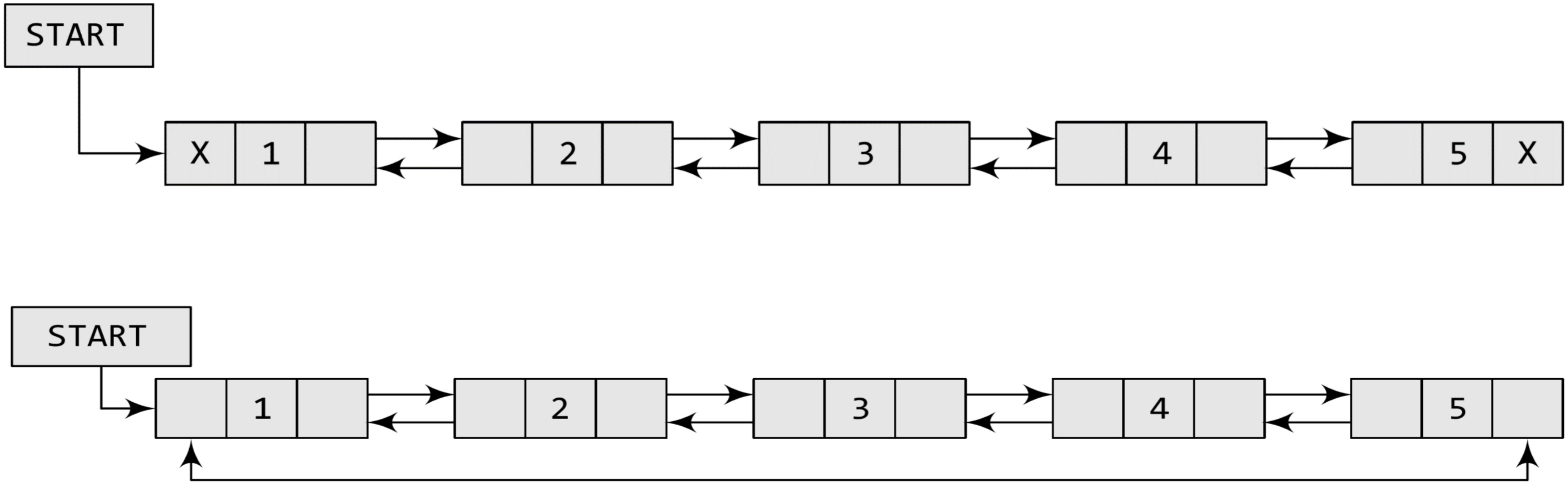
Circular Linked Lists and Multi-Linked Lists

BY

Arun Cyril Jose

Circular Doubly Linked Lists

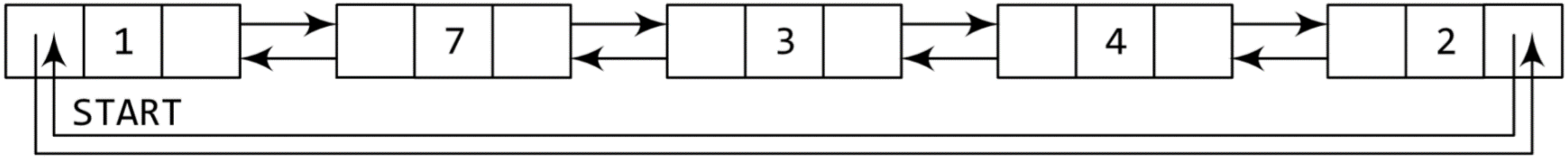


Circular Doubly Linked Lists: Insertion Operations

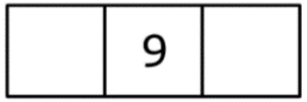
- New node inserted at the beginning.

- New node inserted at the end.

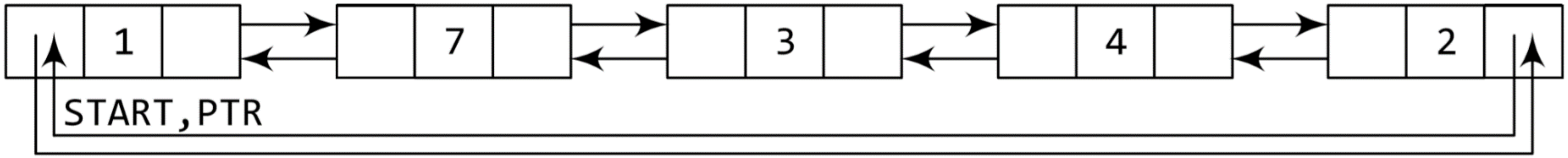
Circular Doubly Linked Lists: Insertion at the beginning



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

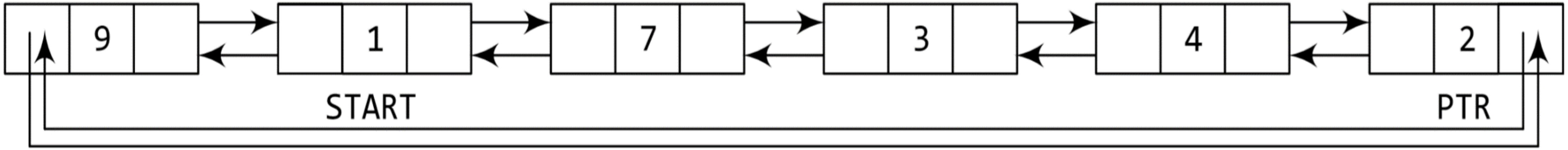


Take a pointer variable PTR that points to the first node of the list.

