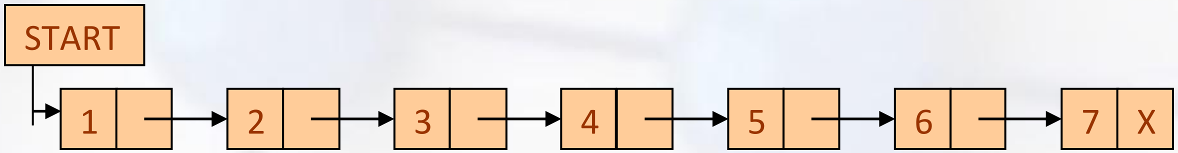
Linked Lists

Introduction

- A linked list is a linear collection of data elements called nodes in which linear representation is given by links from one node to the next node.
- Linked list is a data structure which in turn can be used to implement other data structures. Thus, it acts as building block to implement data structures like stacks, queues and their variations.
- A linked list can be perceived as a train or a sequence of nodes in which each node contains one or more data fields and a pointer to the next node.

Simple Linked List



- In the above linked list, every node contains two parts - one integer and the other a pointer to the next node.
- The left part of the node which contains data may include a simple data type, an array or a structure.
- The right part of the node contains a pointer to the next node (or address of the next node in sequence).
- The last node will have no next node connected to it, so it will store a special value called NULL.

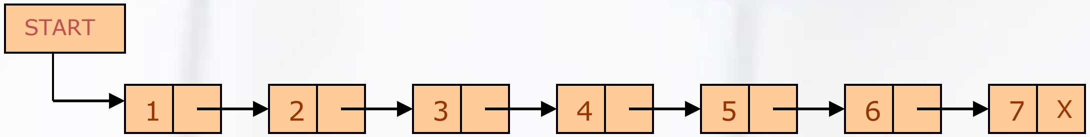
Traversing Linked Lists

- We can traverse the entire linked list using a single pointer variable called START.
- The START node contains the address of the first node; the next part of the first node in turn stores the address of its succeeding node.
- Using this technique the individual nodes of the list will form a chain of nodes.
- If START = NULL, this means that the linked list is empty and contains no nodes.
- In C, we can implement a linked list using the following code:

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

Singly Linked Lists

- 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.

