Linked Lists

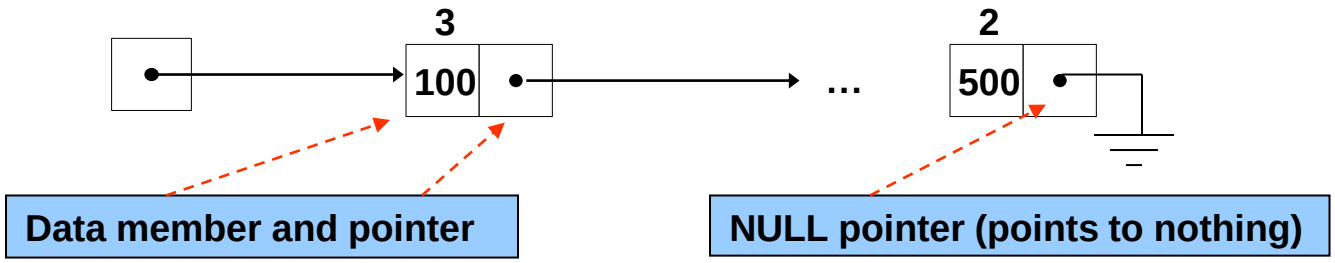
- Introduction: memory representation, allocation and garbage collection.
- Operations: Traversal, insertion and deletion.
- Header linked lists: Grounded and Circular
- Two-way lists: operations on two way linked lists

Introduction

- Linked list
  - Linear collection of self-referential class objects, called nodes
  - Connected by pointer links
  - Accessed via a pointer to the first node of the list
  - Link pointer in the last node is set to null to mark the list's end
- Use a linked list instead of an array when
  - You have an unpredictable number of data elements
  - You want to insert and delete quickly.

Self-Referential Structures

- Self-referential structures
  - Structure that contains a pointer to a structure of the same type
  - Can be linked together to form useful data structures such as lists, queues, stacks and trees
  - Terminated with a **`NULL`** pointer (**`0`**)
- Diagram of two self-referential structure objects linked together

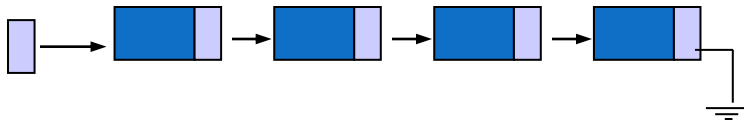



```
struct node {
      int data;
      struct node *nextPtr;
  };
```

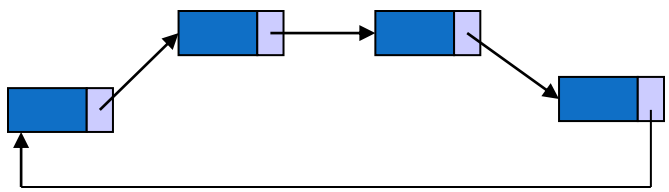
- `nextPtr`
  - Points to an object of type node
  - Referred to as a link

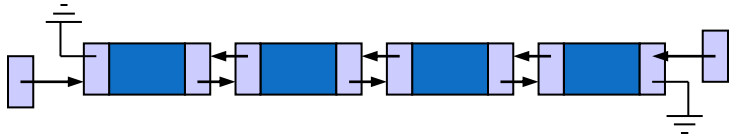
Linked Lists

- Types of linked lists:
  - Singly linked list
    - Begins with a pointer to the first node
    - Terminates with a null pointer
    - Only traversed in one direction



  - Circular, singly linked
    - Pointer in the last node points back to the first node

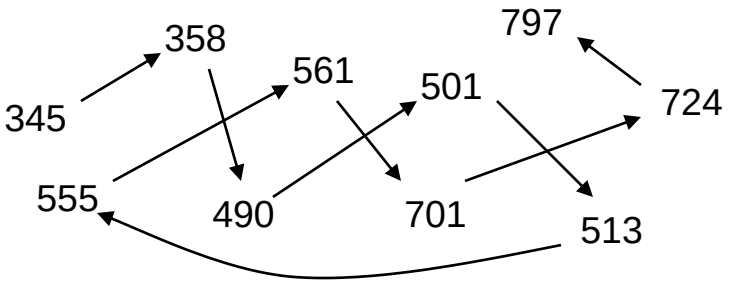




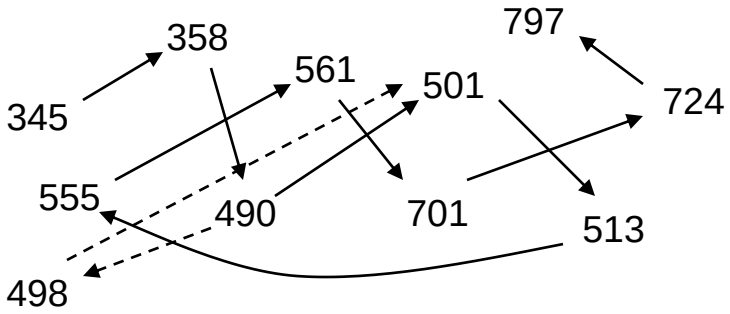
  - Doubly linked list
    - Two "start pointers" – first element and last element
    - Each node has a forward pointer and a backward pointer
    - Allows traversals both forwards and backwards
  - Circular, doubly linked list
    - Forward pointer of the last node points to the first node and backward pointer of the first node points to the last node

Linked Representation of Data

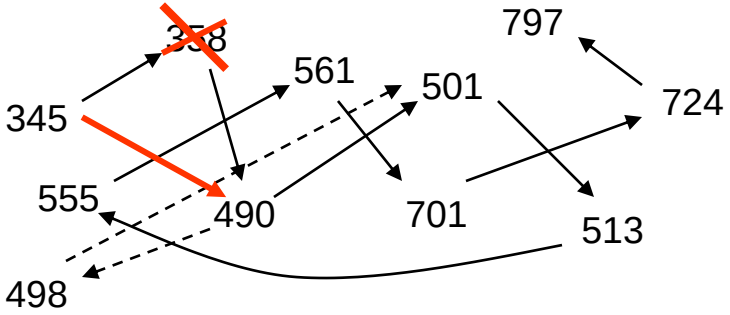
- In a linked representation, data is not stored in a contiguous manner. Instead, data is stored at random locations and the current data location provides the information regarding the location of the next data.



<u>Adding item **498** on to the linked list</u>
Q: What is the cost of adding an item?
Q: how about adding **300** and **800** onto the linked list

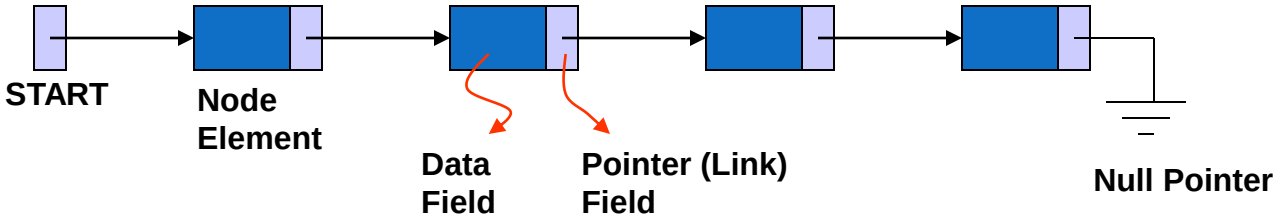


<u>Deleting item **358** from the linked list</u>
Q: What is the cost of deleting an item?
Q: What is the cost of searching for an item?



