Stacks and Queues in List

Learning Objectives:

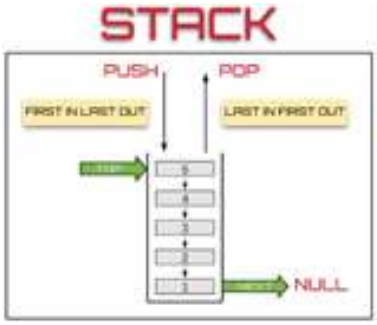
- Understand a stack and a queue
- Perform Insertion and Deletion operations on stacks and queues
- Learn Infix and Postfix expressions
- Convert an infix to postfix
- Evaluation of Postfix Expression

Stack:

A stack is a data structure whose elements are accessed according to the Last-In First-Out (LIFO) principle. This is because in a stack, insertion and deletion of elements can only take place at one end, called top of the stack.

- A stack can is characterized by only two fundamental operations: *push* and *pop*. The push operation adds an item to the top of the stack.
- The pop operation removes an item from the top of the stack, and returns this value to the caller.
- A stack is a *restricted data structure*, because only a small number of operations are performed on it.

Consider the following examples of stacks:



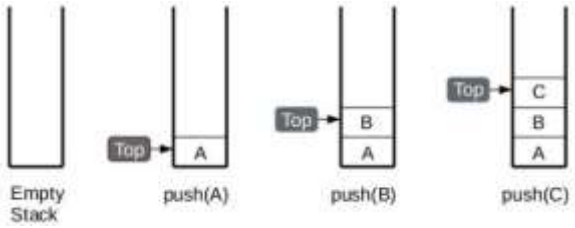
The two operations performed on the stack are:

1. **Push:** Adding (inserting) new element on to the stack.
2. **Pop:** Removing (deleting) an element from the stack

Push operation

Adding new element to the stack list is called push operation. When the stack is empty, the value of top is

1. Basically, an empty stack is initialized with an invalid subscript. Whenever a Push operation is performed, the top is incremented by one and then the new value is inserted on the top of the list till the time the value of top is less than or equal to the size of the stack.

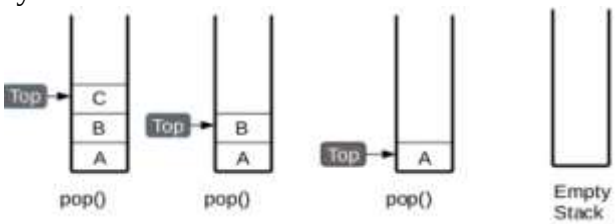


The algorithm for Push operation:

Step1: Start
Step 2: Initialize top with -1.
Step 3: Input the new element.
Step 4: Increment top by one.
Step 5: stack[top]=new element
Step 6: Print "Item Inserted"
Step 7: Stop

Pop operation

Removing existing elements from the stack list is called pop operation. Here we have to check if the stack is empty by checking the value of top. If the value of top is -1, then the stack is empty and such a situation is called Underflow. Otherwise Pop operation can be performed in the stack. The top is decremented by one if an element is deleted from the list.



The algorithm for pop operation:

Step 1: Start
Step 2: If the value of top is -1 go to step 3 else go to step 4
Step 3: Print "Stack Empty" and go to step 7
Step 4: Deleted item = Stack[top]
Step 5: Decrement top by 1
Step 6: print "Item Deleted"
Step 7: Stop

Traversal in a stack

Traversal is moving through the elements of the stack. If you want to display all the elements of the stack, the algorithm will be as follows:

Step 1: Start
Step 2: Check the value of top. If top=-1 go to step 3 else go to step 4
Step 3: Print "Stack Empty" and go to step 7
Step 4: Print the top element of the stack.
Step 5: Decrement top by 1
Step 6: If top= -1 go to step 7 else go to step 4
Step 7: Stop

In Python, Consider the following programs that perform the Push and Pop operation on the stack through append() and pop().

