element capacity

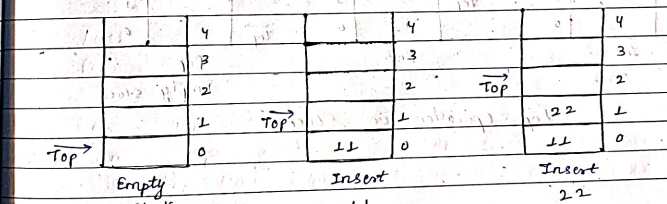
- Push

When an element is added to a stack, the operation is performed by push ( ).

Algorithm :-

Step 1 :- Start
Step 2 :- If top >= size - 1 then
write "stack is overflow"
Step 3 :- Otherwise
read data value 'x'
top = top + 1 ;
stack [top] = x ;
Step 4 :- End

Time Complexity :- O(1)



Push operation on Stack

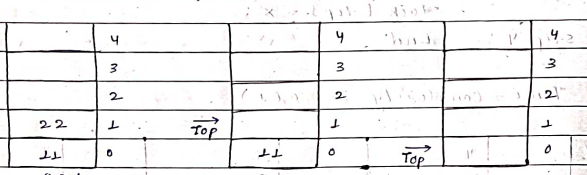
- Pop

When an element is taken off from the stack, the operation is performed by pop( ).

Algorithm :-

Step 1 :- Start
Step 2 :- if top == 1 then
write "stack is underflow"
Step 3 :- otherwise
print "deleted element"
top = top - 1 ;
Step 4 :- End

Time Complexity :- O(1)



Pop Operation on Stack

→ Stack Using Linked List

In a stack push and pop operation are performed at one end called top.

- Push Using Link List

Algorithm :-

- Start
- Create a new node with given value.
- If (top == NULL)
- List is Empty, then set
  new Node → next = NULL
- else
- Not Empty ; then set
  new Node → next = top
- top = new Node
- End

Time Complexity : O(1)

- Pop Using Link List

Algorithm :-

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

Time Complexity : O(n)

(#) Stack as an Abstract Data type

- In Stack ADT Implementation instead of data being stored in each node, the pointer to data is stored.
- The program allocate memory for the data and addressed to the Stack ADT.
- size() - Return the number of element in stack
- isempty() - Return true if the stack is empty.
- is full() - Return true if the stack is full.

Application of Stack

- Stack is used by compiler to check for balancing of parentheses, bracket and braces.
- Stack is used to evaluate a postfix expression.
- Stack is used to convert an infix expression into postfix / prefix form.
- In recursion, all arguement and return value are stored on stack.
- During a function call the return

→ Infix

operand	operator	operand
a	+	b

→ Prefix

operator	operand	operand
+	a	b

→ Postfix

Operand	operand	operator
a	b	+

→ Recursion

* Infix

An infix expression is a mathematical expression in which the operator are placed between the operands.

Ex :- 2 + 3

* Postfix

A postfix expression is a mathematical expression in which the operators are placed after the operands.

Ex :- 2 3 +

* Prefix

A prefix expression is a mathematical expression in which the operators are placed before the operands.

* Recursion

Recursion can be defined as a method through which problems are broken down into smaller sub-problem to find a solution.

→ Types of Recursion

- Direct Recursion

functions in direct recursion call themselves. This procedure comprises a single-step

- Indirect Recursion

Indirect Recursion occur when a function calls other function, which eventually leads back to the original function.

function calling is out of the body

Example :- factorial

```
int fact (int n)
  {
    if (n ≤ 1)
      return 1,
    else
      return n * fact (n-1);
  }
```

Output :-

n = 5

5 * fact (4)
4 * fact (3)
3 * fact (2)
2 * fact (1)
1

4.6=24 4 * f(3)
3.2=6 3 * f(2)
2.1=2 2 * f(1)
1

factorial of 5! = 120

* Stack Operation Condition

→ Lower priority operator will come after higher priority operator.

→ 2 same operator can not come together.

→ At the end of string operator, Reverse the operator.

* Conversion

→ Postfix to Prefix

Ques * - A / BC - / AKL

Ans :-

Symbol	Stack	Expression
*	*	
-	* -	*
A	-	* A
/	- /	* A
B	- /	* AB
C	- /	* ABC
-	- / -	* ABC
	- -	* ABC /
	-	* ABC / -
/	- /	* ABC / -
A	- /	* ABC / - A
K	- /	* ABC / - AK
L	- /	* ABC / - AKL
	←	* ABC / - AKL / -

Conversion - * ABC / - AKL / -

Ques - a / bc - / ade

Ques + * AB / c D

Ques (F - E * D) / C * B ^ A

Symbol	Stack	Expression
(	(	
F	(	F
-	(-	F
E	(-	FE
*	(- *	FE
D	(- *	FED
)	(- *)	FED
	←	FED * -
/	/	FED * -
C	/	FED * - C
*	/ *	FED * - C
		FED * - C /
B	*	FED * - C / B
^	* ^	FED * - C / B
A	* ^	FED * - C / B A
	←	FED * - C / B A ^ *

