20-03-2023

UNIT - 2. STACKS AND QUEUES

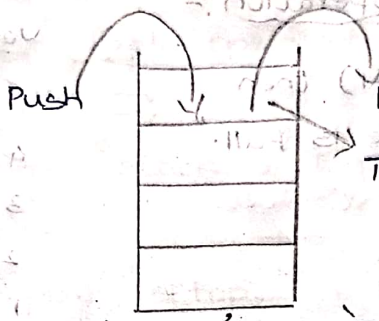
* Stack :-

→ Stack is also known 'LIFO' Data structure.

→ LIFO :- 'Last In First out'

→ For Inserting we use 'Push operation'.

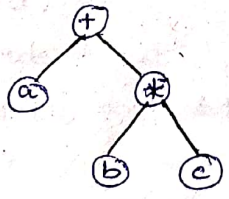
→ For Deletion we use 'Pop operation'.



→ We also use 'peek' and 'Traversal' operations in stacks.

* Applications of stacks :-

1. Recursion.
2. Conversion of Expressions
   Ex :- ① Infix into prefix/postfix
   - 4*5 } Infix expression.
   - a*b }
   - *ab → Prefix expression
   - ab* → postfix expression.
3. Expression Trees.



* Methods for Implementing stacks :-

→ Two ways :-

1. By using 'Arrays'.
2. By using 'Linked List'.

→ For Four operations

① Push ② Pop ③ Peek ④ Traversal.

① Implementing stacks using Arrays :-

→ Initially we initialize Top End as '-1' i.e top = -1.

→ We use Top, s[MAX], x

ⓐ Top :- Top is a 'Index variable' initially 'Top = -1'

ⓑ s[MAX] :- 's' means Array Name and 'MAX' is size.

ⓒ x :- To Read the Input.

[using symbolic constant]

Global
Top = -1, s[MAX]

* Algorithm for Push operation :-

Void Push (int x);

Step-1 :- if (top == MAX-1) then

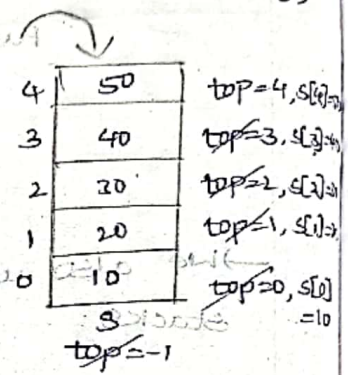
ⓘ Display "stack is Full."

Step-2 :- else

ⓘ Set top = top+1.

ⓘⓘ Set s[top] = x.

Step-3 :- EXIT.



* Algorithm for Pop operation :-

Step-1 :- if (top == -1) then

ⓘ Display "stack is Empty / underflow"

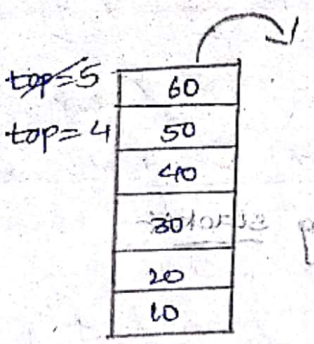
Step-2 :- else

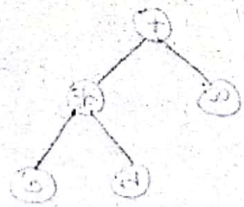
ⓘ Display "Deleting Element is s[top]"

ⓘⓘ Set top = top-1

Step-3 :- EXIT.

* Example :-





s[5] = 60 is Deleted. 4 = 5

push(100) Now top = top+1 4+1 = 5 [∵ top = top+1]

s[5] = 100.

⇒ 4 = 4+1 = 5

pop( ), s[5] = 100 deleted

pop( ), s[4] = 50 deleted

s[0] = 10 deleted then top = -1

* Algorithm for Traversal of stacks:-

step-1:- if (top==-1)

(i) Display "stack is underflow/Empty."

step-2:- else

(i) for(i=top; i>=0; i--)

(a) Display "s[i]."

step-3:- EXIT.

* Algorithm for Peek operation:-

step-1:- if(top==-1)

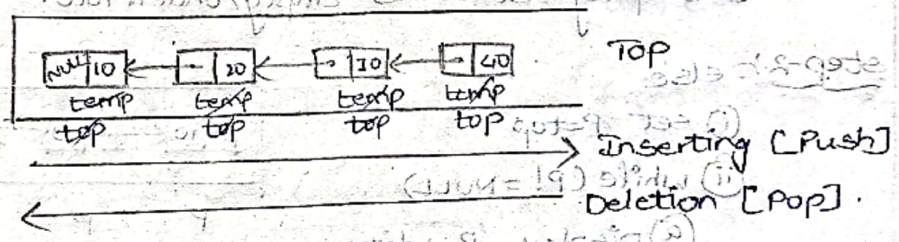
(i) Display "stack is Empty/underflow."

step-2:- else

(i) Display "s[top]"

step-3:- EXIT.

