Unit-4

Linked List

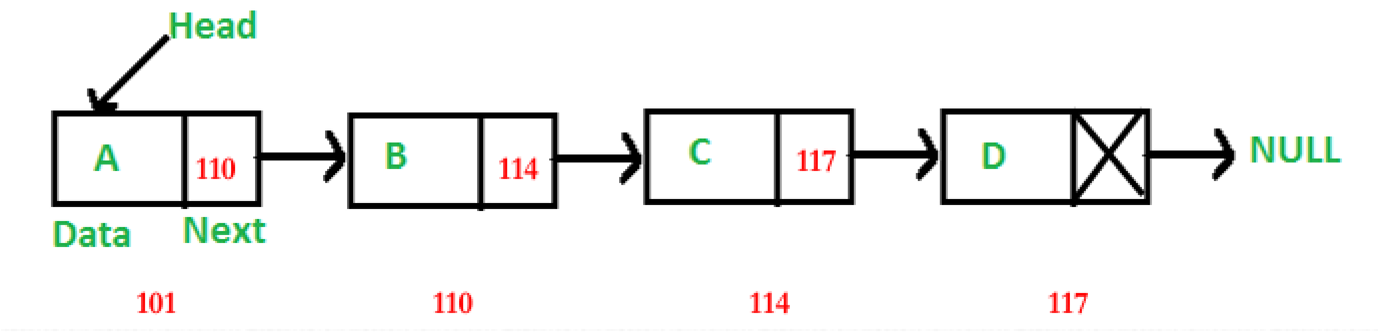
Contents

Differences between Array based and linked based implementation

	Array	Linked List
Definition	Array is a collection of elements having same data type with common name	Linked list is an ordered collection of elements which are connected by links/pointers
Access	Elements can be accessed using index/subscript, random access	Sequential access
Memory structure	Elements are stored in contiguous memory locations	Elements are stored at available memory space
Insertion & Deletion	Insertion and deletion take more timein array	Insertion and deletion are fast and easy
Memory Allocation	Memory is allocated at compile time i.e static memory allocation	Memory is allocated at run time i.edynamic memory allocation
Types	1D,2D, multi-dimensional	SLL, DLL circular linked list
Dependency	Each element is independent	Each node is dependent on each other as address part contains address of next node in the list

Introduction

- Like arrays, Linked List is a linear data structure.
- Unlike arrays, linked list elements are not stored at a contiguous location;
- the elements are linked using pointers



Why Linked List?

- Arrays can be used to store linear data of similar types, but arrays have the following limitations.

1) The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage.

2) Inserting a new element in an array of elements is expensive because the room has to

be created for the new elements and to create room existing elements have to be shifted in a system,

if we maintain a sorted list of IDs in an array id[]. id[] = [1000, 1010, 1050, 2000, 2040].

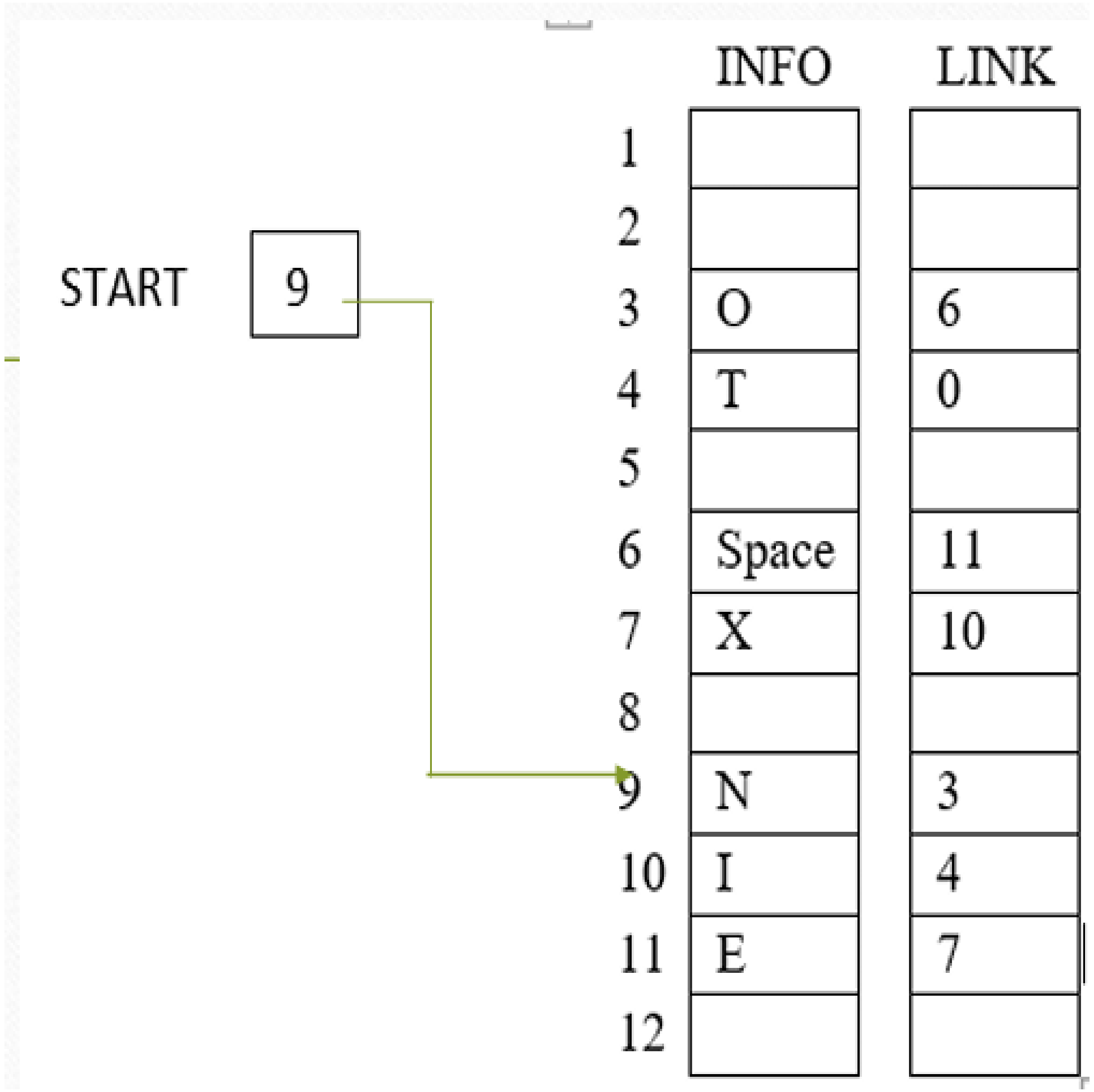
if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000).

Deletion is also expensive with arrays until unless some special techniques are used. For example, to delete 1010 in id[], everything after 1010 has to be moved

Representation:

- A linked list is represented by a pointer to the first node of the linked list. The first node is called the head.
- If the linked list is empty, then the value of the head is NULL.
- Each node in a list consists of at least two parts: 1) data 2) Pointer (Or Reference) to the next node
- In C, we can represent a node using structures. Below is an example of a linked list node with integer data.

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



Let LIST be a linked list. Then LIST will be maintained in memory specified as follows.
First of all, LIST requires two linear arrays we will call them here INFO and LINK- such that INFO[K] and LINK[K] contain, respectively, the information part and the nextpointer field of a node LIST.
LIST also requires a variable name- such as START.
START contains the location of the beginning of the list, and a nextpointer sentinel -denoted by NULL- which indicate the end of the list. Since the subscripts of the array INFO and LINK are usually positive, we will choose NULL=0

START pointing to the first element of the linked list in the memory

- Above picture is of linked list in memory where each node of the list contains a single character. We can obtain the actual list of characters as follows.

START= 9, so INFO[9]=N is the first character.

LINK[9]=3, so INFO[3]=O is the second character.

LINK[3]=6, so INFO[6]= (Blank) is the third character.

LINK[6]=11, so INFO[11]=E is the fourth character.