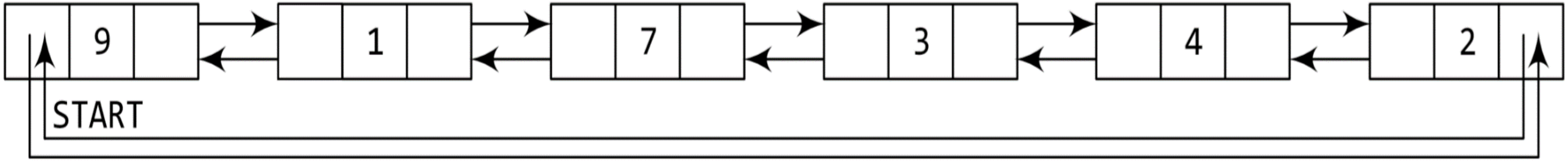


Circular Doubly Linked Lists: Insertion at the beginning

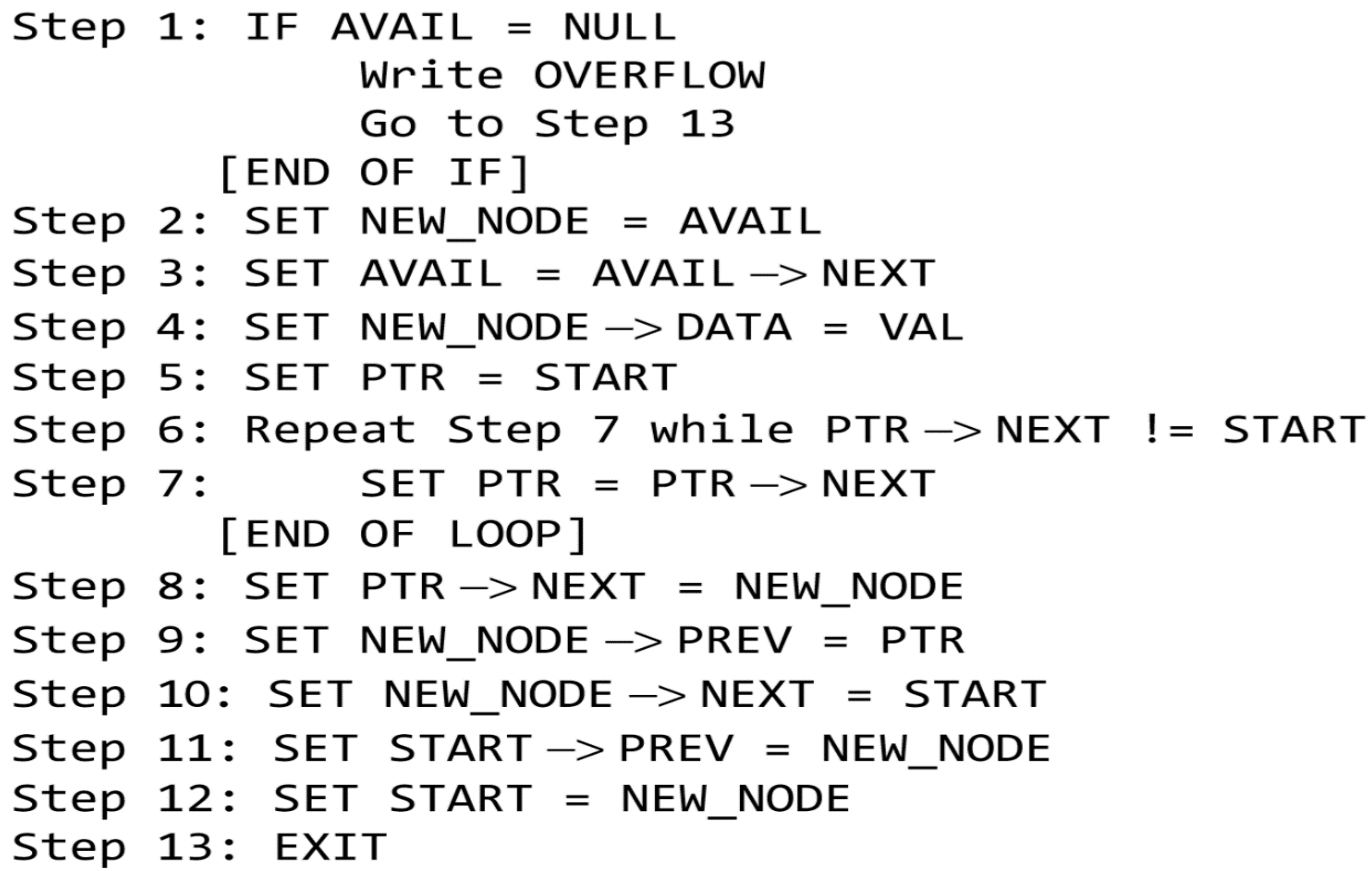
Move PTR so that it now points to the last node of the list. Insert the new node in between PTR and the START node.



START will now point to the new node.