II Implementation of stacks using Linked List:-



structure:-

```
struct node
{
  int data;
  struct node *next;
} *top=NULL, *p, *temp;
```

(*) Algorithm for Push operation:-

step-1:- create a new Node ie temp

step-2:- Input the data to be inserted [using x]

step-3:- (i) Set temp->data=x,

(ii) Set temp->next=NULL.

step-4:- if (top==NULL)

(i) set top=temp.

step-5:- else

(i) set temp→next = top

(ii) Set top = temp.

step-6:- EXIT.

(2) Algorithm for Pop operation :-

step-1:- if (top == NULL) then

(i) Display "stack is Empty."

step-2:- else.

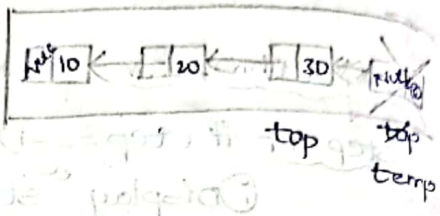
(i) set temp = top.

(ii) set top = top→next.

(iii) set temp→next = NULL.

(iv) Delete temp.

step-3:- EXIT.



(3) Algorithm for Traversal of stacks :-

step-1:- if (top == NULL) then

(i) Display "stack is Empty/underflow."

step-2:- else

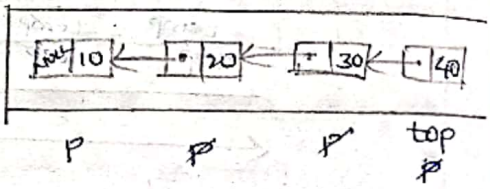
(i) set P = top

(ii) while (P != NULL)

(a) Display P→data

(b) set P = P→next

step-3:- EXIT.



o/p :- 40, 30, 20, 10

(4) Algorithm for Peek operation :-

step-1:- if (top == NULL) then

(i) Display "stack is Empty"

step-2:- else

(i) Display "top→data"

step-3:- EXIT.

* Applications of stacks:-

1] Reversing a string using stacks:-

step-1 :- Read a string using loop [characters one by one].

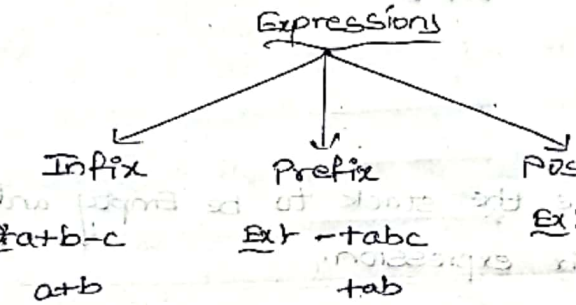
step-2 :- By using push operation, we add characters of given string one by one.

step-3 :- By using pop operation, we add characters one by one to another array. [called b].

step-4 :- Display the array [b].

Step-5 :- EXIT.

2]* Expressions



* Algorithm & Example for conversion from Infix to Prefix Expressions:

Ex:- a+b*c/d-f

⇒ (a+b*c/(d-f)). ⇒))f-d c/c*b+aC.

Input	stack	output
)	)	Empty
)	))	Empty.
f	))	f
-	)) -	f
d	)) -	fd
c	)	fd-
/	) /	fd-
c	) /	fd-c
*	) / *	fd-c
b	) / *	fd-cb
+	) *	fd-cb*/
a	) +	fd-cb*/a.
(	empty	fd-cb*/a+

→ We need to Reverse the output.

fd – cb * / a +

⇒ +a /*bc – fd

(b*c) (d–f)

x / y

+a z

a + z

check → Given exp. (a+b*c/(d–f))

x / y

a + z

a + z

* Algorithm :

Step-1 : Initialize the stack to be empty and reverse the given Infix expression.

Step-2 : Scan the given reverse expression from left to right character by character.

(i) If the input character is Right parenthesis ')' then push it onto the stack.

(ii) else if the input character is Alpha Numeric then append it to the output.

(iii) else if the input is left parenthesis '(' then

(a) Pop the all operators from the stack & append all the operators to output until right parenthesis encountered.

