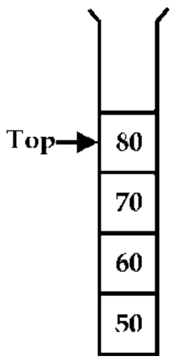
Dear Friends,
In the previous lectures on linked lists, we have discussed the concept of singular linked lists, doubly linked list, and circular linked list. Here, we are going to discuss the application of linked list. That is we are going to discuss the representation of stack and queue data structures using linked list. Also, we will discuss the insertion and deletion operations on stack and queue data structures.

Linked List Representation of Stack

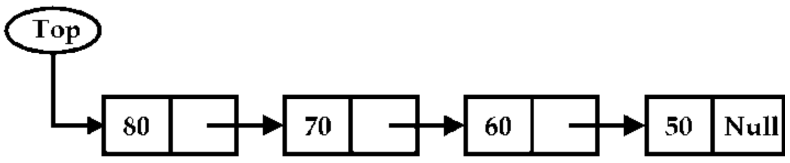
Let us proceed our discussion to representation of stack using linked list. As you all know, stack is a linear data structure which follows the property Last In First Out(LIFO). This means, the last inserted element will be the first to be removed. Because of this nature, stack has only one end indicated by variable Top. Top take care of the element present at the top of the stack. Because stack is the restricted kind of data structure, so only two operations can be applied to stack which are insertion, usually termed as Push and deletion, usually termed as Pop. No other operation like traversal, searching can be performed; otherwise, stack would lost its property of Last In First Out.
Two conditions are checked while push or pop operation on the linked list. One is overflow condition which means, if the stack is full i.e. there is no space in the stack and we want to push an element then that element cannot be inserted and we say overflow occurs. Similarly, if the stack is empty and we try to remove an element, then no element can be removed as stack is empty then we say underflow condition occurs.

We have already discussed the implementation of stack using Array. Arrays put up some limitations when used to implement stack. One of the limitations is the static nature of the array. This means, when array is used to implement stack, the maximum size of the stack must be defined before the execution of the program starts. This is the biggest disadvantage as there may be the overflow when at a moment, the number of elements becomes more than the space allocated to the stack. In case to eliminate or reduce the overflows, if large size of the array is declared, then there may be the wastage of memory because, it may be the case that at most of the time, the number of elements in the stack is very small.

By implementing the stack using the linked list, there is no need to know in advance about the maximum size of the stack. Linked list representation of the stack allows it to grow or shrink without any prior fixed limit. Of course, this is because of the dynamic nature of the linked list. Look at the diagram, Let us see, how a stack data structure can be represented using linked list.



Stack having 4 Elements



Linked List Representation of Elements of Stack shown in adjoining figure

This is the linked list representation of a stack having four elements. One-way linked list is used to represent the elements of the stack. The top most element is at the beginning of the list. Here, the top most element is 80. And the oldest element is at the end of the list. Here, the oldest element is 50. The first element of the linked list here is represented by a pointer variable Top.

Let us discuss, the push and pop operations on the stack using linked list.

Push Operation

As you all are aware of, push operation is the insertion of an element at the top of the stack. In case we are using linked list to represent the stack, the push operation can be performed by inserting an element at the beginning of the linked list. While pushing an element, overflow condition must be checked. Here, the overflow condition occurs when the stack is full and we try to insert an element into it. In case of linked list, the overflow condition will occur when the space required to insert an element is not available which rarely happens. Here is the algorithm depicting the push operation on stack where, the element Data is inserted into the stack using linked list. A stack pointer variable Top points to the top most element or node of the stack.