Linked List

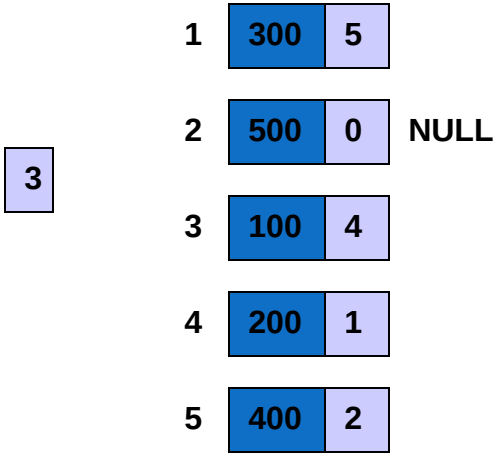
- How do we represent a linked list in the memory
  - Each location has two fields: Data Field and Pointer (Link) Field.
- Linked List Implementation



- `struct node {`

```
    int data;
    struct node *link;
};
struct node my_node;
```

Example:

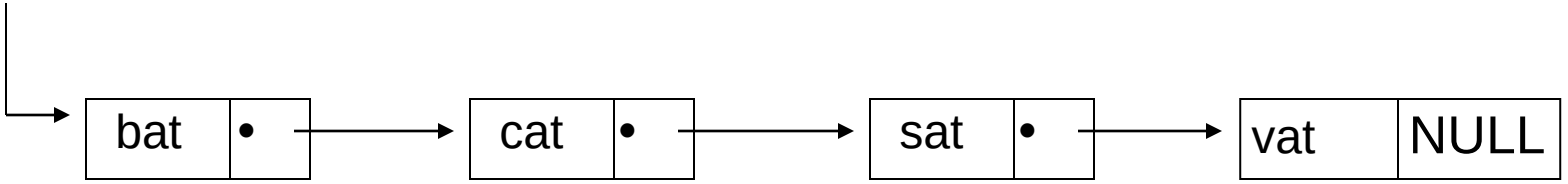


Conventions of Linked List

There are several conventions for the link to indicate the end of the list.

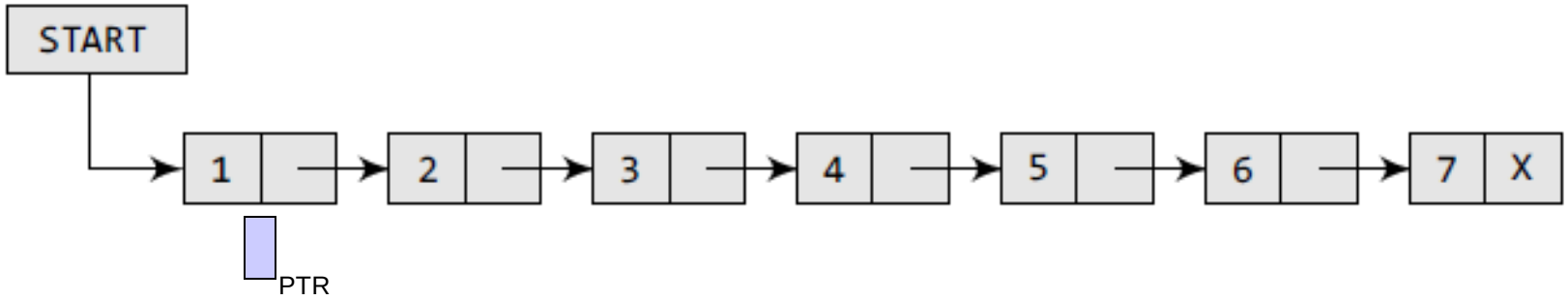
1. a *null link* that points to no node (0 or NULL)
2. a *dummy node* that contains no item
3. a reference back to the first node, making it a *circular list*.

Singly Linked List



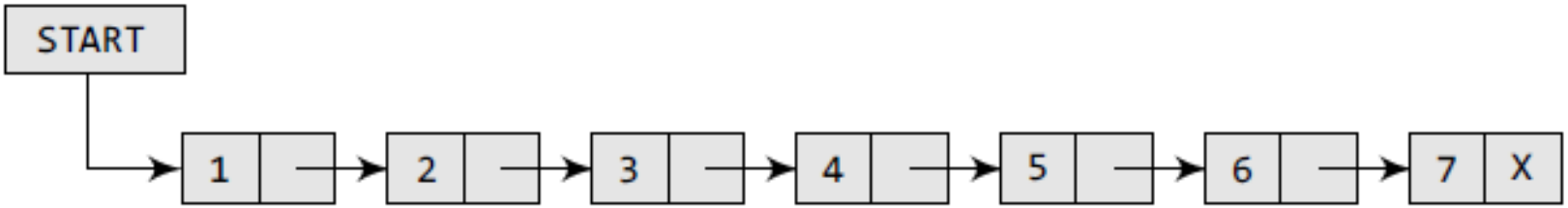
Linked List Manipulation Algorithms

Linked-List Traversal




```
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
```

Print the nos. of nodes in a Linked List

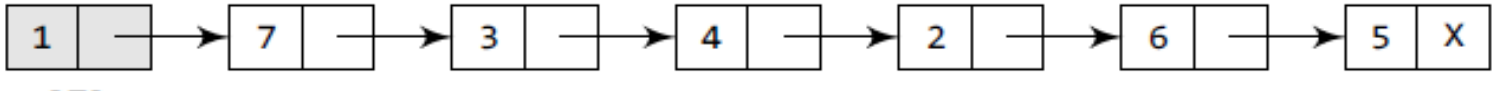



```
Step 1: [INITIALIZE] SET COUNT = 0
Step 2: [INITIALIZE] SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR != NULL
Step 4:                SET COUNT = COUNT + 1
Step 5:                SET PTR = PTR->NEXT
        [END OF LOOP]
Step 6: Write COUNT
Step 7: EXIT
```

Search for an ITEM in a Linked List

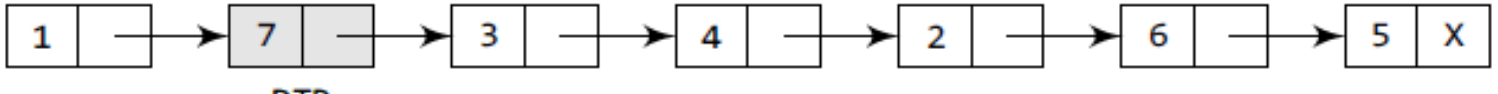
```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Step 3 while PTR != NULL
Step 3:         IF VAL = PTR->DATA
                        SET POS = PTR
                        Go To Step 5
                ELSE
                        SET PTR = PTR->NEXT
                [END OF IF]
        [END OF LOOP]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

Search for an ITEM in a Linked List



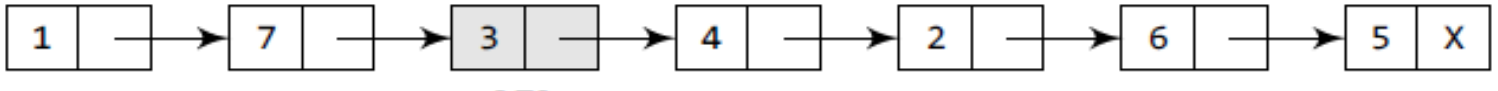
PTR

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



PTR

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



PTR

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

PTR

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

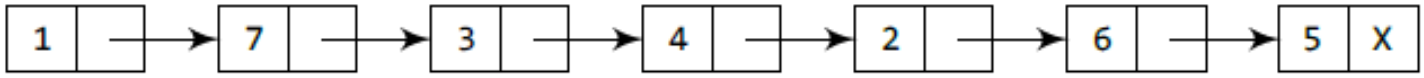
Insert an Item

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
```

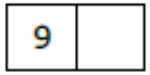
Insert an Item

Inserting a Node at the Beginning of a Linked List

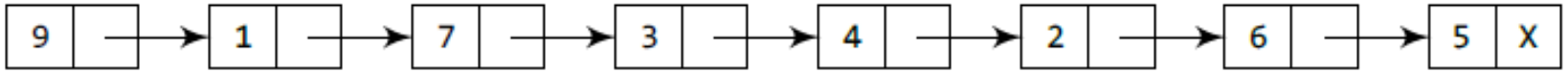


START

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

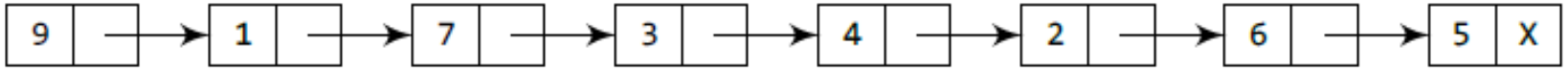


Add the new node as the first node of the list by making the NEXT part of the new node contain the address of START.