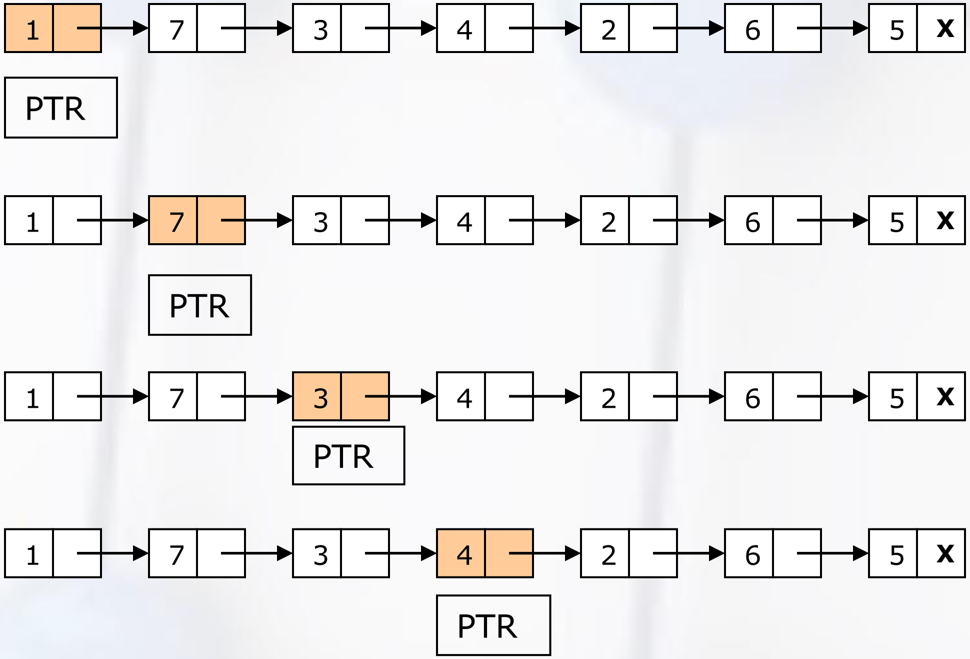


```
ALGORITHM FOR TRAVERSING A LINKED LIST

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

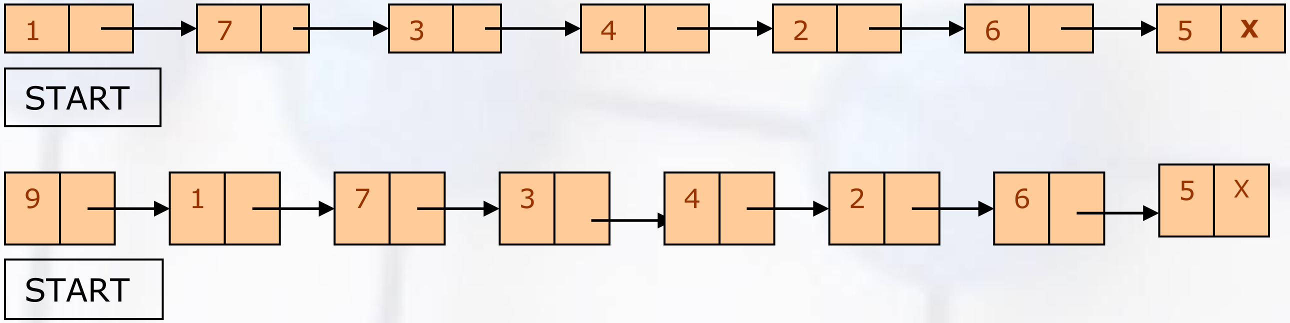
Searching a Linked List

```
ALGORITHM TO SEARCH A LINKED LIST

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

Searching for Val 4 in Linked List

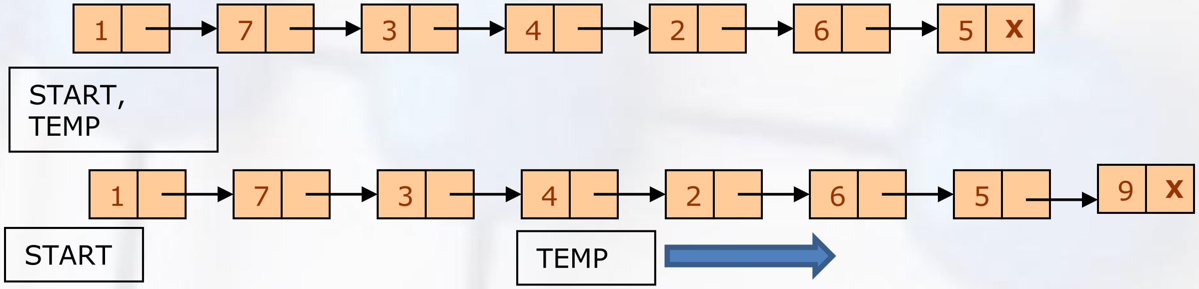


Inserting a Node at the Beginning



```
ALGORITHM TO INSERT A NEW NODE IN THE BEGINNING OF THE
LINKED LIST
Step 1: SET New_Node = GET_NODE()
Step 2: SET New_Node->DATA = VAL
Step 3: SET New_Node->Next = START
Step 4: SET START = New_Node
Step 5: EXIT
```

Inserting a Node at the End



ALGORITHM TO INSERT A NEW NODE AT THE END OF THE LINKED LIST

```
Step 1: SET New_Node = GET_NODE()
Step 2: SET New_Node->DATA = VAL
Step 3: SET New_Node->Next = NULL
Step 4: SET TEMP = START
Step 5: Repeat Step 8 while TEMP->NEXT != NULL
Step 6:              SET TEMP = TEMP ->NEXT
        [END OF LOOP]
Step 7: SET TEMP->NEXT = New_Node
Step 8: EXIT
```

