CHAPTER 3:
LINKED LIST

Content

- Introduction, Representation of Linked List
- Linked List v/s Array
- Types of Linked List - Singly Linked List
- Circular Linked List, Doubly Linked List
- Operations on Singly Linked List and Doubly Linked List
- Stack and Queue using Singly Linked List
- Singly Linked List Application-Polynomial Representation and Addition

Somaiya TRUST

Introduction to Linked List

📌 What is a Linked List?

A **Linked List** is a **linear data structure** where each element (called a **node**) contains:

1. **Data** – the actual value
2. **Pointer (link)** – a reference to the **next node** in the sequence

Unlike arrays, linked lists do **not store elements in contiguous memory**. Instead, they are connected through pointers.

📦 Structure of a Node in C:

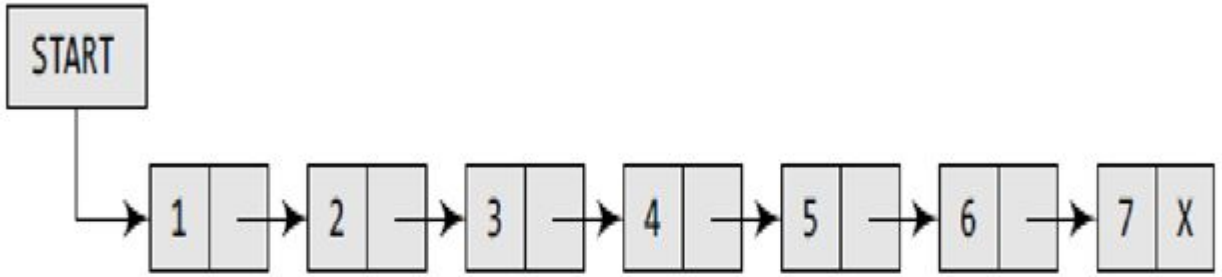
c Copy Edit

```c
struct Node {
    int data;
    struct Node* next;
};
```

TRUST

Representation of Linked List

Basic terminologies

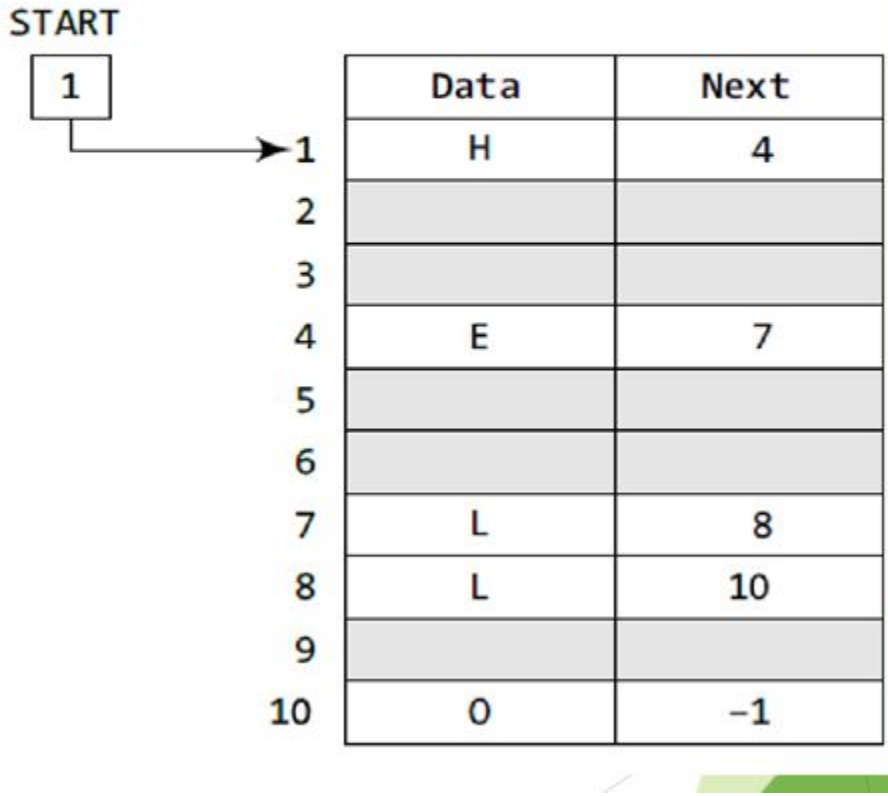


- A linked list, in simple terms, is a linear collection of data elements. These data elements are called *nodes*.
- Linked lists contain a pointer variable START that stores the address of the first node in the list.
- We can traverse the entire list using START which contains the address of the first node; the next part of the first node in turn stores the address of its succeeding node.
- Using this technique, the individual nodes of the list will form a chain of nodes. If START = NULL, then the linked list is empty and contains no nodes.

Representation of Linked List

Examples

- shows a chunk of memory locations which range from 1 to 10.
- The shaded portion contains data for other applications.
- Remember that the nodes of a linked list need not be in consecutive memory locations.

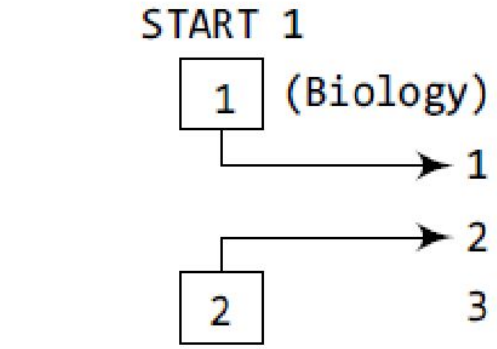


Somaiya TRUST

Representation of Linked List

- we can conclude that
- roll numbers of the students who have opted for
- Biology **are S01, S03, S06, S08, S10, and S11.**

- Similarly, roll numbers of the students who chose
- Computer Science are **S02, S04, S05, S07, and**
- **S09.**



	Roll No	Next
1	S01	3
2	S02	5
3	S03	8
4		
5	S04	7
6		
7	S05	10
8	S06	11
9		
10	S07	12
11	S08	13
12	S09	-1
13	S10	15
14		
15	S11	-1

Linked Lists versus Arrays

- Both arrays and linked lists are a linear collection of data elements.
- But unlike an array, a linked list does not store its nodes in consecutive memory locations.
- linked list does not allow random access of data. Nodes in a linked list can be accessed only in a sequential manner. But like an array, insertions and deletions can be done at any point in the list in a constant time.
- Another advantage of a linked list over an array is that we can add any number of elements in the list. This is not possible in case of an array.
- For example, if we declare an array as int marks[20], then the array can store a maximum of 20 data elements only. There is no such restriction in case of a linked list.

Somaiya TRUST

Memory Allocation and De-allocation for a Linked List

- linked list is represented in the memory.
- If we want to add a node to an already existing linked list in the memory, we first find free space in the memory and then use it to store the information.

START = 1 (Biology)

	Roll No	Marks	Next
1	S01	78	2
2	S02	84	3
3	S03	45	5
4			
5	S04	98	7
6			
7	S05	55	8
8	S06	34	10
9			
10	S07	90	11
11	S08	87	12
12	S09	86	13
13	S10	67	15
14			
15	S11	56	-1

(a)

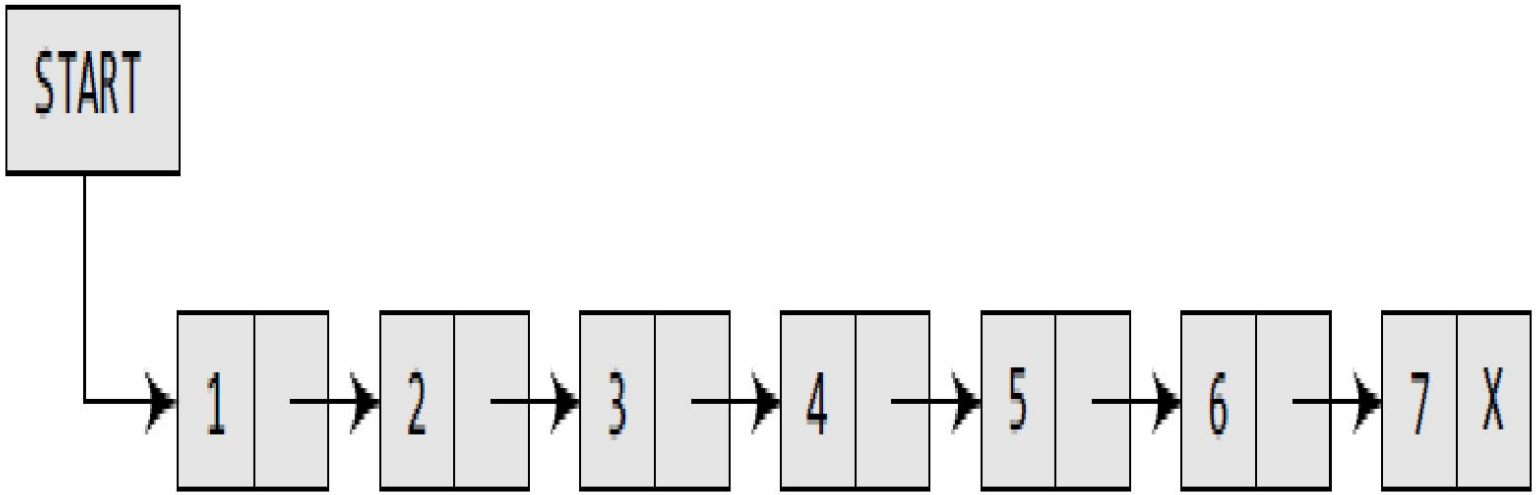
START = 1 (Biology)

	Roll No	Marks	Next
1	S01	78	2
2	S02	84	3
3	S03	45	5
4	S12	75	-1
5	S04	98	7
6			
7	S05	55	8
8	S06	34	10
9			
10	S07	90	11
11	S08	87	12
12	S09	86	13
13	S10	67	15
14			
15	S11	56	4

(b)

SINGLY LINKED LIST

- A singly linked list is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type.
- By saying that the node contains a pointer to the next node, we mean that the node stores the address of the next node in the sequence\
- . A singly linked list allows traversal of data only in one way.



