Arrays & Linked Lists

BY

Arun Cyril Jose

Arrays

- Almost always stored in consecutive memory locations and are referenced by an *index*.
- Collection of similar data elements.
- Set of pairs, index and value.

Array Operations

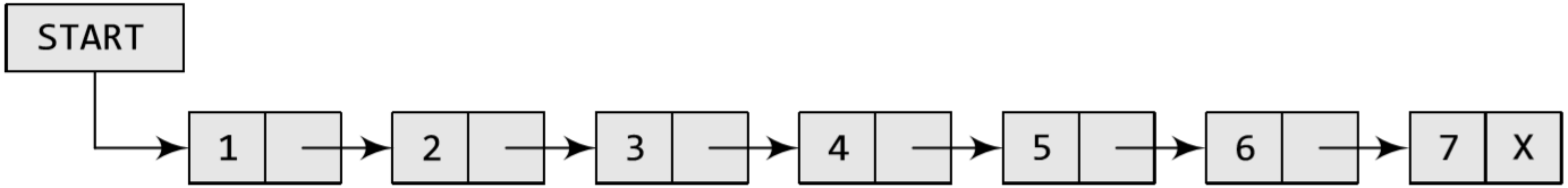
- Traversing an array
- Inserting an element in an array
- Searching an element in an array
- Deleting an element from an array
- Merging two arrays.
- Sorting an array in ascending or descending order.

Linked Lists

- Array elements are stored consecutively.
- Array has a maximum size limit.
- Does not store its elements in consecutive memory locations.
- User can add any number of elements to it.
- **Random access of data is not allowed**.
- Elements can only be accessed in a sequential manner.

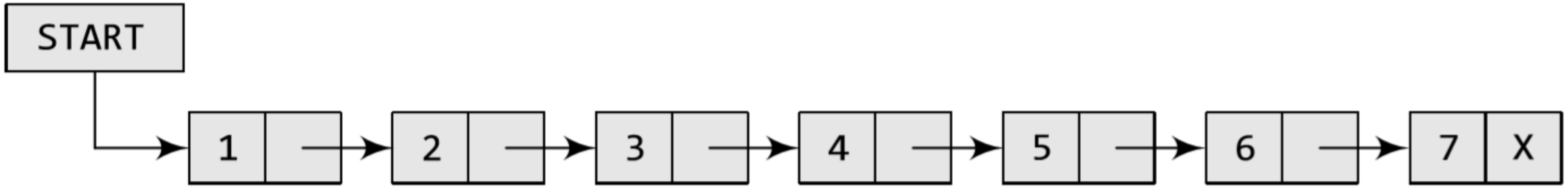
Linked Lists

- Linear collection of data elements.



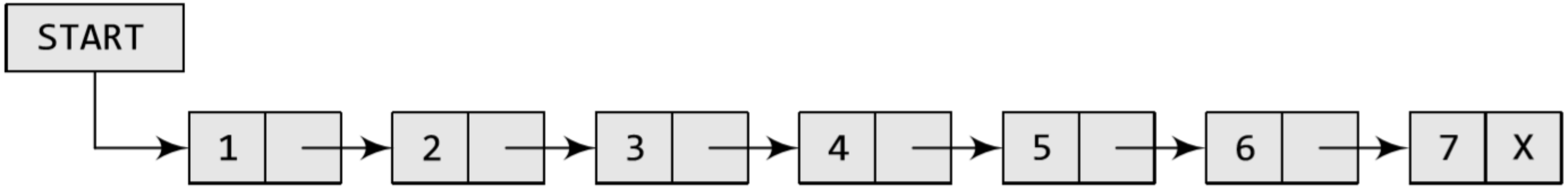
- Data elements → *nodes*.
- Chain of nodes.
- Each node has a data fields and a pointer to the next node.

Linked Lists



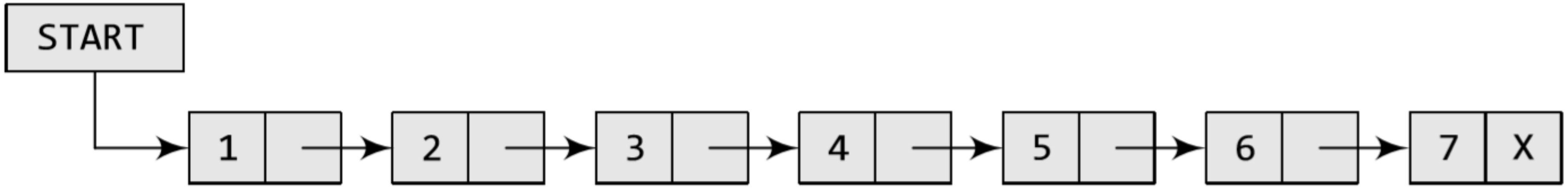
- Last node will have no next node connected to it, so it will store NULL.
- **Self-referential data type**: Every node contains a pointer to another node which is of the same type.
- START stores the address of the first node in the list.

Linked Lists



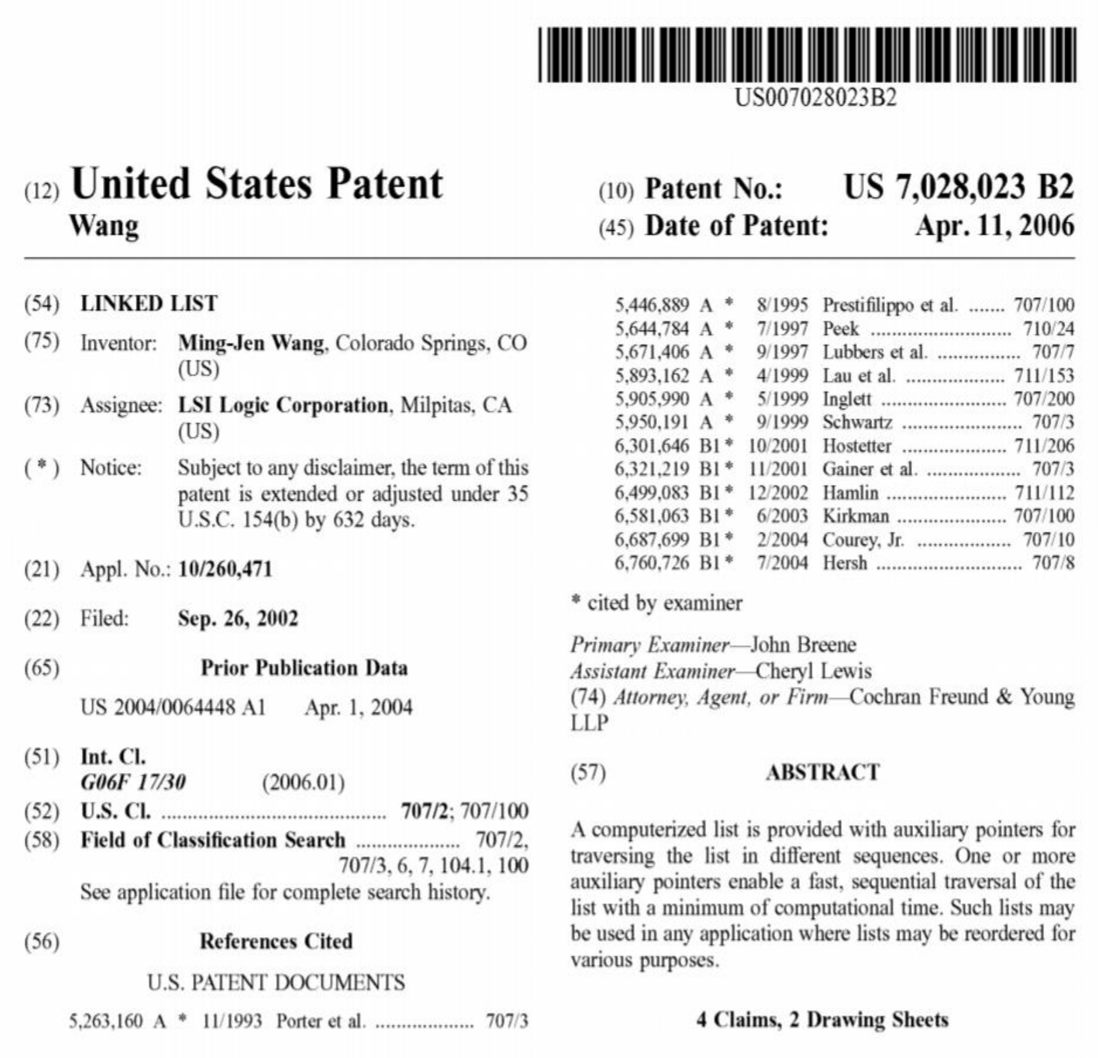
```
struct node  {
        int data;
        struct node *next;
};
```

Linked Lists



- How are linked lists stored in the memory??
- Can two linked lists coexist in the memory??
- Can the data part contain a structure ??