Inserting a Node after Node that ahs Value NUM

ALGORITHM TO INSERT A NEW NODE AFTER A NODE THAT HAS VALUE NUM

```
Step 1: SET New_Node = GET_NODE()
Step 2: SET New_Node->DATA = VAL
Step 3: SET TEMP = START
Step 4: Repeat Steps 5 while TEMP->DATA != NUM
Step 5:   SET TEMP = TEMP->NEXT
         END OF LOOP]
Step 6: New_Node->NEXT = TEMP->NEXT
Step 7: TEMP->NEXT = New_Node
Step 8: EXIT
```

Deleting the First Node

Algorithm to delete the first node from the linked list

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

Deleting the Last Node

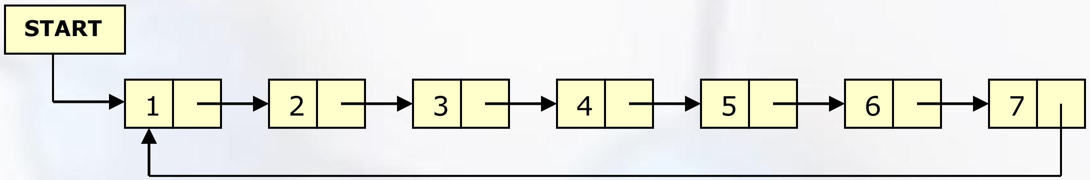
```
ALGORITHM TO DELETE THE LAST NODE OF THE LINKED LIST

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

Deleting the Node After a Given Node

```
ALGORITHM TO DELETE THE NODE AFTER A GIVEN NODE FROM THE
LINKED LIST

Step 1: IF START = NULL, then
               Write UNDERFLOW
               Go to Step 10
        [END OF IF]
Step 2: SET CUR = START
Step 3: SET PREV = NULL
Step 4: Repeat Step 5 and 6 while CUR->DATA != NUM
Step 5:                SET PREV = CUR
Step 6:                SET CUR = CUR->NEXT
        [END OF LOOP]
Step 8: SET PREV->NEXT = CUR->NEXT
Step 9: FREE CUR
Step 10: EXIT
```

Circular Linked List

- In a circular linked list, the last node contains a pointer to the first node of the list. We can have a circular singly listed list as well as circular doubly linked list. While traversing a circular linked list, we can begin at any node and traverse the list in any direction forward or backward until we reach the same node where we had started. Thus, a circular linked list has no beginning and no ending.

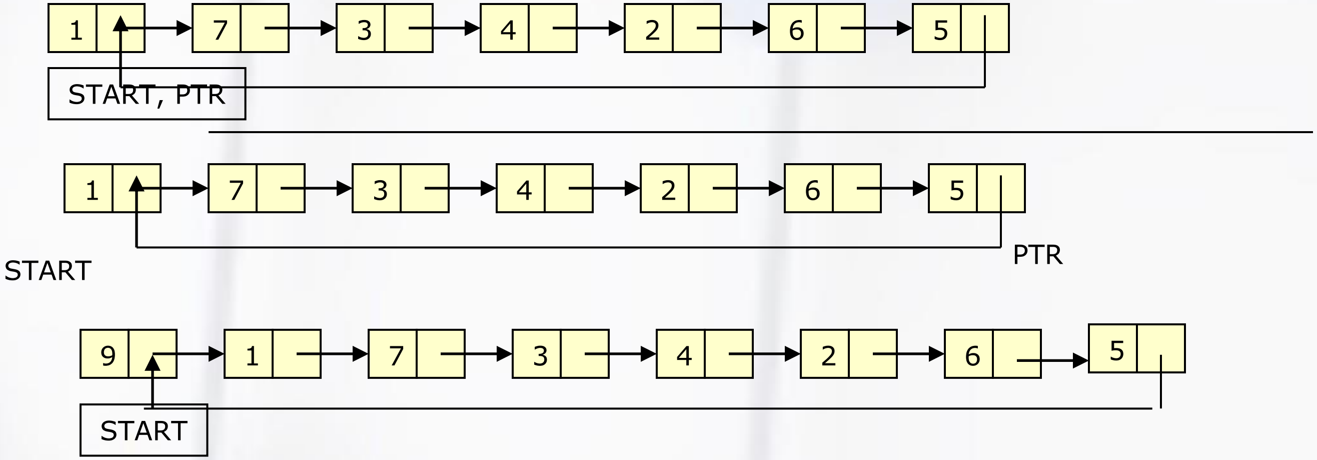


Circular Linked List

Algorithm to insert a new node in the beginning of **circular** the linked list

