Indian Institute of Information Technology Kottayam

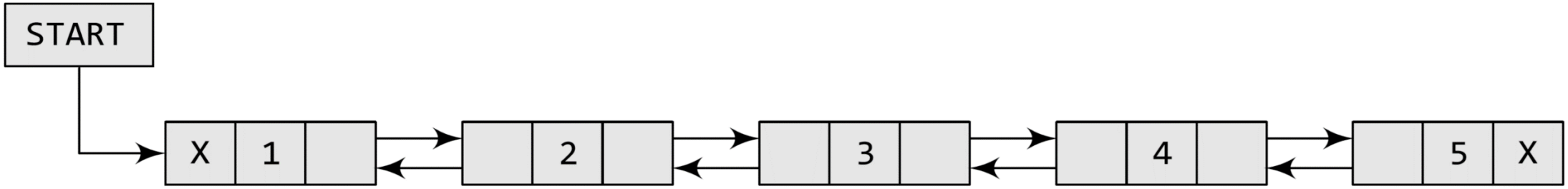
Doubly Linked Lists

BY

Arun Cyril Jose

Doubly Linked Lists

- Two-way linked list.
- Contains a pointer to the next as well as the previous node in the list.

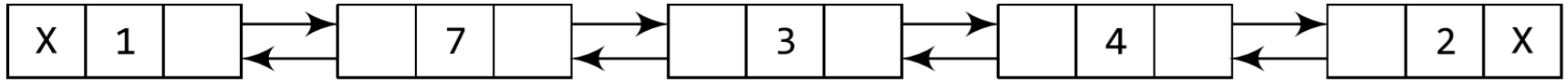


- Enables us to traverse the list in the backward direction.

Insertion Operations on Doubly Linked Lists

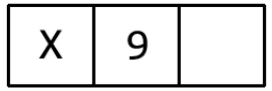
- New node inserted at the beginning.
- New node inserted at the end.
- New node inserted after a given node.
- New node inserted before a given node.

Doubly Linked Lists: Insertion at the beginning

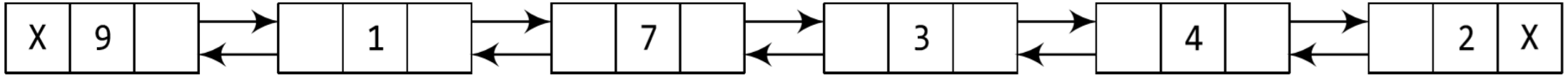


START

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.

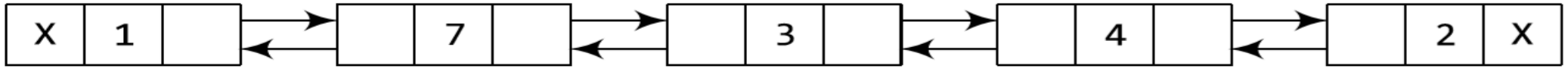


START

Doubly Linked Lists: Insertion at the beginning

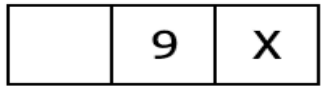
```
Step 1: IF AVAIL = NULL
                Write OVERFLOW
                Go to Step 9
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> PREV = NULL
Step 6: SET NEW_NODE -> NEXT = START
Step 7: SET START -> PREV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

Doubly Linked Lists: Insertion at the end



START

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

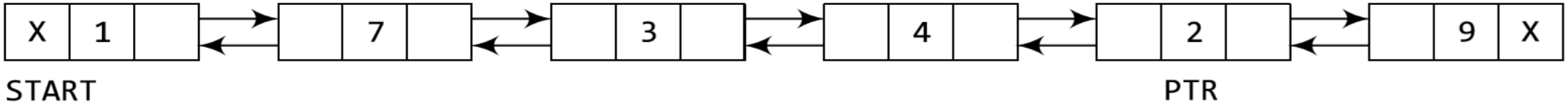


Take a pointer variable PTR and make it point to the first node of the list.

START,PTR

Doubly Linked Lists: Insertion at the end

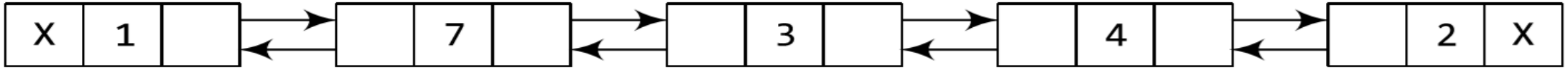
Move PTR so that it points to the last node of the list. Add the new node after the node pointed by PTR.



Doubly Linked Lists: Insertion at the end

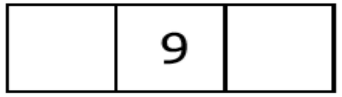
```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 11
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = NULL
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT !=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