START

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



START

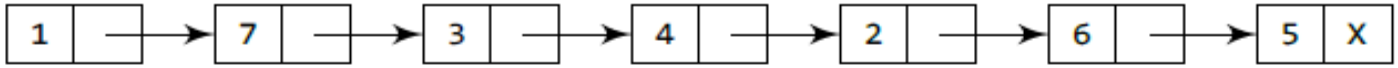
Insert an Item

Inserting a Node at the End of a 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
```

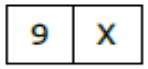
Insert an Item

Inserting a Node at the End of a Linked List

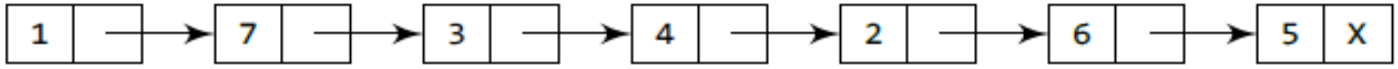


START

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

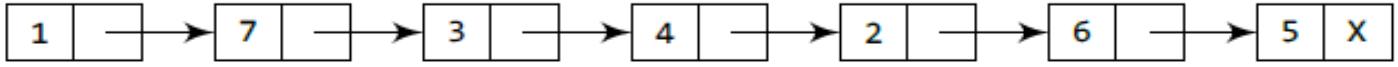


Take a pointer variable PTR which points to START.



START, PTR

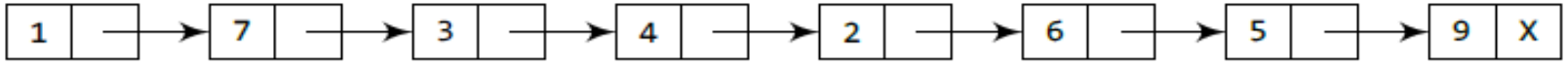
Move PTR so that it points to the last node of the list.



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

Insert an Item

Inserting a Node After 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:Repeat Steps 7 while PTR->DATA != NUM
Step 7:         SET PTR = PTR->NEXT
        [END OF LOOP]
Step 8: SET NEW_NODE->NEXT = PTR→NEXT
Step 9: PTR->NEXT = NEW_NODE
Step 10: EXIT
```

Insert an Item

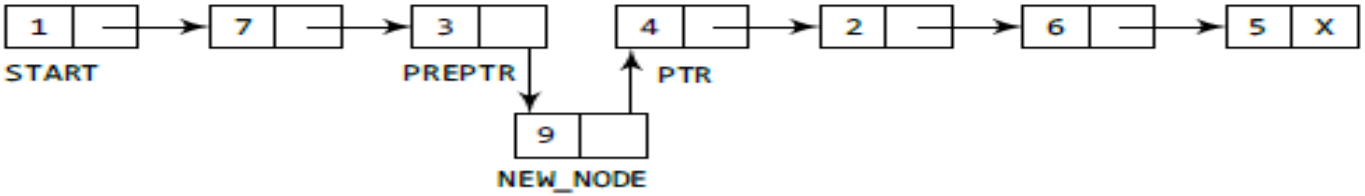
Inserting a Node After a Given Node in a Linked List

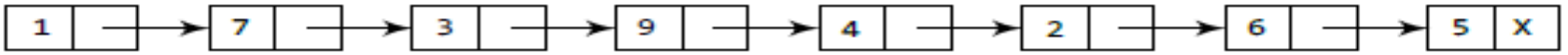
START
Allocate memory for the new node and initialize its DATA part to 9.

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.

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.

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





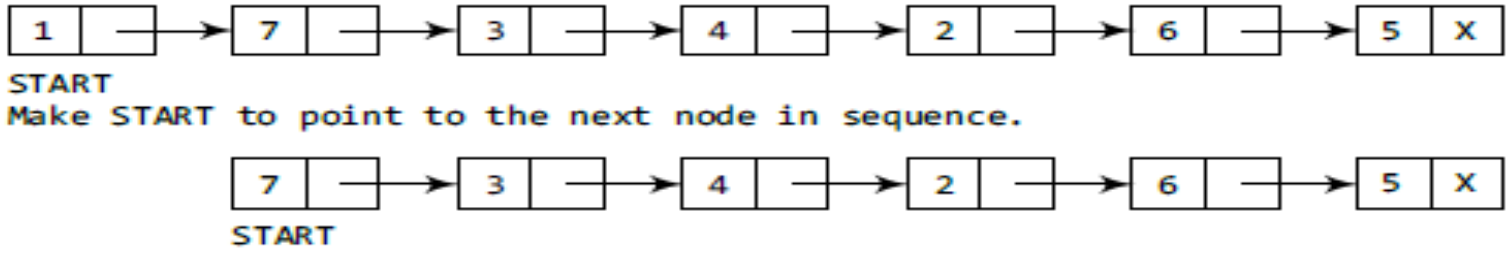
Insert an Item

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: if PTR->DATA = NUM
                SET NEW_NODE-> NEXT = START
                SET START = NEW_NODE and EXIT
Step 7:Repeat while PTR->DATA != NUM
                SET PREPTR = PTR
                SET PTR = PTR->NEXT
        [END OF LOOP]
Step 8: SET NEW_NODE->NEXT = PREPTR→NEXT
Step 9: PREPTR->NEXT = NEW_NODE
Step 10: EXIT
```

Delete an Item

Deleting the First Node from 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 or SET PTR -> LINK = AVAIL and AVAIL = PTR
Step 5: EXIT
```

Delete an Item

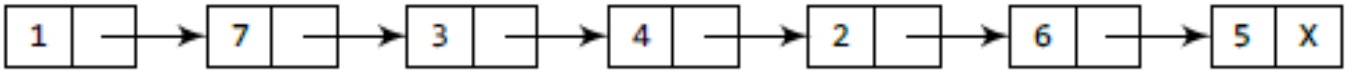
Deleting the Last Node from a Linked List

```
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 or SET PTR -> LINK = AVAIL and AVAIL = PTR

Step 8: EXIT
```