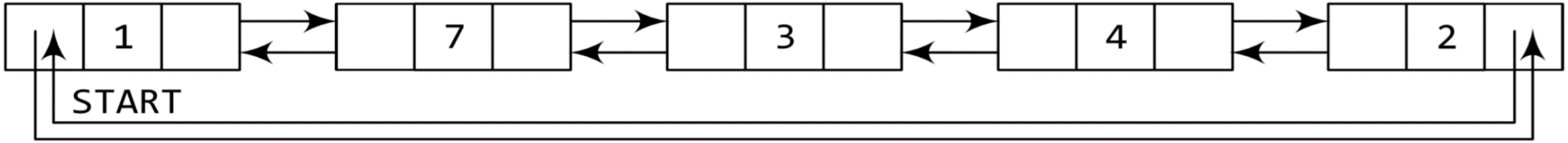


Circular Doubly Linked Lists: Insertion at the beginning

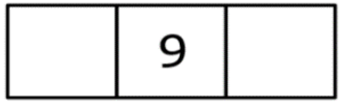


```
Step 1: IF AVAIL = NULL
              Write OVERFLOW
              Go to Step 13
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> NEXT != START
Step 7:       SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET PTR -> NEXT = NEW_NODE
Step 9: SET NEW_NODE -> PREV = PTR
Step 10: SET NEW_NODE -> NEXT = START
Step 11: SET START -> PREV = NEW_NODE
Step 12: SET START = NEW_NODE
Step 13: EXIT
```

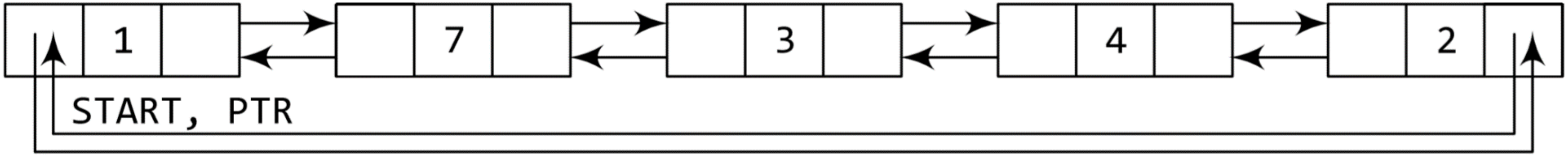
Circular Doubly Linked Lists: Insertion at the end



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

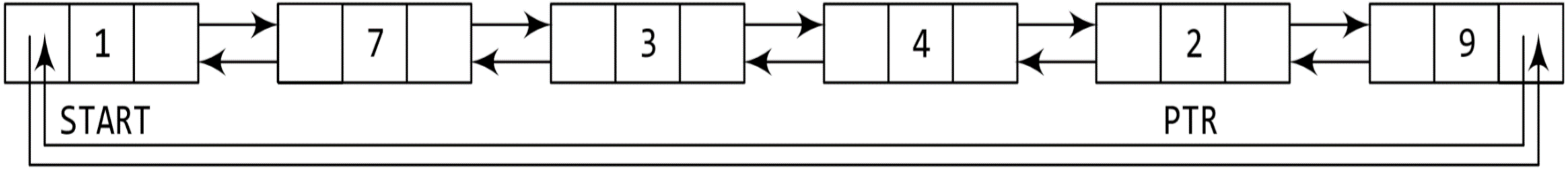


Take a pointer variable PTR that points to the first node of the list.