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

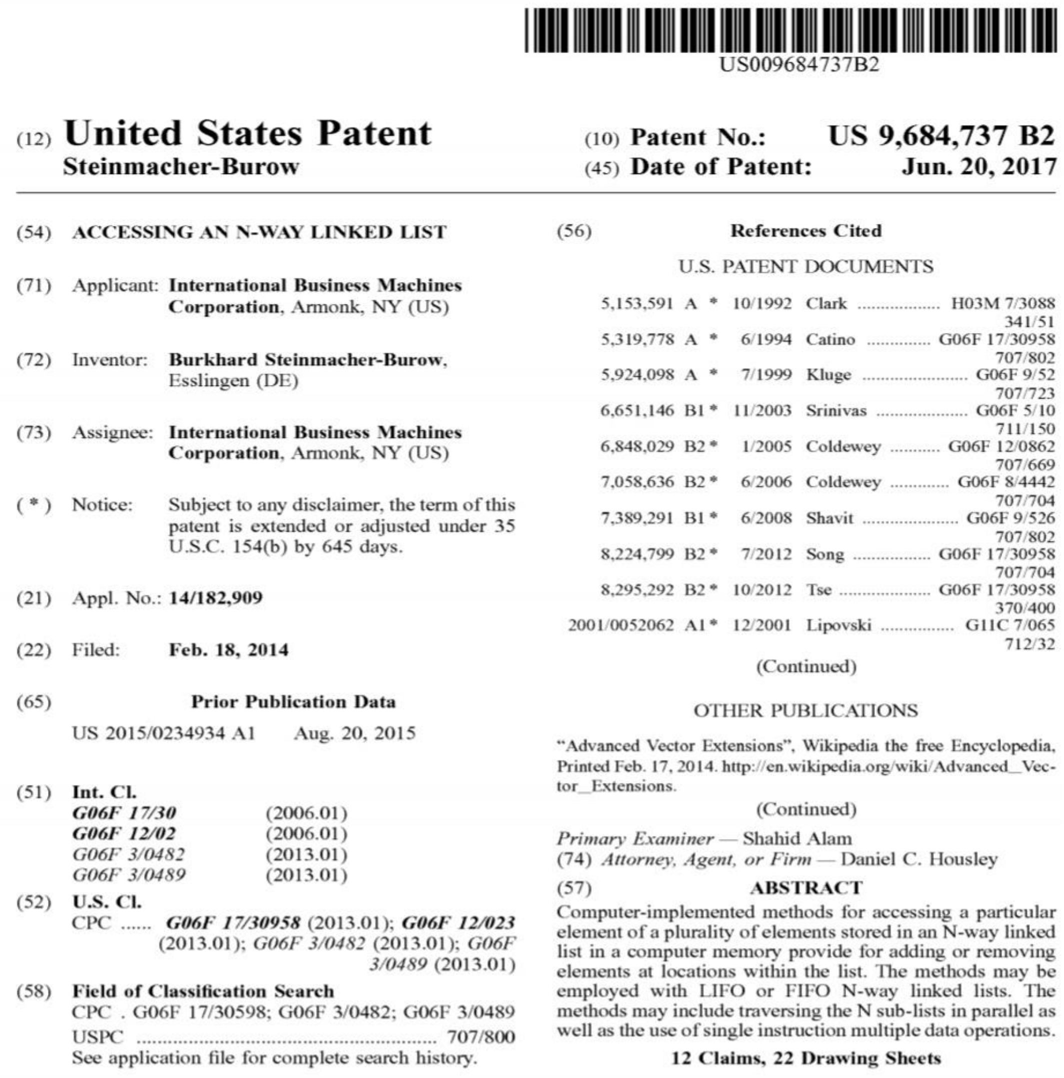
* cited by examiner

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



US009684737B2

(12) **United States Patent**
Steinmacher-Burow

(10) **Patent No.: US 9,684,737 B2**
(45) **Date of Patent: Jun. 20, 2017**

(54) **ACCESSING AN N-WAY LINKED LIST**

(71) Applicant: **International Business Machines Corporation**, Armonk, NY (US)

(72) Inventor: **Burkhard Steinmacher-Burow**, Esslingen (DE)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 645 days.

(21) Appl. No.: **14/182,909**

(22) Filed: **Feb. 18, 2014**

(65) **Prior Publication Data**

US 2015/0234934 A1 Aug. 20, 2015

(51) **Int. Cl.**
G06F 17/30 (2006.01)
G06F 12/02 (2006.01)
G06F 3/0482 (2013.01)
G06F 3/0489 (2013.01)

(52) **U.S. Cl.**
CPC ***G06F 17/30958*** (2013.01); ***G06F 12/023*** (2013.01); *G06F 3/0482* (2013.01); *G06F 3/0489* (2013.01)

(58) **Field of Classification Search**
CPC . G06F 17/30598; G06F 3/0482; G06F 3/0489
USPC 707/800
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,153,591	A *	10/1992	Clark	....	H03M 7/3088 341/51
5,319,778	A *	6/1994	Catino	....	G06F 17/30958 707/802
5,924,098	A *	7/1999	Kluge	....	G06F 9/52 707/723
6,651,146	B1 *	11/2003	Srinivas	....	G06F 5/10 711/150
6,848,029	B2 *	1/2005	Coldewey	....	G06F 12/0862 707/669
7,058,636	B2 *	6/2006	Coldewey	....	G06F 8/4442 707/704
7,389,291	B1 *	6/2008	Shavit	....	G06F 9/526 707/802
8,224,799	B2 *	7/2012	Song	....	G06F 17/30958 707/704
8,295,292	B2 *	10/2012	Tse	....	G06F 17/30958 370/400
2001/0052062	A1 *	12/2001	Lipovski	....	G11C 7/065 712/32

(Continued)

OTHER PUBLICATIONS

"Advanced Vector Extensions", Wikipedia the free Encyclopedia, Printed Feb. 17, 2014. http://en.wikipedia.org/wiki/Advanced_Vector_Extensions.

(Continued)

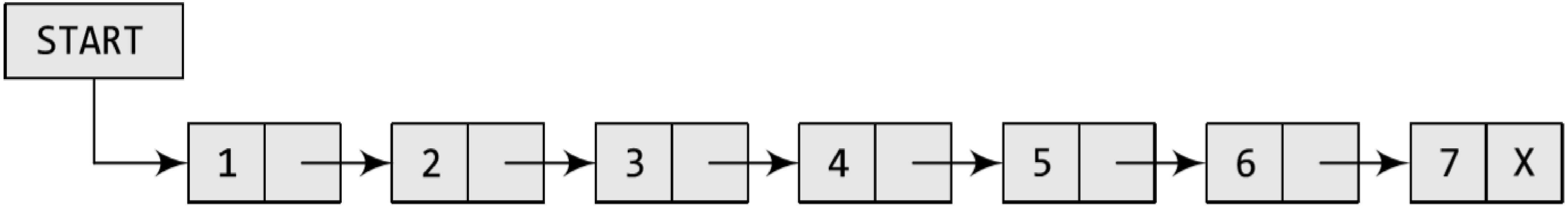
Primary Examiner — Shahid Alam
(74) *Attorney, Agent, or Firm* — Daniel C. Housley

(57) **ABSTRACT**

Computer-implemented methods for accessing a particular element of a plurality of elements stored in an N-way linked list in a computer memory provide for adding or removing elements at locations within the list. The methods may be employed with LIFO or FIFO N-way linked lists. The methods may include traversing the N sub-lists in parallel as well as the use of single instruction multiple data operations.