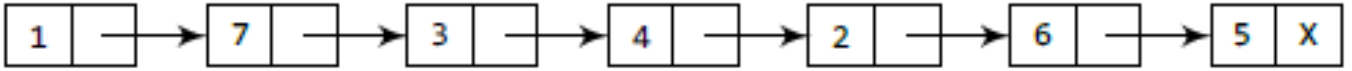
Delete an Item

Deleting the Last Node from a Linked List



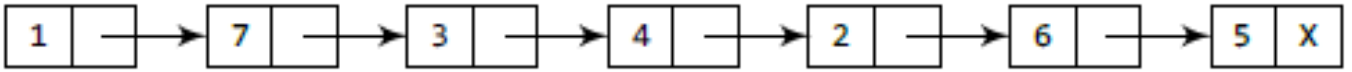
START

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



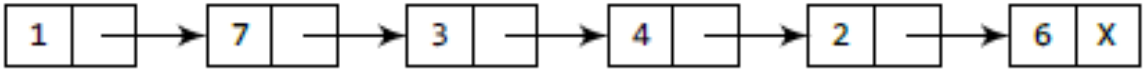
START
PREPTR
PTR

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

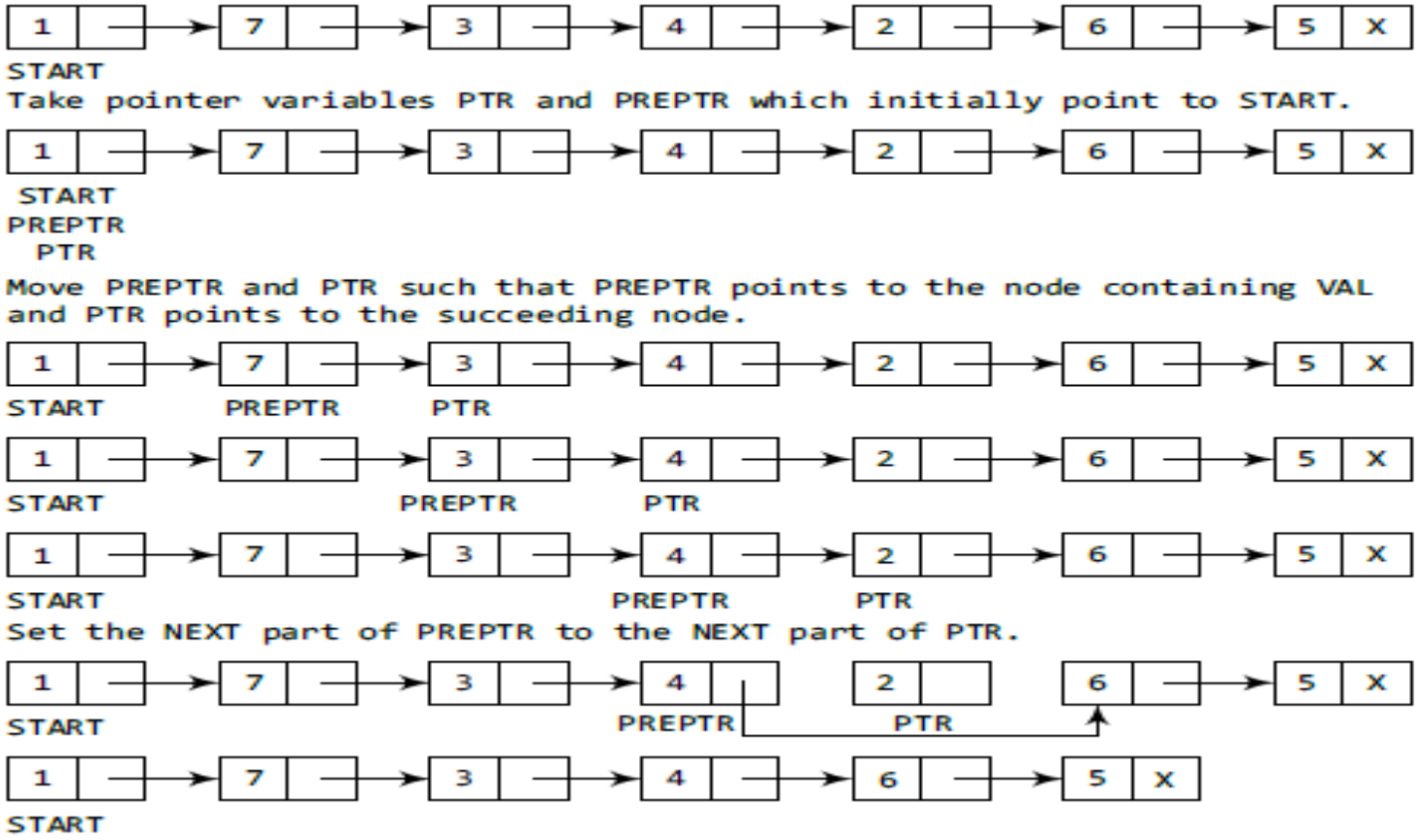
Delete an Item

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 or SET TEMP -> LINK = AVAIL and AVAIL = TEMP
Step 10: EXIT
```

Delete an Item

Deleting the Node After a Given Node in a Linked List



Circular Linked List

Circular Linked-List Traversal

```
Step 1: [INITIALIZE] SET PTR = START
Step 2:        Apply Process to PTR->DATA
Step 3:        PTR = PTR->NEXT

Step 4: Repeat Steps 5 and 6 while PTR != START
Step 5:        Apply Process to PTR->DATA
Step 6:        SET PTR = PTR->NEXT
       [END OF LOOP]
Step 7: EXIT
```

Circular Linked List

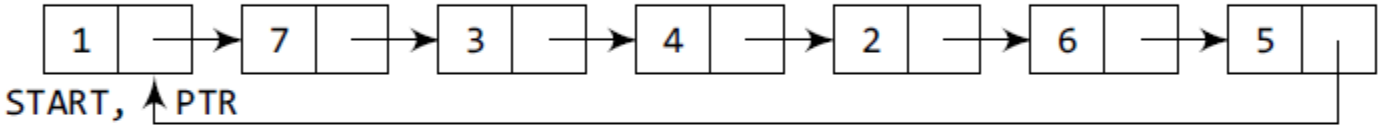
Inserting a Node at the Beginning of a Circular Linked List