Soma
TRUST

Traversing a Linked List

- Traversing a linked list means accessing the nodes of the list in order to perform some processing on them.
- Remember a linked list always contains a pointer variable START which stores the address of the first node of the list.
- End of the list is marked by storing NULL or –1 in the NEXT field of the last node.
- For traversing the linked list, we also make use of another pointer variable PTR which points to the node that is currently being accessed.

```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Steps 3 and 4 while PTR != NULL
Step 3:              Apply Process to PTR->DATA
Step 4:              SET PTR = PTR->NEXT
        [END OF LOOP]
Step 5: EXIT
```

Count no of nodes in LL

```
Step 1: [INITIALIZE] SET COUNT = 0
Step 2: [INITIALIZE] SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR != NULL
Step 4:              SET COUNT = COUNT + 1
Step 5:              SET PTR = PTR->NEXT
        [END OF LOOP]
Step 6: Write COUNT
Step 7: EXIT
```

Somaiya TRUST

Searching for a Value in a Linked List

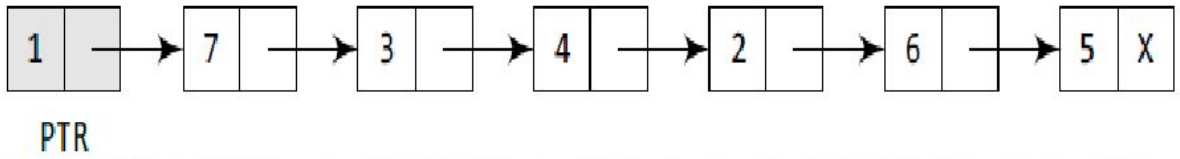
- Searching a linked list means to find a particular element in the linked list.
- In Step 1, we initialize the pointer variable PTR with
- START that contains the address of the first node.
- In Step 2, a while loop is executed which will compare every node's DATA with VAL for which the search is being made.
- If the search is successful, that is, VAL has been found, then the address of that node is stored in POS and the control jumps to the last statement of the algorithm.
- However, if the search is unsuccessful, POS is set to NULL which indicates that VAL is not present in thelinked list.

```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Step 3 while PTR != NULL
Step 3:      IF VAL = PTR -> DATA
                    SET POS = PTR
                    Go To Step 5
             ELSE
                    SET PTR = PTR -> NEXT
             [END OF IF]
        [END OF LOOP]
Step 4: SET POS = NULL
Step 5: EXIT
```

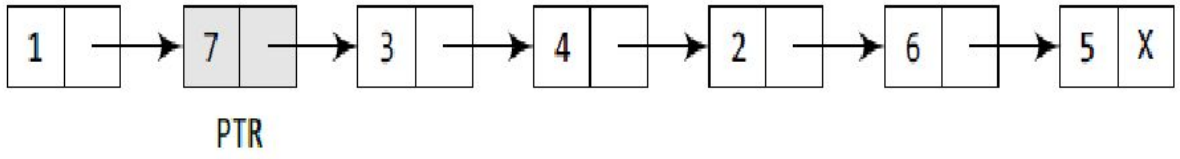
Somaiya TRUST

Searching for a Value in a Linked List

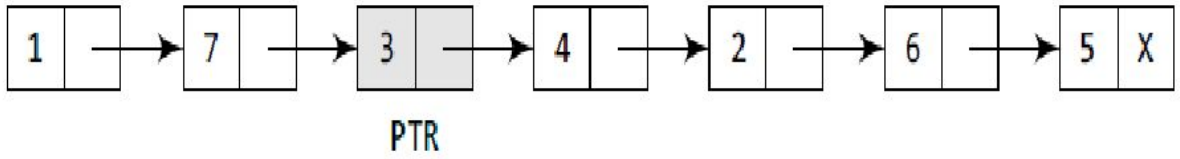
Consider the linked list shown in Fig. 6.11. If we have VAL = 4, then the flow of the algorithm can be explained as shown in the figure.



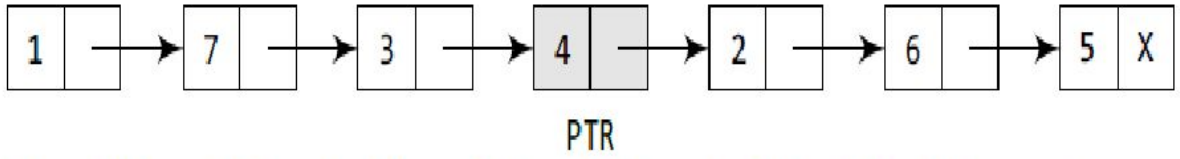
Here PTR -> DATA = 1. Since PTR -> DATA != 4, we move to the next node.



Here PTR -> DATA = 7. Since PTR -> DATA != 4, we move to the next node.



Here PTR -> DATA = 3. Since PTR -> DATA != 4, we move to the next node.



Here PTR -> DATA = 4. Since PTR -> DATA = 4, POS = PTR. POS now stores the address of the node that contains VAL

```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Step 3 while PTR != NULL
Step 3:     IF VAL = PTR->DATA
                SET POS = PTR
                Go To Step 5
            ELSE
                SET PTR = PTR->NEXT
            [END OF IF]
        [END OF LOOP]
Step 4: SET POS = NULL
Step 5: EXIT
```

Inserting a New Node in a Linked List

- Case 1: The new node is inserted at the beginning.
- Case 2: The new node is inserted at the end.
- Case 3: The new node is inserted before a given node.
- Overflow is a condition that occurs when AVAIL = NULL or no free memory cell is present in the system.
- When this condition occurs, the program must give an appropriate message.

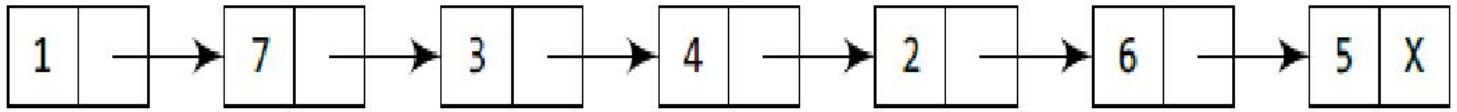
Inserting a Node at the Beginning of a Linked List

- In Step 1, we first check whether memory is available for the new node.
- If the free memory has exhausted, then an OVERFLOW message is printed.
- Otherwise, if a free memory cell is available, then we allocate space for the new node.
- Set its DATA part with the given VAL **and the next part is initialized with the address of the first node of the list, which is stored in START.**
- Now, since the new node is added as the first node of the list, it will now be known as the START node, that is, the START pointer variable will now hold the address of the NEW_NODE

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 7
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET START = NEW_NODE
Step 7: EXIT
```

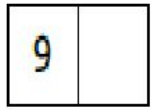
Somaiya TRUST

Inserting a Node at the Beginning of a Linked List

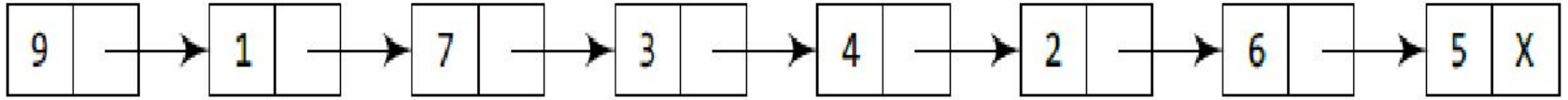


START

Allocate memory for the new node and initialize its DATA part to 9.

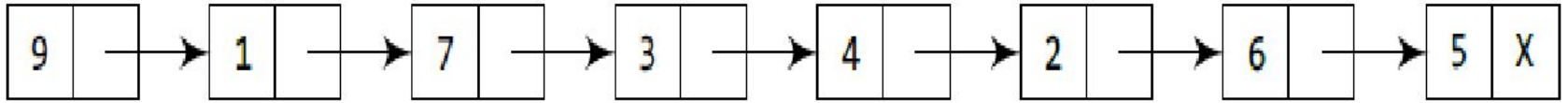


Add the new node as the first node of the list by making the NEXT part of the new node contain the address of START.



START

Now make START to point to the first node of the list.



START

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 7
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET START = NEW_NODE
Step 7: EXIT
```

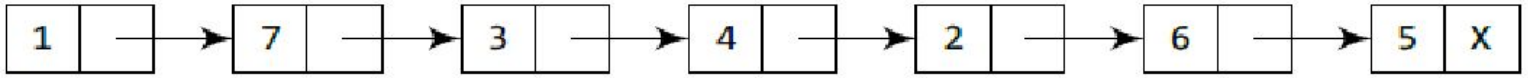
Inserting a Node at the End of a Linked List

- In Step 6, we take a pointer variable PTR and initialize it with START.
- That is, PTR now points to the first node of the linked list. In the while loop, we traverse through the linked list to reach the last node.
- Once we reach the last node, in Step 9, we change the NEXT pointer of the last node to store the address of the new node.
- Remember that the NEXT field of the new node contains NULL, which signifies the end of the linked list.

