Agenda

1 Stacks and Queues

1.1 Fundamentals of the Stacks

1.2 Operations on a Stack

1.3 Fundamentals of the Queues

1.4 Operations on a Queue

- Stacks are dynamic sets in which the element removed from the set by DELETE operation is prespecified.
- In a stack, the element deleted from the set is one most recently inserted: the stack implements a last in, first out, or **LIFO** policy.

- The INSERT operation on a stack is often called **PUSH**, and the DELETE operation, which does not take an element argument, is often called **POP**. These names are allusions to physical stacks, such as spring-loaded stacks of plates used in cafeterias.

Stacks
Fundamentals of the Stacks

- A stack can be implemented as a **constrained version of a linked list**.
- A stack is referenced **via a pointer to the top element of the stack.** The link member in **the last node of the stack is set to NULL** to indicate the bottom of the stack.
- Stack and linked lists are represented identically. The difference between stacks and linked lists is that insertions and deletions may occur **in anywhere in a linked list, but only at the top of a stack**.

Stacks

Operations on a Stack

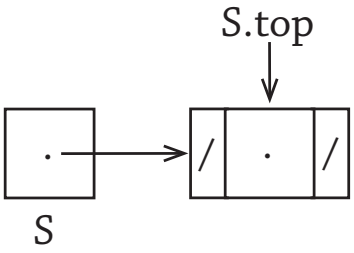
- The primary functions used to manipulate a stack are PUSH and POP. Function PUSH creates a new node and places it on top of the stack.
- Function POP removes a node from the top of the stack, frees the memory that was allocated to the popped node and return the popped value.

```
STACK-PUSH( S, x )
1  x.next = S.top
2  S.top = x
```

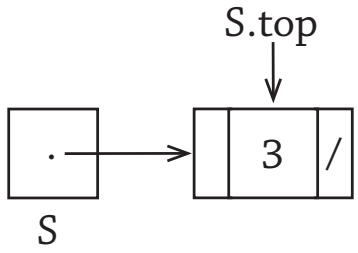
```
STACK-EMPTY( S )
1  if S.top = NIL
2     return "true"
3  else
4     return "false"
```

```
STACK-POP( S )
1  if STACK-EMPTY(S) ≠ "true"
2     x = S.top
3     key = x.key
4     S.top = S.top.next
5     free( x )
6     return key
```

- Following the execution of STACK-PUSH(L, x), where x.key = 3, the empty stack has a new object with key 25 as the new top. This new object points to the NULL. In the figure below, the new situation of the stack is illustrated.

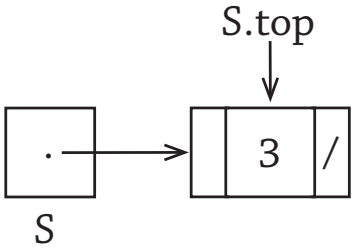


Stack is empty.

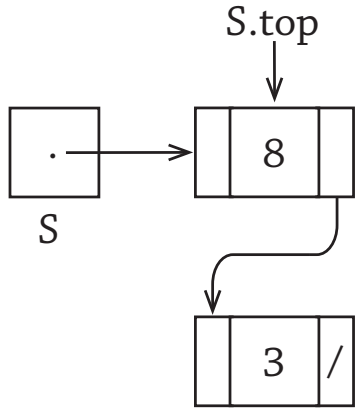


After pushing an object with the key 3.

- Following the execution of STACK-PUSH(L, x), where x.key = 8, the stack has a new object with key 8 as the new top. This new object points to the object with the key 3. In the figure below, the new situation of the stack is illustrated.

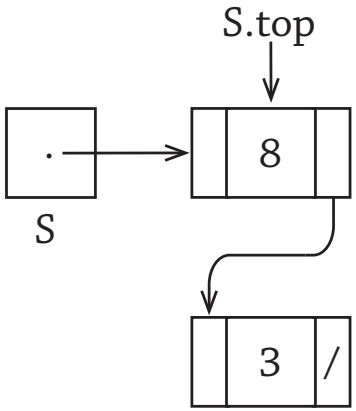


After pushing an object with the key 3.