Algorithm: Push Operation - Insert a new element '$\mathbf{Data}$' into the stack represented by the linked list with a stack pointer variable '$\mathbf{Top}$' pointing to the topmost element of the stack.

```
Step 1:    Allocate memory to a node New from free memory pool.
Step 2:    Set $\mathbf{New} \rightarrow \mathbf{Info} = \mathbf{Data}$
Step 3:    Set $\mathbf{New} \rightarrow \mathbf{Next} = \mathbf{Top}$
Step 4:    Set $\mathbf{Top} = \mathbf{New}$
Step 5:    Exit
```

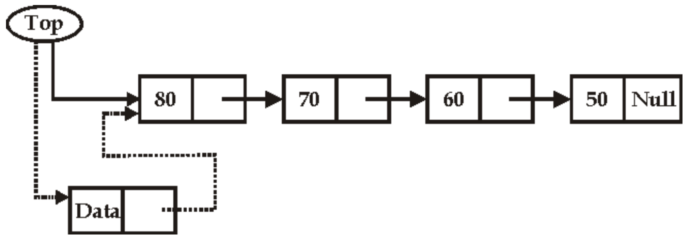


Illustration of Push Operation Performed on Stack

This algorithm inserts a new element 'Data' into the stack represented by the linked list with a stack pointer variable 'Top' pointing to the topmost element of the stack. The first step is to allocate memory to a node New. In step 2, the node is prepared for insertion. That is, the desired element is put into the Info part of this node. In step 3 and 4, the node New is inserted at the beginning of the linked list. That is the Next part of node New points to node 80 and the stack pointer variable Top points to newly inserted node New.

This algorithm inserts one element into the stack. To insert more than one element, algorithm is required to be modified.

Pop Operation

Let us discuss the pop operation on stack data structure. Pop operation is the deletion of an element from the stack. As the stack is Last In first Out data structure, so deletion of an element from the stack is restricted at one end only. So, the pop operation can be performed by deleting an element at the beginning of the linked list. While performing pop operation, underflow condition must be checked. Here, the underflow condition occurs when the stack is empty and we try to delete an element from the stack. Here is an algorithm depicting the pop operation on stack, where, the first element or node pointed by stack pointer variable Top is deleted.

Algorithm: Pop Operation - Delete an element from the stack represented by the linked list and returns the element '$\mathbf{Data}$' which is at the top of the stack.

```
Step 1:    If $\mathbf{Top} = \mathbf{Null}$ Then
              Print "Stack is Empty, Underflow Condition"
                    Exit
           [End If]
Step 2:    Set $\mathbf{Data} = \mathbf{Top} \rightarrow \mathbf{Info}$
Step 3:    Set $\mathbf{Temp} = \mathbf{Top}$
Step 4:    Set $\mathbf{Top} = \mathbf{Top} \rightarrow \mathbf{Next}$
Step 5:    Deallocate the deleted node $\mathbf{Temp}$ to free storage list for its use by
other processes.
Step 6:    Exit
```

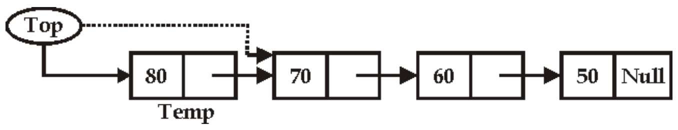


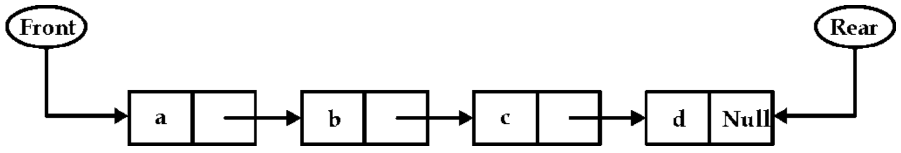
Illustration of Pop Operation Performed on a Stack

The step 1 of the algorithm checks the underflow condition. That means, if the stack pointer variable Top is Null, it means stack is empty and there is no node present. So, the underflow condition occurs and we exit from the program. In the second step, element in the first node is stored in a variable 'Data' for it future usability. Here, the element 80 is stored in variable Data. In step 3, the address of first node is stored in a variable Temp. This is because, at the end, the memory held by this node will be deallocated. In Step 4, the stack pointer variable Top is set to the address of second node. Here, the stack pointer variable Top will point to node 70 as shown here. In step 6, the node 80 whose address is in variable Temp is deallocated and memory held by this node is freed.