Doubly Linked Lists: Insertion after a node



START

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



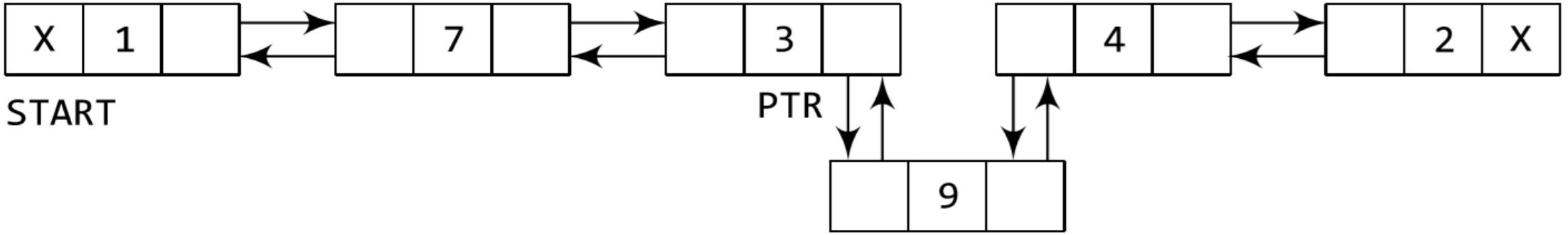
Take a pointer variable PTR and make it point to the first node of the list.

START,PTR

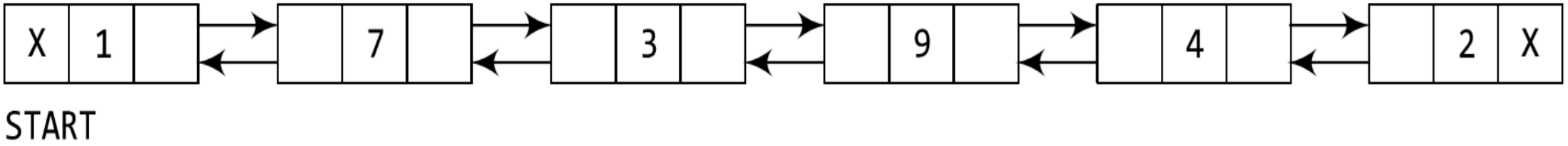
Doubly Linked Lists: Insertion after a node

