14 STACK (NCERT CLASS 12)

INTRODUCTION

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. Consider the following examples of stacks:

1. Ten glass plates placed one above another. (The plate that is kept last has to be taken out first)
2. The tennis balls in a container. (You cannot remove more than one ball at a time)
3. A pile of books
4. A stack of coins

In the above picture coins are kept one above the other and if any additional coin is to be added, it can beadded only on the top. If we want to remove any coin from the stack, the coin on the top of the stack has to be removed first. That means, the coin that was kept last in the stack has to be taken out first.

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. Let us first have a look at the logic to program the Push operation for a stack through the following algorithm:

1. Start
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 is as follows:

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 ATACK

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, we already have pop() and append() functions for popping and adding elements on to the stack.Hence, there is no need to write the code to add and remove elements in the stack. Consider the following programs that perform the Push and Pop operation on the stack through append() and pop().

Program to implement stack (without classes):-

```
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? ")
```

Output:

```
>>>
```

1. PUSH
2. POP
3. Display

Enter your choice: 2
Stack Empty
Do you want to continue or not? y

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: 1
Enter any number :300
Do you want to continue or not? y

1. PUSH
2. POP
3. Display

Enter your choice: 3
300
200
100
Do you want to continue or not? y

1. PUSH
2. POP
3. Display

Enter your choice: 2
Deleted element is : 300
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? n

```
>>>
```

The same program can also be implemented using classes as shown below:

Program to implement a stack(Using classes)

```
s=[]
def push(self):while (c=="y"):
    a=input("Enter any number :")
     stack.s.append(a)
def display(self):
l=len(stack.s)
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
>>>
```

EXPRESSION

An .expression is a combination of variables,constants and operators. Expressions can be written in Infix, Postfix or Prefix notations.

Infix Expression: In this type of notation, the operator is placed between the operands. For example: A+B, A*(B+C), X*Y/Z, etc.

Postfix Expression: In this type of notation, the operator is placed after the operands.

For example: AB+, ABC+*,XYZ/*, etc.

Prefix Expression: In this type of notation, the operator is placed before the operands.

For example: +AB,*A+BC,*X/YZ, etc.

- **Conversion of an infix expression to postfix expression:-**

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 become 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.

- **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

Infix, Prefix and Postfix Notations

Type of Expression	Description	Example
Infix	Operators are placed in between the operands	x * y + z 3 *(4 + 5) (x + y)/(z * 5)
Prefix (Polish)	Operators are placed before the corresponding operands	+z*xy *3+45 /+xy*z5
Postfix (Reverse Polish)	Operators are placed after the corresponding operands	xy*z+ 345+* xy+z5*/

15 QUEUE
(NCERT CLASS 12)

INTRODUCTION

A queue is a container of elements, which are inserted and removed according to the first-in first-out(FIFO) principle.

In a queue, persons who stand in the queue will carry out their work one by one. That means those who stands first in the queue will be allowed to carry out his work first and the person who stands at the second position will be allowed to carry out his work second only. At the same time those who come late will be joining the queue at the end. In simple terms it is called 'first come first out'.

Technically speaking a queue is a linear list, to keep an ordered collection of elements / objects. The principle operations, which can be performed on it are:-

(I) Addition of elements &

(II) Removal of elements.

Addition of element is known as INSERT operation, also known as enqueu-ing. It is done using rear terminal position, i.e. tail end. Removal of element is known as DELETE operation also know as dequeueing. It is done using front terminal position, i.e. head of the list. As the two operations in the queue are performed from different ends, we need to maintain both the access points. These access points are known as FRONT, REAR. FRONT is first element of the queue and REAR is last element. As queue is FIFO implementation, FRONT is used for delete operation and REAR is used for insert operation.

Following are some applications of queue in computers:

(I) In a single processor multi tasking computer, job(s) waiting to be processed form a queue. Same happens when we share a printer with many computers.

(II) Compiling a HLL code.

(III) Using down load manager, for multiple files also uses queue for ordering the files.

(IV) In multiuser OS - job scheduling is done through queue.