12 Claims, 22 Drawing Sheets

Linked Lists: Traversing



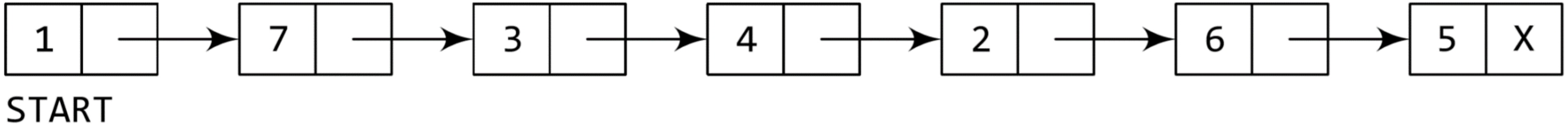
```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Steps 3 and 4 while PTR != NULL
Step 3:           Apply Process to PTR -> DATA
Step 4:           SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: EXIT
```

Linked Lists: Insertion

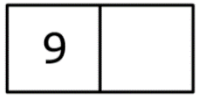
- New node is inserted at the beginning.
- New node is inserted at the end.
- New node is inserted after a given node.
- New node is inserted before a given node.

- **Overflow** occurs when no free memory cell is present in the system.

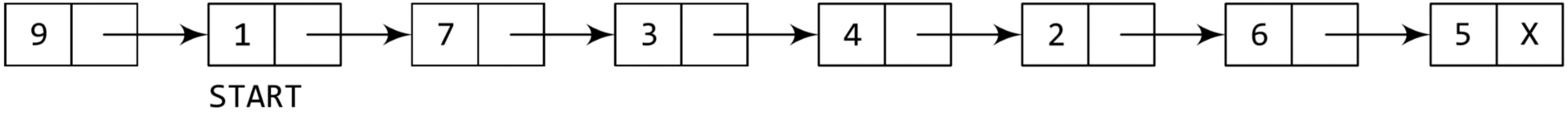
Linked Lists: Insertion at the Beginning



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

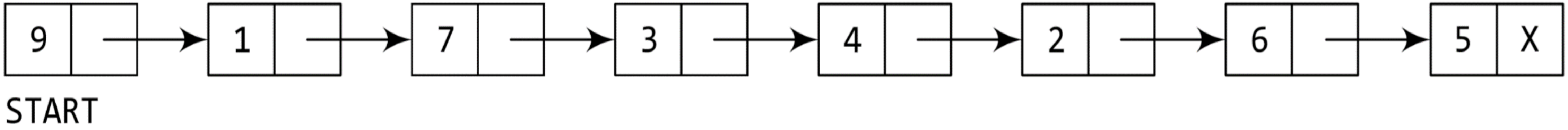


Add the new node as the first node of the list by making the NEXT part of the new node contain the address of START.

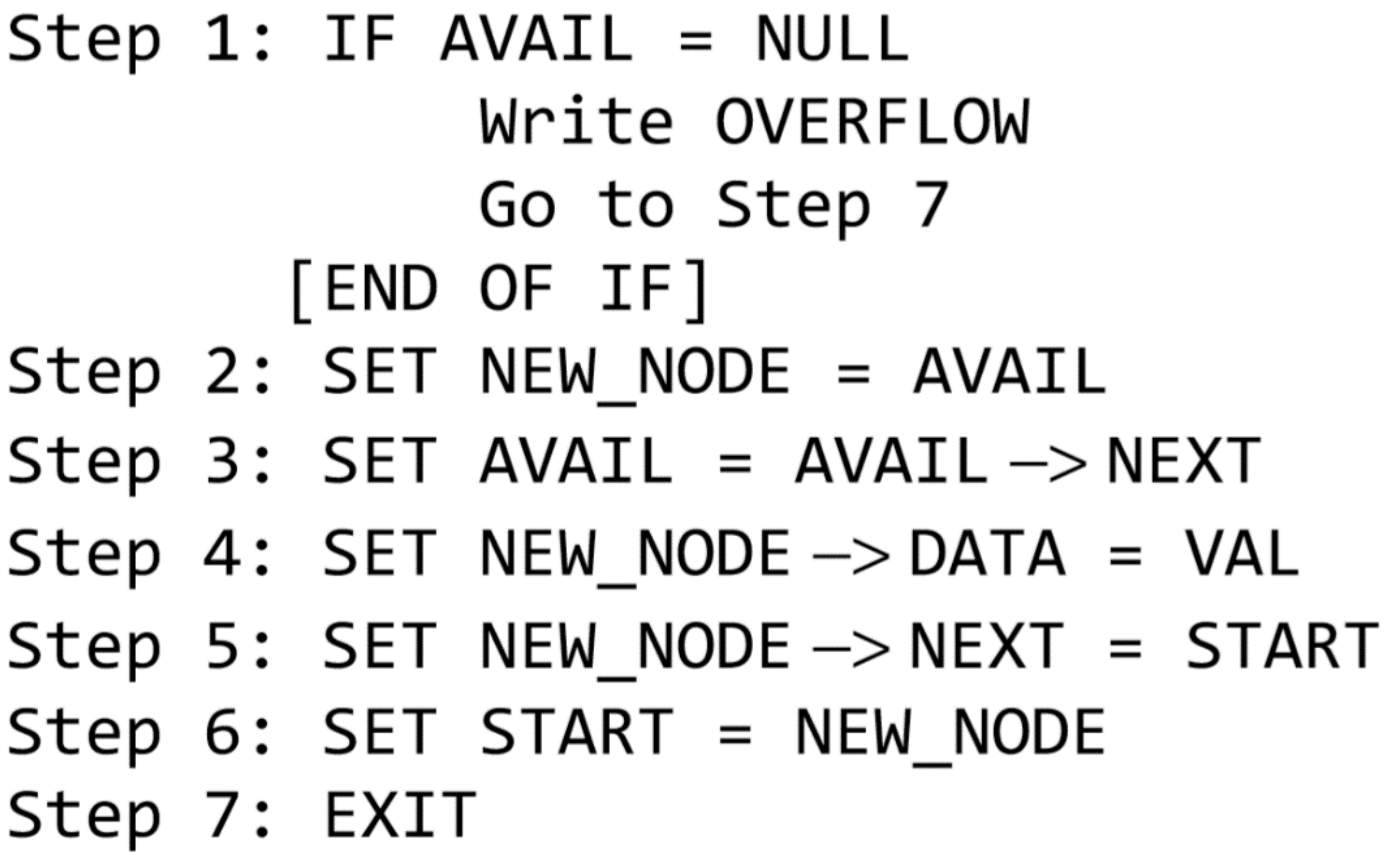


Linked Lists: Insertion at the Beginning

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

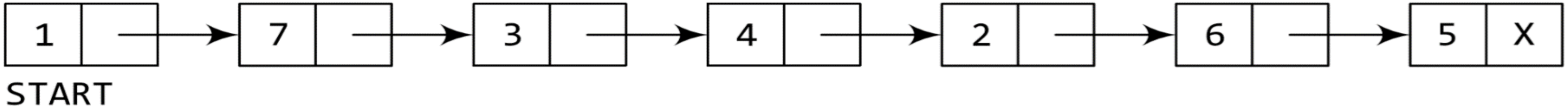


Linked Lists: Insertion at the Beginning

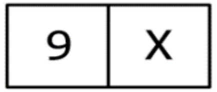


```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 7
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET START = NEW_NODE
Step 7: EXIT
```

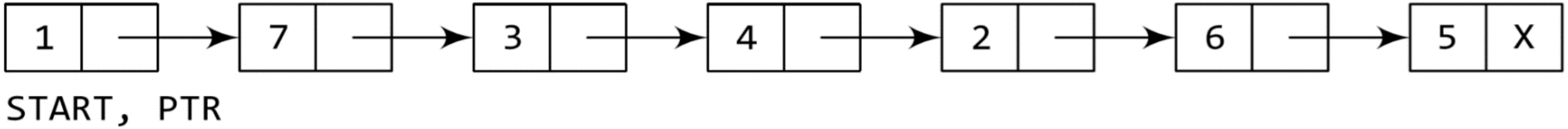
Linked Lists: Insertion at the End



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

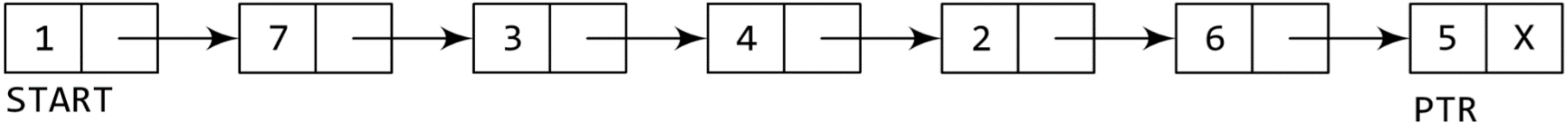


Take a pointer variable PTR which points to START.