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET NEW_NODE->DATA = VAL
Step 5: SET NEW_NODE->NEXT = NULL
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR->NEXT != NULL
Step 8:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 9: SET PTR->NEXT = NEW_NODE
Step 10: EXIT
```

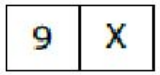
Somaiya TRUST

Inserting a Node at the End of a Linked List

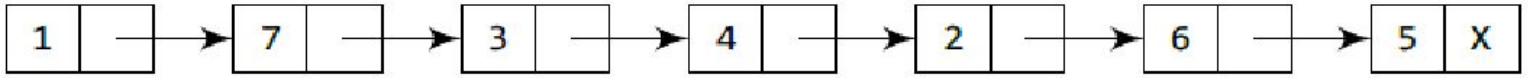


START

Allocate memory for the new node and initialize its DATA part to 9 and NEXT part to NULL.



Take a pointer variable PTR which points to START.



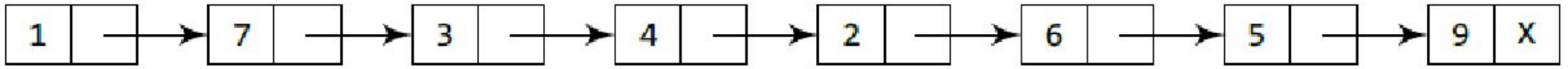
START, PTR

Move PTR so that it points to the last node of the list.

START

PTR

Add the new node after the node pointed by PTR. This is done by storing the address of the new node in the NEXT part of PTR.



START

PTR

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET NEW_NODE->DATA = VAL
Step 5: SET NEW_NODE->NEXT = NULL
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR->NEXT != NULL
Step 8:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 9: SET PTR->NEXT = NEW_NODE
Step 10: EXIT
```

Inserting a Node Before a Given Node in a Linked List

- Suppose we want to add a new node with value 9 before the node containing 3.
- In Step 5, we take a pointer variable PTR and initialize it with START. That is, PTR now points to the first node of the linkedlist.
- Then, we take another pointer variable PREPTR and initialize it with PTR. So now, PTR, PREPTR, and START are all pointing to the first node of the linked list.
- In the while loop, we traverse through the linked list to reach the node that has its value equal to NUM.
- We need to reach this node because the new node will be inserted before this node.
- Once we reach this node, in Steps 10 and 11, we change the NEXT pointers in such a way that the new node is inserted before the desired node.

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET NEW_NODE->DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PTR->DATA != NUM
Step 8:     SET PREPTR = PTR
Step 9:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 10: PREPTR->NEXT = NEW_NODE
Step 11: SET NEW_NODE->NEXT = PTR
Step 12: EXIT
```

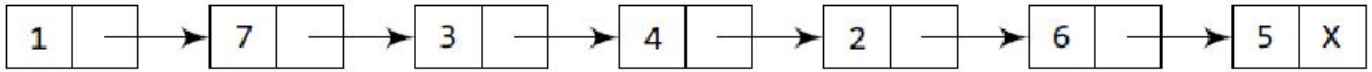
Somaiya TRUST

Inserting a Node Before a Given Node in a Linked List

START

Allocate memory for the new node and initialize its DATA part to 9.

Initialize PREPTR and PTR to the START node.

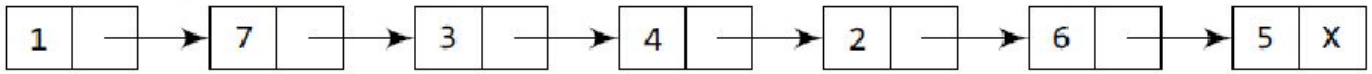


START

PTR

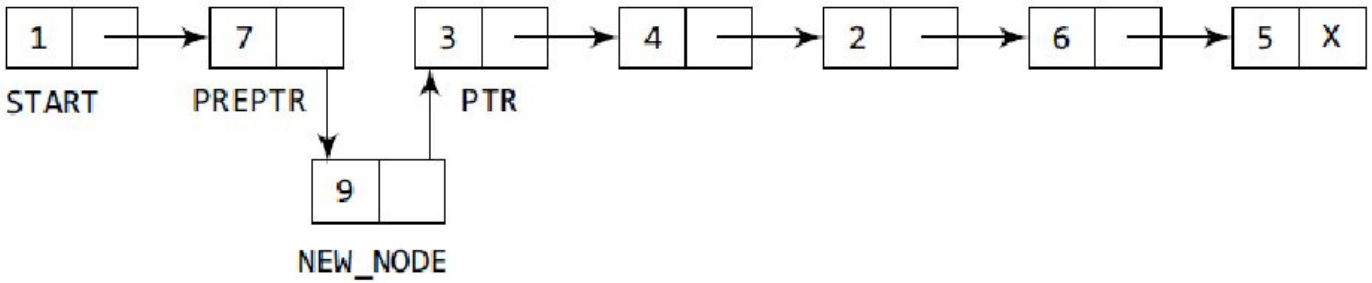
PREPTR

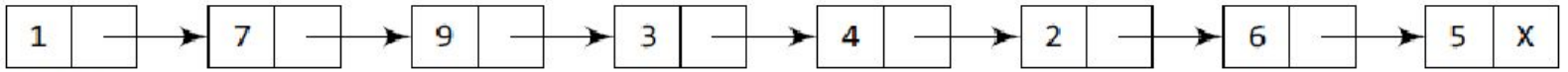
Move PTR and PREPTR until the DATA part of PTR = value of the node before which insertion has to be done. PREPTR will always point to the node just before PTR.



START PREPTR PTR

Insert the new node in between the nodes pointed by PREPTR and PTR.





START

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET NEW_NODE->DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PTR->DATA != NUM
Step 8:     SET PREPTR = PTR
Step 9:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 10: PREPTR->NEXT = NEW_NODE
Step 11: SET NEW_NODE->NEXT = PTR
Step 12: EXIT
```

Deleting a Node from a Linked List

- Case 1: The first node is deleted.
- Case 2: The last node is deleted.
- Case 3: The node after a given node is deleted.
- Underflow is a condition that occurs when we try to delete a node from a linked list that is empty.
- This happens when START = NULL or when there are no more nodes to delete.
- Note that when we delete a node from a linked list, we actually have to free the memory occupied
- by that node.
- The memory is returned to the free pool so that it can be used to store other programs and data.
- Whatever be the case of deletion, we always change the AVAIL pointer so that it points to the address that has been recently vacated.

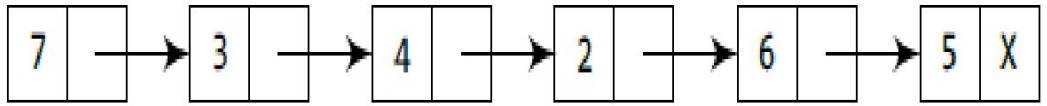
Deleting the First Node from a Linked List

- if there are nodes in the linked list, then we use a pointer variable PTR that is set to point to the first node of the list.
- For this, we initialize PTR with START that stores the address of the first node of the list.
- In Step 3, START is made to point to the next node in sequence and finally the memory occupied by the node pointed by PTR (initially the first node of the list) is freed and returned to the free pool.

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

START

Make START to point to the next node in sequence.



START

Deleting the Last Node from a Linked List

- In Step 2, we take a pointer variable PTR and initialize it with START. That is, PTR now points to the first node of the linked list.
- In the while loop, we take another pointer variable PREPTR such that it always point to one node before the PTR.
- Once we reach the last node and the second last node, we set the NEXT pointer of the second last node to NULL, so that it now becomes the (new) last node of the linked list.
- The memory of the previous last node is freed and returned back to the free pool.

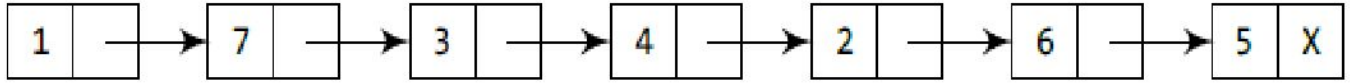
```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 8
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR->NEXT != NULL
Step 4:     SET PREPTR = PTR
Step 5:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 6: SET PREPTR->NEXT = NULL
Step 7: FREE PTR
Step 8: EXIT
```

Somaiya TRUST

Deleting the Last Node from a Linked List

START

Take pointer variables PTR and PREPTR which initially point to START.

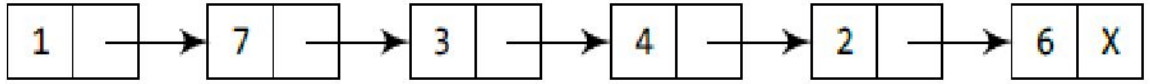


START
PREPTR
PTR

Move PTR and PREPTR such that NEXT part of PTR = NULL. PREPTR always points to the node just before the node pointed by PTR.

START PREPTR PTR

Set the NEXT part of PREPTR node to NULL.



START

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 8
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR->NEXT != NULL
Step 4:     SET PREPTR = PTR
Step 5:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 6: SET PREPTR->NEXT = NULL
Step 7: FREE PTR
Step 8: EXIT
```

Deleting the Node After a Given Node in a Linked List

- In Step 2, we take a pointer variable PTR and initialize it with START. That is, PTR now points to the first node of the linked list.
- In the while loop, we take another pointer variable PREPTR such that it always points to one node before the PTR.
- Once we reach the node containing VAL and the node succeeding it, we set the next pointer of the node containing VAL to the address contained in next field of the node succeeding it.
- The memory of the node succeeding the given node is freed and returned back to the free pool.

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Steps 5 and 6 while PREPTR->DATA != NUM
Step 5:     SET PREPTR = PTR
Step 6:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 7: SET TEMP = PTR
Step 8: SET PREPTR->NEXT = PTR->NEXT
Step 9: FREE TEMP
Step 10: EXIT
```

Somaiya TRUST

Deleting the Node After a Given Node in a Linked List

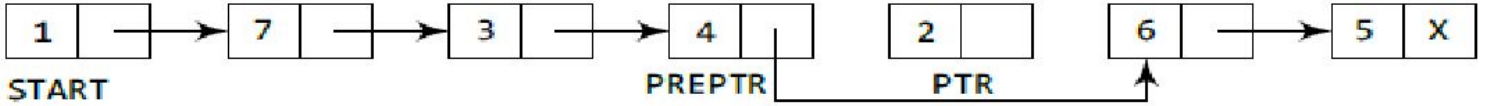
START

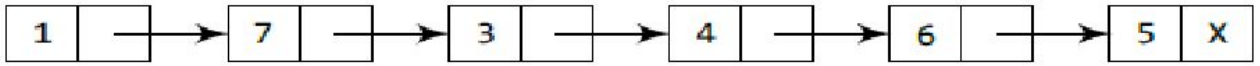
Take pointer variables PTR and PREPTR which initially point to START.