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

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



Doubly Linked Lists: Insertion after a node



Doubly Linked Lists: Insertion after a 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->PREV = NEW_NODE
Step 11: SET PTR->NEXT = NEW_NODE
Step 12: EXIT
```

Doubly Linked Lists: Insertion before a node

START

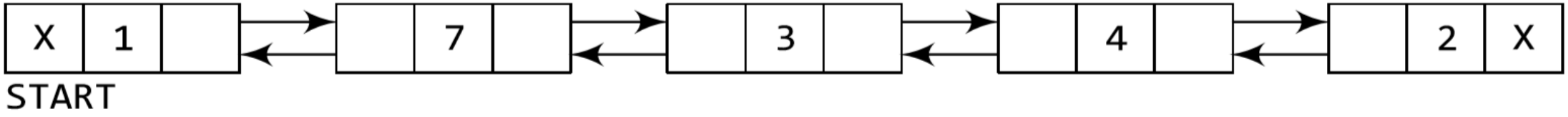
Allocate memory for the new node and initialize its DATA part to 9.

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

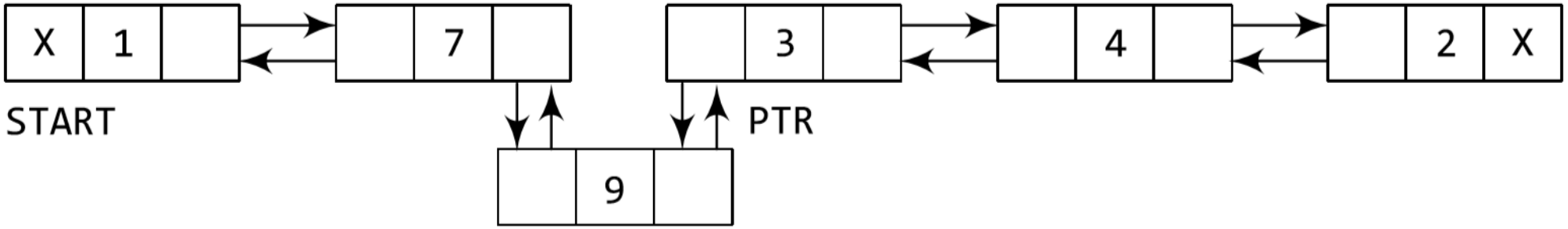
START, PTR

Doubly Linked Lists: Insertion before a node

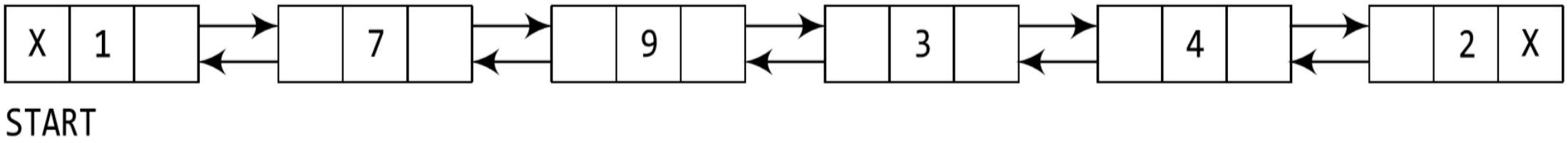
Move PTR further so that it now points to the node whose data is equal to the value before which the node has to be inserted.



Add the new node in between the node pointed by PTR and the node preceding it.



Doubly Linked Lists: Insertion before a node



Doubly Linked Lists: Insertion before a 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
Step 9: SET NEW_NODE -> PREV = PTR -> PREV
Step 10:  SET PTR -> PREV -> NEXT = NEW_NODE
Step 11:  PTR -> PREV = NEW_NODE
Step 12:  EXIT
```

Doubly Linked Lists: Patent


US007533138B1

(12) **United States Patent**
Martin

(10) **Patent No.: US 7,533,138 B1**
(45) **Date of Patent: May 12, 2009**

(54) **PRACTICAL LOCK-FREE DOUBLY-LINKED LIST**

(75) Inventor: **Paul A. Martin**, Arlington, MA (US)

(73) Assignee: **Sun Microsystems, Inc.**, Santa Clara, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 392 days.

(21) Appl. No.: **11/125,910**

(22) Filed: **May 10, 2005**

Related U.S. Application Data

(63) Continuation-in-part of application No. 10/820,661, filed on Apr. 7, 2004.

(60) Provisional application No. 60/571,059, filed on May 14, 2004.

(51) **Int. Cl.**
G06F 17/30 (2006.01)
G06F 9/46 (2006.01)
G06F 9/45 (2006.01)
G06F 13/00 (2006.01)

(52) **U.S. Cl.** **707/206**; 718/100; 717/151; 711/154

(58) **Field of Classification Search** 707/100
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,826,583	B1 *	11/2004	Flood et al.	707/206
6,826,757	B2 *	11/2004	Steele et al.	719/314
7,000,234	B1 *	2/2006	Shavit et al.	719/315

OTHER PUBLICATIONS

"Lock-Free Linked Lists Using Compare-and Swap", author: John D. Valois Dated Jul. 1, 1997.*

(Continued)

Primary Examiner—Neveen Abel Jalil
Assistant Examiner—Tarek Chbouki
(74) *Attorney, Agent, or Firm*—Park, Vaughan & Fleming LLP

