Queues & Stacks

Prof Muliaro Wafula

Array Representation of Queues

- Luggage kept on conveyor belts. The bag which was placed first will be the first to come out at the other end.
- Cars lined at a toll bridge. The first car to reach the bridge will be the first to leave.

In all these examples, we see that the element at the first position is served first. Same is the case with queue data structure. A queue is a **FIFO (First-In, First-Out)** data structure in which the element that is inserted first is the first one to be taken out. The elements in a queue are added at one end called the REAR and removed from the other end called the FRONT. Queues can be implemented by using either arrays or linked lists. In this section, we will see how queues are implemented using each of these data structures.

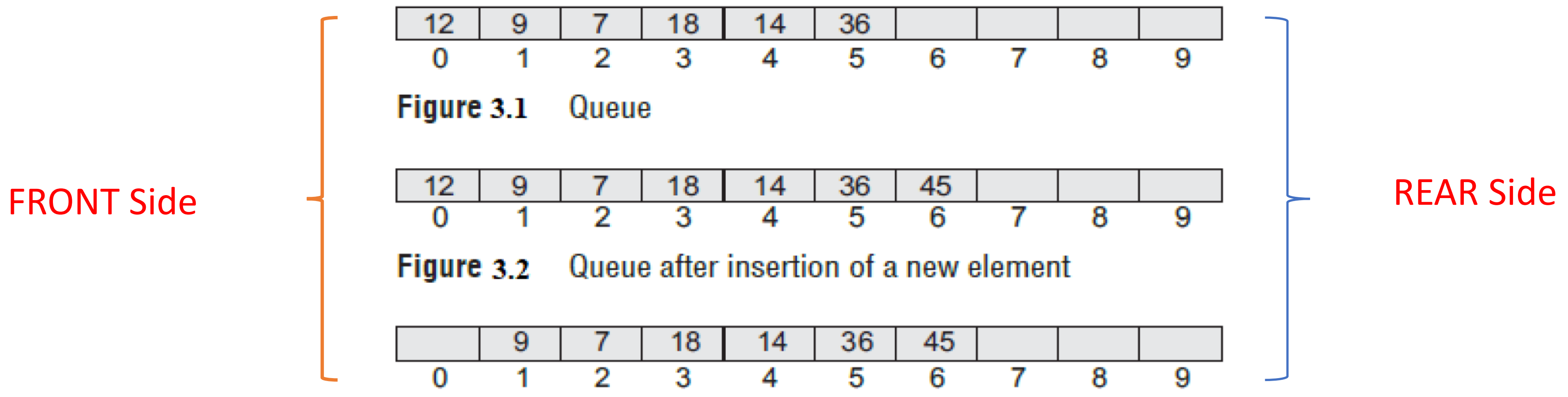


Figure 3.1 Queue

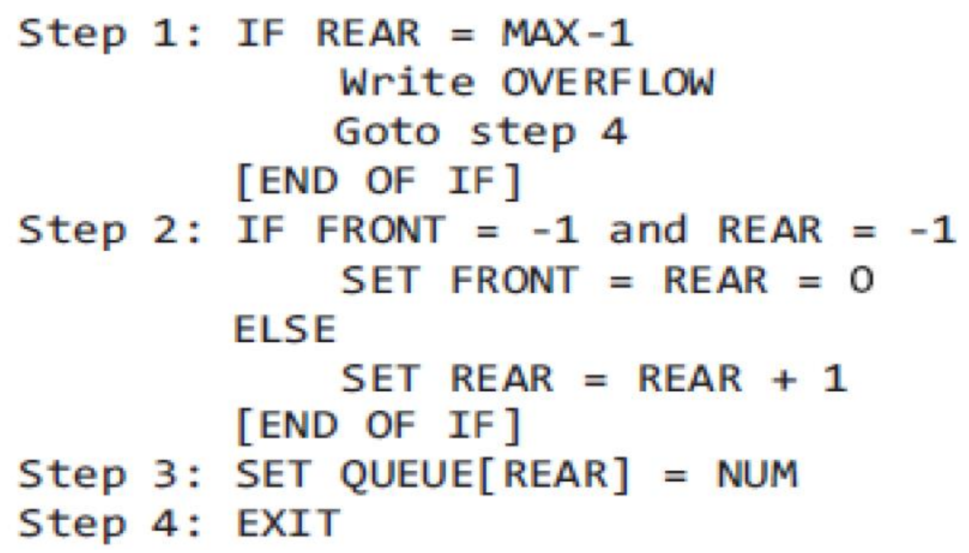
Figure 3.2 Queue after insertion of a new element