QUEUE OPERATIONS

Various operations, which can be performed on a queue are:

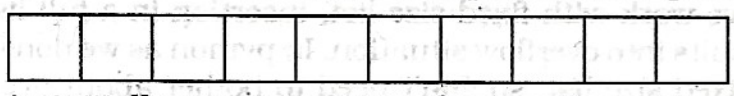
(I) Create a queue having a data structure to store linear list with ordering of elements.

(II) Insert an element will happen using REAR, REAR will be incremented to hold the new value in queue.

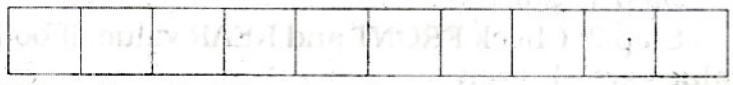
(III) Delete an element will happen using FRONT and FRONT will also be incremented to be able to accessnext element.

Let's understand this with the help of an example:

1. We will use list data type to implement the queue.



2. As initially queue is empty, front and rear should not be able to access any element. The situation can be represented by assigning -1 to both REAR and FRONT as initial value.



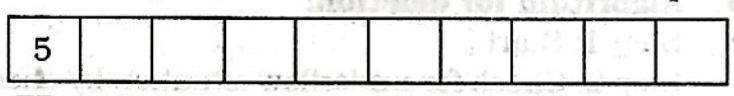
F(front) = -1, R(rear) = -1

Once the queue is created, we will perform various operations on it. Following is the list of operations with its affect on queue:

INSERT(5)

F = F + 1

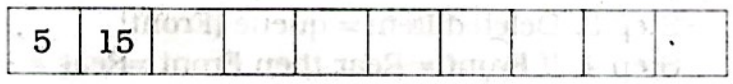
R = R + 1 (as this is the first element of the queue)



FR

INSERT(15)

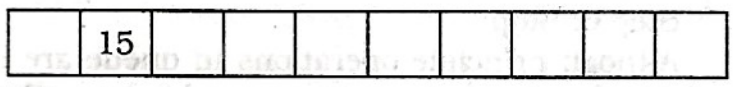
R = R + 1



F R

DELETE()

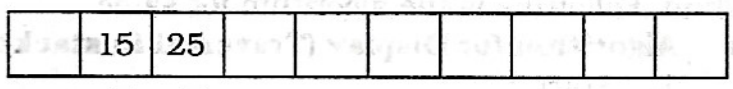
F = F + 1



FR

INSERT(25)

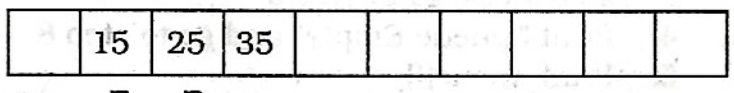
R = R + 1



F R

INSERT(35)

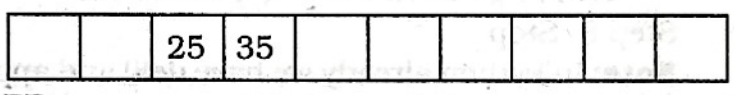
R = R + 1



F R

DELETE()

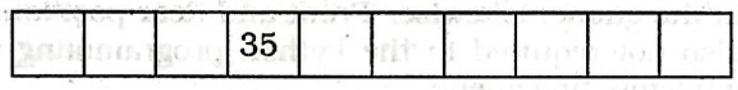
F = F + 1



FR

DELETE()

F + F + 1



FR

DELETE()

As the queue is empty, this is an exception to be handled. We can always say that deletion is attempted from an empty queue, hence not possible. The situation is known as underflow situation. Similarly when we work with fixed size list, insertion in a full list results into overflow situation. In python as we don'thave fixed size list, so don't need to bother about overflow situation. 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

Althogh 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. Following is the algorithm for same.

- **Algorithm for Display (Traversal in stack):**

1. Start
2. Store front value in I
3. Check I position value, if I value is -1 go to step 4 else go to step 5
4. Print "Queue Empty" and go to step 8
5. Print queue[I]
6. I is incremented by 1
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 andaddition 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(0,1):
print a[i]
else:
print("wrong input")
c=raw_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(0,1):
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=raw_input("do you want to continue or not
:")
```

□□□