```
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 PTR = START
Step 6: Repeat Step 7 while PTR -> NEXT != START
Step 7:     PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET NEW_NODE -> NEXT = START
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: SET START = NEW_NODE
Step 11: EXIT
```

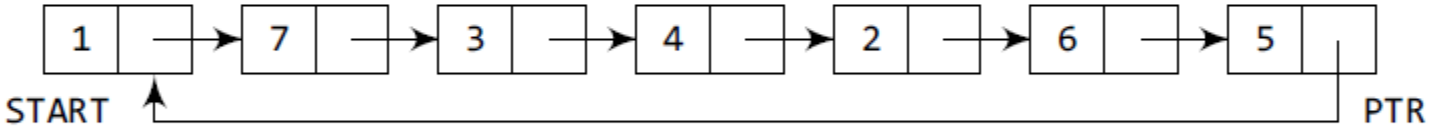
Inserting a Node at the Beginning of a Circular Linked List

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

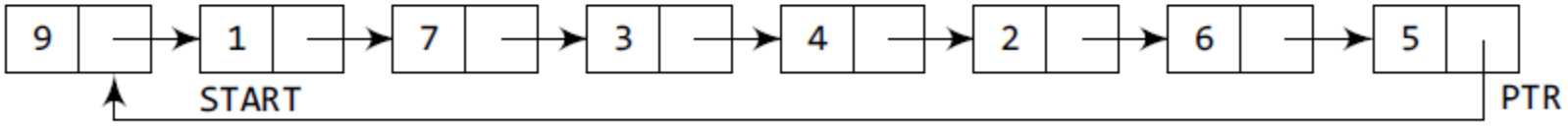
Take a pointer variable PTR that points to the START node of the list.



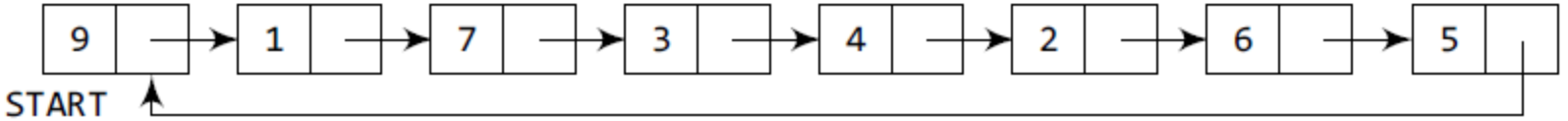
Move PTR so that it now points to the last node of the list.



Add the new node in between PTR and START.



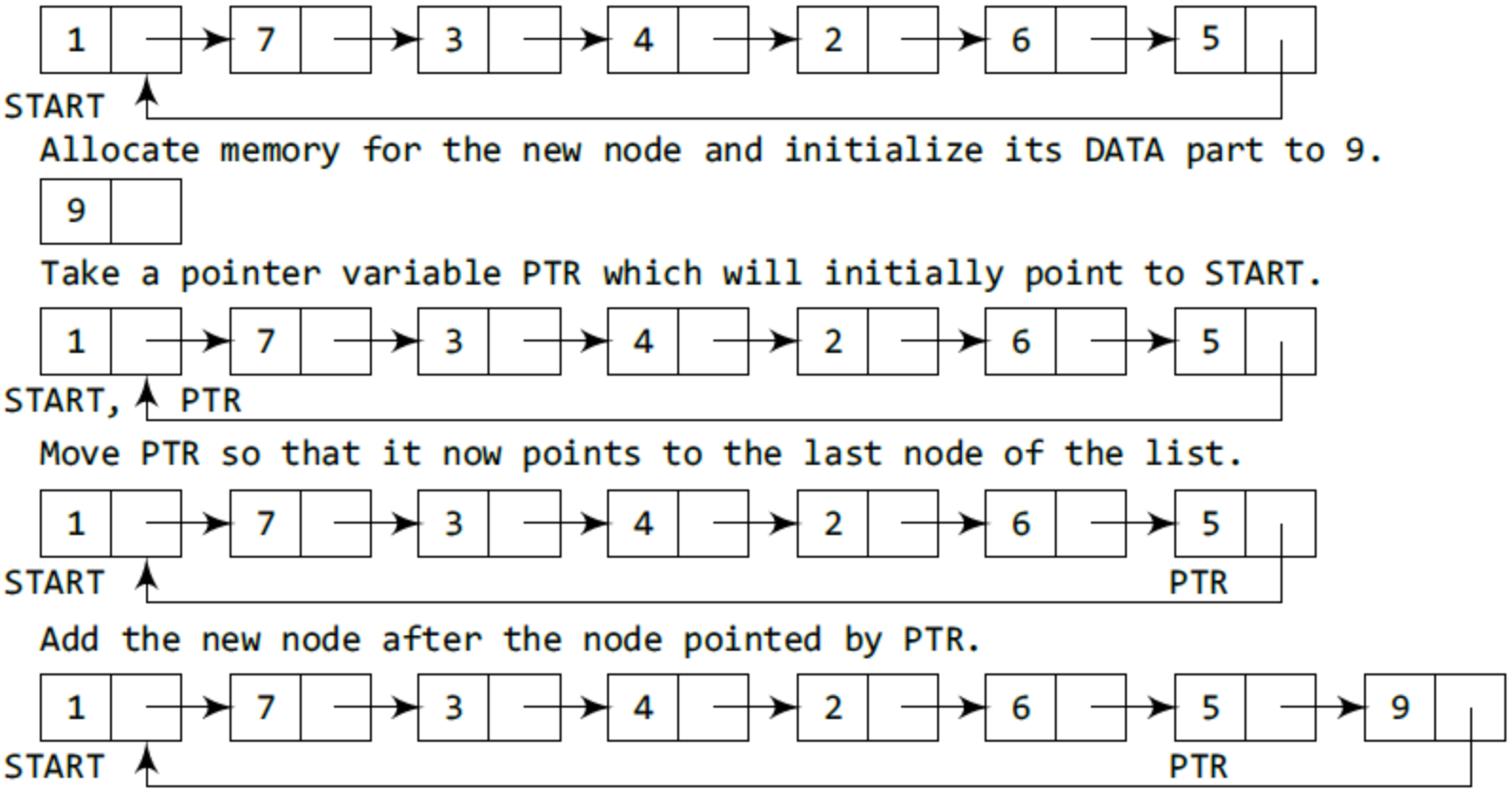
Make START point to the new node.



Inserting a Node at the End of a Circular 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 = 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: EXIT
```

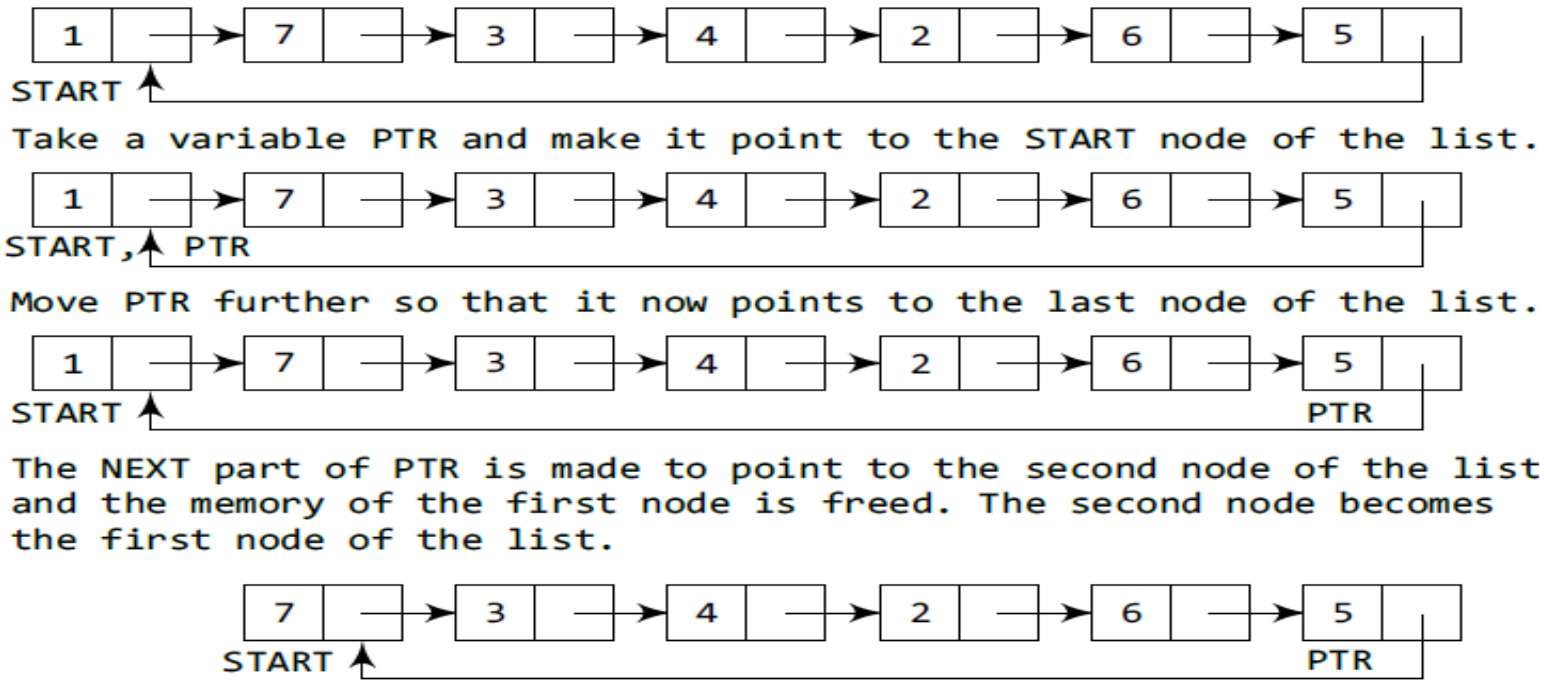
Inserting a Node at the End of a Circular Linked List



Deleting the First Node from a Circular Linked List

```
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: FREE START
Step 7: SET START = PTR -> NEXT
Step 8: EXIT
```

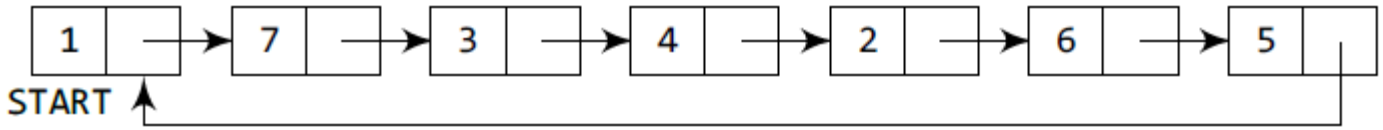
Deleting the First Node from a Circular Linked List