Figure 3.2 Queue after deletion of an element

Checking for Overflow OR Underflow conditions

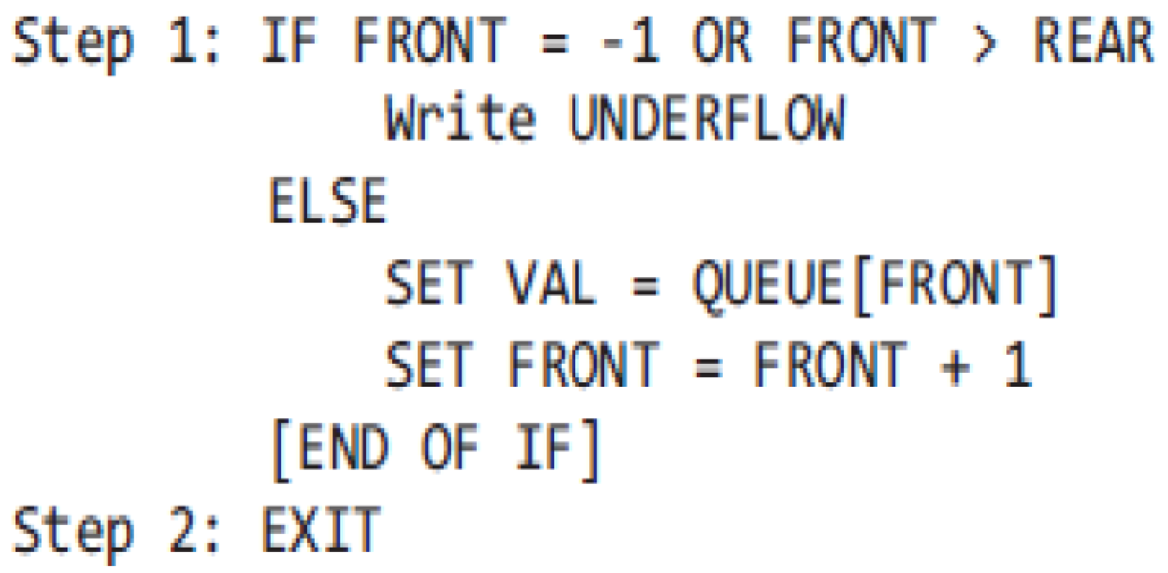
However, before inserting an element in a queue, we must check for overflow conditions. An overflow will occur when we try to insert an element into a queue that is already full. When REAR = MAX – 1, where MAX is the size of the queue, we have an overflow condition. Note that we have written MAX – 1 because the index starts from 0. Similarly, before deleting an element from a queue, we must check for underflow conditions. An underflow condition occurs when we try to delete an element from a queue that is already empty. If FRONT = –1 and REAR = –1, it means there is no element in the queue.

Algorithm for enqueue



```
Step 1: IF REAR = MAX-1
             Write OVERFLOW
             Goto step 4
        [END OF IF]
Step 2: IF FRONT = -1 and REAR = -1
             SET FRONT = REAR = 0
        ELSE
             SET REAR = REAR + 1
        [END OF IF]
Step 3: SET QUEUE[REAR] = NUM
Step 4: EXIT
```

Algorithm for Dequeue



```
Step 1: IF FRONT = -1 OR FRONT > REAR
             Write UNDERFLOW
        ELSE
             SET VAL = QUEUE[FRONT]
             SET FRONT = FRONT + 1
        [END OF IF]
Step 2: EXIT
```