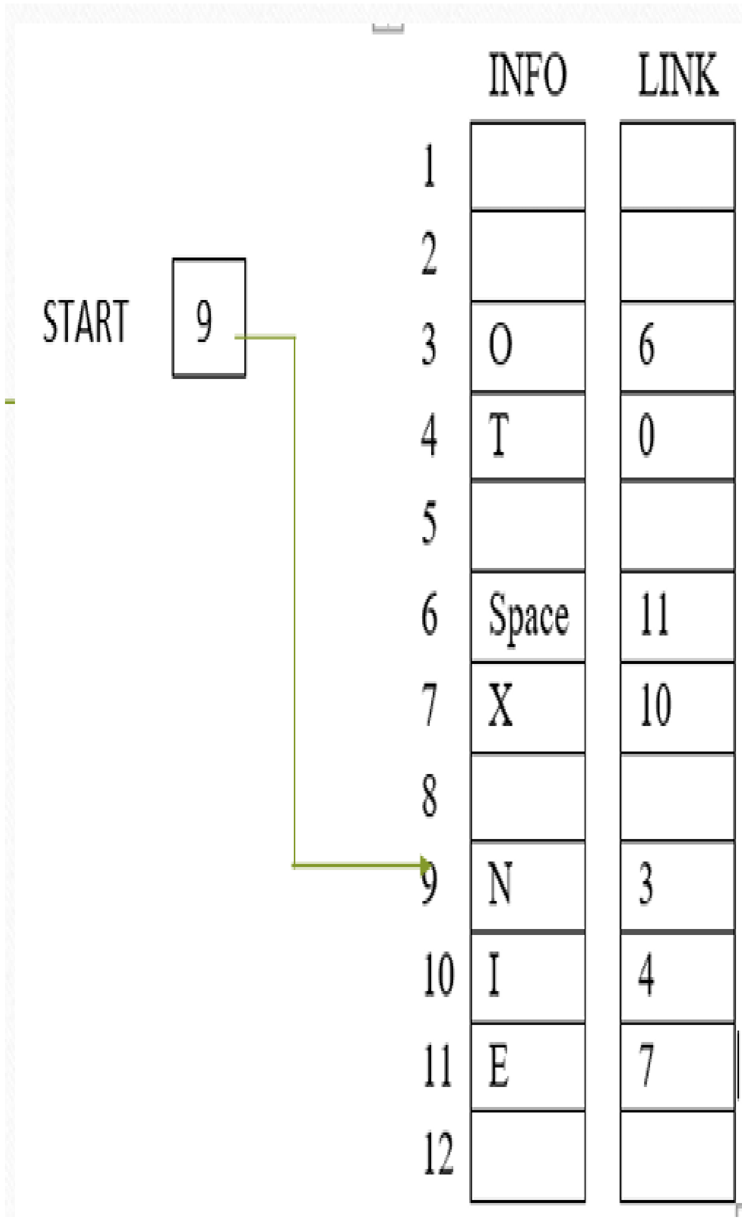
LINK[11]=7, so INFO[7]=X is the fifth character.

LINK[7]=10, so INFO[10]=I is the sixth character.

LINK[10]=4, so INFO[4]=T is the seventh character.

LINK[4]=0, the NULL value, so the List has ended.

In other words, NO EXIT is the character string

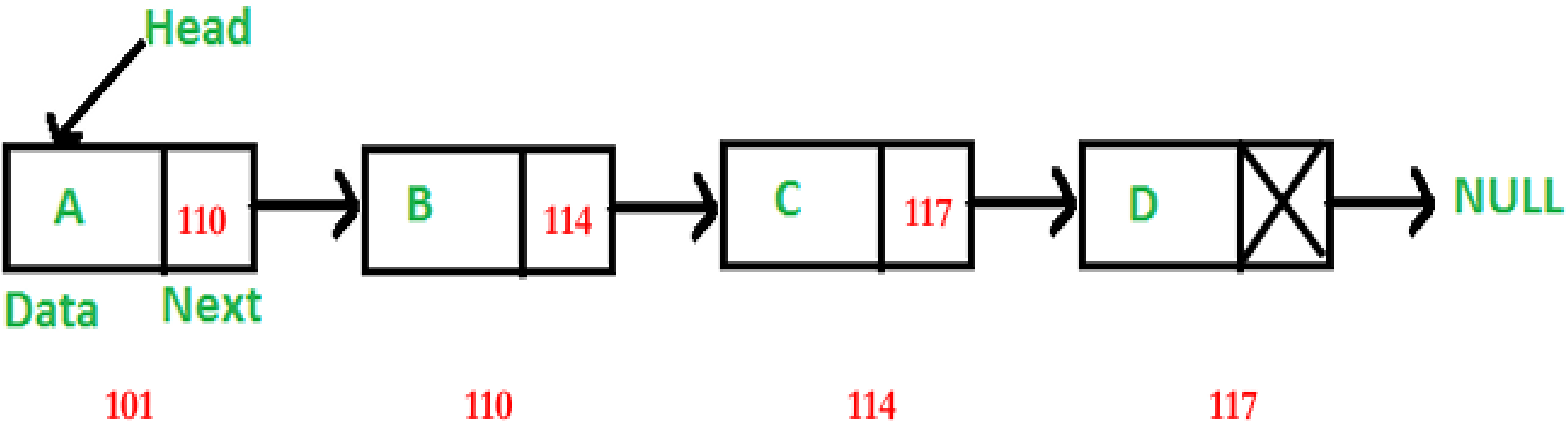


Types of Linked List

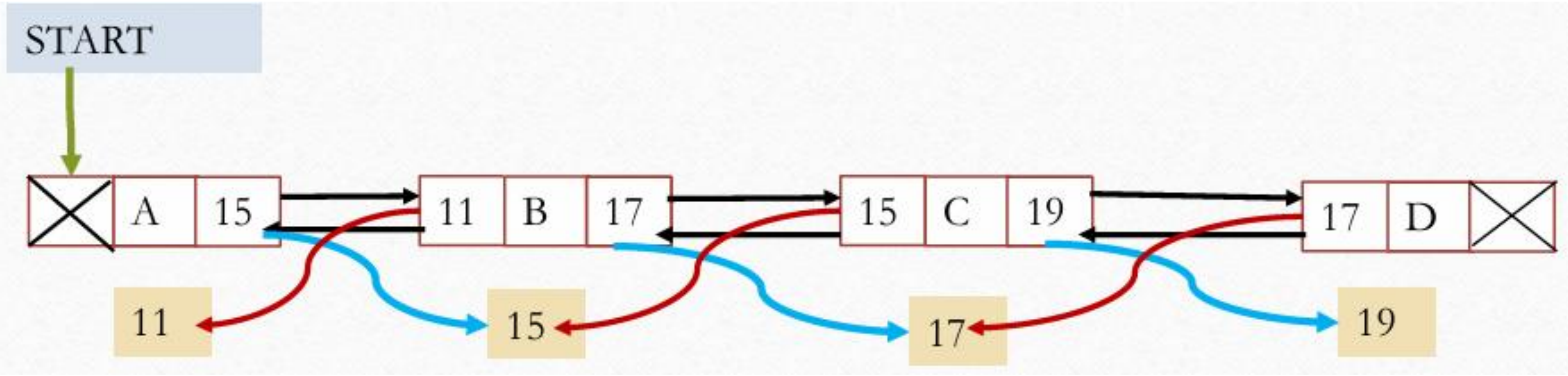
Following are the various types of linked list.

- Simple Linked List − Item navigation is forward only.
- Doubly Linked List − Items can be navigated forward and backward.
- Circular Linked List − Last item contains link of the first element as next and the first element has a link to the last element as previous