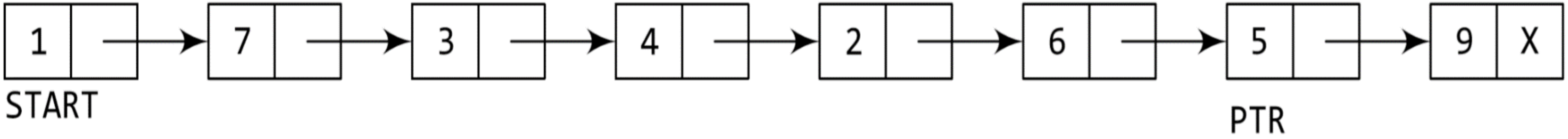


Linked Lists: Insertion at the End

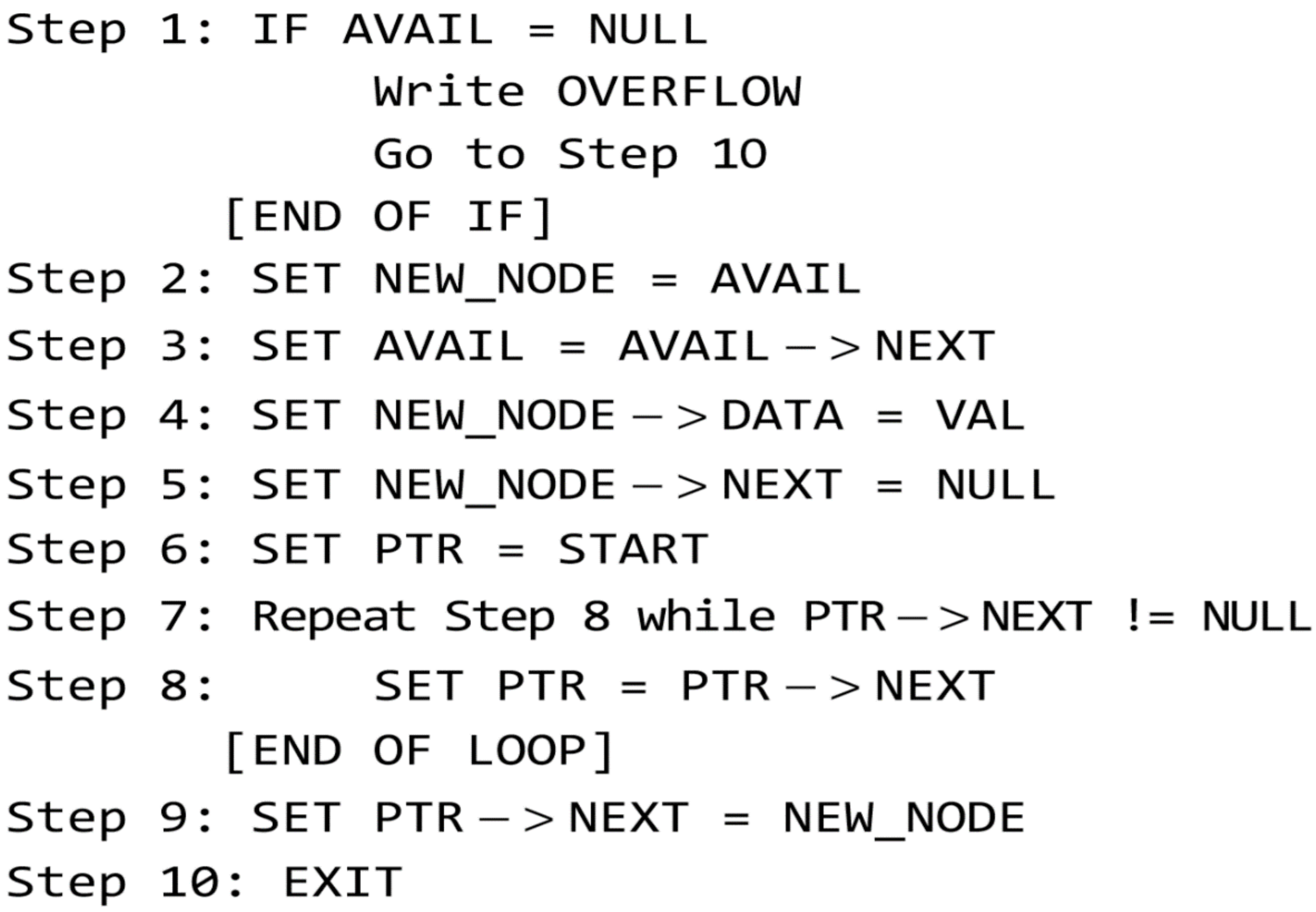
Move PTR so that it points to the last node of the list.