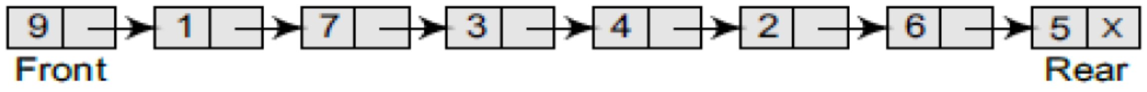


Figure 3.4 Linked queue

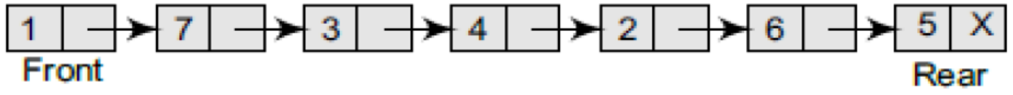


Figure 3.5 Linked queue

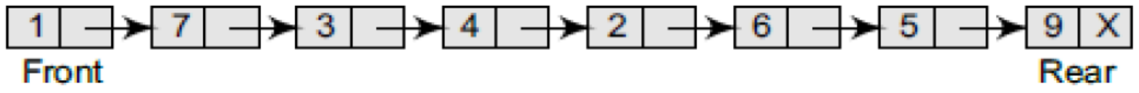


Figure 3.6 Linked queue after inserting a new node

```
Step 1: Allocate memory for the new node and name
        it as PTR
Step 2: SET PTR -> DATA = VAL
Step 3: IF FRONT = NULL
            SET FRONT = REAR = PTR
            SET FRONT -> NEXT = REAR -> NEXT = NULL
        ELSE
            SET REAR -> NEXT = PTR
            SET REAR = PTR
            SET REAR -> NEXT = NULL
        [END OF IF]
Step 4: END
```

Algorithm to insert an element in a linked queue

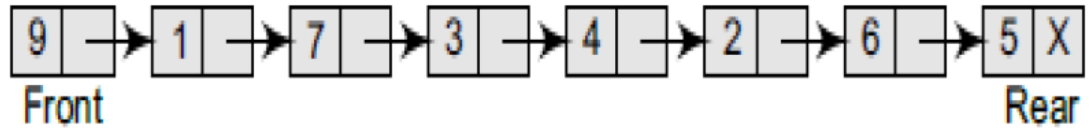


Figure 3.7 Linked queue

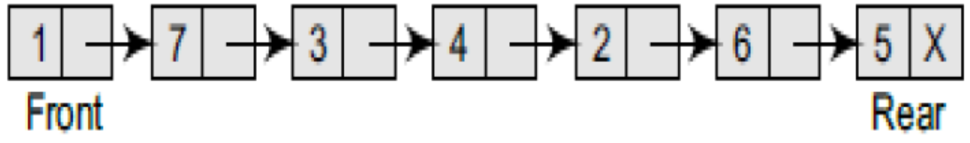


Figure 3.8 Linked queue after deletion of an element

```
Step 1: IF FRONT = NULL
            Write "Underflow"
            Go to Step 5
        [END OF IF]
Step 2: SET PTR = FRONT
Step 3: SET FRONT = FRONT -> NEXT
Step 4: FREE PTR
Step 5: END
```

Circular Queue

For insertion, we now have to check for the following three conditions:

- If front = 0 and rear = MAX – 1, then the circular queue is full. Look at the queue given in Fig. 3.12 which illustrates this point.
- If rear != MAX – 1, then rear will be incremented and the value will be inserted as illustrated in Fig. 3.13.
- If front != 0 and rear = MAX – 1, then it means that the queue is not full. So, set rear = 0 and insert the new element there, as shown in Fig. 3.14.

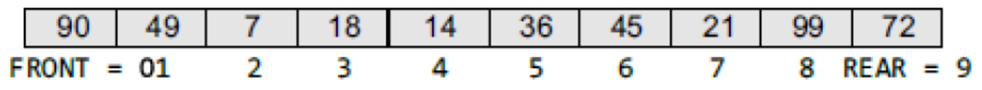


Figure 3.12 Full queue

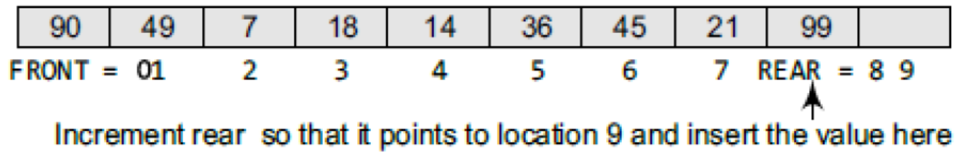

90 49 7 18 14 36 45 21 99
FRONT = 01 2 3 4 5 6 7 REAR = 8 9
Increment rear so that it points to location 9 and insert the value here