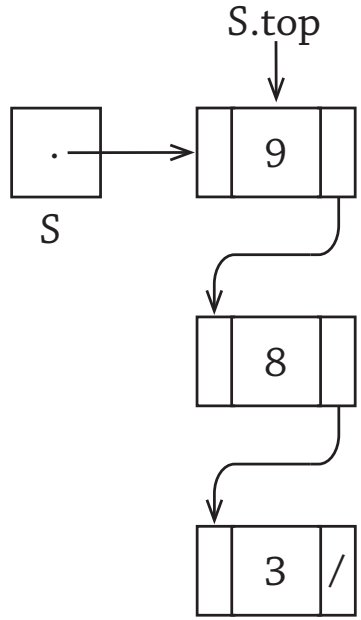


After pushing an object with the key 8.

- Following the execution of STACK-PUSH(S, x), where x.key = 9, the stack has a new object with key 9 as the new top. This new object points to the object with the key 8. In the figure below, the new situation of the stack is illustrated.

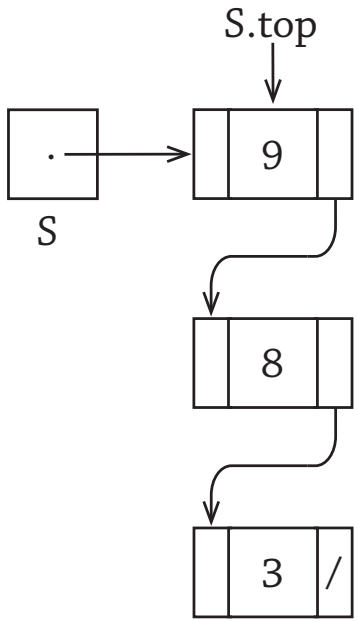


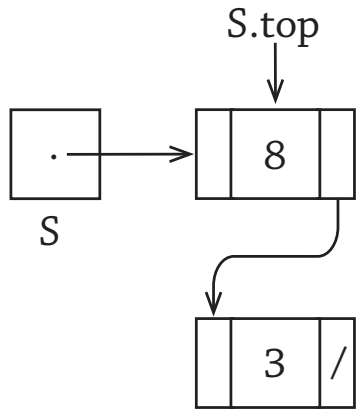
After pushing an object with the key 8.



After pushing an object with the key 9.

- Following the execution of STACK-POP(S), the object with the key 9 will be popped from the stack S. The new top of the stack S points to the object with the key 8. In the figure below, the new situation of the stack is illustrated.





After applying POP operation.

Queues

Fundamentals of the Queues

- In a queue, the element deleted is always the one that has been set for the longest time: the queue implements a first in-first out, or **FIFO** policy.
- The queue has a head and a tail. When an element is added, it takes its place at the tail of the queue, just as a newly arriving customer takes a place at the end of the line.
- The element deleted is always the one at the head of the queue, like the customer at the head of the line who has waited the longest.

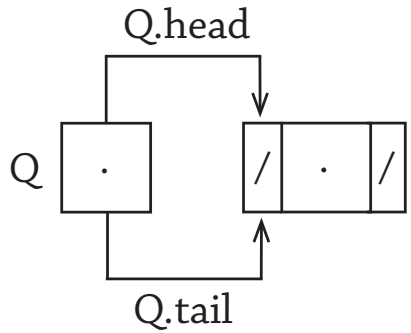
- We call the INSERT operation on a queue ENQUEUE, and we call the DELETE operation DEQUEUE; like the stack operation POP, DEQUEUE takes no element argument.

```
QUEUE-EMPTY( Q )
1  if Q.head = NIL
2    return "true"
3  else
4    return "false"
```

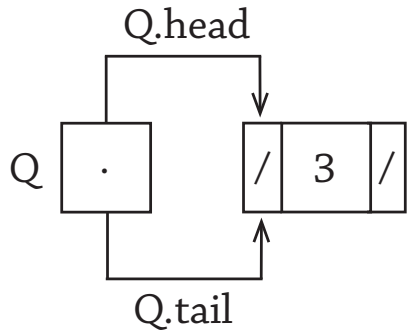
```
QUEUE-ENQUEUE( Q, x )
1  if QUEUE-EMPTY( Q )
2    x.next = NIL
3    Q.head = x
4    Q.tail = x
5  else
6    x.next = NIL
7    Q.tail.next = x
8    Q.tail = x
```

```
QUEUE-DEQUEUE( Q )
1  if QUEUE-EMPTY( Q ) ≠ "true"
2    x = Q.head
3    key = x.key
4    Q.head = x.next
5    free( x )
6    return key
```

- Following the execution of QUEUE-ENQUEUE(Q, x), where x.key = 3, the empty queue has a new object with key 3 as the new head and tail. This new object points to the NULL. In the figure below, the new situation of the stack is illustrated.