START
PREPTR
PTR

Move PREPTR and PTR such that PREPTR points to the node containing VAL and PTR points to the succeeding node.

Set the NEXT part of PREPTR to the NEXT part of PTR.

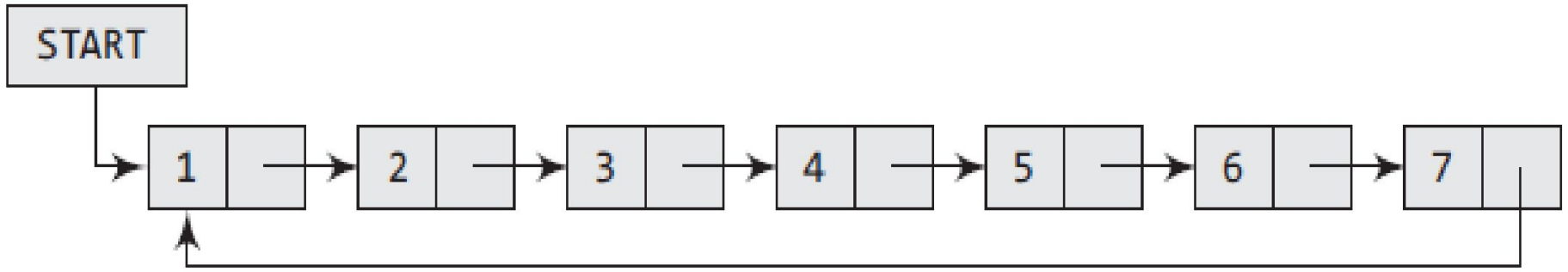





```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Steps 5 and 6 while PREPTR->DATA != NUM
Step 5:     SET PREPTR = PTR
Step 6:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 7: SET TEMP = PTR
Step 8: SET PREPTR->NEXT = PTR->NEXT
Step 9: FREE TEMP
Step 10: EXIT
```

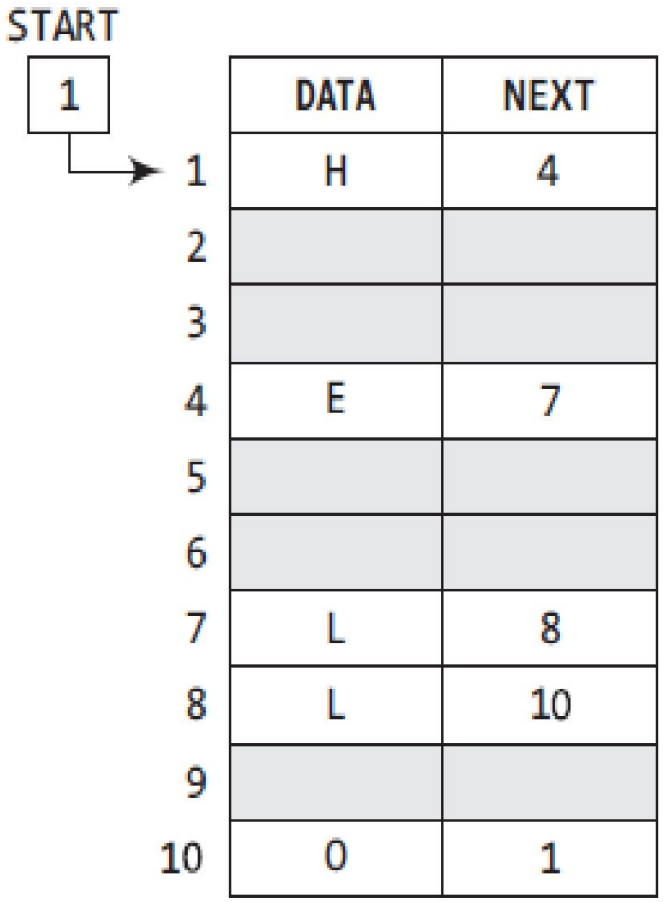
CIRCULAR LINKED LIST(CLL)

- In a circular linked list, the last node contains a pointer to the first node of the list.
- We can have a circular singly linked list as well as a circular doubly linked list.
- While traversing a circular linked list, we can begin at any node and traverse the list in any direction, forward or backward, until we reach the same node where we started.
- Thus, a circular linked list has no beginning and no ending.
- The only downside of a circular linked list is the complexity of iteration. Note that there are no NULL values in the NEXT part of any of the nodes of list.



Example CIRCULAR LINKED LIST

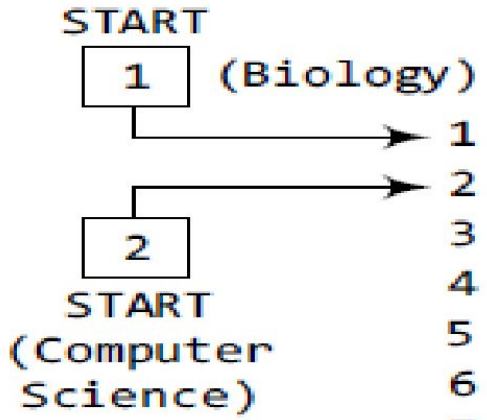
- Circular linked lists are widely used in operating systems for task maintenance.
- When we are surfing the Internet, we can use the Back button and the Forward button to move to the previous pages that we have already visited.
- How is this done? The answer is simple.
- A circular linked list is used to maintain the sequence of the Web pages visited.
- Traversing this circular linked list either in forward or backward direction helps to revisit the pages again using Back and Forward buttons.



	DATA	NEXT
1	H	4
2		
3		
4	E	7
5		
6		
7	L	8
8	L	10
9		
10	0	1

Somaiya TRUST

CLL Memory Representation



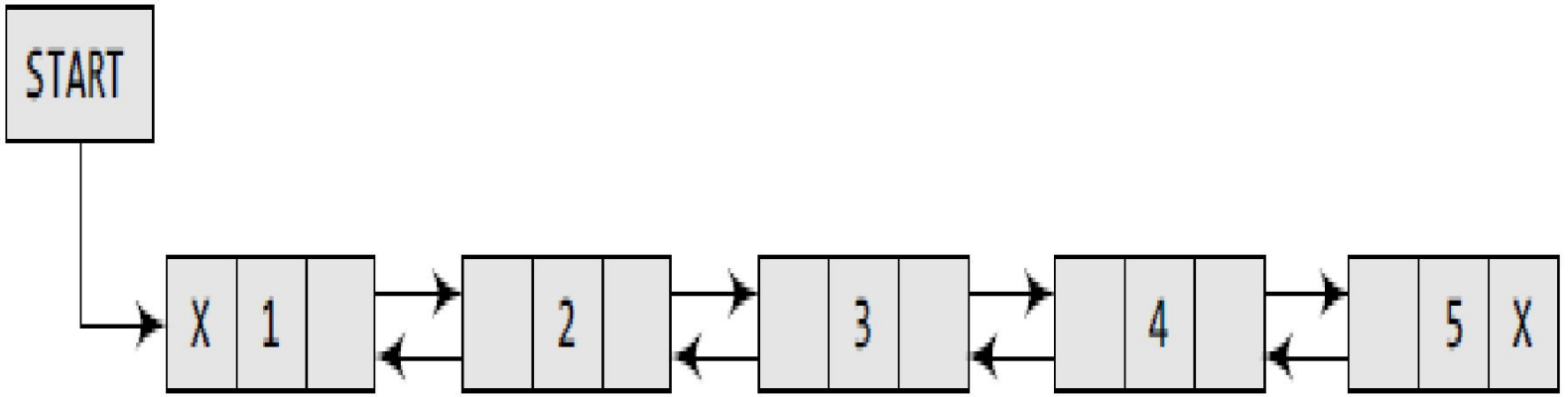
	Roll No	NEXT
1	S01	3
2	S02	5
3	S03	8
4		
5	S04	7
6		
7	S05	10
8	S06	11
9		
10	S07	12
11	S08	13
12	S09	1
13	S10	15
14		
15	S11	2

Somaiya TRUST

DOUBLY LINKED LISTS

- A doubly linked list or a two-way linked list is a more complex type of linked list that contains a pointer to the next as well as the previous node in the sequence.
- Therefore, it consists of three parts—data, a pointer to the next node, and a pointer to the previous node
- In C, the structure of a doubly linked list can be given as,

```
struct node
{
struct node *prev;
int data;
struct node *next;
};
```



Somaiya TRUST

DLL Memory representation

- The PREV field of the first node and the NEXT field of the last node will contain NULL.
- The PREV field is used to store the address of the preceding node, which enables us to traverse the list in the backward direction.
- Thus, we see that a doubly linked list calls for more space per node and more expensive basic operations.
- However, a doubly linked list provides the ease to manipulate the elements of the list as it maintains pointers to nodes in both the directions (forward and backward).
- The main advantage of using a doubly linked list is that it makes searching twice as efficient

START = 1

	DATA	PREV	NEXT
1	H	-1	3
2			
3	E	1	6
4			
5			
6	L	3	7
7	L	6	9
8			
9	0	7	-1

Somaiya TRUST

Inserting a New Node in a Doubly Linked List

- Case 1: The new node is inserted at the beginning.
- Case 2: The new node is inserted at the end.
- Case 3: The new node is inserted after a given node.