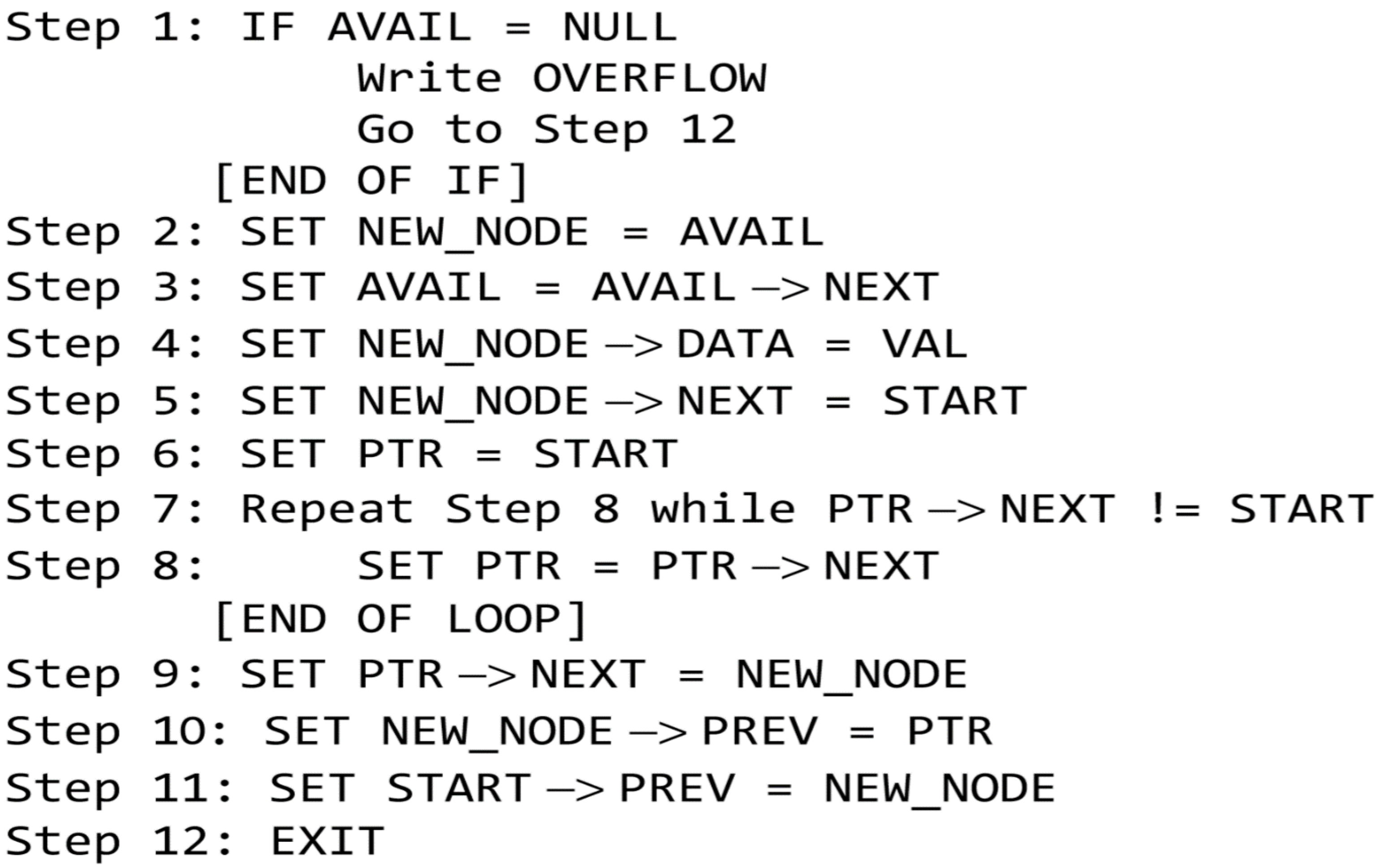


Circular Doubly Linked Lists: Insertion at the end

Move PTR to point to the last node of the list so that the new node can be inserted after it.