Linked List Representation of Queue

Let us proceed our discussion to representation of queue using linked list. Queue is a linear data structure which follows the property, First In first Out(FIFO) or First Come First Serve(FCFS). This means, the First inserted element will be the first to be removed. Queue has two ends, one is Front and other is Rear. Front variable indicates the oldest element inserted into the queue and Rear variable indicates the last element inserted into the queue. Because queue is the restricted kind of data structure, so only two operations can be applied to queue which are insertion, and deletion. Insertion operation takes place at Rear end and deletion operation takes place at Front end.
Two conditions are checked while insertion and deletion operation on the queue. One is overflow condition which means, if the queue is full i.e. there is no space in the queue and we want to insert an element then that element cannot be inserted and we say overflow occurs. Similarly, if the queue is empty and we want to remove an element, then no element can be removed as queue is empty then we say underflow condition occurs.

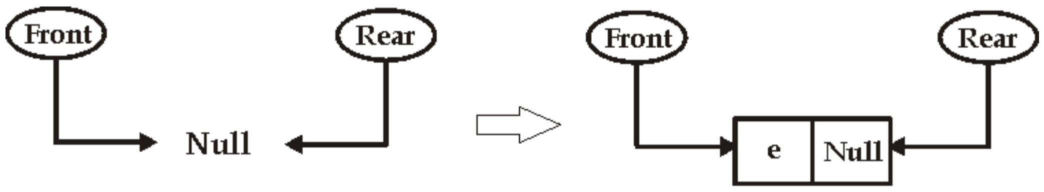
By implementing the queue using the linked list, there is no need to know in advance about the maximum size of the queue. Linked list representation of the queue allows it to grow or shrink without any prior fixed limit. Of course, this is because of the dynamic nature of the linked list. Look at the diagram, Let us see, how a queue data structure can be represented using linked list.



A Queue Maintained using a Linked List

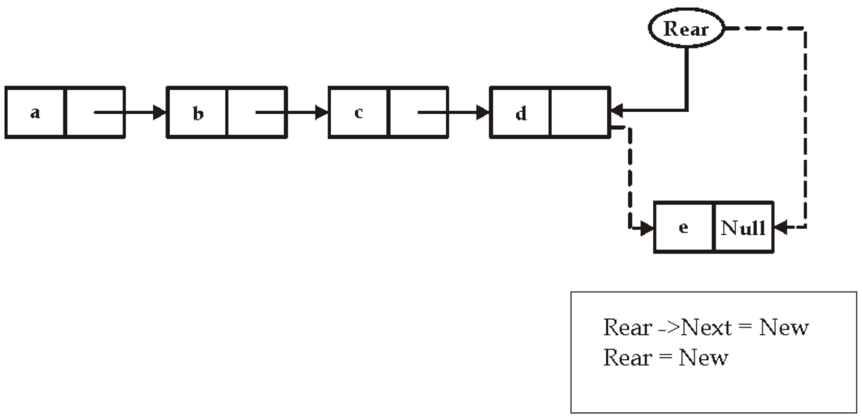
In this linked list representation of queue, there are four nodes. A one-way linked list is used to represent the elements of the queue but with a small difference. Here, in case of queue, the address of the last node is also stored in a pointer variable known as Rear. As shown in this figure, the Front variable holds the address of node ‘a’ which is the first node of the queue. The pointer variable Rear holds the address of node ‘d’ which is the last node of the queue. Insertion of new node will take place at the rear end i.e. after node ‘d’. Only node ‘a’ can be deleted from this queue as node ‘a’ is at the front of the queue.