Figure 3.13 Queue with vacant locations

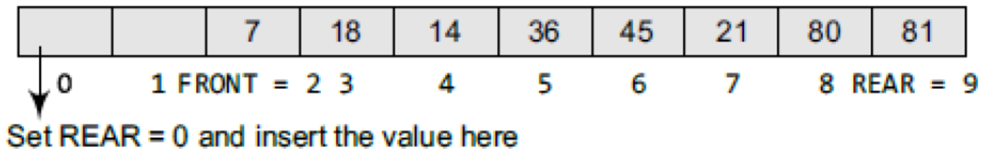

7 18 14 36 45 21 80 81
0 1 FRONT = 2 3 4 5 6 7 8 REAR = 9
Set REAR = 0 and insert the value here

Figure 3.14 Inserting an element in a circular queue

```
Step 1: IF FRONT = 0 and Rear = MAX - 1
              Write "OVERFLOW"
              Goto step 4
          [End OF IF]
Step 2: IF FRONT = -1 and REAR = -1
              SET FRONT = REAR = 0
          ELSE IF REAR = MAX - 1 and FRONT != 0
              SET REAR = 0
          ELSE
              SET REAR = REAR + 1
          [END OF IF]
Step 3: SET QUEUE[REAR] = VAL
Step 4: EXIT
```

Dequeue

A deque (pronounced as 'deck' or 'dequeue') is a list in which the elements can be inserted or deleted at either end. It is also known as a head-tail linked list because elements can be added to or removed from either the front (head) or the back (tail) end.

However, no element can be added and deleted from the middle. In the computer's memory, a deque is implemented using either a circular array or a circular doubly linked list.

In a deque, two pointers are maintained, LEFT and RIGHT, which point to either end of the deque. The elements in a deque extend from the LEFT end to the RIGHT end and since it is circular, Dequeue[N–1] is followed by Dequeue[0]. Consider the deques shown in Fig. 3.18.

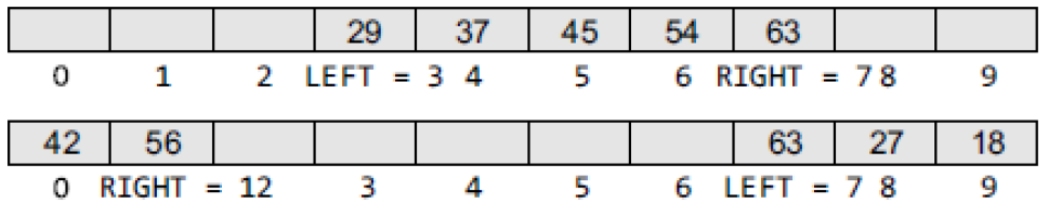


Figure 3.18 Double-ended queues

Priority Queues

A priority queue is a data structure in which each element is assigned a priority. The priority of the element will be used to determine the order in which the elements will be processed. The general rules of processing the elements of a priority queue are

- An element with higher priority is processed before an element with a lower priority.
- Two elements with the same priority are processed on a first-come-first-served (FCFS) basis.

A priority queue can be thought of as a modified queue in which when an element has to be removed from the queue, the one with the highest-priority is retrieved first. The priority of the element can be set based on various factors. Priority queues are widely used in operating systems to execute the highest priority process first. The priority of the process may be set based on the CPU time it requires to get executed completely.

Stacks

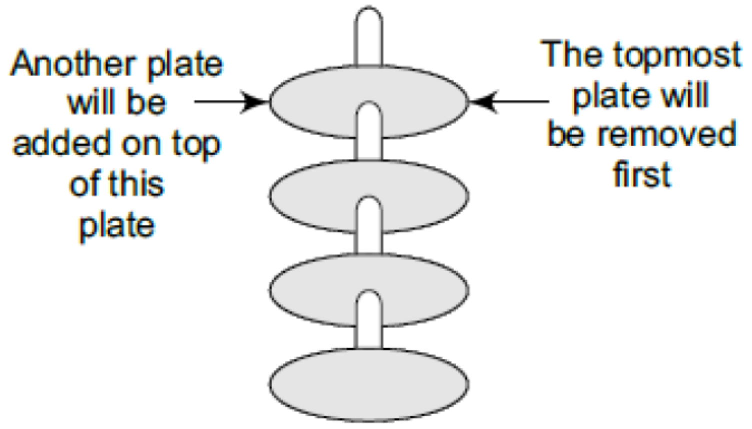

Another plate will be added on top of this plate
The topmost plate will be removed first

Figure 3.21 Stack of plates

A stack is a linear data structure which uses the same principle, i.e., the elements in a stack are added and removed only from one end, which is called the TOP.