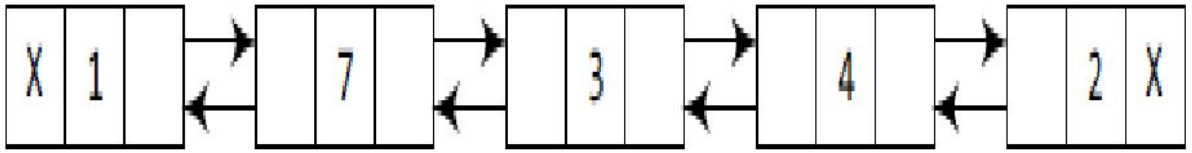
Inserting a Node at the beginning of a Doubly Linked List

- In Step 1, we first check whether memory is available for the new node.
- If the free memory has exhausted, then an OVERFLOW message is printed.
- Otherwise, if free memory cell is available, then we allocate space for the new node.
- Set its DATA part with the given VAL and the NEXT part is initialized with the address of the first node of the list, which is stored in START.
- Now, since the new node is added as the first node of the list, it will now be known as the START node, that is, the START pointer variable will now hold the address of NEW_NODE.

```
Step 1: IF AVAIL = NULL
                Write OVERFLOW
                Go to Step 9
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> PREV = NULL
Step 6: SET NEW_NODE -> NEXT = START
Step 7: SET START -> PREV = NEW_NODE
Step 8: SET START = NEW_NODE
Step 9: EXIT
```

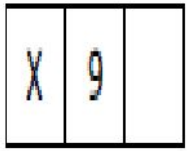
Somaiya TRUST

Inserting a Node at the beginning of a Doubly Linked List

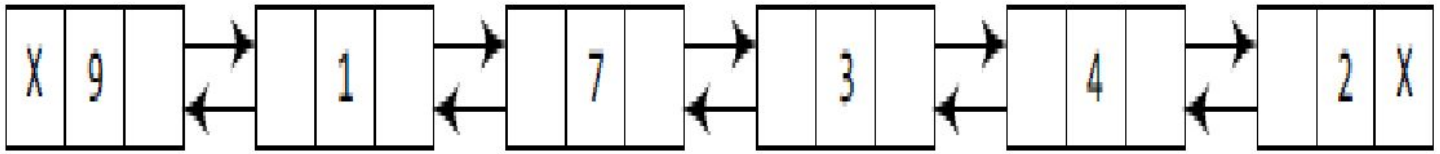


START

Allocate memory for the new node and initialize its DATA part to 9 and PREV field to NULL



Add the new node before the START node. Now the new node becomes the first node of the list.



START

```
Step 1: IF AVAIL = NULL
                Write OVERFLOW
                Go to Step 9
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> PREV = NULL
Step 6: SET NEW_NODE -> NEXT = START
Step 7: SET START -> PREV = NEW_NODE
Step 8: SET START = NEW_NODE
Step 9: EXIT
```

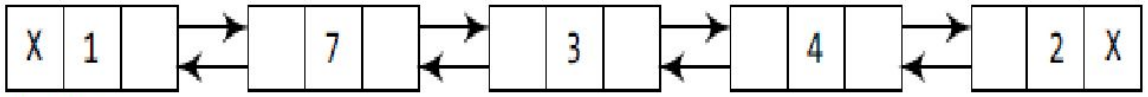
Inserting a Node at the End of a Doubly Linked List

- In Step 6, we take a pointer variable PTR and initialize it with START.
- In the while loop, we traverse through the linked list to reach the last node.
- Once we reach the last node, in Step 9, we change the NEXT pointer of the last node to store the address of the new node.
- Remember that the NEXT field of the new node contains NULL which signifies the end of the linked list.
- The PREV field of the NEW_NODE will be set so that it points to the node pointed by PTR (now the second last node of the list).

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 11
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = NULL
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != NULL
Step 8:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: SET NEW_NODE -> PREV = PTR
Step 11: EXIT
```

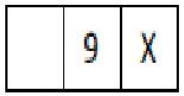
Somaiya TRUST

Inserting a Node at the End of a Doubly Linked List

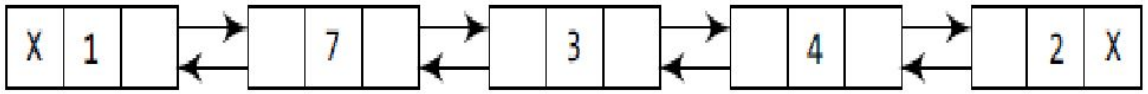


START

Allocate memory for the new node and initialize its DATA part to 9 and its NEXT field to NULL.

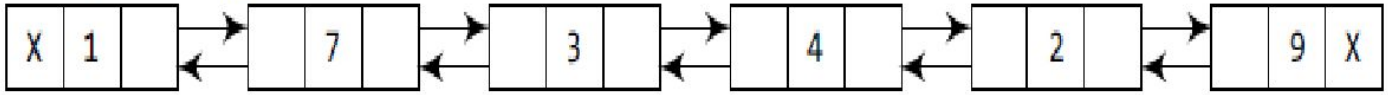


Take a pointer variable PTR and make it point to the first node of the list.



START,PTR

Move PTR so that it points to the last node of the list. Add the new node after the node pointed by PTR.



START PTR

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 11
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = NULL
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != NULL
Step 8:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: SET NEW_NODE -> PREV = PTR
Step 11: EXIT
```

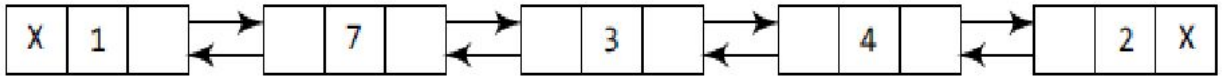
Inserting a Node After a Given Node in a Doubly Linked List

- In Step 5, we take a pointer PTR and initialize it with START.
- That is, PTR now points to the first node of the linked list.
- In the while loop, we traverse through the linked list to reach the node that has its value equal to NUM.
- We need to reach this node because the new node will be inserted after this node.
- Once we reach this node, we change the NEXT and PREV fields in such a way that the new node is inserted after the desired node.

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> DATA != NUM
Step 7:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET NEW_NODE -> NEXT = PTR -> NEXT
Step 9: SET NEW_NODE -> PREV = PTR
Step 10: SET PTR -> NEXT = NEW_NODE
Step 11: SET PTR -> NEXT -> PREV = NEW_NODE
Step 12: EXIT
```

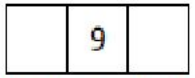
Somaiya TRUST

Inserting a Node After a Given Node in a Doubly Linked List

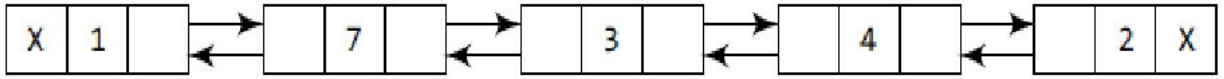


START

Allocate memory for the new node and initialize its DATA part to 9.

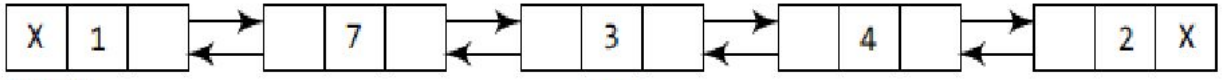


Take a pointer variable PTR and make it point to the first node of the list.



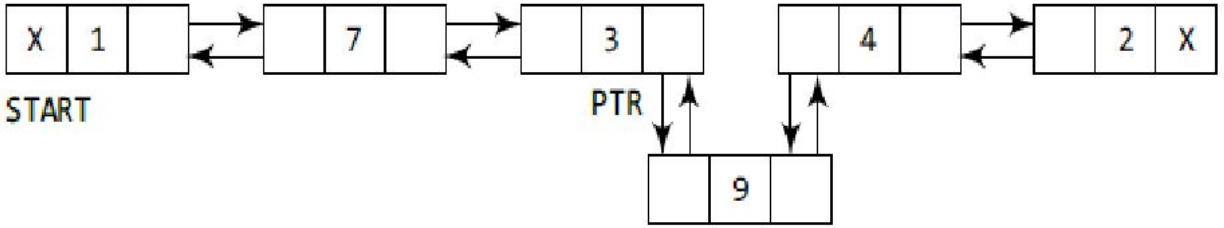
START,PTR

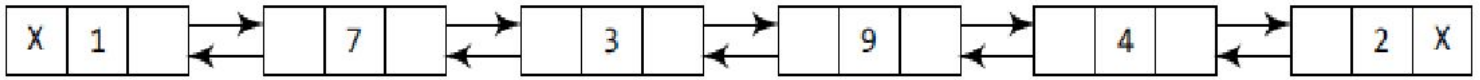
Move PTR further until the data part of PTR = value after which the node has to be inserted.



START PTR

Insert the new node between PTR and the node succeeding it.





START

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> DATA != NUM
Step 7:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET NEW_NODE -> NEXT = PTR -> NEXT
Step 9: SET NEW_NODE -> PREV = PTR
Step 10: SET PTR -> NEXT = NEW_NODE
Step 11: SET PTR -> NEXT -> PREV = NEW_NODE
Step 12: EXIT
```

Deleting a Node from a Doubly Linked List

- Case 1: The first node is deleted.
- Case 2: The last node is deleted.
- Case 3: The node after a given node is deleted.

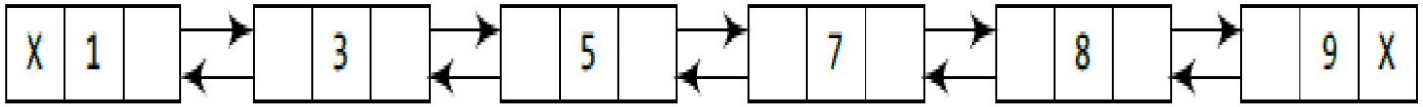
Deleting the First Node from a Doubly Linked List

- In Step 1 of the algorithm, we check if the linked list exists or not.
- If START =NULL, then it signifies that there are no nodes in the list and the control is transferred to the last statement of the algorithm.
- However, if there are nodes in the linked list, then we use a temporary pointer variable PTR that is set to point to the first node of the list.
- For this, we initialize PTR with START that stores the address of the first node of the list.
- In Step 3, START is made to point to the next node in sequence and finally the memory occupied by PTR (initially the first node of the list) is freed and returned to the free pool.