Circular Doubly Linked Lists: Insertion at the end



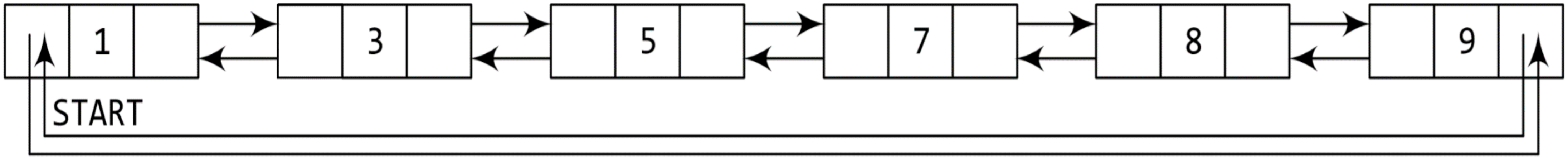
```
Step 1: IF AVAIL = NULL
             Write OVERFLOW
             Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT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

Circular Doubly Linked Lists: Deletion Operations

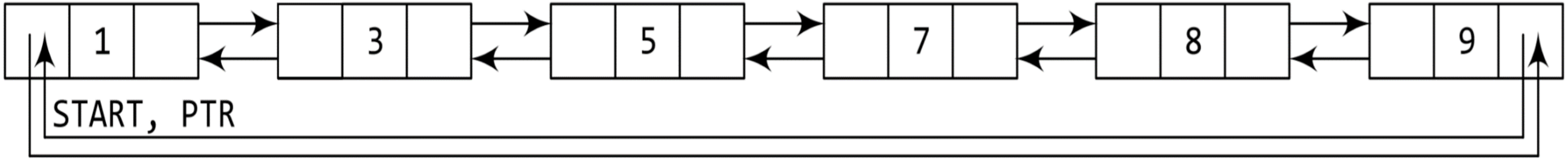
- First node is deleted.

- Last node is deleted.

Circular Doubly Linked Lists: Deletion at the beginning

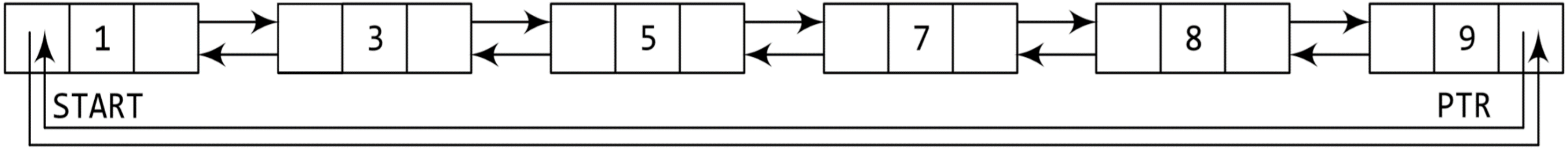