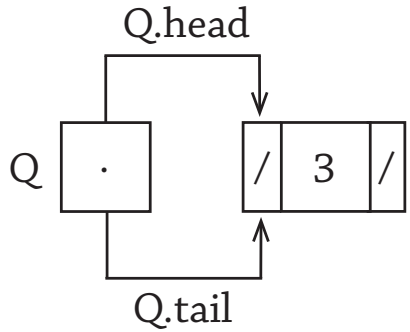


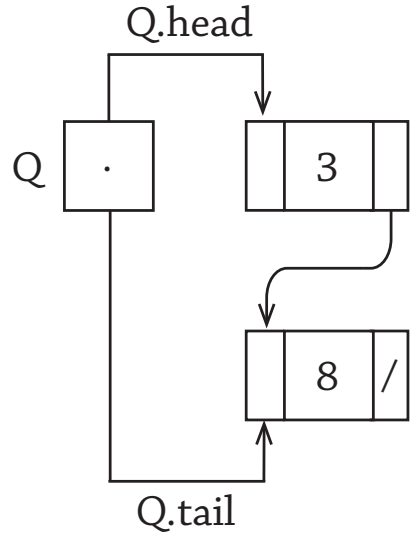
Queue is empty.



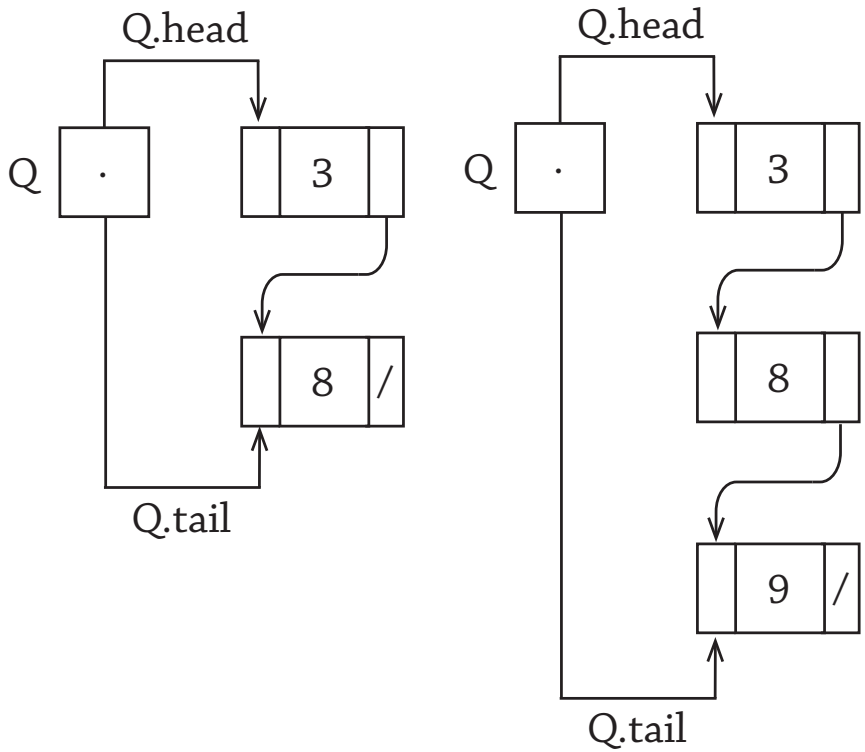
After enqueuing an object with the key 3.

- Following the execution of QUEUE-ENQUEUE(Q, x), where x.key = 8, the queue has a new object with key 8 as the new tail. This new object points to the NULL. In the figure below, the new situation of the stack is illustrated.





- Following the execution of QUEUE-ENQUEUE(Q, x), where x.key = 9, the queue has a new object with key 9 as the new head and tail. This new object points to the NULL. In the figure below, the new situation of the stack is illustrated.



- Following the execution of QUEUE-DEQUEUE(Q), the object with the key 3 will be dequeued from the Queue Q. The new head of the queue Q points to the object with the key 8. In the figure below, the new situation of the queue is illustrated.