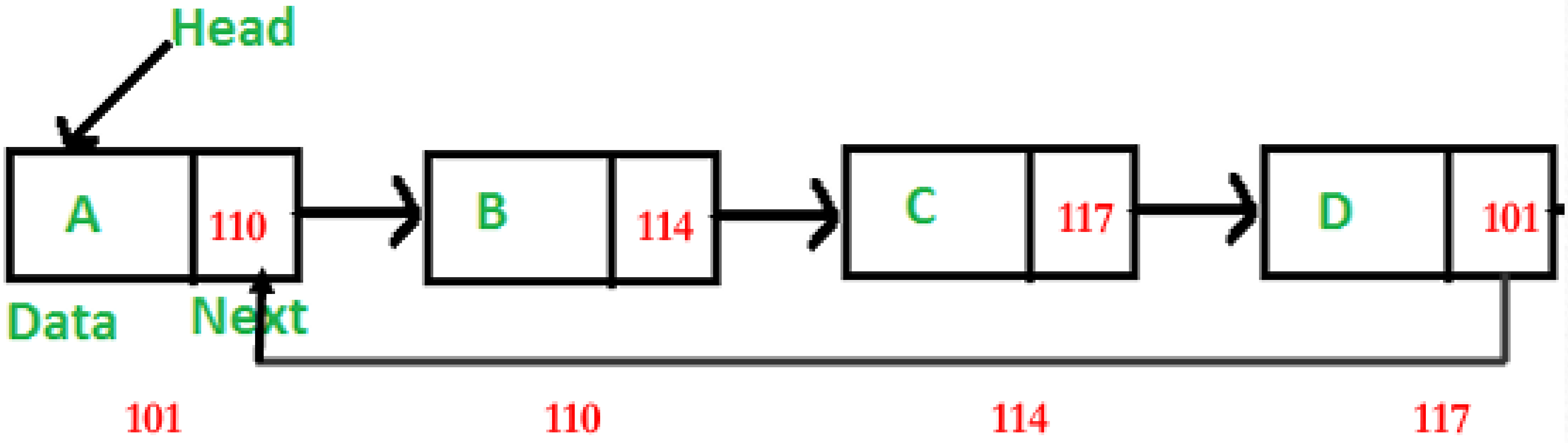
Simple Linked List



Doubly Linked List

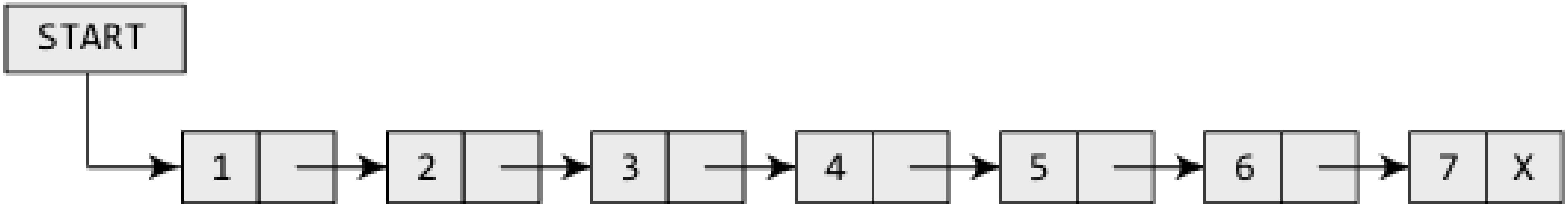


Circular Linked List



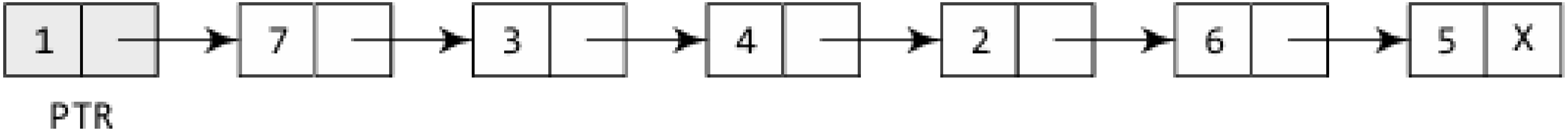
Singly Linked List

- 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 sequence. A singly linked list allows traversal of data only in one way



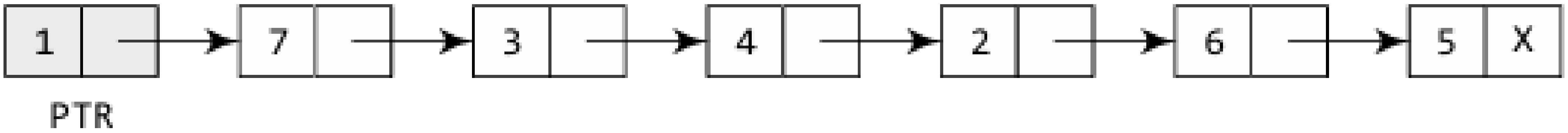
Traversing a Linked List

```
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
```



To print the number of nodes in a linked list

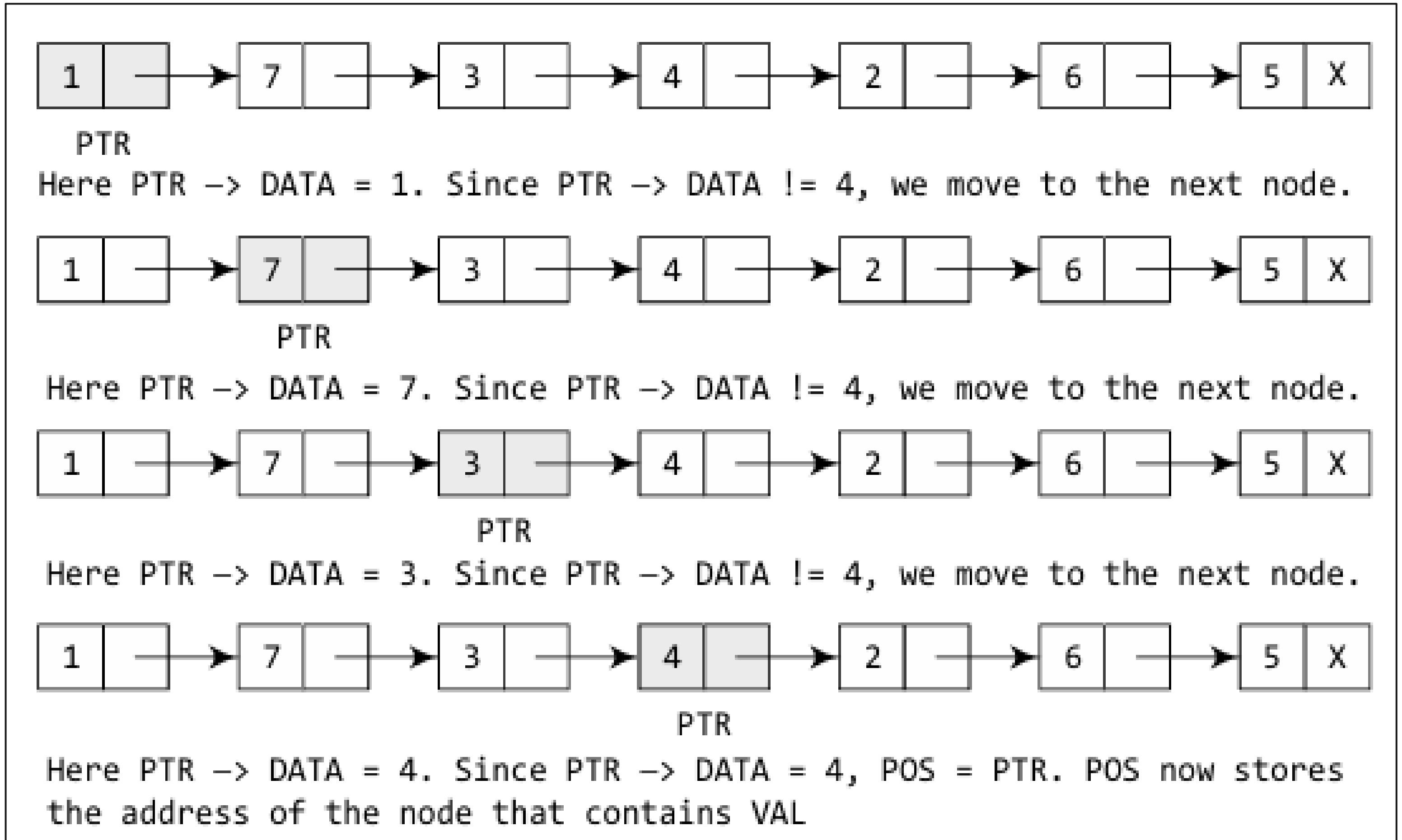
```
Step 1: [INITIALIZE] SET COUNT = 0
Step 2: [INITIALIZE] SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR != NULL
Step 4:             SET COUNT = COUNT + 1
Step 5:             SET PTR = PTR -> NEXT
       [END OF LOOP]
Step 6: Write COUNT
Step 7: EXIT
```



Searching for a value in a Linked 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
```

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



Inserting a new node in a Linked List

Case 1: The new node is inserted at the beginning.

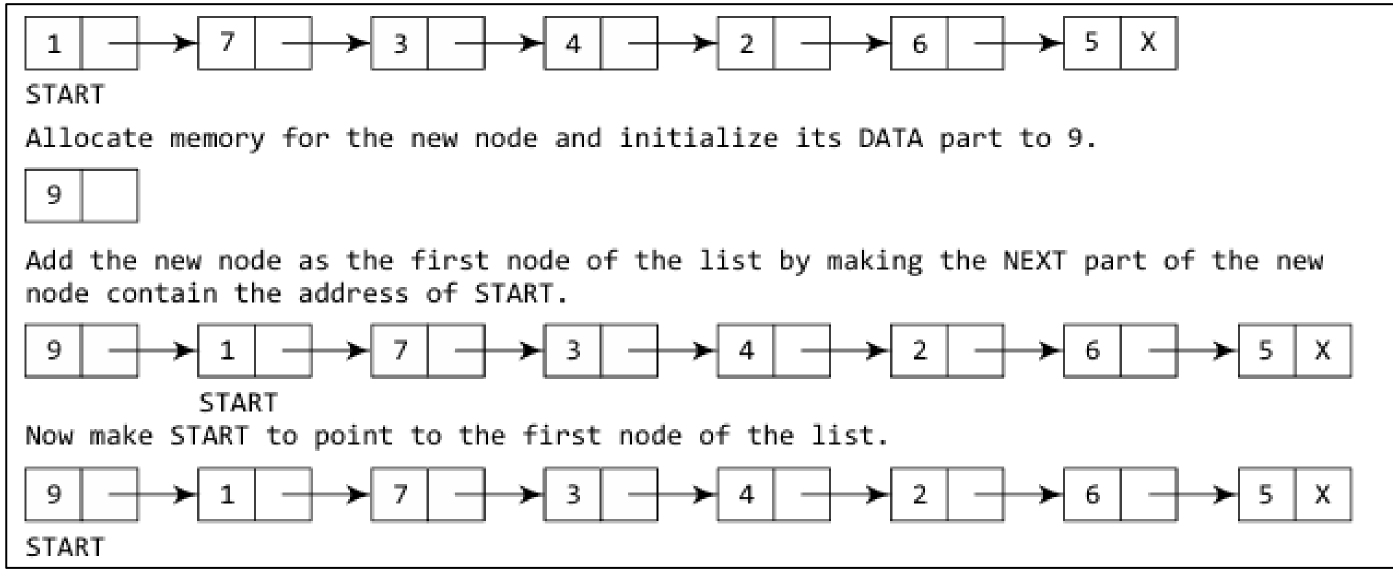
Case 2: The new node is inserted at the end.

Case 3: The new node is inserted after a given node.

Case 4: The new node is inserted before a given node

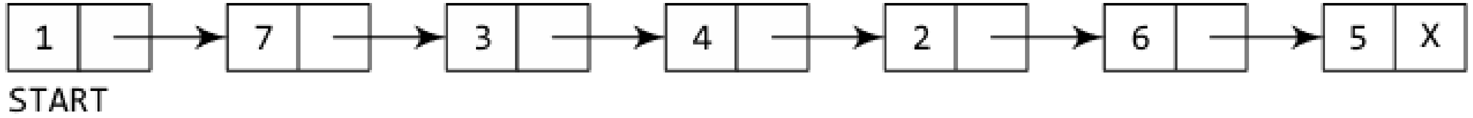
Inserting a Node at the Beginning of a Linked List