```
Step 1: IF AVAIL = NULL, then
               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 PTR = START
Step 6: Repeat Step 7 while PTR->NEXT != START
Step 7:               PTR = PTR->NEXT
Step 8: SET New_Node->Next = START
Step 8: SET PTR->NEXT = New_Node
Step 6: SET START = New_Node
Step 7: EXIT
```

Circular Linked List



Circular Linked List

Algorithm to insert a new node at the end of the circular linked list

```
Step 1: IF AVAIL = NULL, then
                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 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
```

Circular Linked List

```
Algorithm to insert a new node after a node that has value NUM

Step 1: IF AVAIL = NULL, then
                 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 Step 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
```

Circular Linked List

Algorithm to delete the first node from the circular linked list

```
Step 1: IF START = NULL, then
                    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 IF]
Step 5: SET PTR->NEXT = START->NEXT
Step 6: FREE START
Step 7: SET START = PTR->NEXT
Step 8: EXIT
```

Circular Linked List

Algorithm to delete the last node of the **circular** linked list

```
Step 1: IF START = NULL, then
                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 PREPTR = PTR
Step 5:                SET PTR = PTR->NEXT
        [END OF LOOP]
Step 6: SET PREPTR->NEXT = START
Step 7: FREE PTR
Step 8: EXIT
```

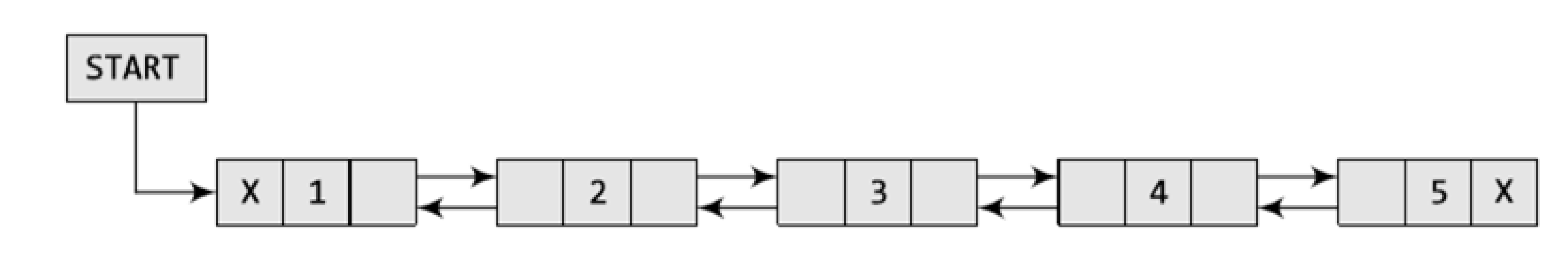
Circular Linked List

Algorithm to delete the node after a given node from the circular linked list

```
Step 1: IF START = NULL, then
                    Write UNDERFLOW
                    Go to Step 9
          [END OF IF]
Step 2: SET PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Step 5 and 6 while PREPTR->DATA != NUM
Step 5:             SET PREPTR = PTR
Step 6:             SET PTR = PTR->NEXT
          [END OF LOOP]
Step 7: SET PREPTR->NEXT = PTR->NEXT
Step 8: FREE PTR
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 previous node in the sequence. Therefore, it consists of three parts and not just two. The three parts are data, a pointer to the next node and a pointer to the previous node



Doubly Linked List

- In C language, the structure of a doubly linked list is given as,

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

- The prev field of the first node and the next field of the last node will contain NULL. The prev field is used to store the address of the preceding node. This would enable to traverse the list in the backward direction as well.