Conversion :- FED * - C / B A ^ *

Ques Convert to Prefix

5, 8, 16, /, 2, 2, *, -, +

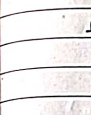
5 is an operand push it into the stack

Stack

8 is an operand push it into the stack

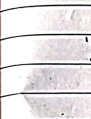
Stack

16 is an operand push it into the stack



Stack

'/' is an operator pop two expression from stack 16/8 = 2



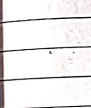
pop

push

2 is an operand push it into the stack

push

2 is an operand push it into the stack



'*' is an operator pop two expression from stack

2 * 2 = 4

4
2
5

Pop

'-' is an operator pop two expression from stack

4 - 2 = 2

2
5

Pop

'+' is an operator pop two expression from stack

2 + 5 = 7

7

→ Prefix to Postfix

Ques * - A / B C - / A K L

L K A / - C B / A - *

Symbol	Stack	Expression
L		L
K		LK
A		LKA
/	/	LKA
-	/-	LKA/
C	-	LKA/C
B	-	LKA/CB
/	-/	LKA/CB
A	-/	LKA/CBA
-	-/-	LKA/CBA/

-	LKA/CBA/
-	LKA/CBA/-
-*	LKA/CBA/-
←	LKA/CBA/-*-

Conversion :- LKA/CBA/-*-

Ques + , - , * , 2 , 2 , 1 , 16 , 8 , 5

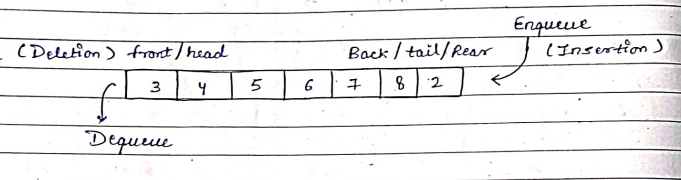
5 , 8 , 16 , / , 2 , 2 , * , - , +

Same as Question 1

Answer :- 7

Queues

- A Queue is used for storing and managing data in a specific order.
- It follows the principal FIFO (first In, first Out), where the first element added to the queue is the first element to be removed
- It operates like a line where element are added at one end (rear) and removed from the other end (front)



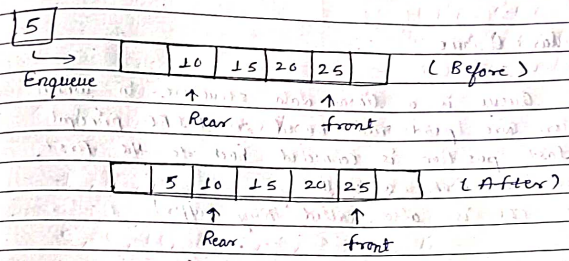
Operations on Queue

* Enqueue

Insert the element at the end of the queue (Rear)

- Check if the queue is full.
- If queue is full, return overflow and exit
- If queue is not full, increment the rear pointer to point next empty space.

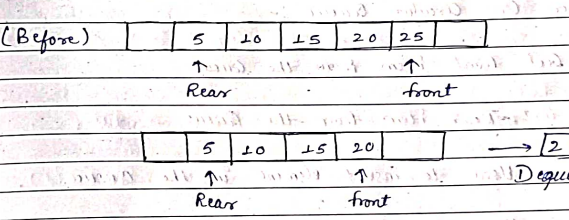
- Add element to the queue, where rear is pointing.



* Dequeue

This operation removes and return an element that is at the front end of the queue.

- If queue is empty, return underflow and exit
- If queue is not empty, access the data where the front is pointing
- Remove element from queue, where front is pointing



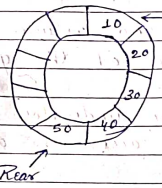
Types of Queues :-

* Circular Queue

Circular Queue is a linear data structure in which the operation are performed based on FIFO principal and last position is connected back to the first position to make a circle.

It is also called 'Ring Buffer'

Time complexity for circular Queue is O(1).

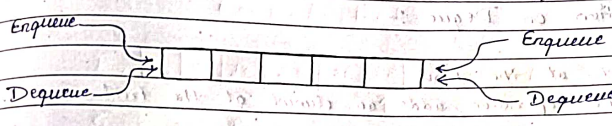


Operation On Circular Queue :-

- Front - Get front item from the Queue
- Rear - Get Rear item from the Queue
- Enqueue - Used to insert element in the Queue.
- Dequeue - Used to delete element in the Queue.

* Double - Ended Queue (Deque)

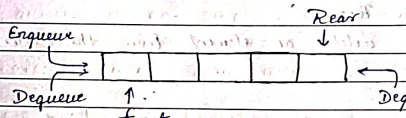
Deque is a linear data structure where the insertion and deletion operation are performed from both end it is generalized version of que.



Types of Deque :-

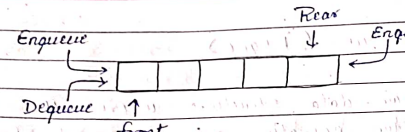
→ Input Restricted Queue

In Input restricted queue insertion operation can be performed at only one end while deletion can be performed from both end.



→ Output Restricted Queue

In Output restricted queue deletion operation can be performed at only one end while insertion can be performed from both end.



Operation on Deque :-

- Insert at the front
  This operation adds an element at the front.
- Insert at the Rear
  This operation adds an element to the rear.
- Delete from the front
  This operation deletes an element from the front.
- Delete from the Rear
  This operation deletes an element from the rear.
- Check Empty
  This operation check if the deque is empty
- Check full
  This operation check if the deque is full.

* Time complexity of Deque is constant i.e O(1).

* Priority Queue

A priority queue is a abstract data type (ADT) where element has some priority the element with highest priority would come first in queue.

Types of Priority Queue

→ Ascending Priority Queue
  Smallest element has the highest priority.

→ Descending Priority Queue
  Largest element has the highest priority.

* Time complexity of priority Queue is O(logn).

- The priority of an element in a priority queue will determine the order in which the element are removed.

Operation on Priority Queue

- Inserting Element in priority Queue
  Maximum priority
- Removing element in priority Queue
  Minimum priority.

* Priority queue works under hip data structure.