```
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
```

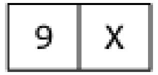


Inserting a Node at the End of a Linked List

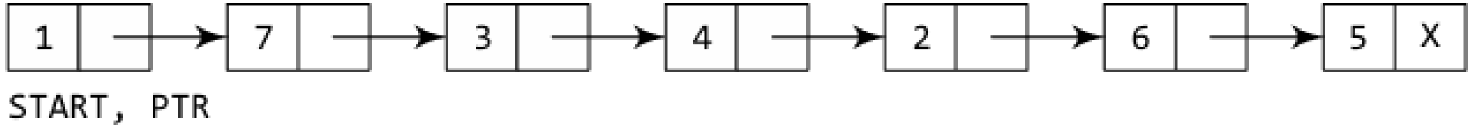
- Insert node 9 at end



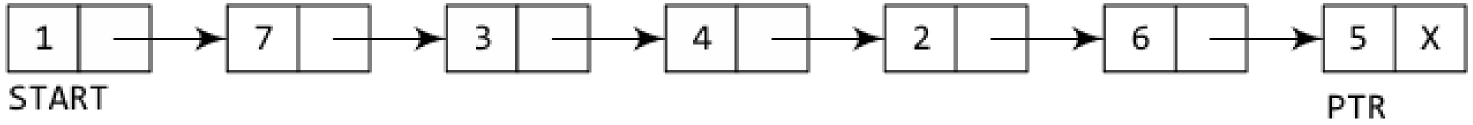
Allocate memory for the new node and initialize its DATA part to 9 and NEXT part to NULL.



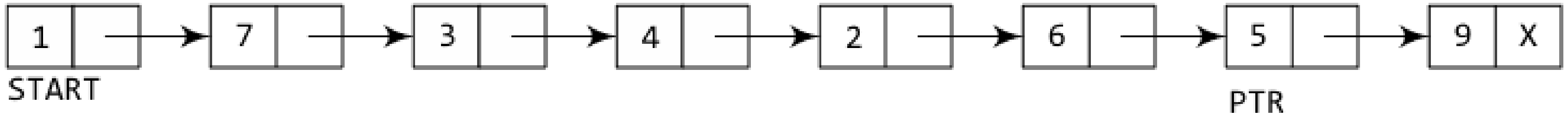
Take a pointer variable PTR which points to START.



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



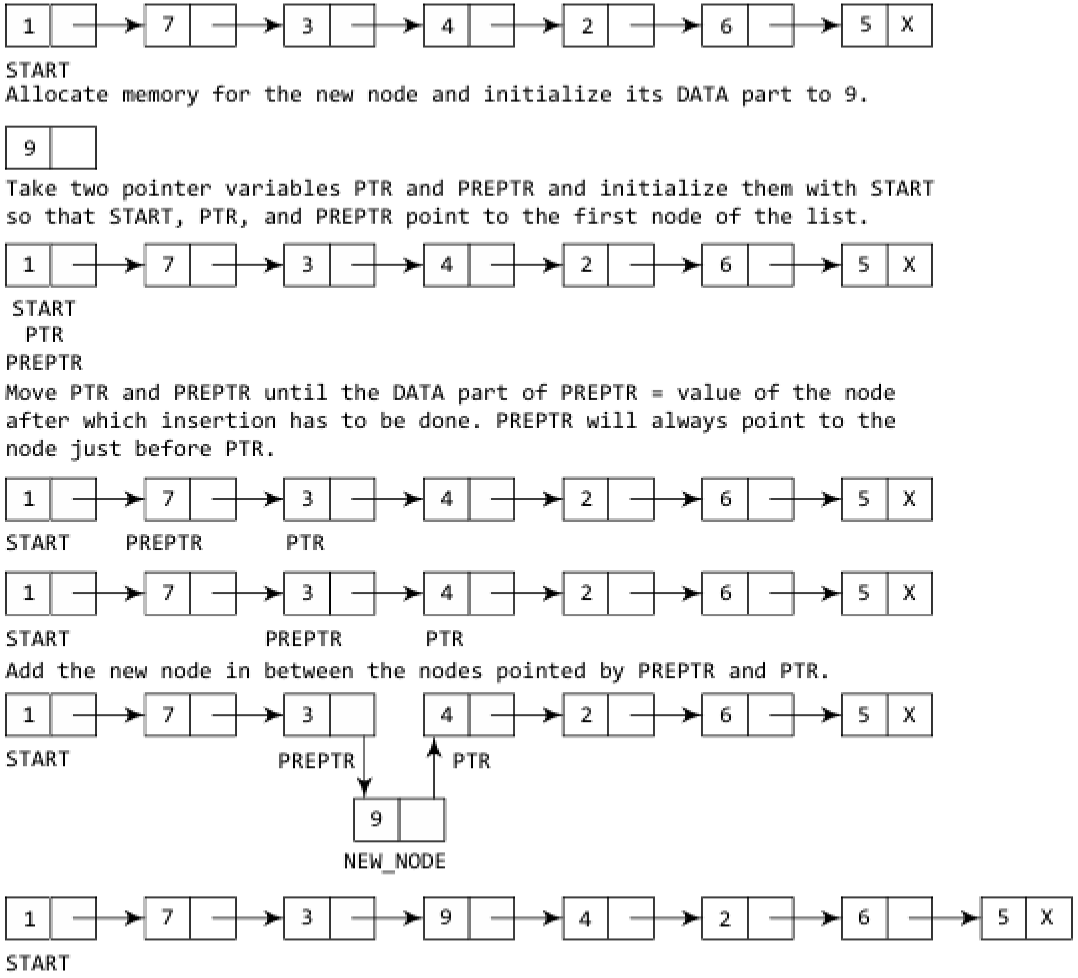
Add the new node after the node pointed by PTR. This is done by storing the address of the new node in the NEXT part of PTR.




```
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
```

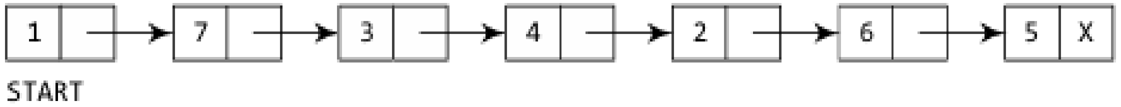
Insert a new node after a node that has value NUM

```
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
```

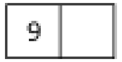


Inserting a Node Before a Given Node in a Linked List

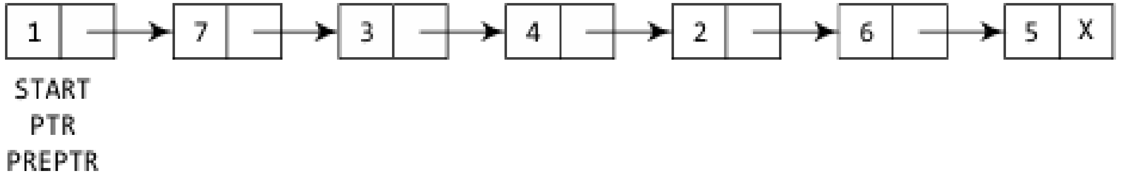
```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
       [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PTR -> DATA != NUM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



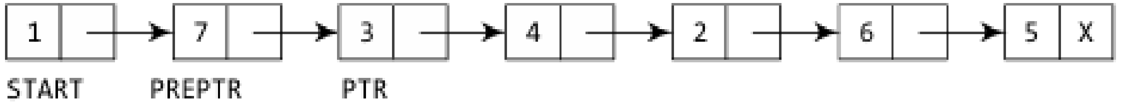
Allocate memory for the new node and initialize its DATA part to 9.