(b) Pop the right parenthesis ')' from the stack and discard it.

(iv) else if Top of stack is Right parenthesis ')' then push the input character onto the stack.

(v) else

(a) Repeat the loop until the stack is not empty and input character is lower precedence than the top of the stack.

(A) Pop the stack and append the popped elements into stack output.

(B) Push the Input Operator into the stack.

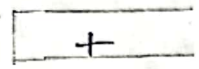
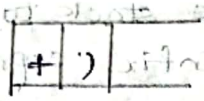
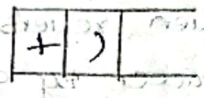
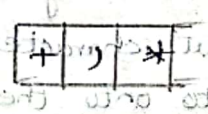
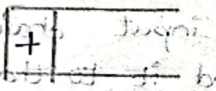
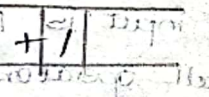
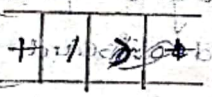
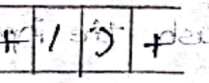
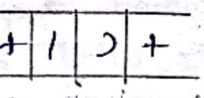
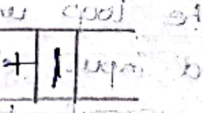
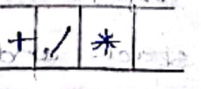
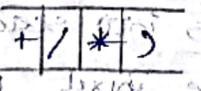
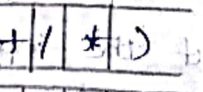
Step-3 : If the end of the input string is encountered then Iterate the loop until the stack is not empty. Pop the stack and append the remaining input string to the output.

Step-4 : EXIT.

* Example :-

Infix expression :- (A-B) *(C+D) / (E*F) + G.

Reversing :- G +) F * E (/) D + C (*) B - A (

Input	stack	output.
G	Empty	G
+		G
)		G
F		G F
*		G F E
(		G F E *
/		G F E *
)		G F E * D C
D		G F E * D
+		G F E * D
C		G F E * D C
(		G F E * D C +
*		G F E * D C +
)		G F E * D C +
B		G F E * D C + B
-		G F E * D C + B *
A		G F E * D C + B A
(		G F E * D C + B A -

* Queues :-

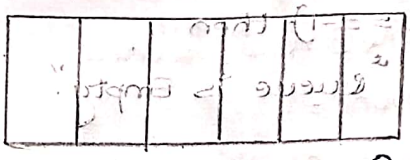
→ 'Queue' is a 'FIFO Data structure' [First In First Out].

→ Always 'Insertion operation' performs at 'Rear End' only.

→ Always 'Deletion operation' performs at 'Front-End' only.

→ 'Insertion operation' is also known as 'EnQueue operation'.

→ 'Deletion operation' is also known as 'Dequeue operation'.



Q[MAX]; → symbolic constant.

front = rear = -1

* Types of Queues :-

→ There are three types of queues.

① Linear Queue

② Circular Queue

③ 'Double Ended Queue'

① Implementation of Linear queue by Arrays :-

* Algorithm for Inserting an element into queue :-

step-1 :- if (rear ≥ MAX-1) then

① display "Queue is overflow".

step-2 :- else

① set rear = rear+1

② set Q[rear] = x.

step-3 :- if (rear == 0)

① set front = 0.

step-4 :- EXIT.

② * Algorithm for Deleting an element from queue:

step-1 :- if (front == -1) then
  (i) Display "Queue is underflow."

step-2 :- else
  (i) Display "Deleting element @[front]"
  (ii) set front = front+1

step-3 :- if (front > rear)
  (i) Set front = rear = -1

step-4 :- EXIT.

③ Algorithm for Traversal of queue :-

step-1 :- if (front == -1) then
  (i) Display "Queue is Empty"

step-2 :- else
  (i) for(i=front; i<=rear; i++)
    (a) Display "@[i]"

step-3 :- EXIT.

* Implementation of queues using Linked List :-

① Algorithm for inserting a node into queue :-

```
struct node
{ int data;
  struct node *next;
};* front=NULL, *rear=NULL, *p, *temp;
```

step-1 :- create a new node i.e. temp.

step-2 :- Input the data to be inserted using 'x' variable.

step-3 :- set temp->data=x;
  set temp->next=NULL.

step-4 :- if (front==NULL) then
  (i) set front=temp.
  (ii) set rear=temp.

step-5 :- else

(i) set rear → next = temp

(ii) set rear = temp

step-6 :- EXIT.