Take a pointer variable PTR that points to the first node of the list.

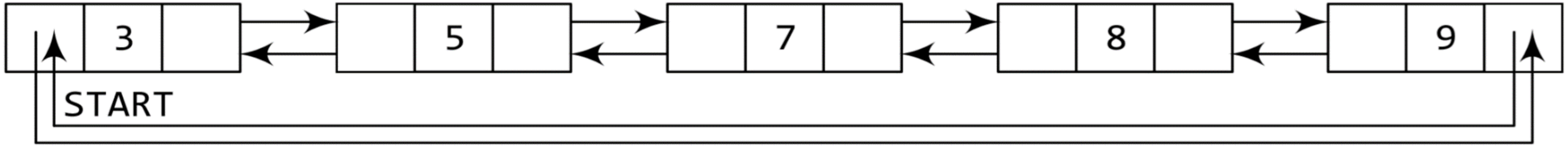


Circular Doubly Linked Lists: Deletion at the beginning

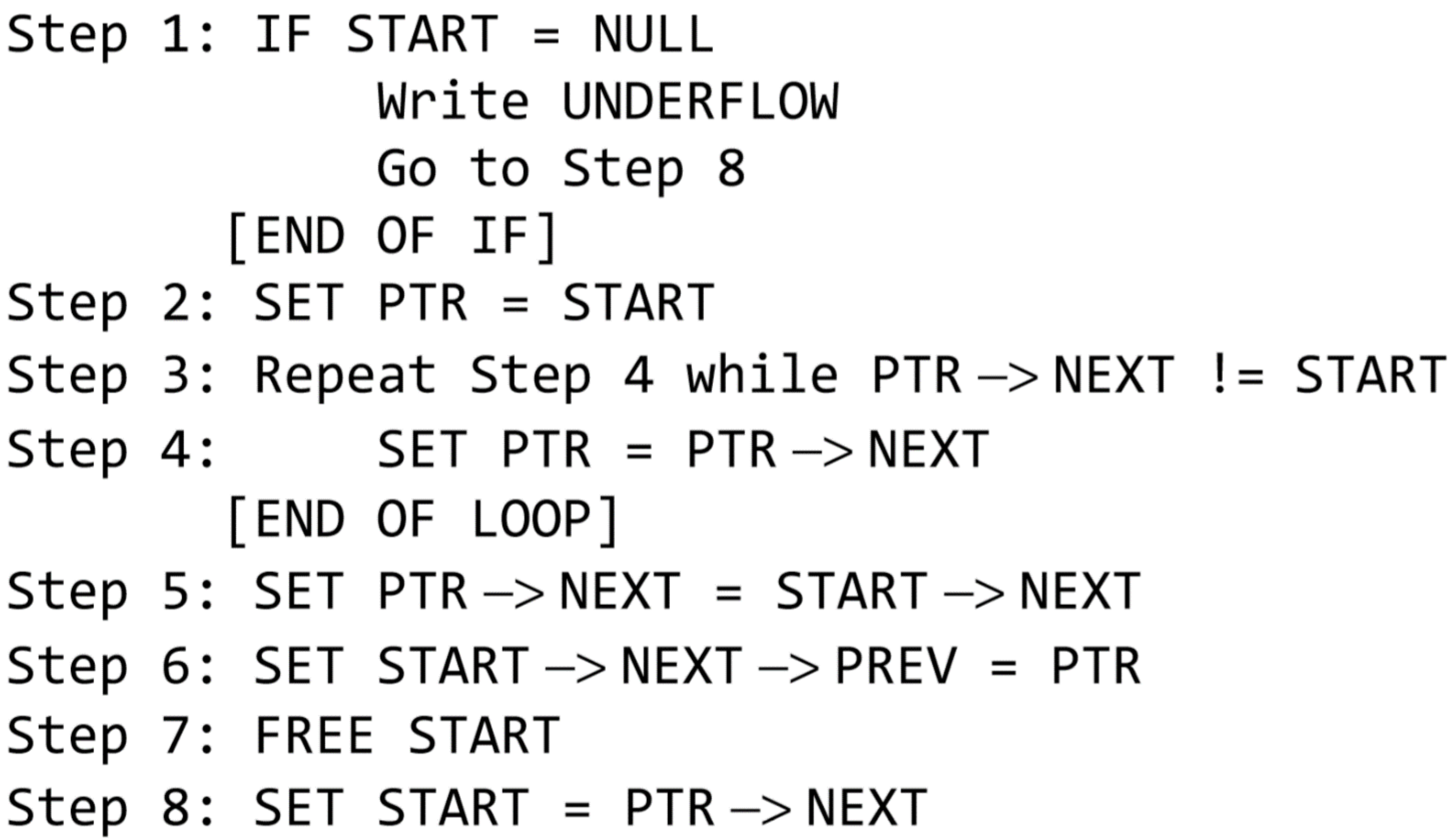
Move PTR further so that it now points to the last node of the list.



Make START point to the second node of the list. Free the space occupied by the first node.

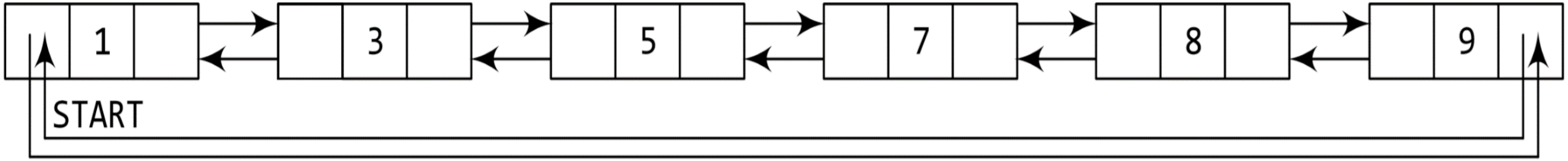


Circular Doubly Linked Lists: Deletion at the beginning

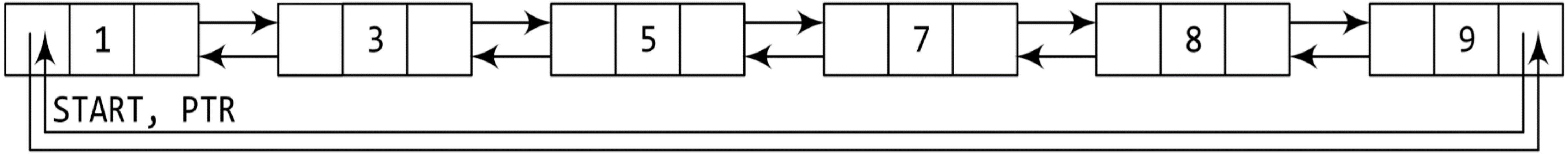