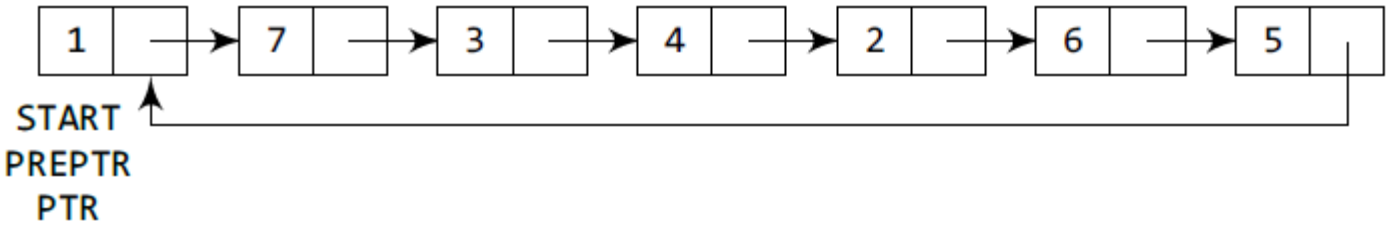
Deleting the Last Node from a Circular Linked List

```
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 != START
Step 4:         SET PREPTR = PTR
Step 5:         SET PTR = PTR -> NEXT
          [END OF LOOP]
Step 6: SET PREPTR -> NEXT = START
Step 7: FREE PTR
Step 8: EXIT
```

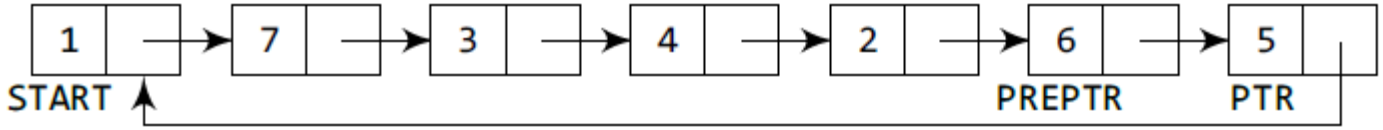
Deleting the Last Node from a Circular Linked List



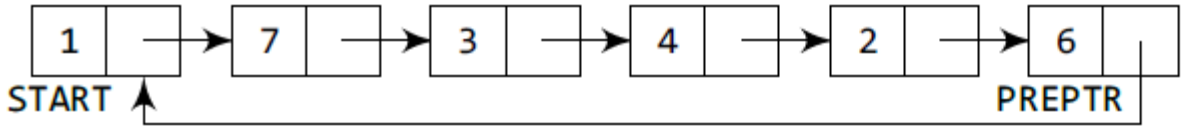
Take two pointers PREPTR and PTR which will initially point to START.



Move PTR so that it points to the last node of the list. PREPTR will always point to the node preceding PTR.



Make the PREPTR's next part store START node's address and free the space allocated for PTR. Now PREPTR is the last node of the list.



Header Linked Lists

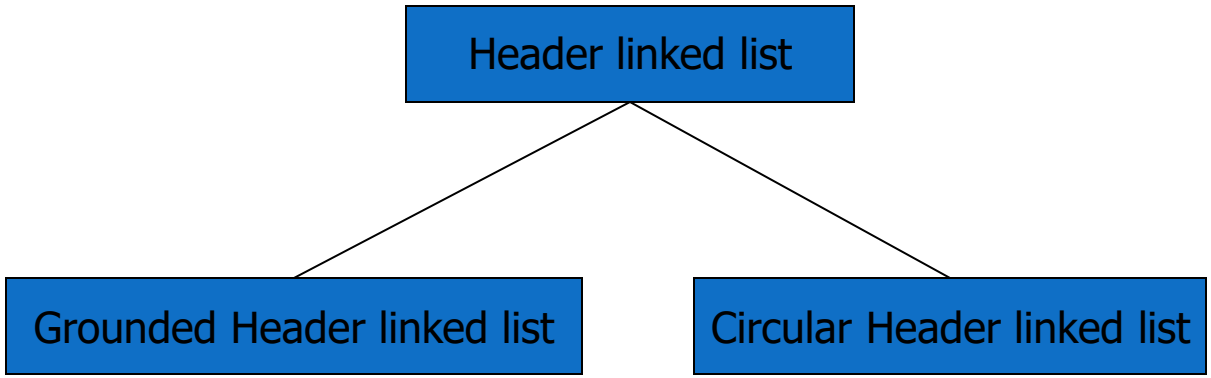
- Header linked list is a linked list which always contains a special node called the Header Node, at the beginning of the list.
- It has two types:

  a) Grounded Header List

  Last Node Contains the NULL Pointer

  b) Circular Header List

  Last Node Points Back to the Header Node



Grounded Header Link List

- A grounded header list is a header list where the last node contains the null pointer.
- The term "**grounded**" comes from the fact that many texts use the electrical ground symbol to indicate the null pointer.

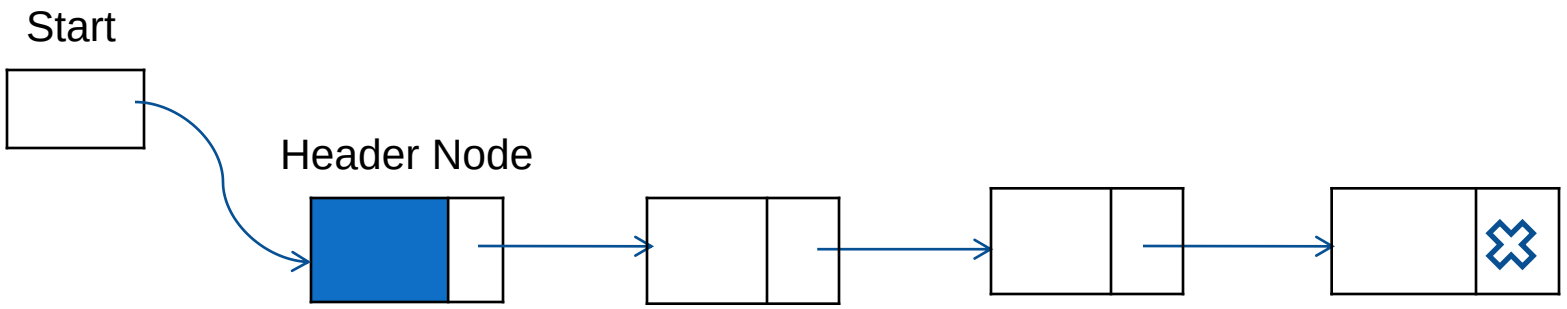


Figure: Grounded Header Link List

Circular Header Linked List

- A circular header Link list is a header list where the last node points back to the header node.
- The chains do not indicate the last node and first node of the link list.
- In this case, external pointers provide a frame reference because last node of a circular link list does not contain null pointer.