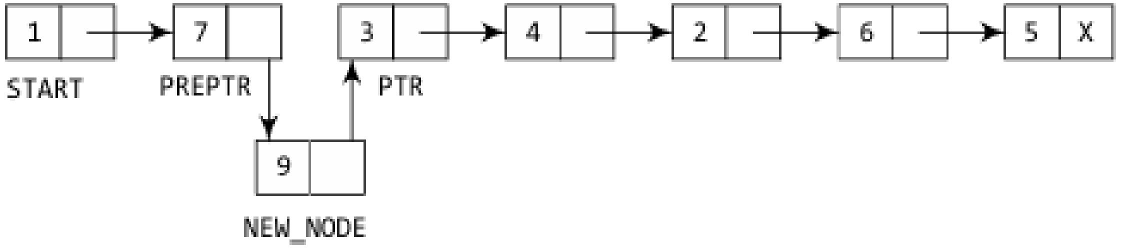
Initialize PREPTR and PTR to the START node.

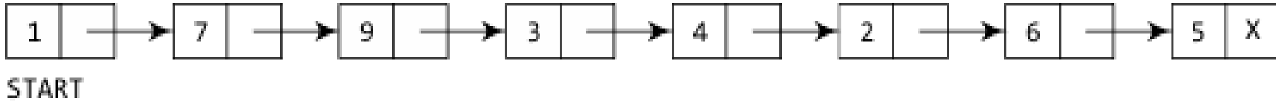


Move PTR and PREPTR until the DATA part of PTR = value of the node before which insertion has to be done. PREPTR will always point to the node just before PTR.



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





Deleting a node from a Linked List

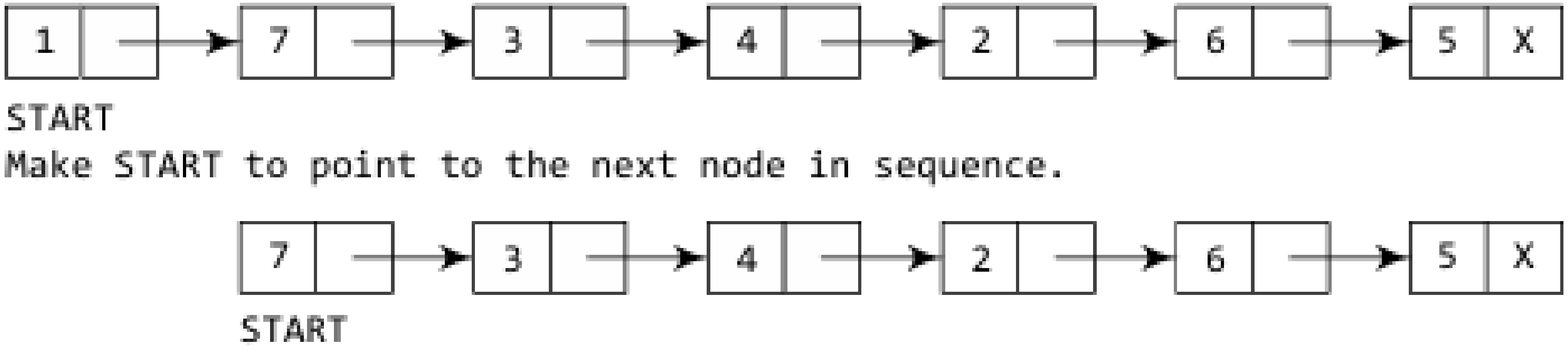
Case 1: The first node is deleted.

Case 2: The last node is deleted.

Case 3: The node after a given node is deleted.

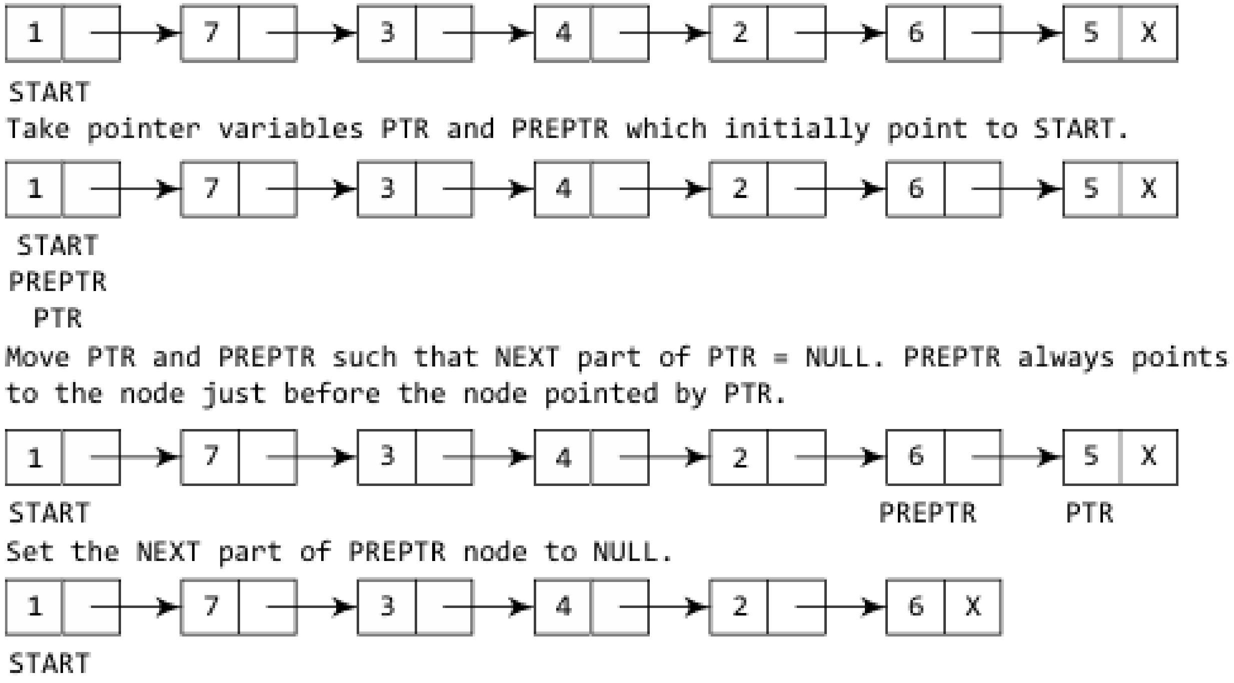
Deleting the first node of a linked list

```
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
```



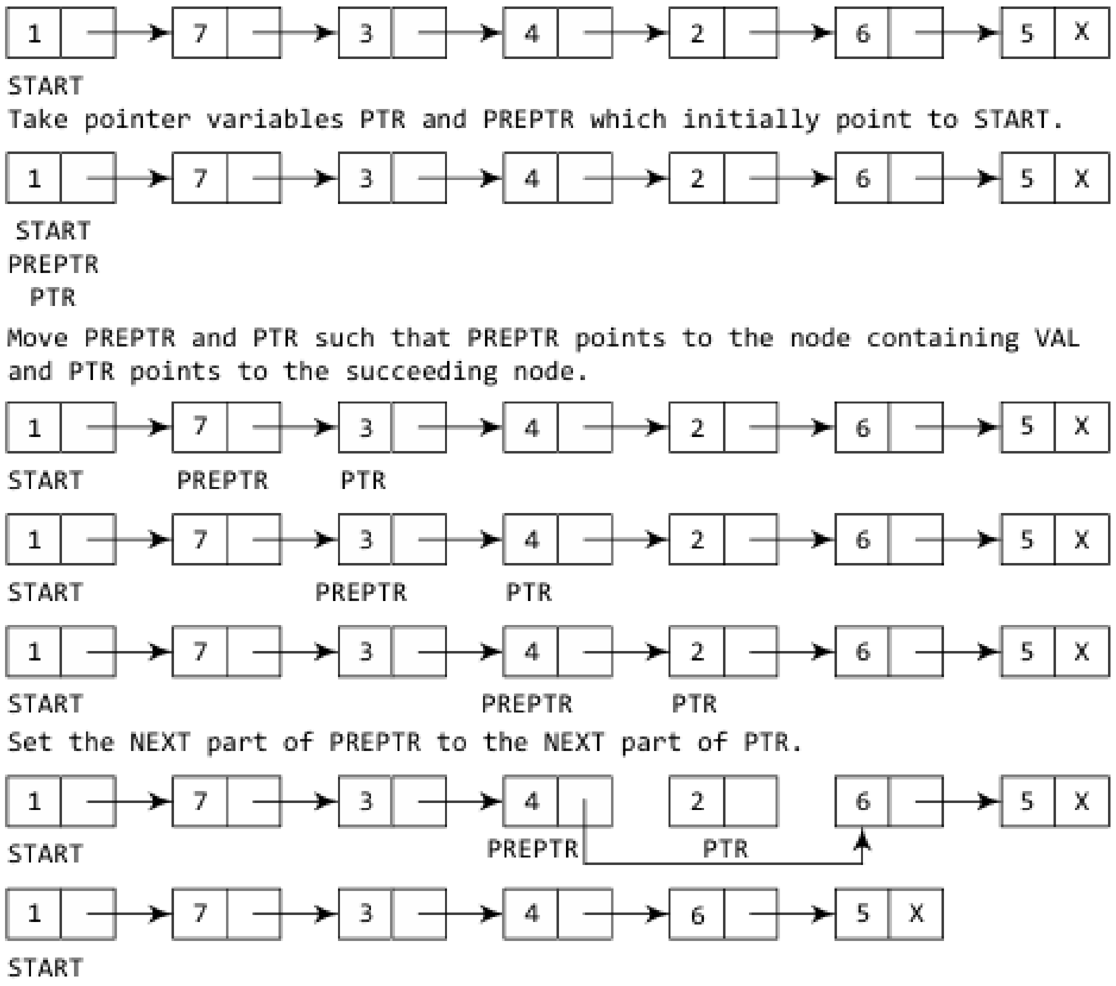
Deleting the Last Node from a Linked List

```
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