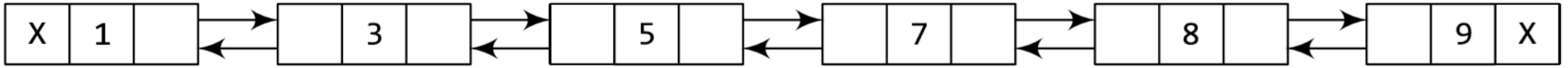
(57) **ABSTRACT**

One embodiment of the present invention provides a system that supports inserting or deleting nodes at any location within a doubly-linked list which is lock-free, wherein lock-free means that the doubly-linked list can be simultaneously accessed by multiple processes without requiring the processes to perform locking operations (non-blocking) and furthermore that a finite number of steps performed by a process will guarantee progress by at least one process (lock-free). During operation, the system receives a reference to a target node to be deleted from the doubly-linked list. Next, the system atomically marks a forward pointer in the target node to indicate that the target node is deleted, wherein the forward pointer contains the address of an immediately following node in the doubly-linked list, and wherein the marking operation does not destroy the address of the immediately following node. Additional cleanup steps are then done by this or any other process. The system may also receive a new node which is accessible by only the requesting thread and may then insert the new node into the doubly linked list after a reference node. The system accomplishes this by setting the new node's backward pointer to the reference node and forward pointer to the successor of the reference node. Next, the system atomically changes the forward pointer of the reference node from the successor node to the new node. Additional cleanup steps are then done by this or any other process. An update operation that atomically performs a delete of an old node and an insert of its replacement node is also described.

Deletion Operations on Doubly Linked Lists

- First node is deleted.
- Last node is deleted.
- Node after a given node is deleted.
- Node before a given node is deleted

Doubly Linked Lists: Deletion of First Node



START

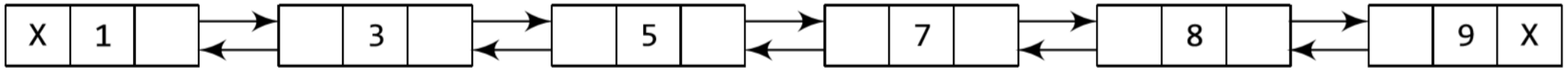
Free the memory occupied by the first node of the list and make the second node of the list as the START node.

START

Doubly Linked Lists: Deletion of First Node

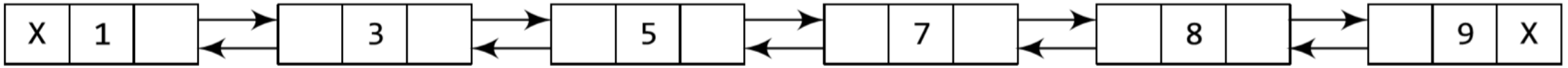
```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 6
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET START = START -> NEXT
Step 4: SET START -> PREV = NULL
Step 5: FREE PTR
Step 6: EXIT
```

Doubly Linked Lists: Deletion of Last Node



START

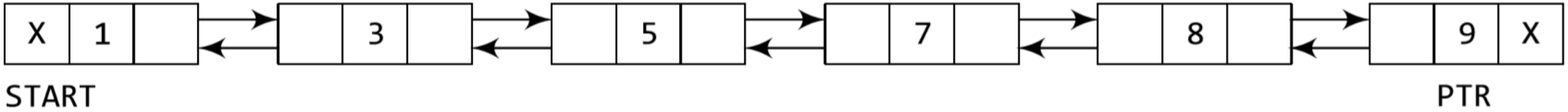
Take a pointer variable PTR that points to the first node of the list.