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 6
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET START = START->NEXT
Step 4: SET START->PREV = NULL
Step 5: FREE PTR
Step 6: EXIT
```

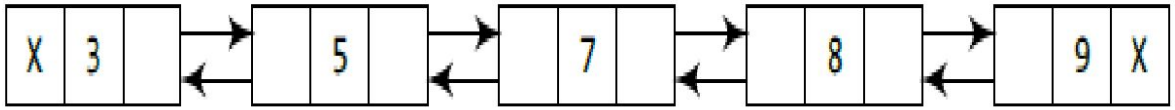
Somaiya TRUST

Deleting the First Node from a Doubly Linked List



START

Free the memory occupied by the first node of the list and make the second node of the list as the START node.



START

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 6
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET START = START->NEXT
Step 4: SET START->PREV = NULL
Step 5: FREE PTR
Step 6: EXIT
```

Somaiya TRUST

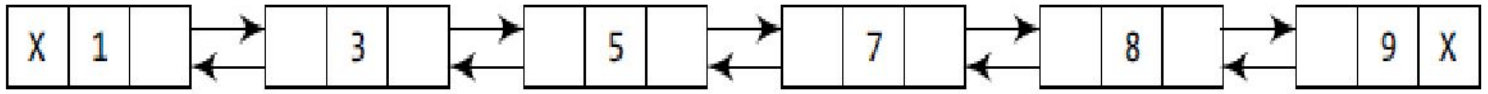
Deleting the Last Node from a Doubly Linked List

- In Step 2, we take a pointer variable PTR and initialize it with START.
- That is, PTR now points to the first node of the linked list.
- The while loop traverses through the list to reach the last node.
- Once we reach the last node, we can also access the second last node by taking its address from the PREV field of the last node.
- To delete the last node, we simply have to set the next field of second last node to NULL, so that it now becomes the (new) last node of the linked list.
- The memory of the previous last node is freed and returned to the free pool.

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 7
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR->NEXT != NULL
Step 4:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 5: SET PTR->PREV->NEXT = NULL
Step 6: FREE PTR
Step 7: EXIT
```

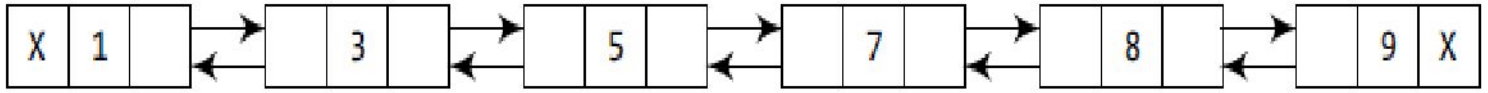
Somaiya TRUST

Deleting the Last Node from a Doubly Linked List



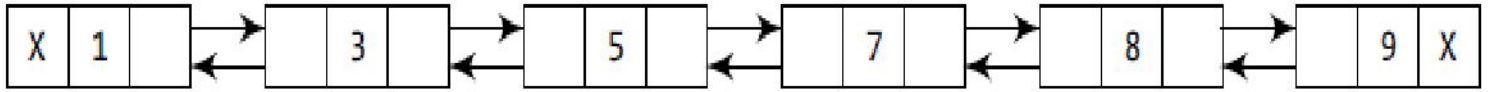
START

Take a pointer variable PTR that points to the first node of the list.



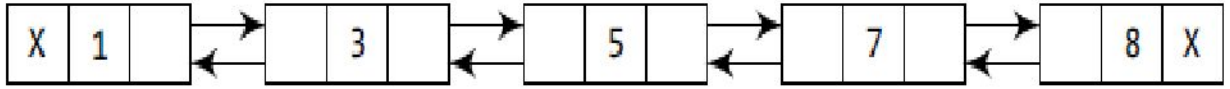
START,PTR

Move PTR so that it now points to the last node of the list.



START PTR

Free the space occupied by the node pointed by PTR and store NULL in NEXT field of its preceding node.



START

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 7
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR->NEXT != NULL
Step 4:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 5: SET PTR->PREV->NEXT = NULL
Step 6: FREE PTR
Step 7: EXIT
```

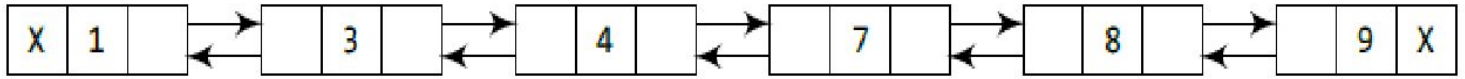
Deleting the Node After a Given Node in a Doubly Linked List

- In Step 2, we take a pointer variable PTR and initialize it with START.
- That is, PTR now points to the first node of the doubly linked list.
- The while loop traverses through the linked list to reach the given node.
- Once we reach the node containing VAL, the node succeeding it can be easily accessed by using the address stored in its NEXT field.
- The NEXT field of the given node is set to contain the contents in the NEXT field of the succeeding node.
- Finally, the memory of the node succeeding the given node is freed and returned to the free pool.

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 9
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR->DATA != NUM
Step 4:       SET PTR = PTR->NEXT
        [END OF LOOP]
Step 5: SET TEMP = PTR->NEXT
Step 6: SET PTR->NEXT = TEMP->NEXT
Step 7: SET TEMP->NEXT->PREV = PTR
Step 8: FREE TEMP
Step 9: EXIT
```

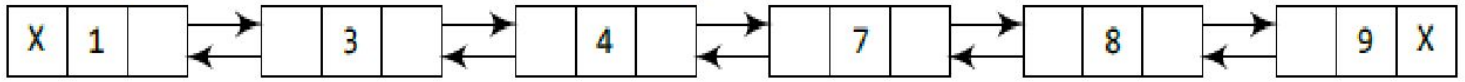
Somaiya TRUST

Deleting the Node After a Given Node in a Doubly Linked List



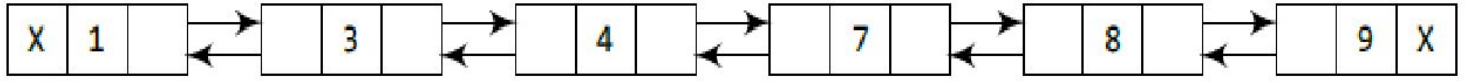
START

Take a pointer variable PTR and make it point to the first node of the list.



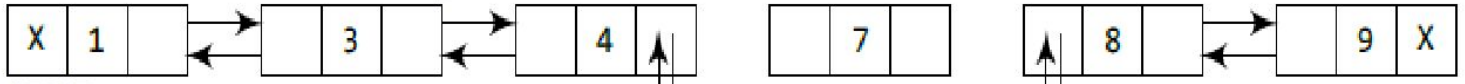
START,PTR

Move PTR further so that its data part is equal to the value after which the node has to be inserted.



START PTR

Delete the node succeeding PTR.



START PTR

START

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 9
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> DATA != NUM
Step 4:       SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET TEMP = PTR -> NEXT
Step 6: SET PTR -> NEXT = TEMP -> NEXT
Step 7: SET TEMP -> NEXT -> PREV = PTR
Step 8: FREE TEMP
Step 9: EXIT
```

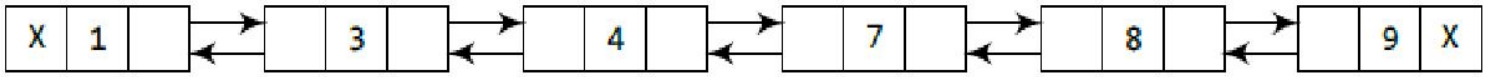
Deleting the Node Before a Given Node in a Doubly Linked List

- In Step 2, we take a pointer variable PTR and initialize it with START.
- That is, PTR now points to the first node of the linked list.
- The while loop traverses through the linked list to reach the desired node.
- Once we reach the node containing VAL, the PREV field of PTR is set to contain the address of the node preceding the node which comes before PTR.
- The memory of the node preceding PTR is freed and returned to the free pool.