Program to implement stack

```
s=[]
c="y"
while (c=="y"):
print "1. PUSH"
print "2. POP "
print "3. Display"
choice=input("Enter your choice: ")
if (choice==1):
        a=input("Enter any number :")
        s.append(a)
elif (choice==2):
        if (s==[]):
        print "Stack Empty"
else:
        print "Deleted element is : ",s.pop()
        elif (choice==3):
        l=len(s)
        for i in range(l-1,-1,-1):
        print s[i]
else:
        print("Wrong Input")
        c=raw_input("Do you want to continue or not? ")
```

Program to implement a stack(Using classes)

```
class stack:
s=[]
def push(self):
a=input("Enter any number :")
stack.s.append(a)
def display(self):
l=len(stack.s)
for i in range(l-1,-1,-1):
print stack.s[i]
a=stack()
c="y"
while (c=="y"):
print "1. PUSH"
print "2. POP "
print "3. Display"
choice=input("Enter your choice: ")
        if (choice==1):
```

```
            a.push()
    elif (choice==2):
            if (a.s==[]):
                    print "Stack Empty"
            else:
                    print "Deleted element is : ",a.s.pop()
    elif (choice==3):
            a.display()
    else:
            print("Wrong Input")
            c=raw_input("Do you want to continue or not? ")
```

output:

Output:

```
>>>
1. PUSH
2. POP
3. Display
Enter your choice: 1
Enter any number :100
Do you want to continue or not? y
1. PUSH
2. POP
3. Display
Enter your choice: 1
Enter any number :200
Do you want to continue or not? y
1. PUSH
2. POP
3. Display
Enter your choice: 3
200
100
Do you want to continue or not? y
1. PUSH
2. POP 3.
Display
Enter your choice: 2
Deleted element is : 200
Do you want to continue or not? y
1. PUSH
2. POP
3. Display
Enter your choice: 2
Deleted element is : 100
Do you want to continue or not: y
1. PUSH
2. POP
```

```
3. Display
Enter your choice: 2
Stack Empty
Do you want to continue or not? n
>>>
```

Application of Stacks

It is used to reverse a word. You push a given word to stack –letter by letter and then pop letter from the stack.

- "Undo" mechanism in text editor.
- Backtracking: This is a process when you need to access the most recent data element in a series of elements. Once you reach a dead end, you must backtrack.
- Language Processing: Compiler' syntax check for matching braces in implemented by using stack.
- Conversion of decimal number to binary.
- To solve tower of Hanoi.
- Conversion of infix expression into prefix and postfix.

Application of stacks in infix expression to postfix expression conversion

- Infix expression operand1 operator operand2 for example a+b
- Postfix expression operand1 operand2 operator for example ab+
- Prefix expression operator operand1 operand2 for example +ab

Some example of infix expression and their corresponding postfix expression

Infix expression postfix expression

a*(b-c/e abc-*e/

a+b)*c -d/e ab+cd-*e/

a+b*c/d -e+f abc*+de-/f+

Algorithm to convert infix expression to postfix expression using stack

The following algorithm shows the logic to convert an infix expression to an equivalent postfix expression:

Step 1: Start

Step 2: Add "("(left parenthesis) and ")" (right parenthesis) to the start and end of the expression(E).

Step 3: Push "("left parenthesis onto stack.

Step 4: Check all symbols from left to right and repeat step 5 for each symbol of 'E' until the stack becomes empty.

Step 5: If the symbol is:

i) an operand then add it to list.

ii) a left parenthesis "("then push it onto stack.

iii) an operator then:

a) Pop operator from stack and add to list which has the same or higher precedence than the incoming operator.

b) Otherwise add incoming operator to stack.

iv) A right parenthesis")" then:

a) Pop each operator from stack and add to list until a left parenthesis is encountered.

b) Remove the left parenthesis.

Step 6: Stop

EXAMPLE: Let us illustrate the procedure *InfixToPostfix* with the following arithmetic expression:
Input: (A + B)^ C - (D * E) / F) (infix form)

Read symbol	*Stack*	*Output*
Initial	(	
1	((	
2	((	A
3	((+	A
4	((+	AB
5	(	AB+
6	(^	AB+
7	(^	AB + C
8	(–	AB + C ^
9	(– (	AB + C ^
10	(– (	AB + C ^ D
11	(– (*	AB + C ^ D
12	(– (*	AB + C ^ DE
13	(–	AB + C ^ DE *
14	(– /	AB + C ^ DE *
15	(– /	AB + C ^ DE * F
16		AB + C ^ DE * F / –

Output: A B + C ^ DE * F / – (postfix form)

Evaluation of Postfix Expression

The algorithm to evaluate a postfix expression is as follows:

Step 1: Start

Step 2: Check all symbols from left to right and repeat steps 3 & 4 for each symbol of expression 'E' until all symbols are over.

i) If the symbol is an operand, push it onto stack.

ii) If the symbol is an operator then

a) Pop the top two operands from stack and apply an operator in between them.

b) Evaluate the expression and place the result back on stack.

Step 3: Set result equal to top element on the stack.