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

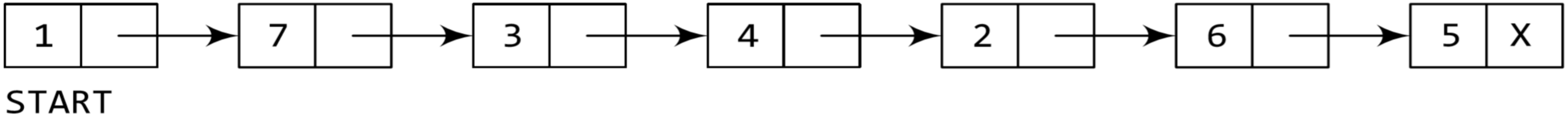


Linked Lists: Insertion at the End

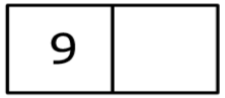


```
Step 1: IF AVAIL = NULL
              Write OVERFLOW
              Go to Step 10
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET NEW_NODE->DATA = VAL
Step 5: SET NEW_NODE->NEXT = NULL
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR->NEXT != NULL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

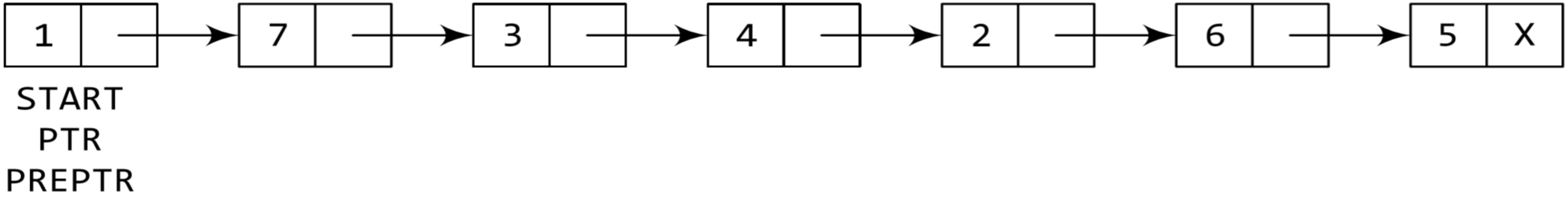
Linked Lists: Insertion After a Given Node



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

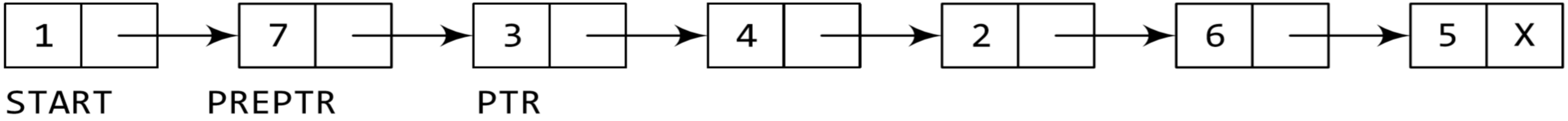


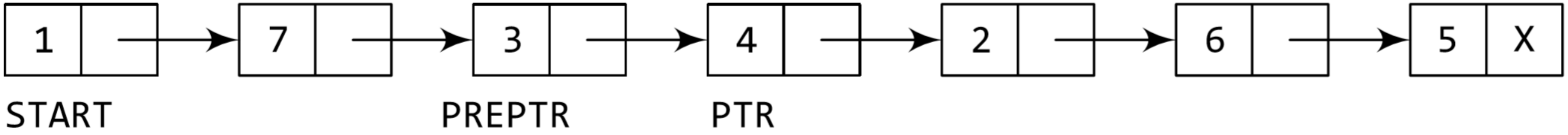
Take two pointer variables PTR and PREPTR and initialize them with START so that START, PTR, and PREPTR point to the first node of the list.



Linked Lists: Insertion After a Given Node

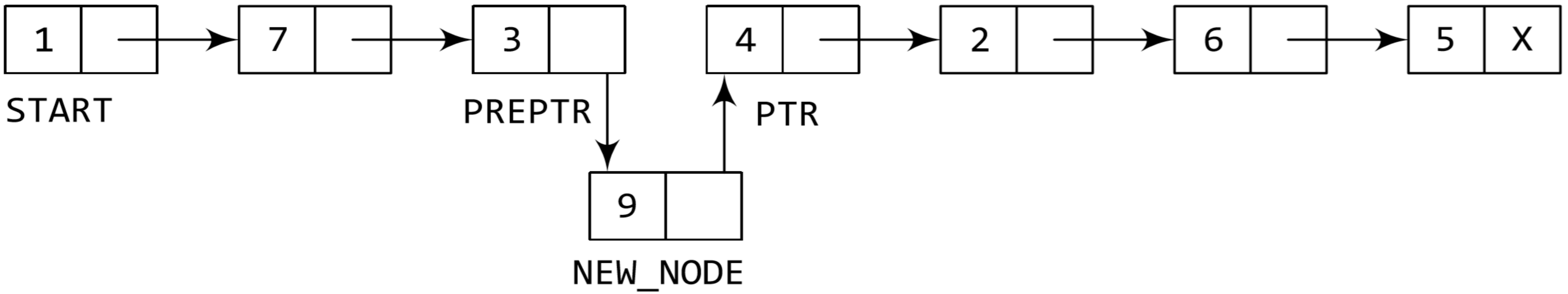
Move PTR and PREPTR until the DATA part of PREPTR = value of the node after which insertion has to be done. PREPTR will always point to the node just before PTR.

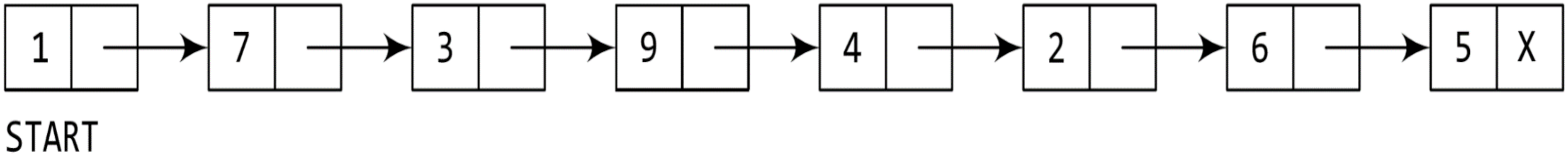




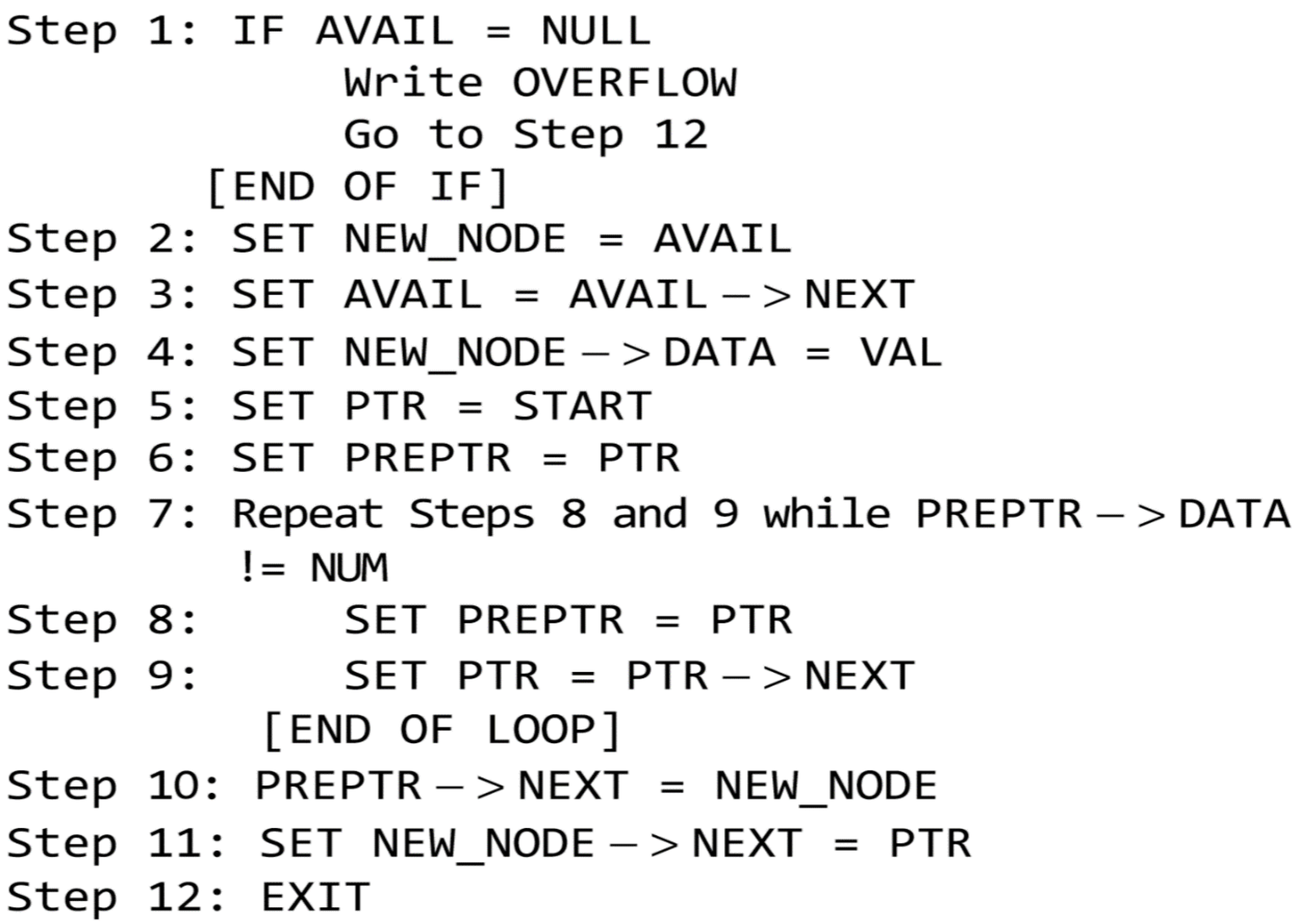
Linked Lists: Insertion After a Given Node

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





Linked Lists: Insertion After a Given Node



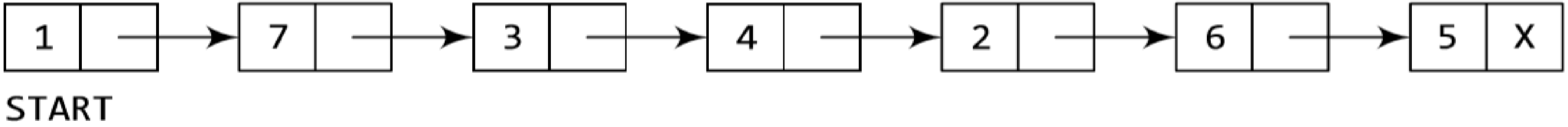
```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PREPTR -> DATA
  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