```
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
```



Sort LinkedList in ascending order

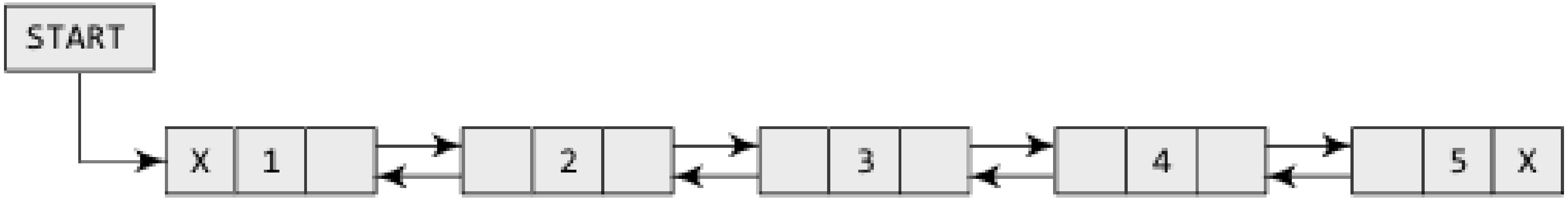
```
Step 1: IF START = NULL OR START->NEXT = NULL
        Write "List is too short to sort"
        Go to Step 9
[END OF IF]
Step 2: SET PTR1 = START
Step 3: Repeat Steps 4 to 7 while PTR1 ≠ NULL
Step 4: SET PTR2 = PTR1->NEXT
Step 5: Repeat Steps 6 and 7 while PTR2 ≠ NULL
 Step 6: IF PTR1->DATA > PTR2->DATA
              SET TEMP = PTR1->DATA
              SET PTR1->DATA = PTR2->DATA
              SET PTR2->DATA = TEMP
         [END OF IF]
 Step 7: SET PTR2 = PTR2->NEXT
   [END OF INNER LOOP]
   Step 8: SET PTR1 = PTR1->NEXT
[END OF OUTER LOOP]
Step 9: EXIT
```

Merging the node

```
Step 1: IF LIST1 = NULL
        SET MERGED_LIST = LIST2
        Go to Step 9
[END OF IF]
Step 2: IF LIST2 = NULL
        SET MERGED_LIST = LIST1
        Go to Step 9
[END OF IF]
Step 3: Initialize pointers:
        SET PTR1 = LIST1
        SET PTR2 = LIST2
        SET MERGED_LIST = NULL
        SET LAST = NULL
Step 4: Repeat Steps 5–7 while PTR1 ≠ NULL AND PTR2 ≠ NULL
Step 5: IF PTR1->DATA ≤ PTR2->DATA
                SET TEMP = PTR1
                SET PTR1 = PTR1->NEXT
            ELSE
                SET TEMP = PTR2
                SET PTR2 = PTR2->NEXT
            [END OF IF]
Step 6: IF MERGED_LIST = NULL
                SET MERGED_LIST = TEMP
                SET LAST = TEMP
            ELSE
                SET LAST->NEXT = TEMP
                SET LAST = TEMP
            [END OF IF]
Step 7: [END OF LOOP]
Step 8: IF PTR1 ≠ NULL
            SET LAST->NEXT = PTR1
        ELSE IF PTR2 ≠ NULL
            SET LAST->NEXT = PTR2
        [END OF IF]
Step 9: EXIT
```

Doubly Linked List

- A doubly linked list or a two-way linked list is a more complex type of linked list which 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




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

Memory representation of a doubly linked list

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	O	7	–1

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.

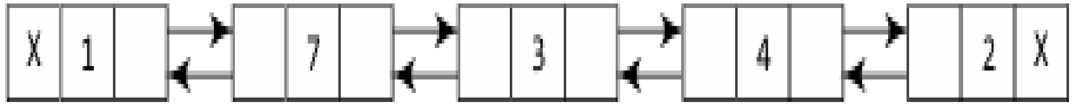
Case 3: The new node is inserted after a given node.

Case 4: The new node is inserted before a given node

Inserting a Node at the Beginning of a Doubly Linked List

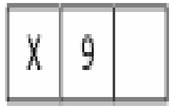
Suppose we want to add a new node with data 9 as the first node of the list. Then the following changes will be done in the linked list.

```
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
```

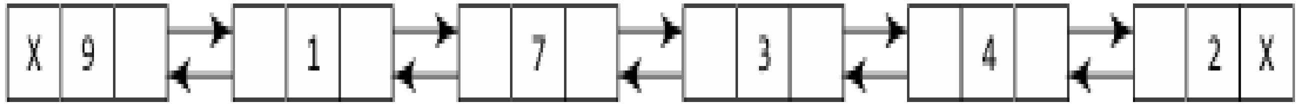


START

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



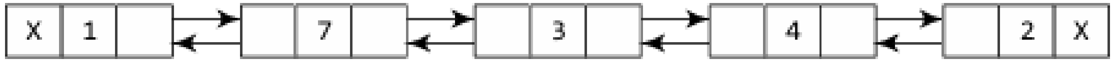
Add the new node before the START node. Now the new node becomes the first node of the list.



START

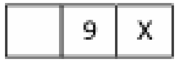
Inserting a Node at the End end of a Doubly Linked List