Let us discuss the insertion of a new element in the queue. As discussed, the insertion of new element takes place at the rear end of the queue. Before insertion of new element, the overflow condition must be checked. i.e. if queue is already full and we want to insert a new element then it will be an overflow condition as it is not possible to insert the new node in a full queue. Below is the diagram along with algorithm for insertion of an element ‘e’ into the queue. The element ‘e’ is inserted at the Rear end of the list.



```
Rear  = New
Front = New
```

This insertion of a New element ‘*e*’ in the empty queue



This insertion of a New element 'e' in the queue

Algorithm: Insert a given element '$\mathbf{Data}$' in a queue which is implemented using a linked list '$\mathbf{Q}$' having variable $\mathbf{Front}$ which contains the address of 1st element of the queue and variable $\mathbf{Rear}$ which contains the address of last element of the queue.

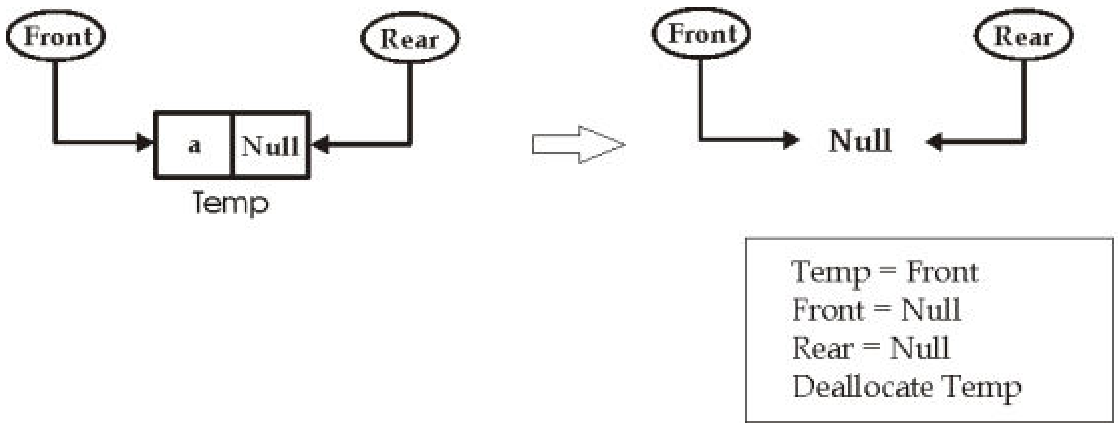
```
Step 1:     Allocate memory to node $\mathbf{New}$
Step 2:     Set $\mathbf{New} \rightarrow \mathbf{Info} = \mathbf{Data}$ And $\mathbf{New} \rightarrow \mathbf{Next} = \mathbf{Null}$
Step 3:     If $\mathbf{Rear} = \mathbf{Null}$ Then
                Set $\mathbf{Front} = \mathbf{New}$ And $\mathbf{Rear} = \mathbf{New}$
            Else
                Set $\mathbf{Rear} \rightarrow \mathbf{Next} = \mathbf{New}$ And $\mathbf{Rear} = \mathbf{New}$
            [End If]
Step 4:     Exit
```

Here, in this algorithm, first of all, memory is allocated to a node New. This node is prepared for insertion into the queue. As this node is to be inserted at the rear end, so this node is going to be the last node. Therefore, the Next part of node New is Null and the Info part of the node New will be 'e' as element to be inserted is 'e'. Two cases arise while inserting this New node. First, if the linked list is empty, in this case both the Front and Rear variables will be Null. So, after insertion of node New, both the Front and Rear variables will be set to address of node New as shown here in the diagram.