Linked Lists: Deletion

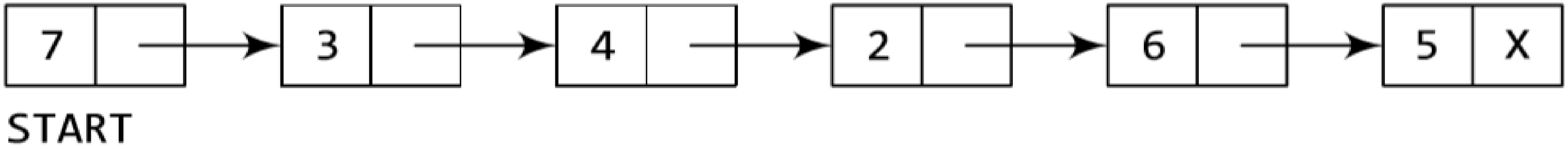
- First node is deleted.
- Last node is deleted.
- Node after a given node is deleted.

- **Underflow** occurs when we try to delete a node from a linked list that is empty.
- When we delete a node from a linked list, **we have to free the memory** occupied by that node.

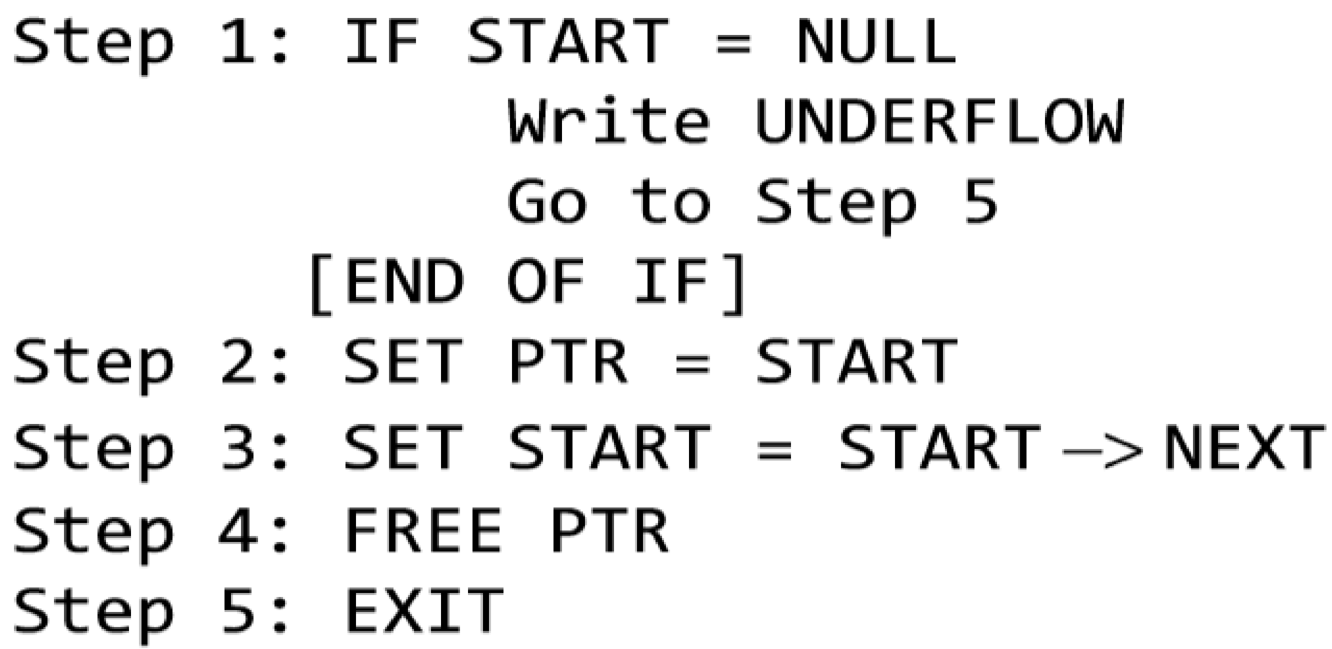
Linked Lists: Deletion of First Node



Make START to point to the next node in sequence.

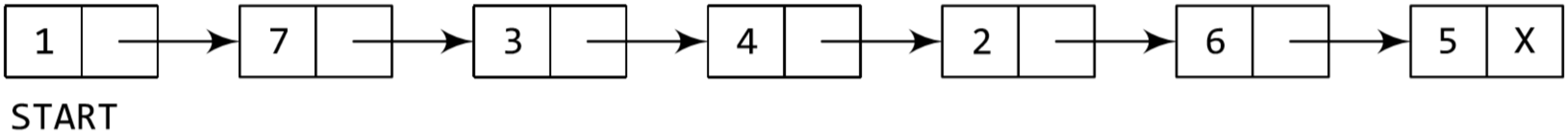


Linked Lists: Deletion of First Node

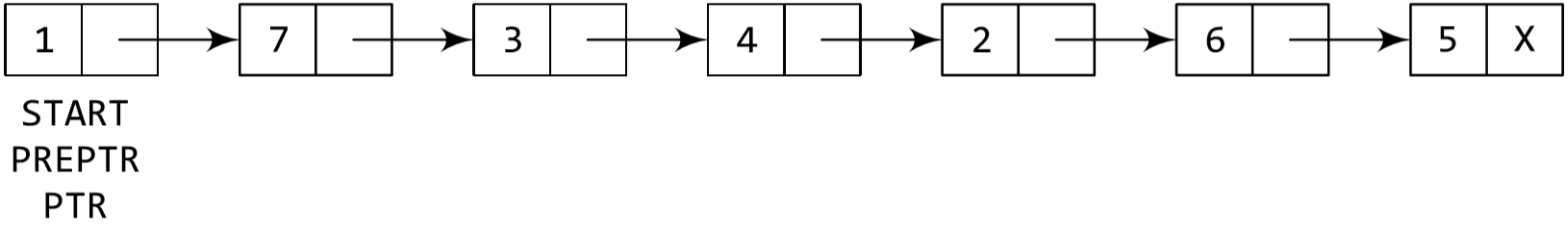


```
Step 1: IF START = NULL
                Write UNDERFLOW
                Go to Step 5
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET START = START -> NEXT
Step 4: FREE PTR
Step 5: EXIT
```

Linked Lists: Deletion of Last Node

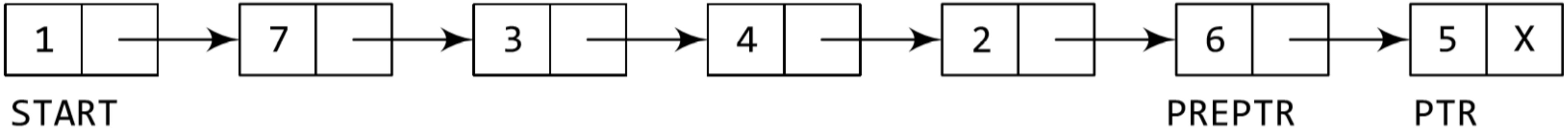


Take pointer variables PTR and PREPTR which initially point to START.