(2) Algorithm for Deleting a Node at begining :-

step-1 :- if (front == NULL)

Display "Circular queue is Empty."

step-2 :- else

(i) set temp = front

(ii) set front = front → next

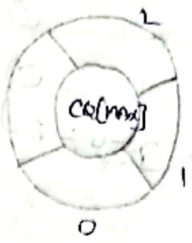
(iii) set temp → next = NULL;

* Circular Queue :-

Front = Rear = -1.

Front = (Front + 1) % MAX

Rear = (Rear + 1) % MAX.



(1) * Algorithm for Enqueue Operation (or) Inserting an Element into circular Queue :-

CQ[MAX], front = -1, rear = -1
void insert CQ(int data)

step-1 :- if (front == (rear+1) % MAX)
  (i) Display "circular queue is Full."

step-2 :- else
  (i) set rear = (rear+1) % MAX
  (ii) set CQ[rear] = data.
  (iii) if (front == -1)
    (a) set front = 0.

step-3 :- EXIT.

(2) Algorithm for Dequeue operation :-

step-1 :- if (front == -1) then
  (i) Display "circular queue is underflow."

step-2 :- else
  (i) Display "Deleting element CQ[front]."
  (ii) if (front = ~~front~~ = rear)
    (a) set front = -1
    (b) set rear = -1
  (iii) else
    (a) set front = (front+1) % MAX.

step-3 :- EXIT.

(3) Algorithm for Traversal of circular Queue :-

step-1 :- if (front == -1) then
  (i) Display "circular queue is Empty."

step-2 :- else
  (i) if (front <= rear)
    (a) for (i = front ; i <= rear ; i++)
      (A) Display "CQ[i]."

(ii) else.

(a) for (i=front; i<=MAX-1; i++)

(A) Display cq[i].

(b) for (i=0; i<=rear; i++)

(A) Display cq[i].

step-3:- EXIT.

(or)

step-3:- else (front<=rear)

(i) for(i=front; i<=rear; i++)

(a) Display "cq[i]".

(ii) else.

(a) i=front+1

(b) do

{

cq[i]

i=(i+1)%MAX

}

while(i!=rear)

(c) Display cq[rear]

step-3:- EXIT.

* Priority queue:-

(1) Ascending PQ.

(2) Descending PQ.

* Algorithm for Enqueue operation:-

Enqueue PQ(int data)

PQ[MAX], itemcount=0

step-1:- if (PQ is Full(-))

(i) if (item count == 0)

(a) set PQ[itemcount++] = data.

(ii) else

(a) for (i=itemcount-1; i>=0; i--).

(A) if (data > PQ[i])

(I) PQ[i+1] = PQ[i];

(B) else

(I) break;

(b) set PQ[i+1] = data;

(c) set itemcount = itemcount+1

step-2:- EXIT.

* Algorithm for Dequeue of PQ :-

step-1 :- if (! isempty( ))

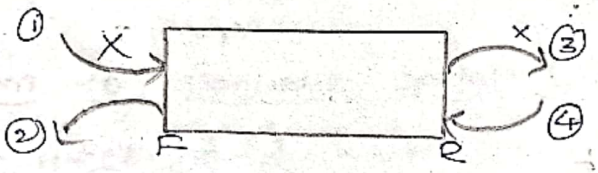
(i) return PQ [--itemcount].

step-2 :- EXIT.

* isfull & isempty :-

```
bool isFull ( )
{
   return itemcount==MAX;
}
bool isempty ( )
{
   return itemcount==0;
}.
```

* Double Ended Queue :-



→ The Double Ended queues are classified into

(1) Input Restricted. [Insert at Rear only]. ①, ③, ④ ✓

(2) output Restricted. [Deletion at Front only]. ①, ②, ④ ✓

* Algorithm for Enqueue operation at Front End :-

DQ[MAX], front = rear = -1

step-1 :- if (front==0)

Display "Insertion not possible at Front end."

step-2 :- else

(i) if (front==-1)

(a) set front = front+1

(b) DQ [front] = data

(c) if (rear == -1)

(A) set rear=0

(ii) else

(a) set front = front-1

(b) DQ [front] = data

step-3 :- EXIT.

* Algorithm for Deletion at Rear End :-

step-1 :- if (front == -1)

(i) Display "Queue is Empty."

step-2 :- else

(i) if (rear == front)

(a) set front = -1

(b) set rear = -1

(ii) else

(a) display Deleting element DQ[rear].

(b) set rear = rear - 1

step-3 :- EXIT.

* Applications of Queues :-

(1) Online Reservations.

(2) schooling

(3) Mails.

(4) Priority Queues.