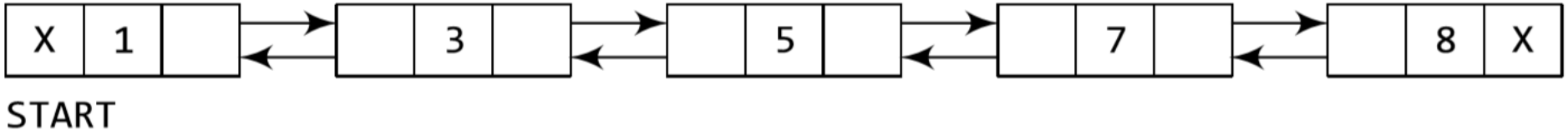
START,PTR

Doubly Linked Lists: Deletion of Last Node

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



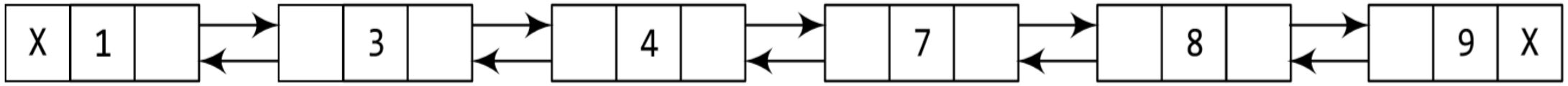
Free the space occupied by the node pointed by PTR and store NULL in NEXT field of its preceding node.



Doubly Linked Lists: Deletion of Last Node

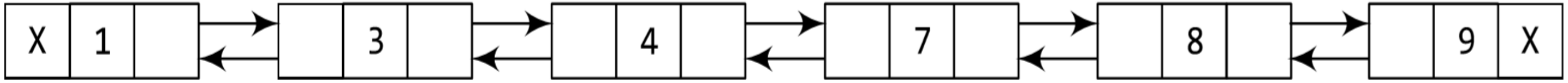
```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 7
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> NEXT != NULL
Step 4:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET PTR -> PREV -> NEXT = NULL
Step 6: FREE PTR
Step 7: EXIT
```

Doubly Linked Lists: Deletion of a Node After a Specific Node



START

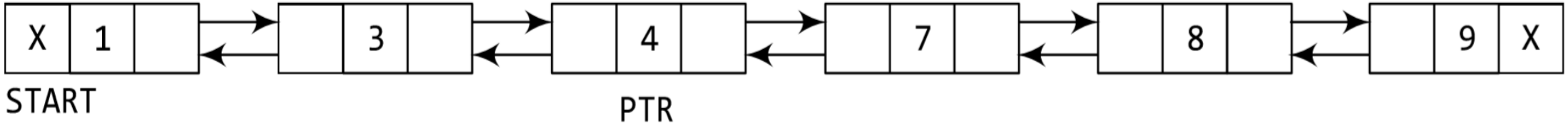
Take a pointer variable PTR and make it point to the first node of the list.



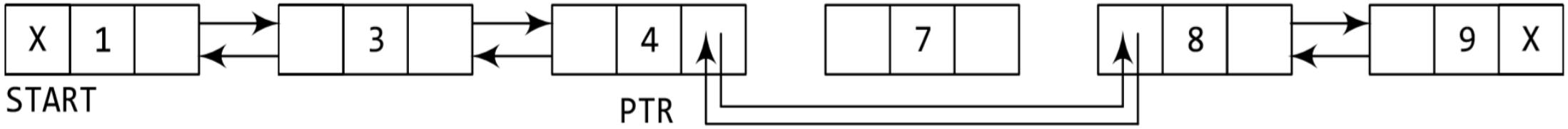
START,PTR

Doubly Linked Lists: Deletion of a Node After a Specific Node

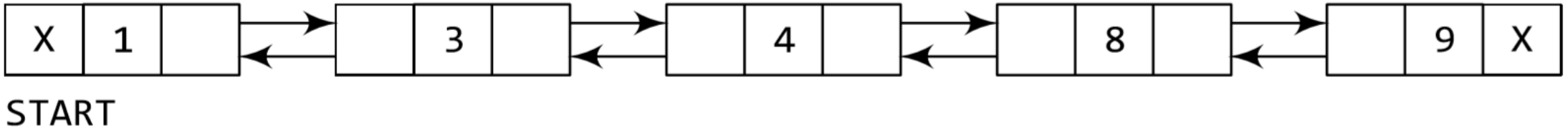
Move PTR further so that its data part is equal to the value after which the node has to be inserted.



Delete the node succeeding PTR.



Doubly Linked Lists: Deletion of a Node After a Specific Node



Doubly Linked Lists: Deletion of a Node After a Specific Node

```
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
```