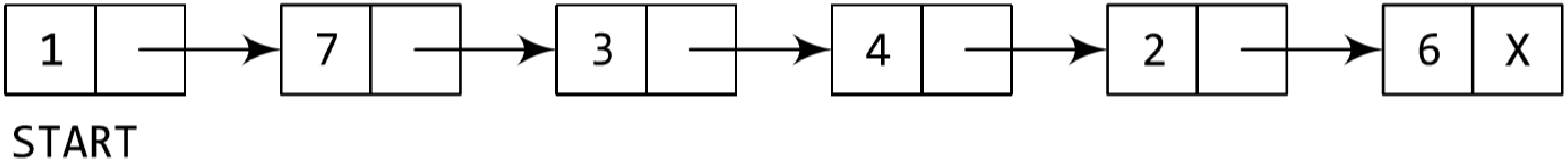


Linked Lists: Deletion of Last Node

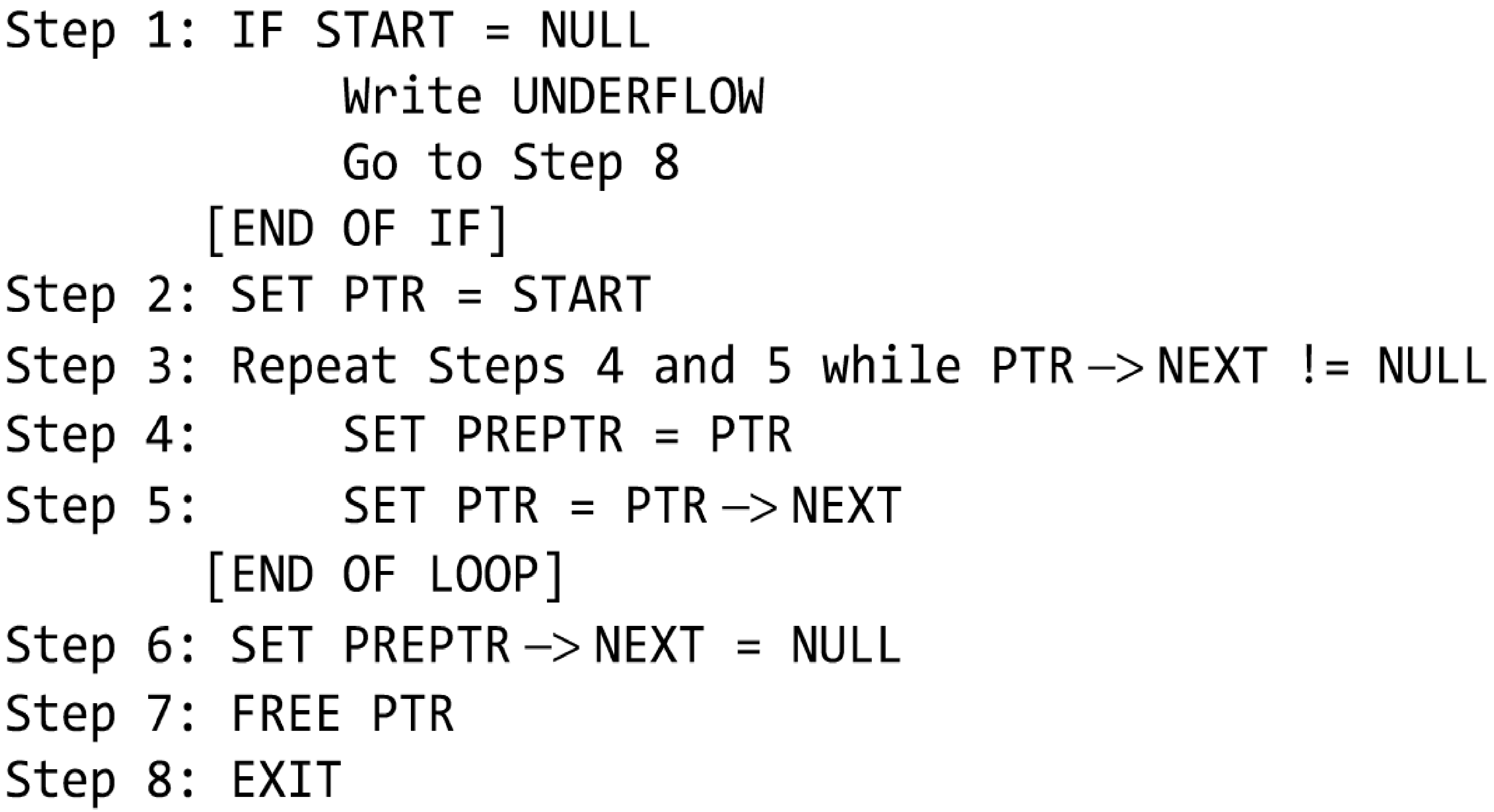
Move PTR and PREPTR such that NEXT part of PTR = NULL. PREPTR always points to the node just before the node pointed by PTR.



Set the NEXT part of PREPTR node to NULL.