```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 8
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> NEXT != START
Step 4:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET PTR -> NEXT = START -> NEXT
Step 6: SET START -> NEXT -> PREV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

Circular Doubly Linked Lists: Deletion at the end

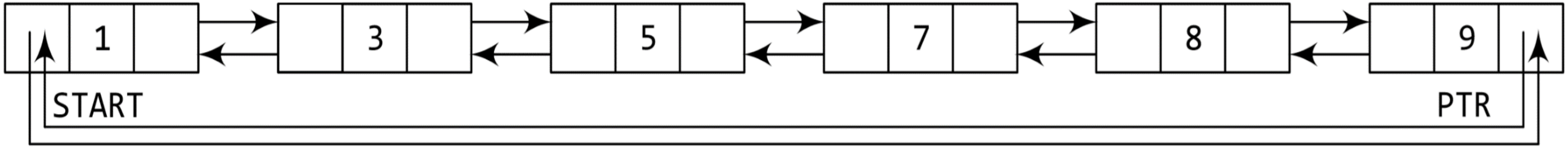


Take a pointer variable PTR that points to the first node of the list.

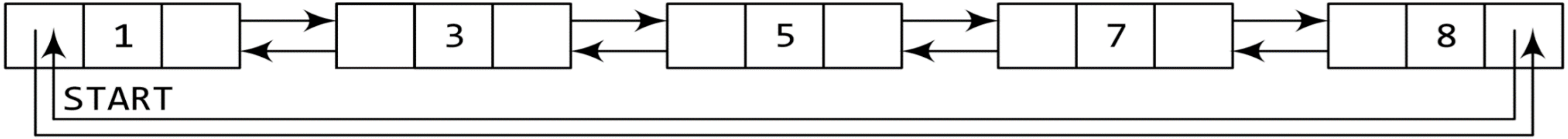


Circular Doubly Linked Lists: Deletion at the end

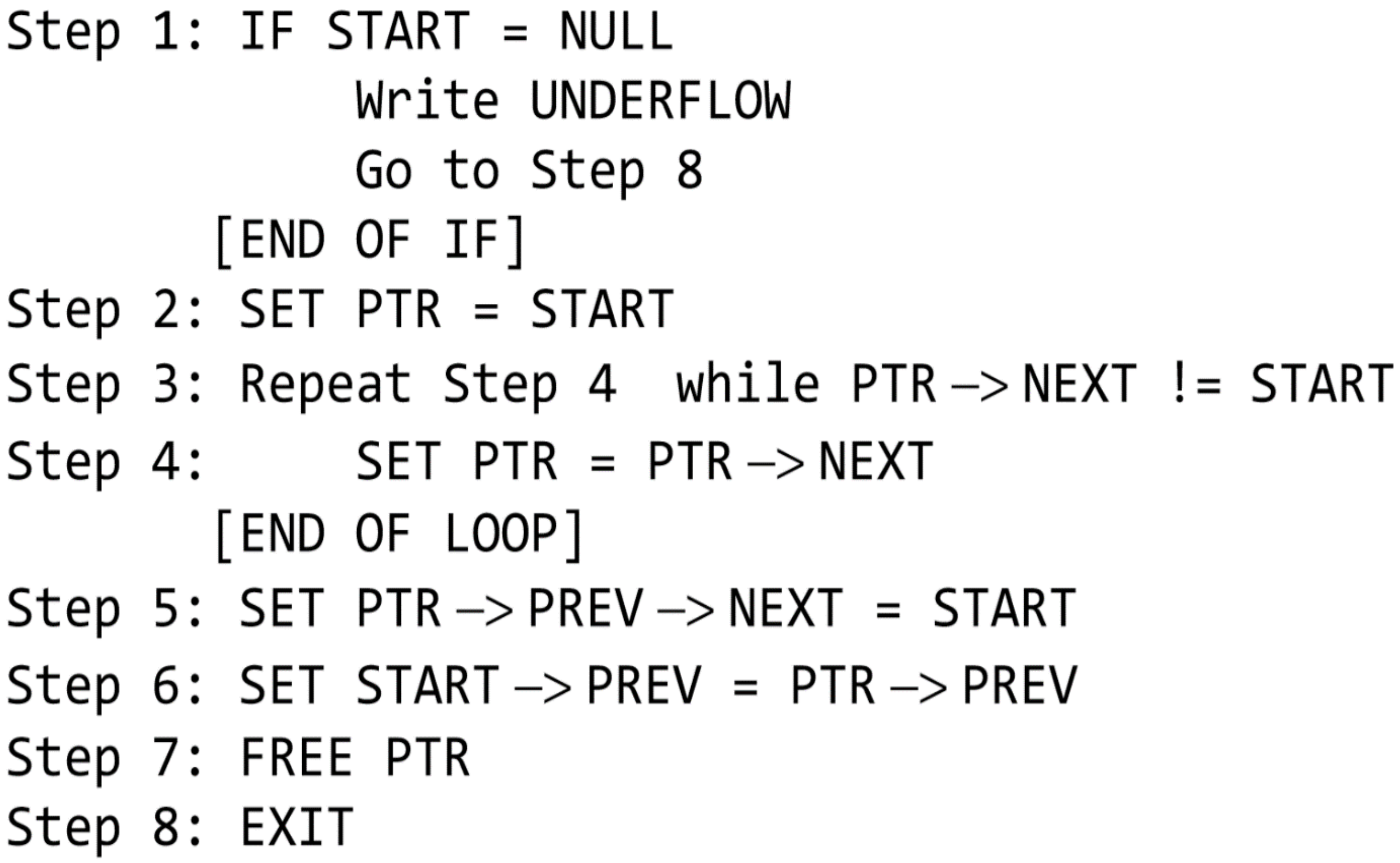
Move PTR further so that it now points to the last node of the list.