Step 4: Stop

Example: Evaluate the following postfix expression showing stack status after every step
8, 2, +, 5, 3, -, *, 4 /

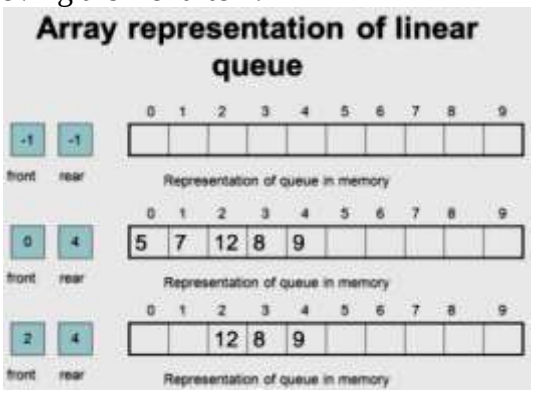
token scanned from postfix expression	Stack status bold letter shows the top of the stack) after processing the scanned token	Operation performed
8	**8**	Push 8
2	8, **2**	Push 2
+	**10**	Op2=pop(i.e 2 Op1=pop(i.e 8 Push(op1+op2) i.e. 8+2
5	10, **5**	Push(5)
3	10, 5, **3**	Push(3)
-	10, **2**	Op2=pop(i.e. 3 Op1=pop(i.e. 5 Push(op1-op2) i.e. 5-3
*	**20**	Op2=pop(i.e. 2 Op1=pop(i.e. 10 Push(op1-op2) i.e. 10*2
4	20, **4**	Push 4
/	5	Op2=pop(i.e. 4 Op1=pop(i.e. 20 Push(op1/op2) i.e. 20/4
NULL	Final result 5	Pop 5 and return 5

Queue

Queue is a linear data structure which follows First In First Out FIFO rule in which a new item is added
at the rear end and deletion of item is from the front end of the queue. In a FIFO data structure, the first
element added to the queue will be the first one to be removed.

A queue is a linear data structure which follows an ordered collection of items where an item is inserted at one end called the "rear" and an existing item is removed at the other end, called the "front".

- Queue is also called as **FIFO** list i.e. **First-In First-Out**.
- In the queue only two operations are allowed enqueue and dequeue.
- Enqueue means to insert an item into back of the queue.
- Dequeue means removing the front item.



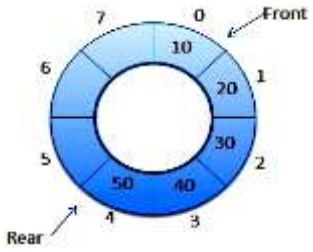
Types of Queues:

Queue can be of four types:

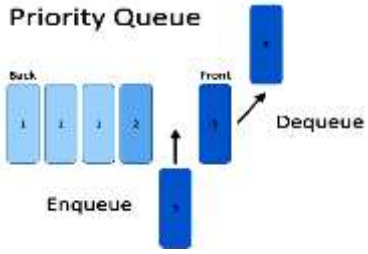
- Simple Queue
- Circular Queue
- Priority Queue
- De-queue (Double Ended Queue)

Simple Queue: In simple queue insertion occurs at the rear end of the list and deletion occurs at the front end of the list.

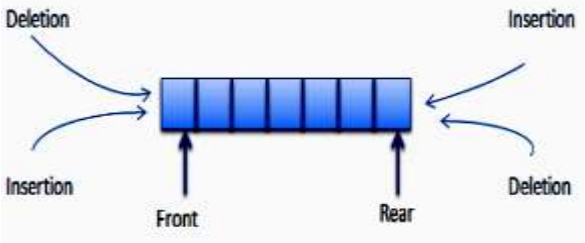
Circular Queue: A circular queue is a queue in which all nodes are treated as circular such that the last node follows the first node.



Priority Queue: A priority queue is a queue that contains items that have some present priority. An element can be inserted or removed from any position depending upon some priority.



Dequeue: It is a queue in which insertion and deletion takes place at the both ends.



Operation on Queues:

- **Queue( )**: It creates a new queue that is empty.
- **enqueue(item)**: It adds a new item to the rear of the queue.
- **dequeue( )**: It removes the front item from the queue.
- **isEmpty( )**: It tests to see whether the queue is empty.