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 9
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR->DATA != NUM
Step 4:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 5: SET TEMP = PTR->PREV
Step 6: SET TEMP->PREV->NEXT = PTR
Step 7: SET PTR->PREV = TEMP->PREV
Step 8: FREE TEMP
Step 9: EXIT
```

Somaiya TRUST

Deleting the Node Before a Given Node in a Doubly Linked List

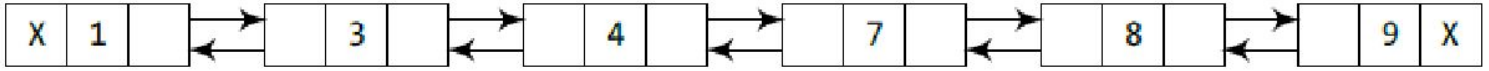


START

Take a pointer variable PTR that points to the first node of the list.

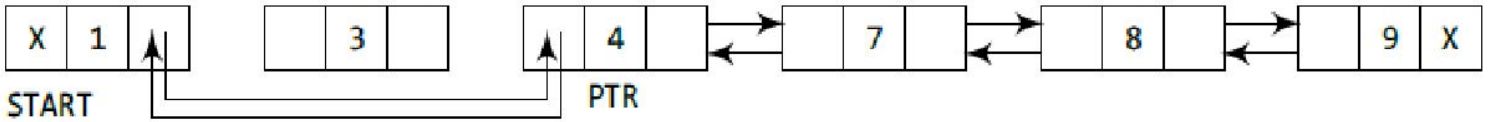
START,PTR

Move PTR further till its data part is equal to the value before which the node has to be deleted.



START PTR

Delete the node preceding PTR.



START PTR

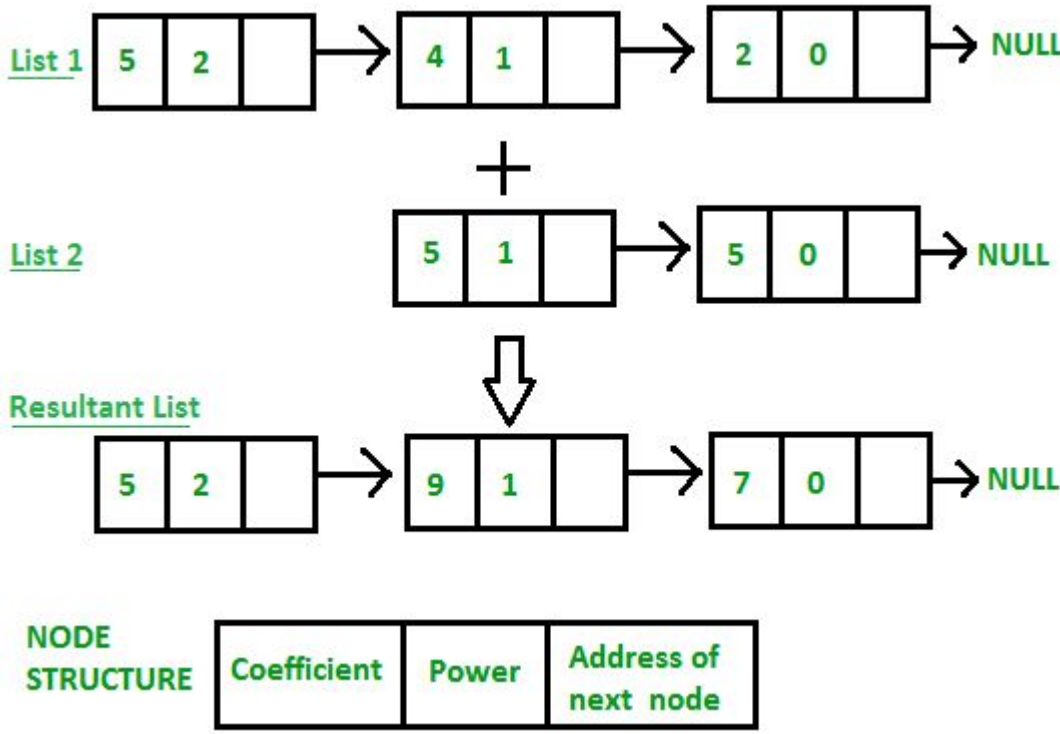
START

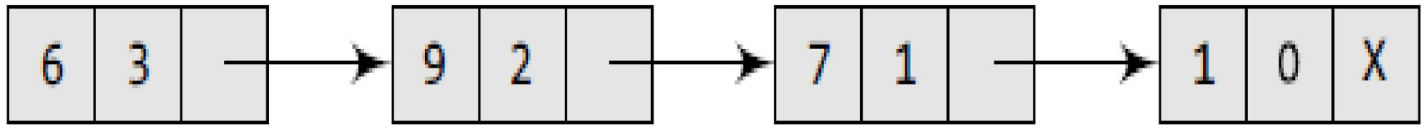
```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 9
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR->DATA != NUM
Step 4:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 5: SET TEMP = PTR->PREV
Step 6: SET TEMP->PREV->NEXT = PTR
Step 7: SET PTR->PREV = TEMP->PREV
Step 8: FREE TEMP
Step 9: EXIT
```

Somaiya TRUST

APPLICATIONS OF LINKED LISTS

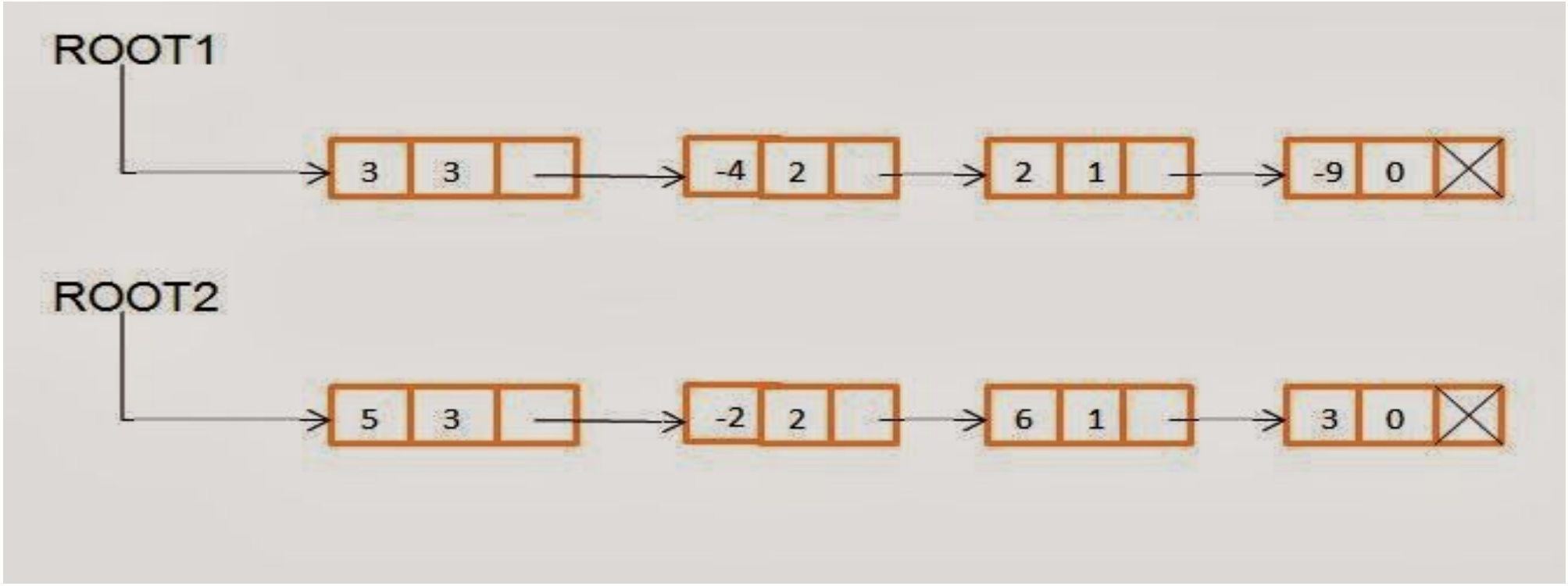
- Linked lists can be used to represent polynomials and the different operations that can be performed on them.
- **Polynomial Representation**
- Consider a polynomial $6x^3 + 9x^2 + 7x + 1$.
- Every individual term in a polynomial consists of two parts, a coefficient and a power.
- Here, 6, 9, 7, and 1 are the coefficients of the terms that have 3, 2, 1, and 0 as their powers respectively.
- Every term of a polynomial can be represented as a node of the linked list.

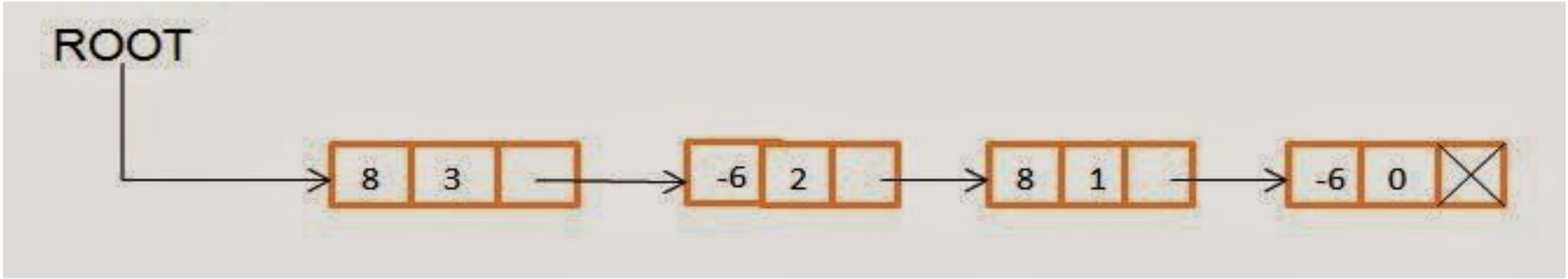




Somaiya TRUST

POLYNOMIAL ADDITION





POLYNOMIAL ADDITION USING LL

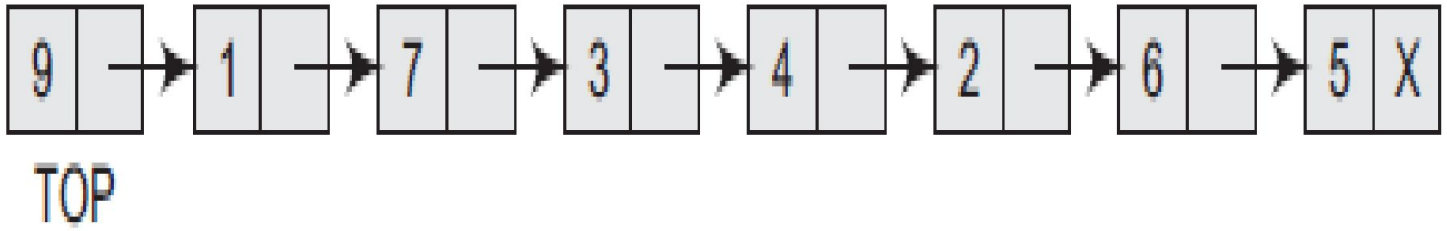
Representation of a polynomial using the linked list is beneficial when the operations on the polynomial like addition and subtractions are performed.

The resulting polynomial can also be traversed very easily to display the polynomial.

Somaiya TRUST

LINKED REPRESENTATION 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.



Somaiya TRUST

OPERATIONS ON A LINKED STACK - PUSH

- A linked stack supports all the three stack operations, that is, push, pop, and peek.

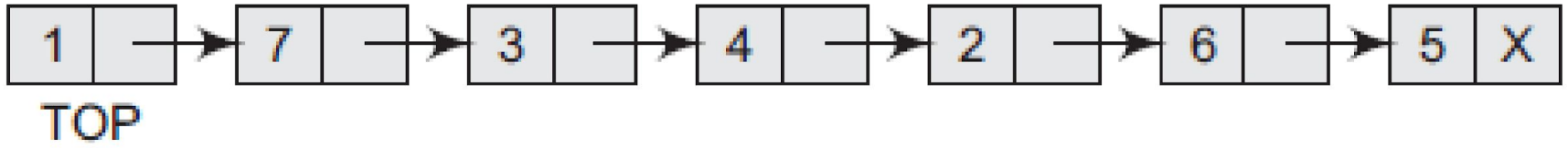
Push Operation

- The push operation is used to insert an element into the stack. The new element is added at the topmost position of the stack.
- In Step 1, memory is allocated for the new node.
- In Step 2, the DATA part of the new node is initialized with the value to be stored in the node.
- In Step 3, we check if the new node is the first node of the linked list. This is done by checking if TOP = NULL.
- In case the IF statement evaluates to true, then NULL is stored in the NEXT part of the node and the new node is called TOP.
- However, if the new node is not the first node in the list, then it is added before the first node of the list (that is, the TOP node) and termed as TOP.