Hence, a stack is called a LIFO (Last-In-First-Out) data structure, as the element that was inserted last is the first one to be taken out.

Array rep of Stack

In the computer's memory, stacks can be represented as a linear array. Every stack has a variable called TOP associated with it, which is used to store the address of the topmost element of the stack. It is this position where the element will be added to or deleted from.

There is another variable called MAX, which is used to store the maximum number of elements that the stack can hold. If TOP = NULL, then it indicates that the stack is empty and if TOP = MAX–1, then the stack is full. (You must be wondering why we have written MAX–1. It is because array indices start from 0.) Look at Fig. 3.22.

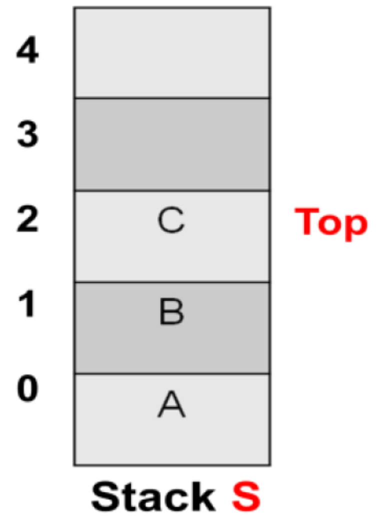
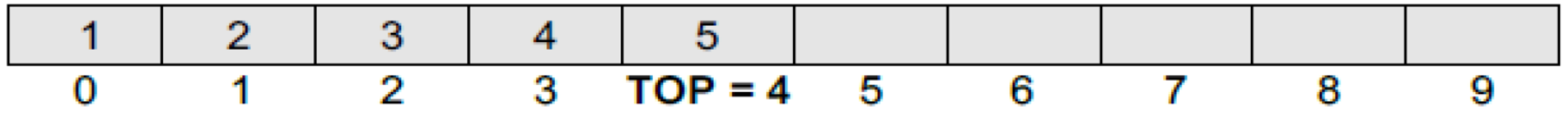


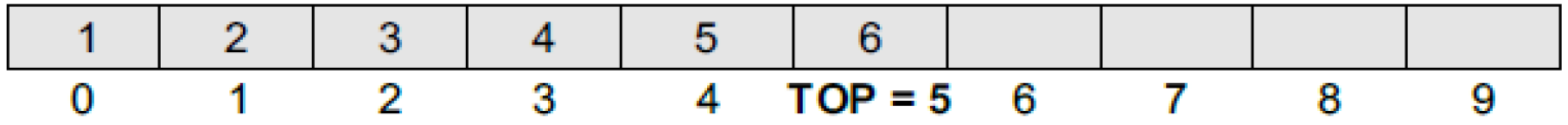
Fig. 3.22 Stack

Push Operation

- The push operation is used to insert an element into the stack.
- The new element is added at the topmost position of the stack.
- To insert an element with value 6, we first check if TOP=MAX–1.
- If the condition is false, then we increment the value of TOP and store the new element at the position given by stack[TOP].

The **push** operation adds an element to the top of the stack and the **pop** operation removes the element from the top of the stack. The **peek** operation returns the value of the topmost element of the stack.

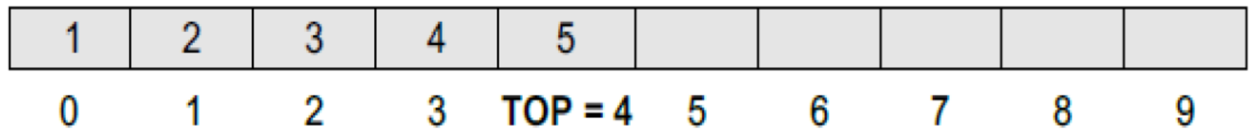


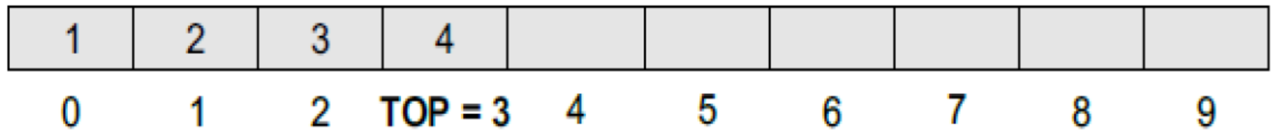


```
Step 1: IF TOP = MAX-1
            PRINT "OVERFLOW"
            Goto Step 4
        [END OF IF]
Step 2: SET TOP = TOP + 1
Step 3: SET STACK[TOP] = VALUE
Step 4: END
```