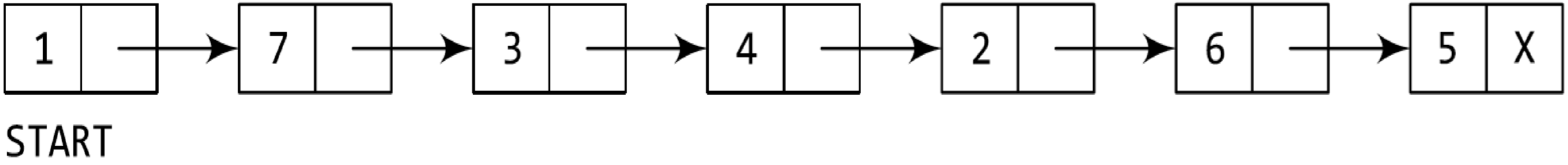


Linked Lists: Deletion of Last Node

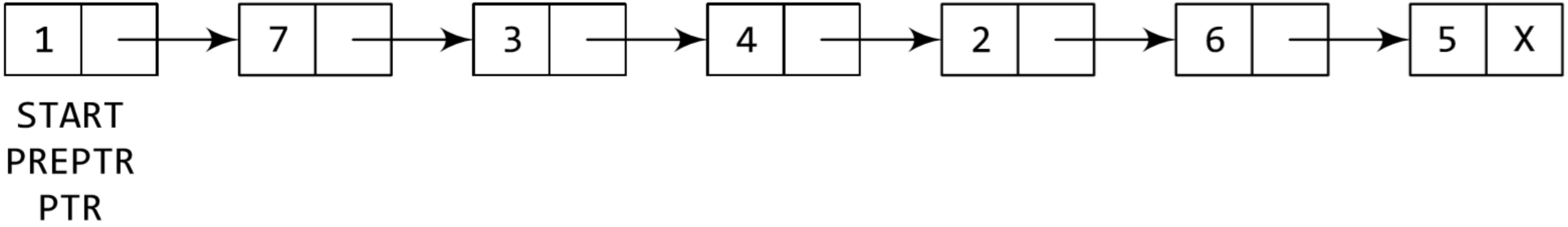


```
Step 1: IF START = NULL
              Write UNDERFLOW
              Go to Step 8
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR -> NEXT != NULL
Step 4:       SET PREPTR = PTR
Step 5:       SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 6: SET PREPTR -> NEXT =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

Linked Lists: Deletion of a Node After a Specific Node

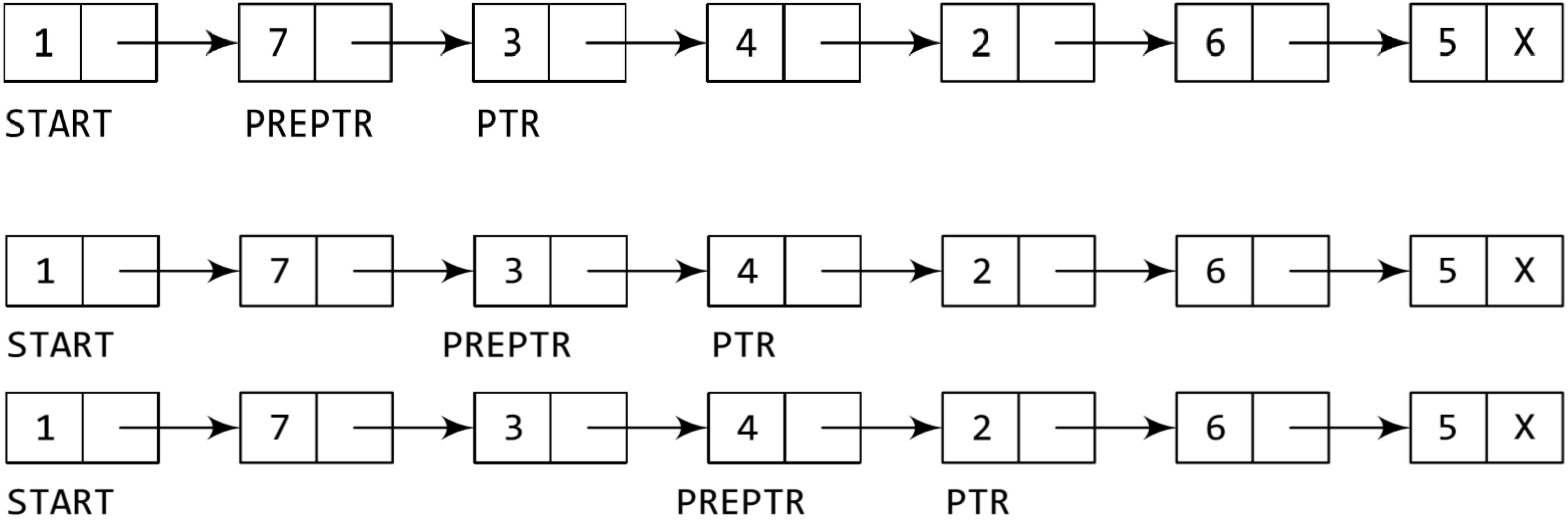


Take pointer variables PTR and PREPTR which initially point to START.

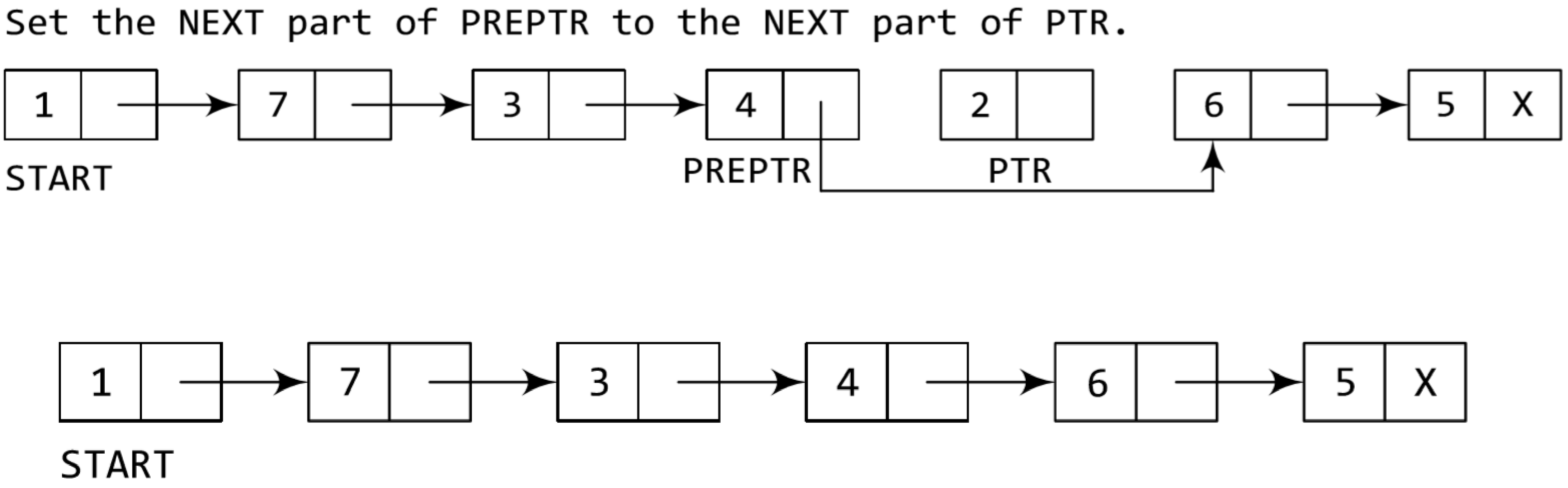


Linked Lists: Deletion of a Node After a Specific Node

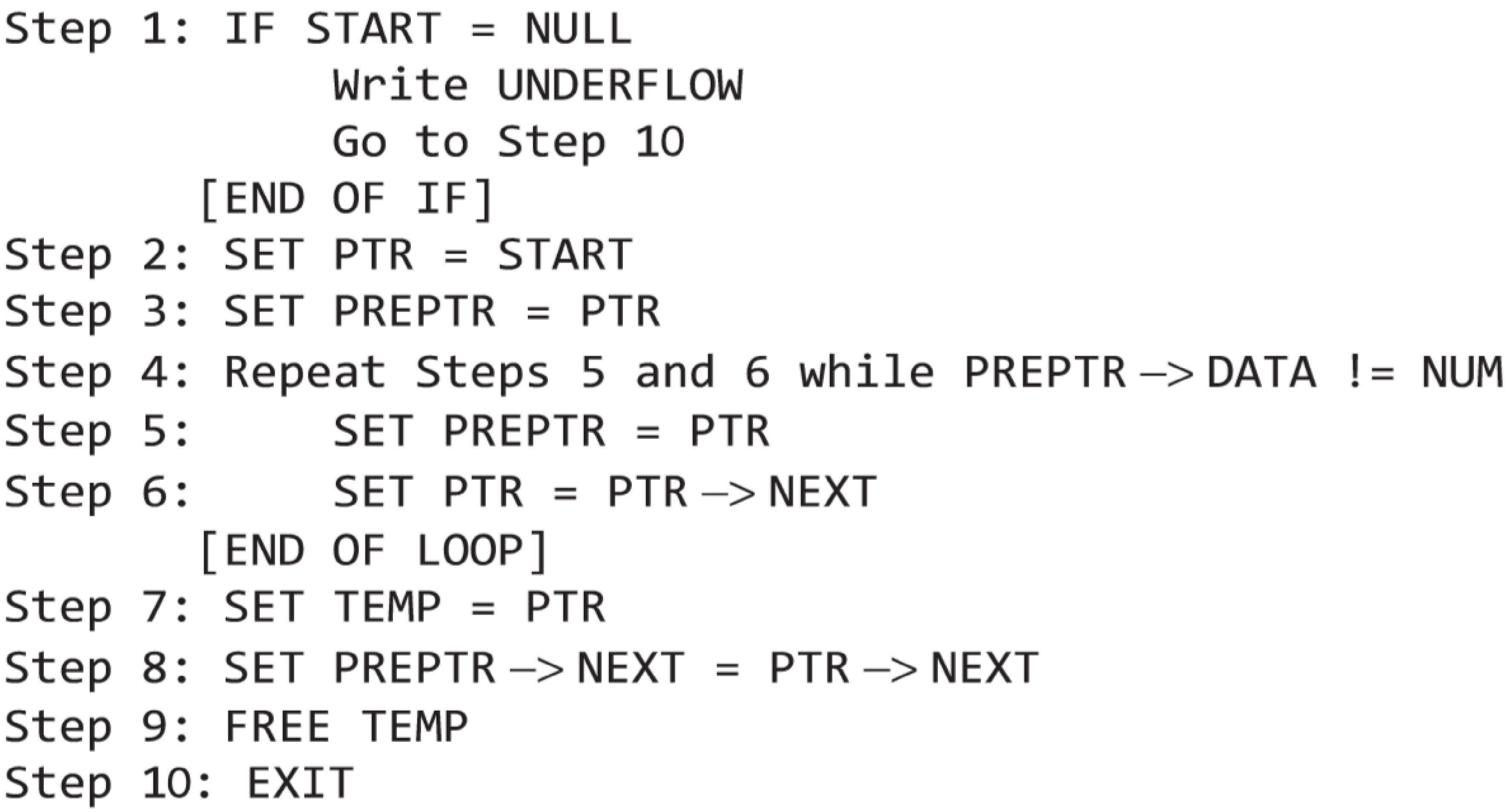
Move PREPTR and PTR such that PREPTR points to the node containing VAL and PTR points to the succeeding node.



Linked Lists: Deletion of a Node After a Specific Node



Linked Lists: Deletion of a Node After a Specific Node



```
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
```