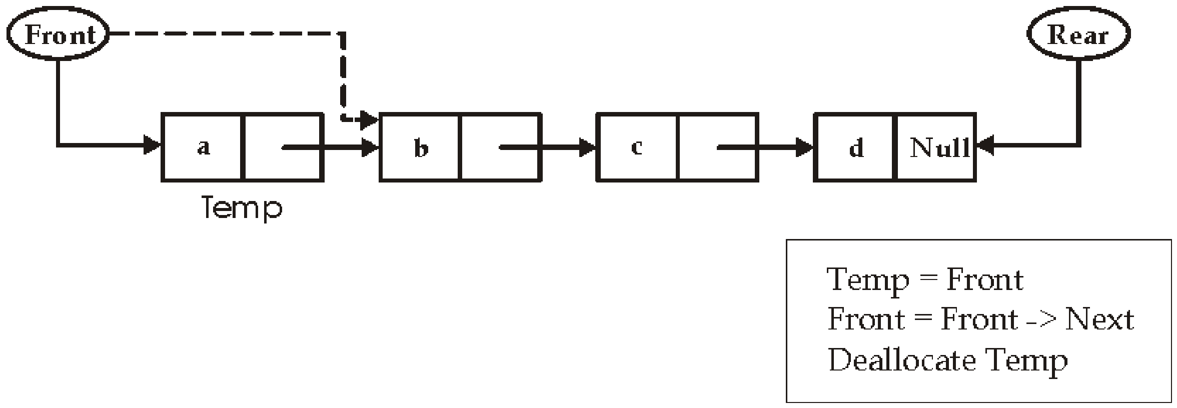
Secondly, if the linked list is not empty then node New will be inserted at the rear end by setting the Next part of last node to address of node New. Here, note that the address of last node is directly available in variable Rear and therefore, we need not to traverse the linked list to reach at the last node. After the node New is inserted, the variable Rear is set to the address of newly inserted node New as shown in the diagram here.

Deletion

Let us discuss the deletion operation on queue data structure. As the queue is First In First Out data structure, so deletion of an element from the queue is restricted at the Front end only. This means, the deletion of an element from the queue will be performed at the beginning of the linked list. While deleting an element from the queue, underflow condition must be checked. Here, the underflow condition occurs when the queue is empty and we try to delete an element from the queue. Here is the algorithm depicting the deletion operation on queue. The element at the front end of the queue is deleted.



Deletion of an element from the Queue having only one node



Deletion of an element from the Queue

Algorithm: Removes an element from a queue which is maintained using a linked list ‘Q’ having variable 'Front' which contains the address of 1st element of the queue and variable 'Rear' which contains the address of last element of the queue.

```
Step 1:    If $\mathbf{Front} = \mathbf{Null}$ Then
               Print “Queue is Empty”
               Exit
           [End If]
Step 2:    Set $\mathbf{Data} = \mathbf{Front} \rightarrow \mathbf{Info}$
Step 3:    Set $\mathbf{Temp} = \mathbf{Front}$
Step 4:    If $\mathbf{Front} = \mathbf{Rear}$ Then
               Set $\mathbf{Front} = \mathbf{Null}$ And $\mathbf{Rear} = \mathbf{Null}$
           Else
               Set $\mathbf{Front} = \mathbf{Front} \rightarrow \mathbf{Next}$
           [End If]
Step 5:    Deallocate memory taken by node Temp
Step 6:    Exit
```

The step 1 of the algorithm checks the underflow condition. That means, if the pointer variable Front is Null, i.e. if the queue is empty then underflow condition occurs and we exit from the program. In the second step, element in the first node is stored in a variable ‘Data’ for it future usability. Here, the element of first node which is ‘a’ is stored in variable Data. In step 3, the address of first node is stored in a variable Temp. This is because, at the end, the memory held by this node will be deallocated. At this stage two conditions occur. First, if the queue has only one node, then after deletion of this node, the queue will become empty. So, the variables Front and Rear becomes Null as shown in the figure. Secondly, if the queue has more than one elements then the second node of the queue will become the first node. Now, Front will point to the second node. At the end, the deleted node Temp is deallocated and memory held by variable Temp is freed.

To summarize, we have discussed the the linked list representation of the stack and queue data structure. The two operations insertion and deletion possible on stack and queue data structures using linked list are also discussed.
Thank You..