Pop Operation

- The pop operation is used to delete the topmost element from the stack.
- However, before deleting the value, we must first check if TOP=NULL because if that is the case, then it means the stack is empty and no more deletions can be done.
- To delete the topmost element, we first check if TOP=NULL. If the condition is false, then we decrement the value pointed by TOP.

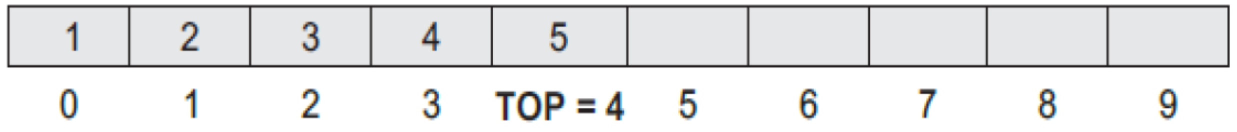




```
Step 1: IF TOP = NULL
            PRINT "UNDERFLOW"
            Goto Step 4
        [END OF IF]
Step 2: SET VAL = STACK[TOP]
Step 3: SET TOP = TOP - 1
Step 4: END
```

Peek Operation

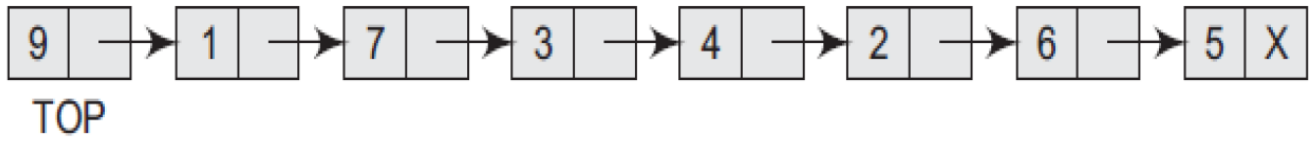
- Peek is an operation that returns the value of the topmost element of the stack without deleting it from the stack.
- However, the Peek operation first checks if the stack is empty, i.e., if TOP = NULL, then an appropriate message is printed, else the value is returned.
- Here, the Peek operation will return 5,as it is the value of the topmost element of the stack.



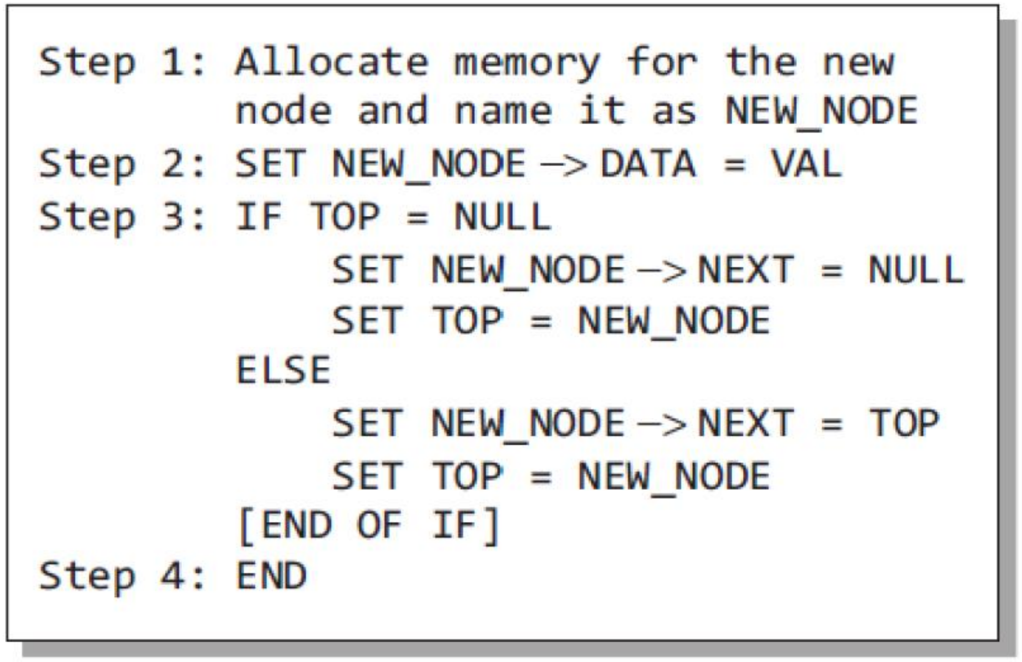
```
Step 1: IF TOP = NULL
             PRINT "STACK IS EMPTY"
             Goto Step 3
Step 2: RETURN STACK[TOP]
Step 3: END
```