Following are the formal steps for INSERT and DELETE operations

Algorithm for insertion:

Step 1: Start
Step 2: Check FRONT and REAR value, if both the values are -1, then
FRONT and REAR are incremented by 1 other wise Rear is incremented by one.
Step 3: Add new element at Rear. (i.e.) queue[Rear]=new element.
Step 4: Stop

Algorithm for deletion:

Step 1: Start
Step 2: Check for underflow situation by checking value of Front = -1
If it is display appropriate message and stop
Otherwise
Step 3: Deleted item = queue [Front]

Step 4: If Front = Rear then Front =Rear = -1
Otherwise Front is incremented by one
Step 5: Print "Item Deleted"
Step 6: Stop

Although principle operations in queue are Insert and Delete, but as a learner, we need to know the contents of queue at any point of time. To handle such requirement we will add traversal operation in our program.

Algorithm for Display

Step 1. Start
Step 2. Store front value in I
Step 3. Check I position value, if I value is -1 go to step 4 else go to step 5
Step 4. Print "Queue Empty" and go to step 8
Step 5. Print queue[I]
Step 6. I is incremented by 1
Step 7. Check I position value, if I value is equal to rear+1 go to step 8 else go to step 5
Step 8: Stop

Note: In Python already we have **del()** and **append()** functions for deletion of elements at the front and addition of elements at the rear. Hence, no need of writing special function for add and remove elements in the queue. Likewise, 'Front and Rear positions' are also not required in the Python programming while implementing queue.

Example:
Write a program to implement Queue using list.
Code: (without using class)

```
a=[]
c='y'
while c=='y':
    print "1. INSERT"
    print "2. DELETE "
    print "3. Display"
    choice=input("enter your choice ")
    if (choice==1):
        b=input("enter new number ")
        a.append(b)
    elif (choice==2):
        if (a==[]):
        print("Queue Empty")
    else:
        print "deleted element is:",a[0]
        del a[0]
    elif (choice==3):
        l=len(a)
    for i in range(0,l):
        print a[i]
    else:
```

```
                print("wrong input")
        c=input("do you want to continue or not ")
```

Program: (Using class)

```
class queue:
        q=[]
        def insertion(self):
                a=input("enter any number: ")
                queue.q.append(a)
        def deletion(self):
                if (queue.q==[]):
                        print "Queue empty"
                else:
                print "deleted element is: ",queue.q[0]
                del queue.q[0]
        def display(self):
                l=len(queue.q)
                for i in range(0,l):
                        print queue.q[i]
                        a=queue()
                        c="y"
                while (c=="y"):
                        print "1. INSERTION"
                        print "2. DELETION "
                        print "3. DISPLAY"
                        choice=input("enter your choice: ")
                        if (choice==1):
                                a.insertion()
                        elif (choice==2):
                                a.deletion()
                        elif (choice==3):
                                a.display()
                        else:
                                print("wrong input")
                        c=input("do you want to continue or not :")
```

Applications of queue in computers:

- In a single processor multi-tasking computer, job(s) waiting to be processed form a queue.
- sharing a printer with many computers.
- Compiling a HLL code
- Using down load manager, for multiple files also uses queue for ordering the files.
- In multiuser OS - job scheduling is done through queue.

EXERCISE

1. Expand the following:
   (i) LIFO (ii) FIFO
2. What is stack?
3. What is Queue?
4. What are all operations possible in data structure?
5. Give one example of infix expression.
6. Give one example of postfix expression.
7. Give one example of prefix expression.
8. Convert (A+B)*C in to postfix form.
9. Evaluate using stack 10, 3,*,30,2,*,-
10. Converting following Infix expression to Postfix notation:
    a) (A+B)*C+D/E-F
    b) P+Q*(R-S)/T
    c) (True And False) || (False And True)
11. Evaluation the following Postfix Expression:
    a) 20,8,4,/,2,3,+,*,-
    b) 15,3,2,+,/,7,+,2,*
    c) False, Not, True, And, True, False, Or, And
    d) True, False, Not, And, False, True, Or, And
12. Write the push operation of stack containing names using class.
13. Write the pop operation of stack containing numbers using class.
14. Write the insertion operation of queue containing character using class.
15. Write the deletion operation of queue containing numbers using class.
16. Write any two example of stack operation.
17. Write any two example of pop operation.
18. Write an algorithm to evaluate postfix expression.
19. Write an algorithm to convert infix to postfix.
20. Write an algorithm to implement push operation.
21. Write an algorithm to implement pop operation.
22. Write an algorithm to implement insertion operation of queue.
23. Write an algorithm to implement deletion operation of queue.
24. Write a function to push any student's information to stack.
25. Write a function to add any customer's information to queue.