Free the space occupied by PTR.



Circular Doubly Linked Lists: Deletion at the end



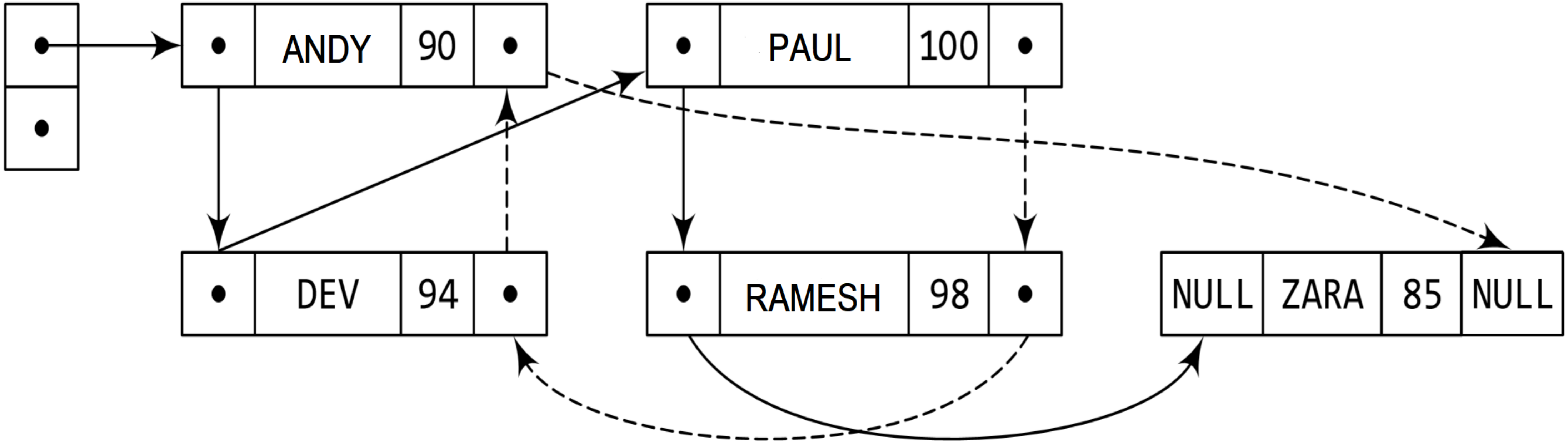
```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 8
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4  while PTR->NEXT != START
Step 4:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 5: SET PTR->PREV->NEXT = START
Step 6: SET START->PREV = PTR->PREV
Step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

Multi-Linked Lists

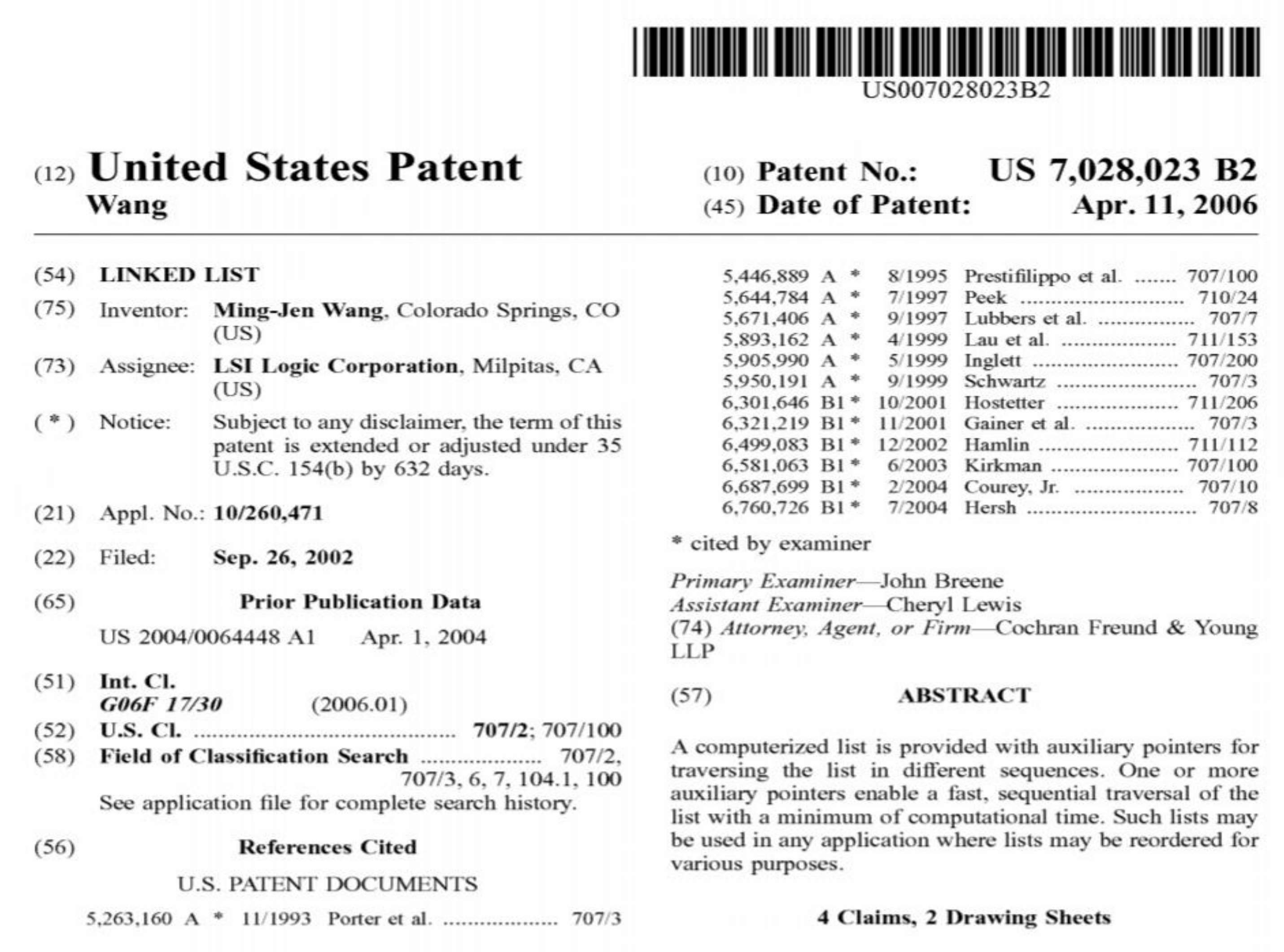
- Each node can have *n* number of pointers to other nodes.
- Already encountered ?

- Generally used to organize multiple orders of one set of elements.

Multi-Linked Lists



Linked Lists



US007028023B2

(12) **United States Patent**
Wang

(10) **Patent No.:** **US 7,028,023 B2**
(45) **Date of Patent:** **Apr. 11, 2006**

(54) **LINKED LIST**

(75) Inventor: **Ming-Jen Wang**, Colorado Springs, CO (US)

(73) Assignee: **LSI Logic Corporation**, Milpitas, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 632 days.

(21) Appl. No.: **10/260,471**

(22) Filed: **Sep. 26, 2002**

(65) **Prior Publication Data**

US 2004/0064448 A1 Apr. 1, 2004

(51) **Int. Cl.**
G06F 17/30 (2006.01)

(52) **U.S. Cl.** **707/2**; 707/100

(58) **Field of Classification Search** 707/2, 707/3, 6, 7, 104.1, 100
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,263,160	A *	11/1993	Porter et al.	707/3
5,446,889	A *	8/1995	Prestifilippo et al.	707/100
5,644,784	A *	7/1997	Peek	710/24
5,671,406	A *	9/1997	Lubbers et al.	707/7
5,893,162	A *	4/1999	Lau et al.	711/153
5,905,990	A *	5/1999	Inglett	707/200
5,950,191	A *	9/1999	Schwartz	707/3
6,301,646	B1 *	10/2001	Hostetter	711/206
6,321,219	B1 *	11/2001	Gainer et al.	707/3
6,499,083	B1 *	12/2002	Hamlin	711/112
6,581,063	B1 *	6/2003	Kirkman	707/100
6,687,699	B1 *	2/2004	Courey, Jr.	707/10
6,760,726	B1 *	7/2004	Hersh	707/8

* cited by examiner

Primary Examiner—John Breene
Assistant Examiner—Cheryl Lewis
(74) *Attorney, Agent, or Firm*—Cochran Freund & Young LLP

(57) **ABSTRACT**

A computerized list is provided with auxiliary pointers for traversing the list in different sequences. One or more auxiliary pointers enable a fast, sequential traversal of the list with a minimum of computational time. Such lists may be used in any application where lists may be reordered for various purposes.

4 Claims, 2 Drawing Sheets

Polynomial Representation

- $6x^3 + 9x^2 + 7x + 1$.