Linked rep of Stack

In a linked stack, every node has two parts—one that stores data and another that stores the address of the next node. The START pointer of the linked list is used as TOP. All insertions and deletions are done at the node pointed by TOP. If TOP = NULL, then it indicates that the stack is empty. The linked representation of a stack is shown in below figure.

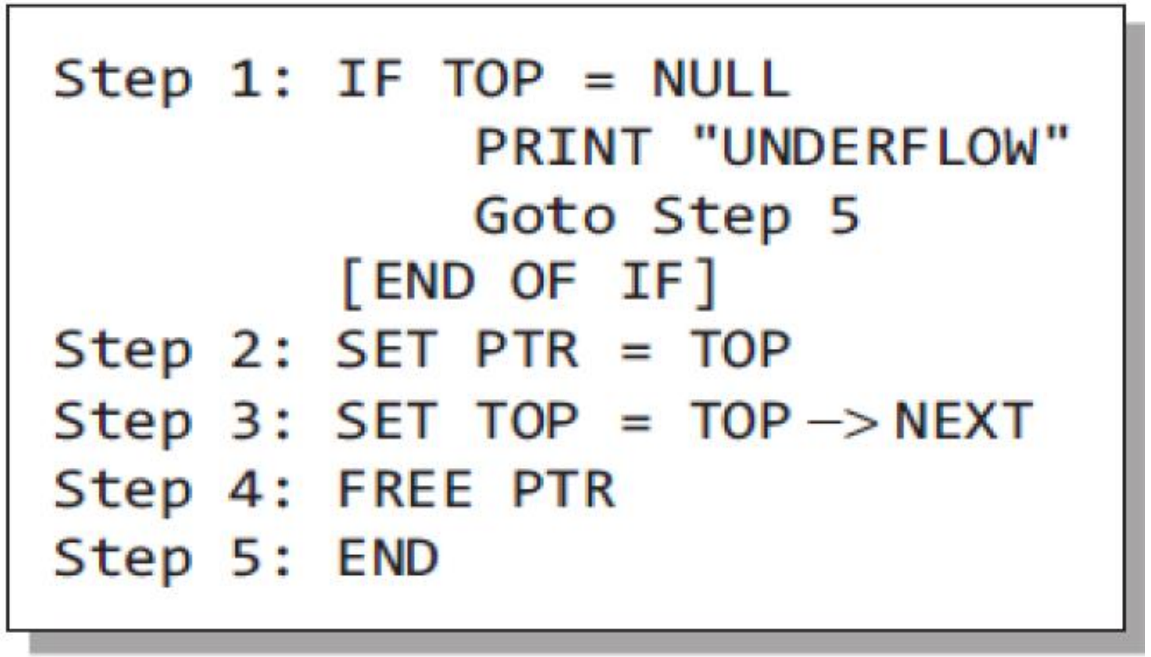


PUSH Operation



```
Step 1: Allocate memory for the new
        node and name it as NEW_NODE
Step 2: SET NEW_NODE -> DATA = VAL
Step 3: IF TOP = NULL
            SET NEW_NODE -> NEXT = NULL
            SET TOP = NEW_NODE
        ELSE
            SET NEW_NODE -> NEXT = TOP
            SET TOP = NEW_NODE
        [END OF IF]
Step 4: END
```

POP Operation



```
Step 1: IF TOP = NULL
              PRINT "UNDERFLOW"
              Goto Step 5
        [END OF IF]
Step 2: SET PTR = TOP
Step 3: SET TOP = TOP -> NEXT
Step 4: FREE PTR
Step 5: END
```