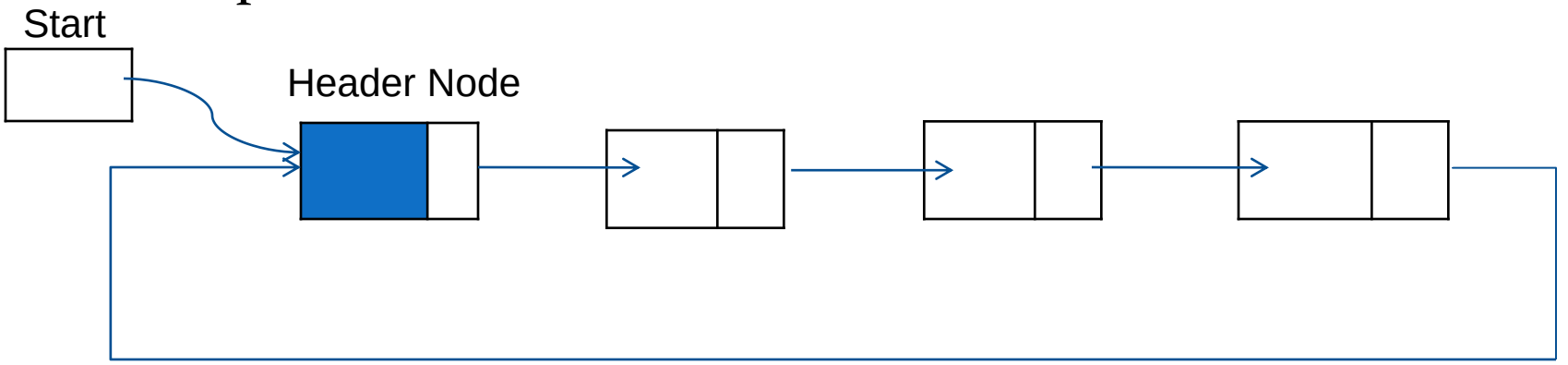


Figure: Circular Header Link List

Benefit of using Header Node

- One way to simplify insertion and deletion is never to insert an item before the first or after the last item and never to delete the first node
- You can set a header node at the beginning of the list containing a value smaller than the smallest value in the data set
- You can set a trailer node at the end of the list containing a value larger than the largest value in the data set.
- These two nodes, header and trailer, serve merely to simplify the insertion and deletion algorithms and are not part of the actual list.
- The actual list is between these two nodes.

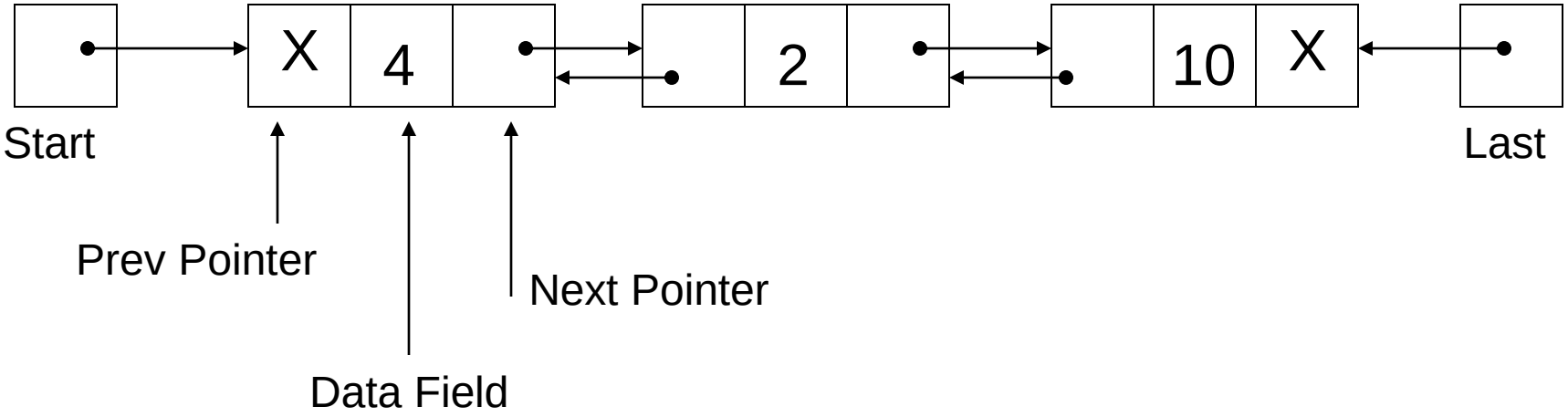
Two-way lists

- A two-way list is a linear collection of data elements, called nodes, where each node N is divided into three parts:
  - Information field
  - Forward Link which points to the next node
  - Backward Link which points to the previous node
- The starting address or the address of first node is stored in START / FIRST pointer .
- Another pointer can be used to traverse list from end. This pointer is called END or LAST.

Two-way lists(cont...)

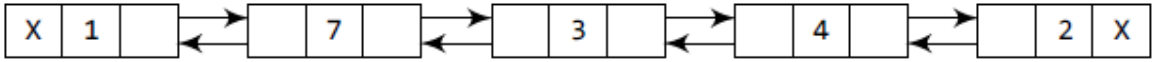
- Every node (except the last node) contains the address of the next node, and every node (except the first node) contains the address of the previous node.
- A two-way list (doubly linked list) can be traversed in either direction.

Representations of Two-way lists



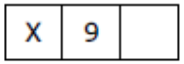
Insert an Item

Inserting a Node at the Beginning of a Doubly Linked List

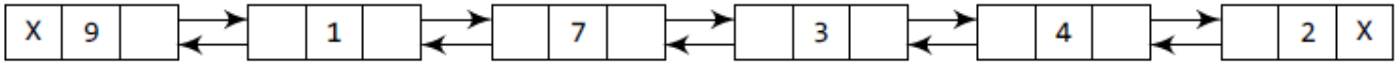


START

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



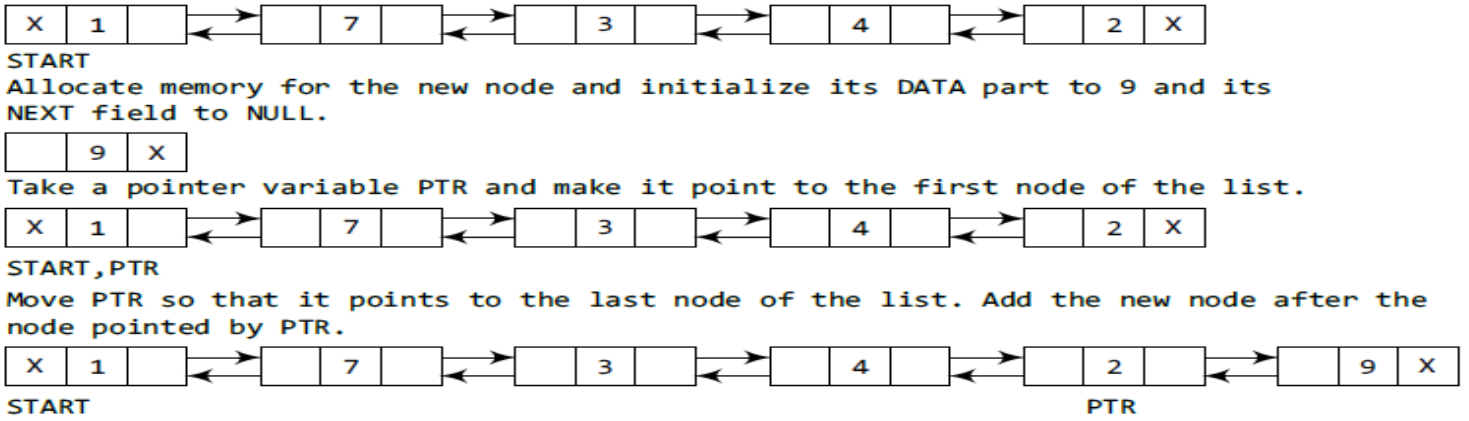
Add the new node before the START node. Now the new node becomes the first node of the list.