```
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
```

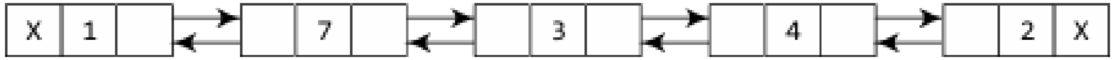


START

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

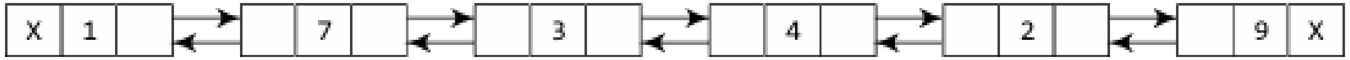


Take a pointer variable PTR and make it point to the first node of the list.



START,PTR

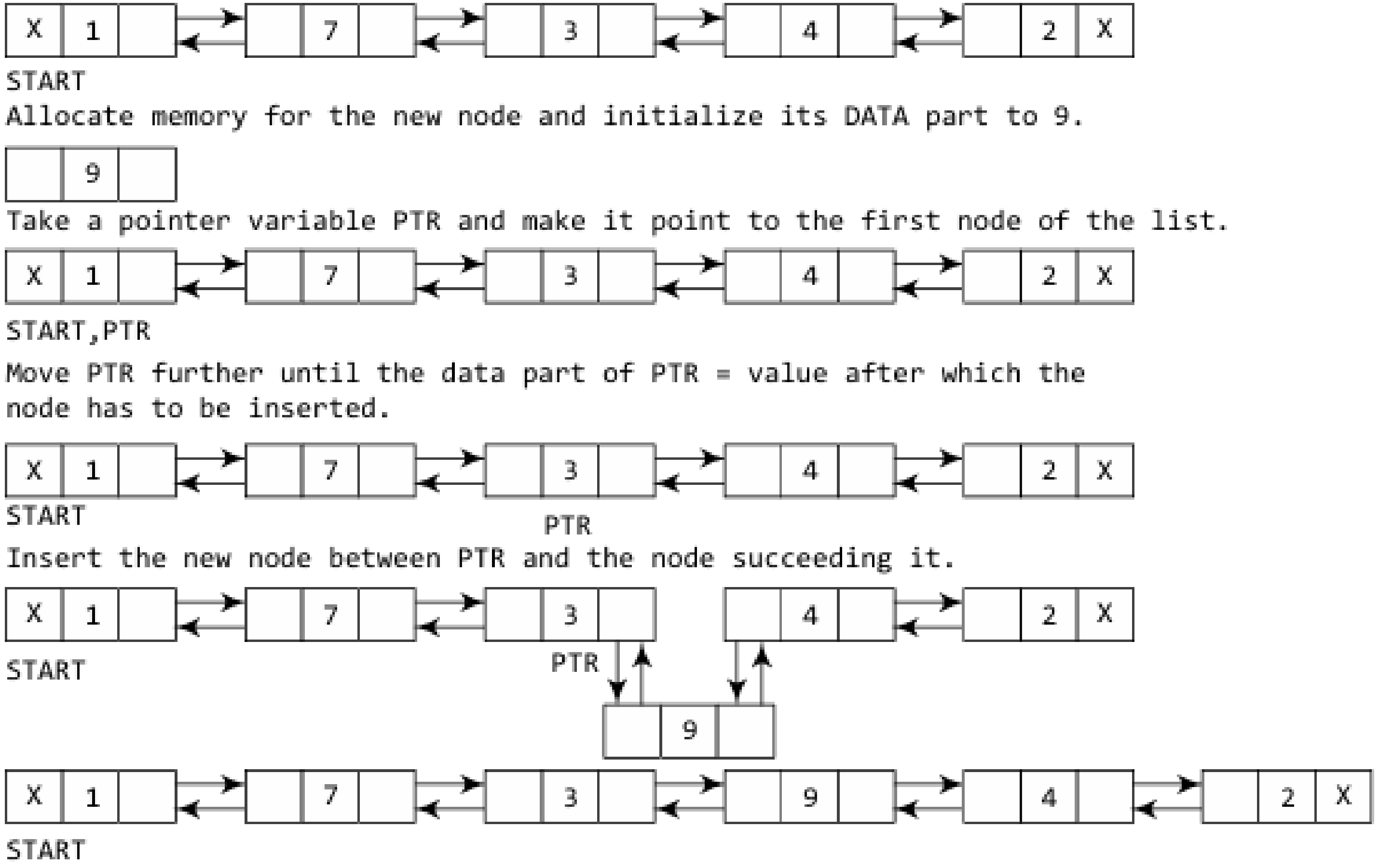
Move PTR so that it points to the last node of the list. Add the new node after the node pointed by PTR.



START PTR

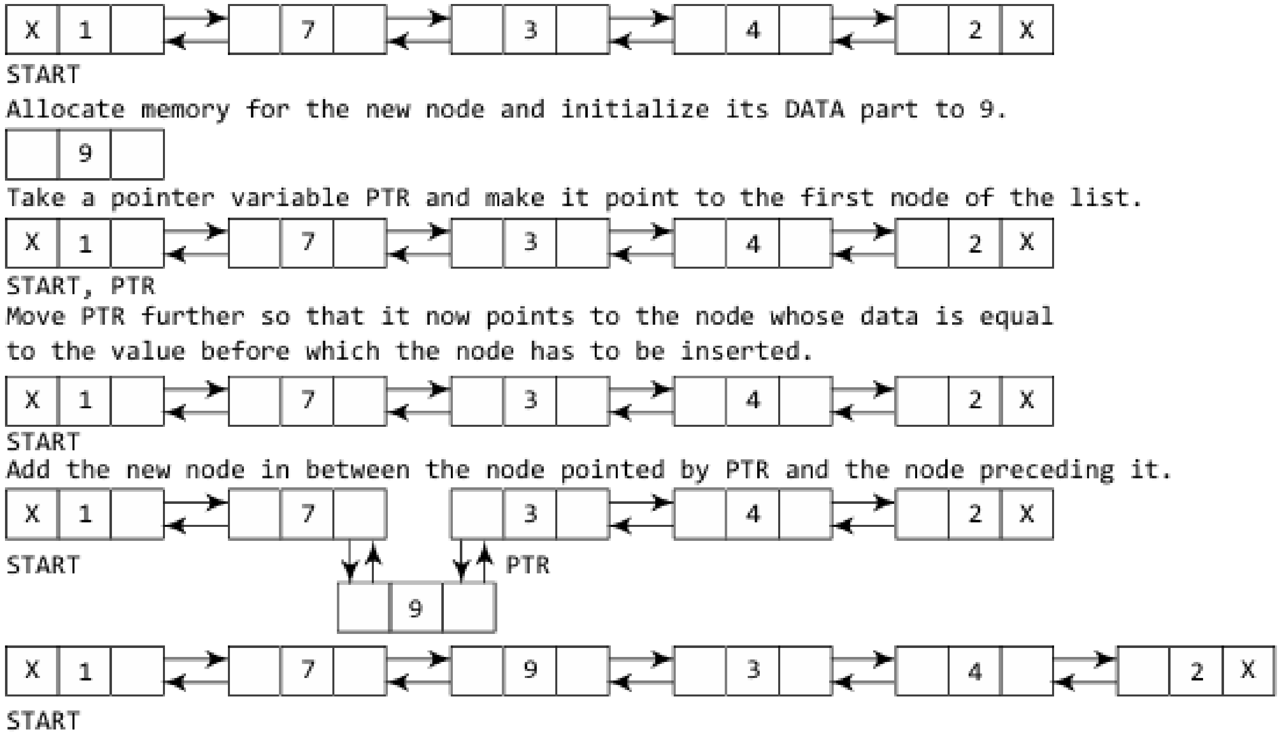
Algorithm to insert a new node after a given node

```
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: 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
```



Algorithm to insert a new node before 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: 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 = NEW_NODE
Step 11: SET PTR -> PREV -> NEXT = NEW_NODE
Step 12: EXIT
```



Deleting a node from a doubly Linked List

Case 1: The first node is deleted.

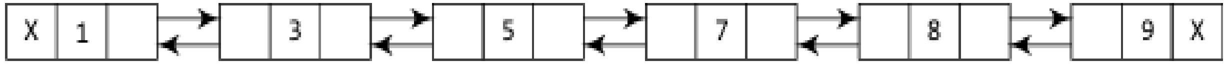
Case 2: The last node is deleted.

Case 3: The node after a given node is deleted.

Case 4: The node before a given node is deleted.

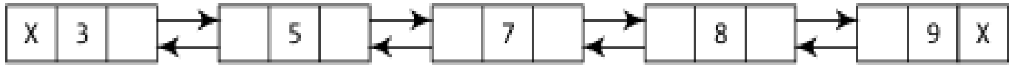
Deleting the First Node from a Doubly Linked List

```
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
```



START

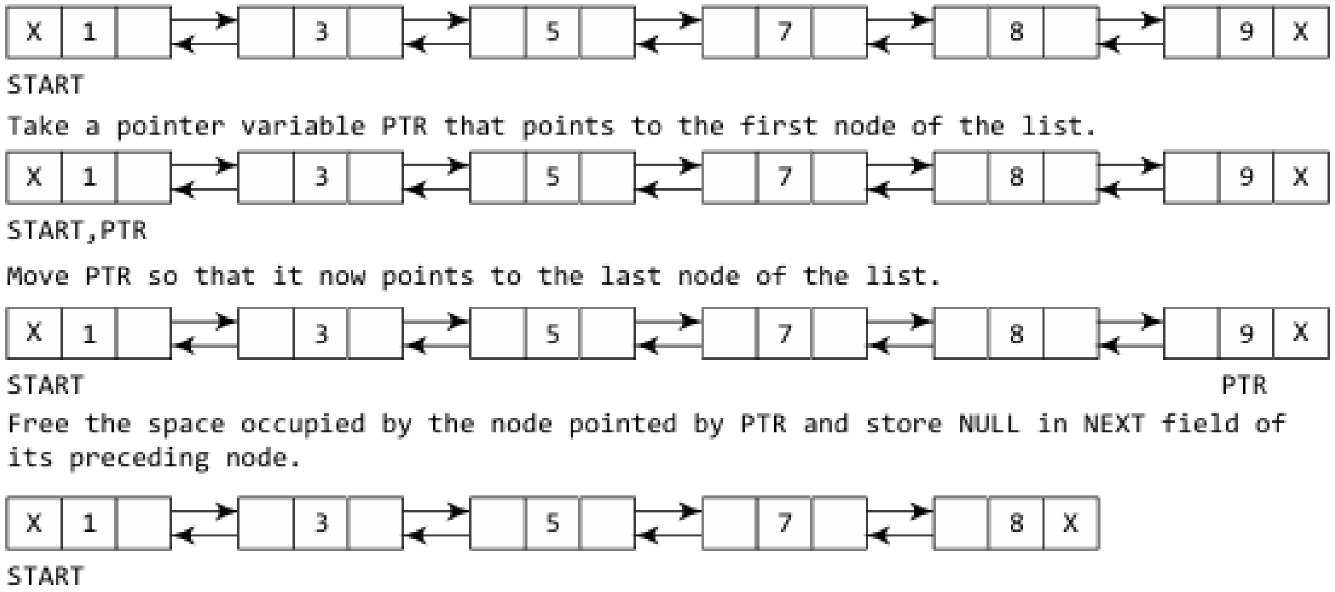
Free the memory occupied by the first node of the list and make the second node of the list as the START node.