```
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
```

Somaiya TRUST

Push operation



- To insert an element with value 9, we first check if TOP=NULL. If this is the case, then we allocate memory for a new node, store the value in its DATA part and NULL in its NEXT part. The new node
- will then be called TOP. However, if TOP!=NULL, then we insert the new node at the beginning of
- the linked stack and name this new node as TOP.

Somaiya TRUST

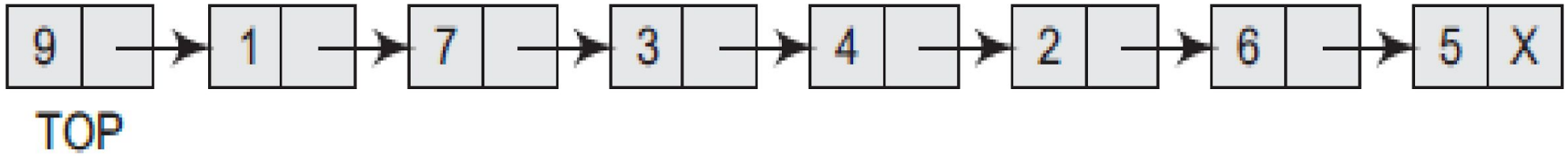
Pop Operation

- In Step 1, we first check for the UNDERFLOW condition.
- In Step 2, we use a pointer PTR that points to TOP.
- In Step 3, TOP is made to point to the next node in sequence.
- In Step 4, the memory occupied by PTR is given back to the free pool.

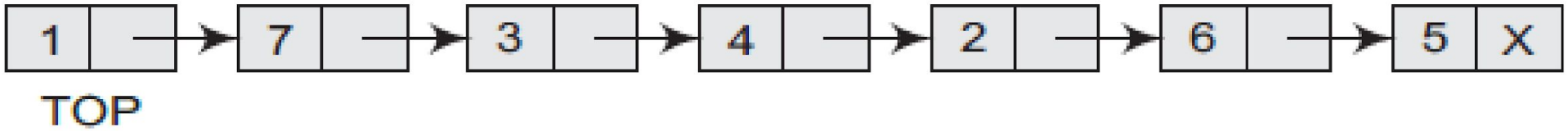
```
Step 1: IF TOP = NULL
            PRINT "UNDERFLOW"
            Goto Step 5
        [END OF IF]
Step 2: SET PTR = TOP
Step 3: SET TOP = TOP -> NEXT
Step 4: FREE PTR
Step 5: END
```

Somaiya TRUST

Pop operation



- before deleting the value, we must first check if TOP=NULL, because if this is the case, then it means that the stack is empty and no more deletions can be done.
- If an attempt is made to delete a value from a stack that is already empty, an UNDERFLOW message is printed.
- In case TOP!=NULL, then we will delete the node pointed by TOP, and make TOP point to the second element of the linked stack.



LINKED REPRESENTATION OF QUEUEs

- Drawback is that the array must be declared to have some fixed size.
- If we allocate space for 50 elements in the queue and it hardly uses 20–25 locations, then half of the space will be wasted.
- And in case we allocate less memory locations for a queue that might end up growing large and large, then a lot of re-allocations will have to be done, thereby creating a lot of overhead and consuming a lot of time.
- In case the queue is a very small one or its maximum size is known in advance, then the array implementation of the queue gives an efficient implementation.
- But if the array size cannot be determined in advance, the other alternative, i.e., the linked representation is used.

Somaiya TRUST

LINKED REPRESENTATION OF QUEUE

- In a linked queue, every element has two parts, one that stores the data and another that stores the address of the next element.
- The START pointer of the linked list is used as FRONT.
- Here, we will also use another pointer called REAR, which will store the address of the last element in the queue.
- All insertions will be done at the rear end and all the deletions will be done at the front end. If FRONT = REAR = NULL, then it indicates that the queue is empty.

Operations on Linked Queues -Enqueue

- In Step 1, the memory is allocated for the new node.
- In Step 2, the DATA part of the new node is initialized with the value to be stored in the node.
- In Step 3, we check if the new node is the first node of the linked queue.
- This is done by checking if FRONT = NULL.
- If this is the case, then the new node is tagged as FRONT as well as REAR.
- Also NULL is stored in the NEXT part of the node (which is also the FRONT and the REAR node).
- However, if the new node is not the first node in the list, then it is added at the REAR end of the linked queue (or the last node of the queue)

```
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
```

Somaiya TRUST

Operations on Linked Queues: Enqueue

Figure 8.6 Linked queue

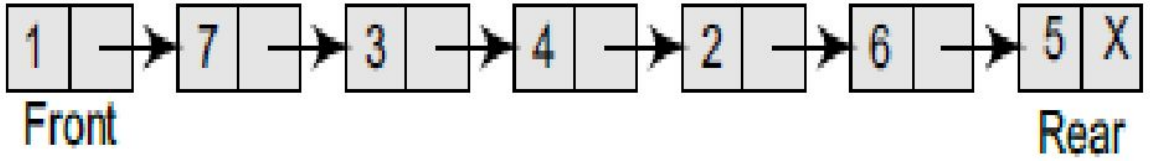
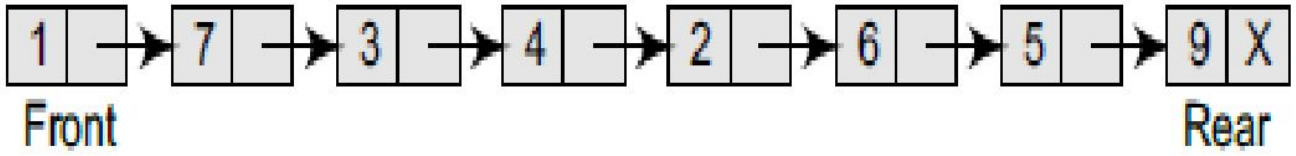


Figure 8.7 Linked queue



Somaiya TRUST

Operations on Linked Queues -Dequeue

- In Step 1, we first check for the underflow condition.
- If the condition is true, then an appropriate message is displayed,
- otherwise in Step 2, we use a pointer PTR that points to FRONT.
- In Step 3, FRONT is made to point to the next node in sequence.
- In Step 4, the memory occupied by PTR is given back to the free pool.

```
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
```

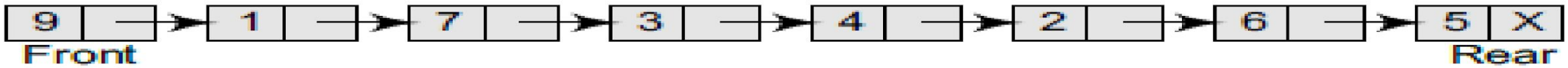
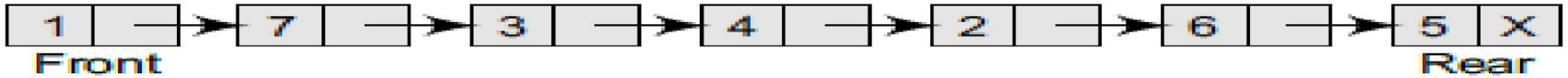


Figure 8.10 Linked queue



SAMPLE QUESTIONS

QUESTION NO.	SAMPLE QUESTIONS MODULE 3
1	Make a comparison between a linked list and a linear array. Which one will you prefer to use and when? Take example of insertion of numbers into array and Linked list. Apply insertion at beginning and compare the result.
2	Write algorithm for SLL insertion and deletion
3	Why is a doubly linked list more useful than a singly linked list?
4	Write an algorithm for performing various Insertions operations on doubly linked list a) Insertion at the beginning b) Insertion at the end c) Insertion in the middle
5	Form a Single linked list to store students' roll numbers [1,2,3,4,5,6]. Use the single linked list to insert the record of a new student at the beginning of the list. Show step wise execution of insertion. Delete the last record from the list. Show step wise deletion process using pointers
6	Write an algorithm to delete the middle element of a singly linked list. Show the representation
7	Explain Circular Linked list
8	Write an algorithm for performing various deletion operations on doubly linked list a) Deletion at the beginning b) Deletion at the end c) Deletion in the middle
9	Explain memory representation of Linked List with diagram
10	Write algorithms for performing the various operations performed on Linked List a) Traversing a list b) counting number of nodes in a list c) Searching an element in the list

SAMPLE QUESTIONS

QUESTION NO.	SAMPLE QUESTIONS MODULE 3
11	Write and explain the algorithm for Linked implementation of stack
12	Write and explain the algorithm for Linked implementation of Queue
13	Explain how polynomial addition is performed using Linked List

Somaiya TRUST