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

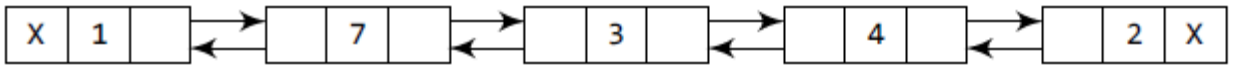


Step 1: IF AVAIL = NULL
Write **OVERFLOW**
Go to Step 11
[END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = NULL
Step 6: SET PTR = START
Step 7: *Repeat Step 8* while PTR -> NEXT != NULL
Step 8: SET PTR = PTR ->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

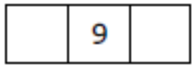
```
Step 1: IF AVAIL = NULL
              Write OVERFLOW
              Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> DATA != NUM
Step 7:        SET PTR = PTR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

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.



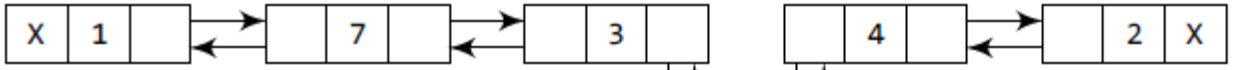
Take a pointer variable PTR and make it point to the first node of the list.

START,PTR

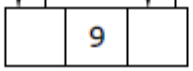
Move PTR further until the data part of PTR = value after which the node has to be inserted.

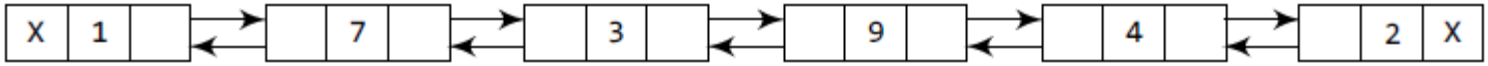
START PTR

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



START PTR

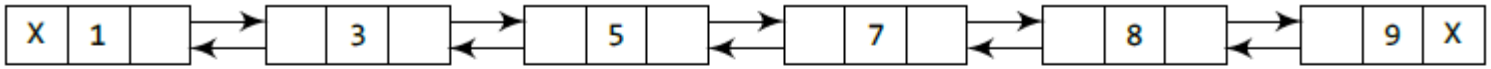




START

Delete an Item

Deleting the First Node from a Doubly Linked List



START

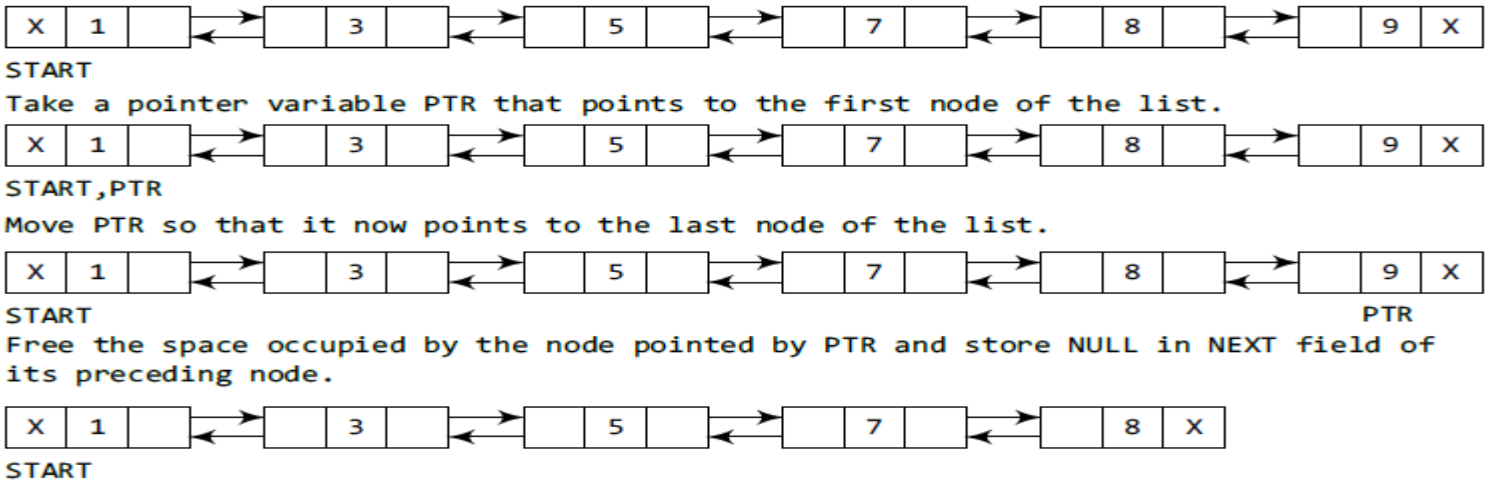
Free the memory occupied by the first node of the list and make the second node of the list as the START node.

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
```

Delete an Item

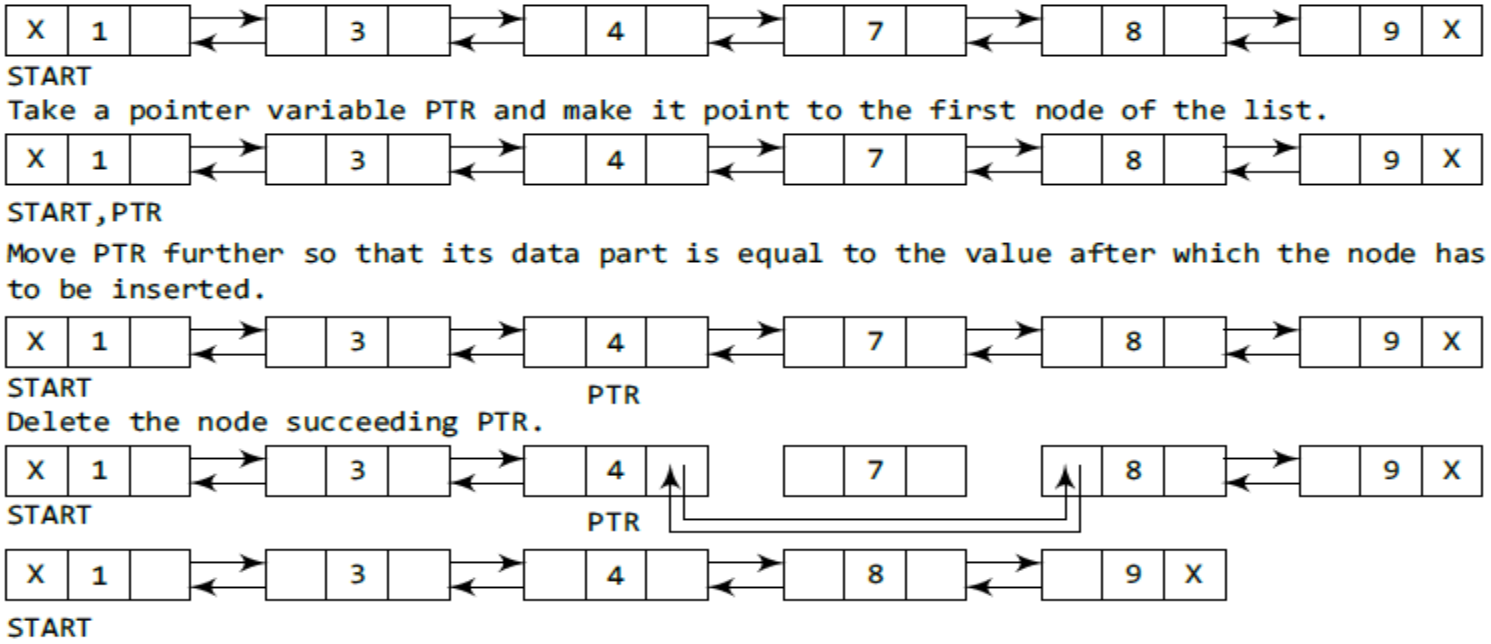
Deleting the Last Node from a Doubly Linked List




```
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
```

Delete an Item

Deleting the Node After a Given Node from 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

Deletion in 2-way Doubly LL

- DELTWL(INFO,FORW,BACK,START,AVAIL,LOC)
- 1.SET FORW[BACK[LOC]]=FORW[LOC]
- BACK[FORW[LOC]]=BACK[LOC]
- 2. SET FORW[LOC]=AVAIL and AVAIL=LOC
- 3.Exit

Insertion in 2-way Doubly LL

- INSTTWL(INFO,FORW,BACK,START,AVAIL,LOCA,LOCB,ITEM)
- 1. IF AVAIL=NULL Set OVERFLOW and Exit
- 2. SET NEW=AVAIL and AVAIL=FORW[AVAIL], INFO[NEW]=ITEM
- 3.SET FORW[LOCA]=NEW,FORW[NEW]=LOCB,
- BACK[LOCB]=NEW, BACK[NEW]=LOCA
- 4.Exit

Thank You