START

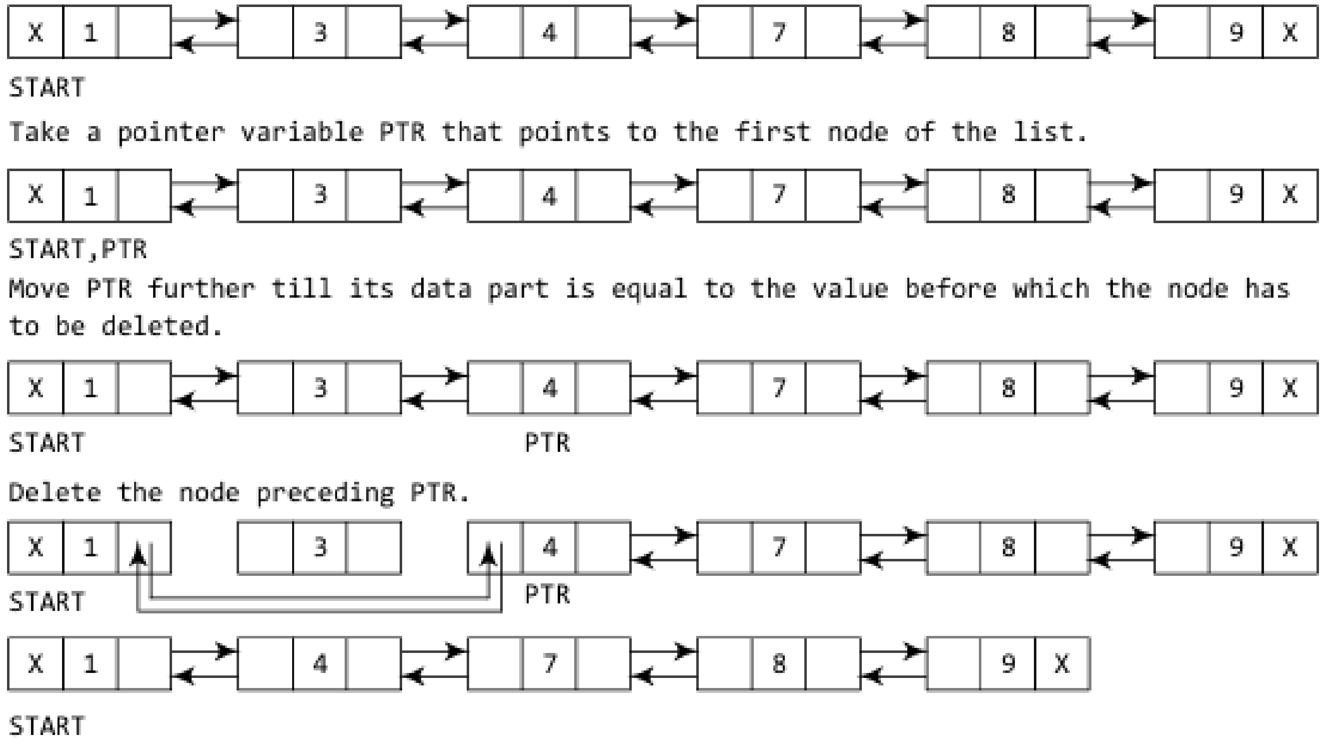
Deleting the Last Node from a Doubly Linked List

```
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
```



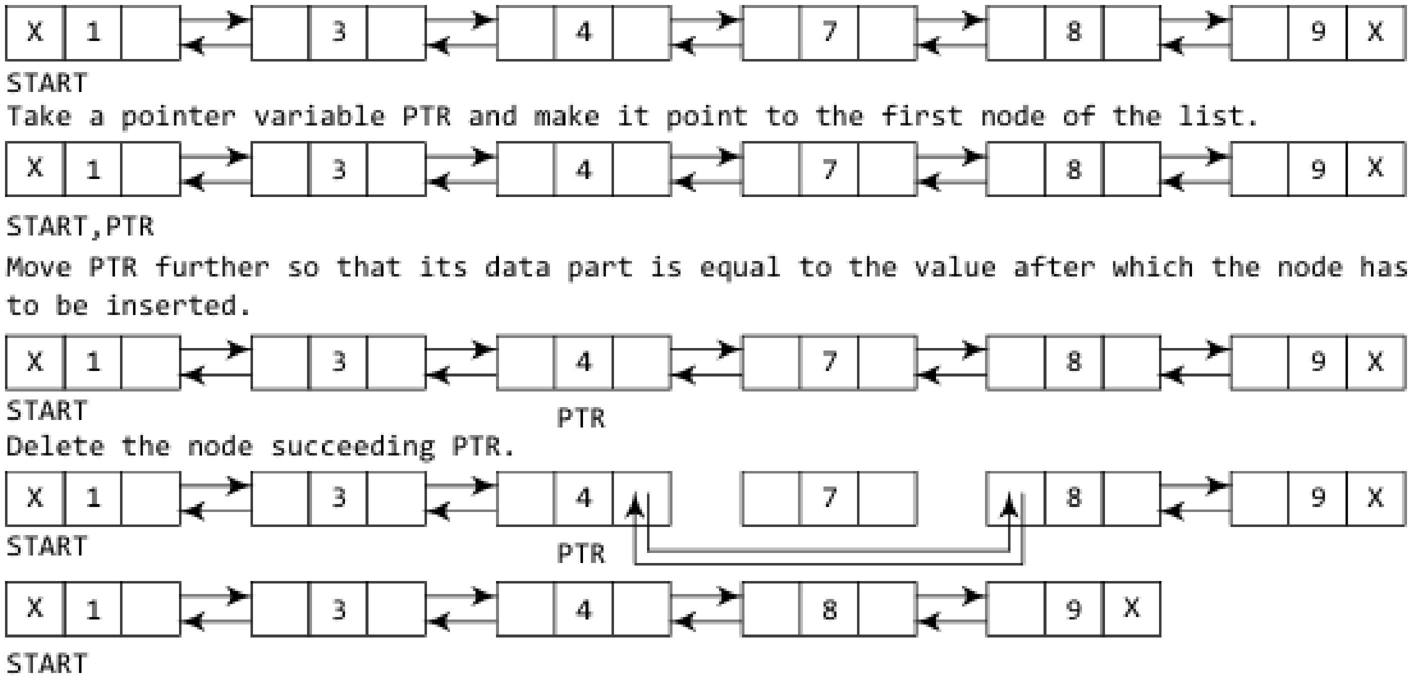
Deleting the Node Before a Given Node in a Doubly Linked List

```
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
```



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

```
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
```



Applications of LinkedList

- Polynomial representation Let us see how a polynomial is represented in the memory using a linked list. Consider a polynomial 6x3 + 9x2 + 7x + 1. Every individual term in a polynomial consists of two parts, a coefficient and a power.
- 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.

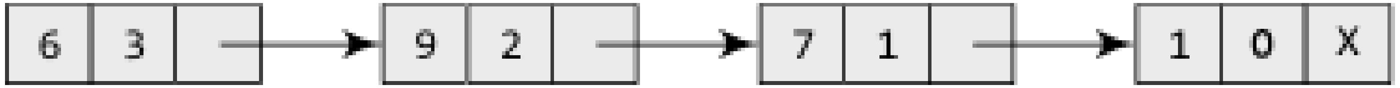


Figure 6.74 Linked representation of a polynomial

Real-World Examples of Linked List Applications

System/Software	Use Case	Type of Linked List Used
Microsoft Word or Google Docs	Undo/Redo functionality	Doubly Linked List
Google Chrome / Firefox	Browser history navigation	Doubly Linked List
Spotify / VLC Media Player	Managing playlists (next/previous track navigation)	Doubly Linked List
Linux Kernel	Process control blocks (scheduling)	Singly / Circular Linked List
Windows File System (NTFS)	Directory structures	Singly Linked List
Facebook Graph API	Social connections & friend lists	Adjacency List using Linked Lists
Redis (In-memory DB)	List datatype for fast data manipulation	Doubly Linked List
Compiler Design Tools	Syntax tree representation	Linked Lists (varies by compiler)
Job Scheduling Systems	Task management and execution order	Circular Linked List
Memory Allocators (malloc)	Free memory block management	Singly Linked List