Doubly Linked List

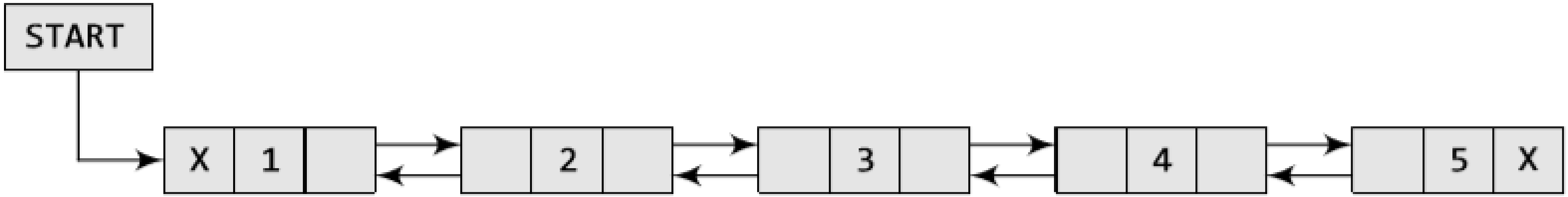


Figure 6.37 Doubly linked list

Doubly Linked List

Algorithm to insert a new node in the beginning of the doubly linked list

```
Step 1: IF AVAIL = NULL, then
            Write OVERFLOW
            Go to Step 8
      [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 = New_Node
Step 8: EXIT
```

Doubly Linked List

```
Algorithm to insert a new node at the end of the doubly
linked list

Step 1: IF AVAIL = NULL, then
                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: New_Node->PREV = PTR
Step 11: EXIT
```

Doubly Linked List

Algorithm to insert a new node after a node that has value NUM

```
Step 1: IF AVAIL = NULL, then
               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 8 while PTR->DATA != NUM
Step 7:                SET PTR = PTR->NEXT
        [END OF LOOP]
Step 8: New_Node->NEXT = PTR->NEXT
Step 9: SET New_Node->PREV = PTR
Step 10: SET PTR->NEXT = New_Node
Step 11: EXIT
```

Doubly Linked List

Algorithm to delete the first node from the doubly linked list

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

Doubly Linked List

Algorithm to delete the last node of the doubly linked list

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

Doubly Linked List

Algorithm to delete the node after a given node from the doubly linked list

```
Step 1: IF START = NULL, then
             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
```

Circular Doubly Linked List

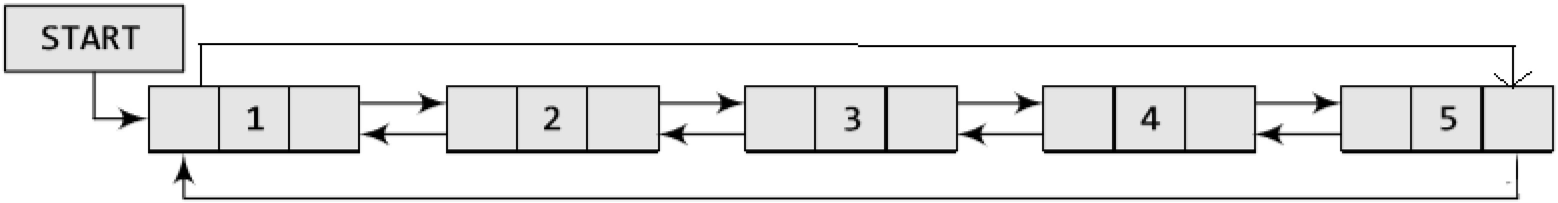
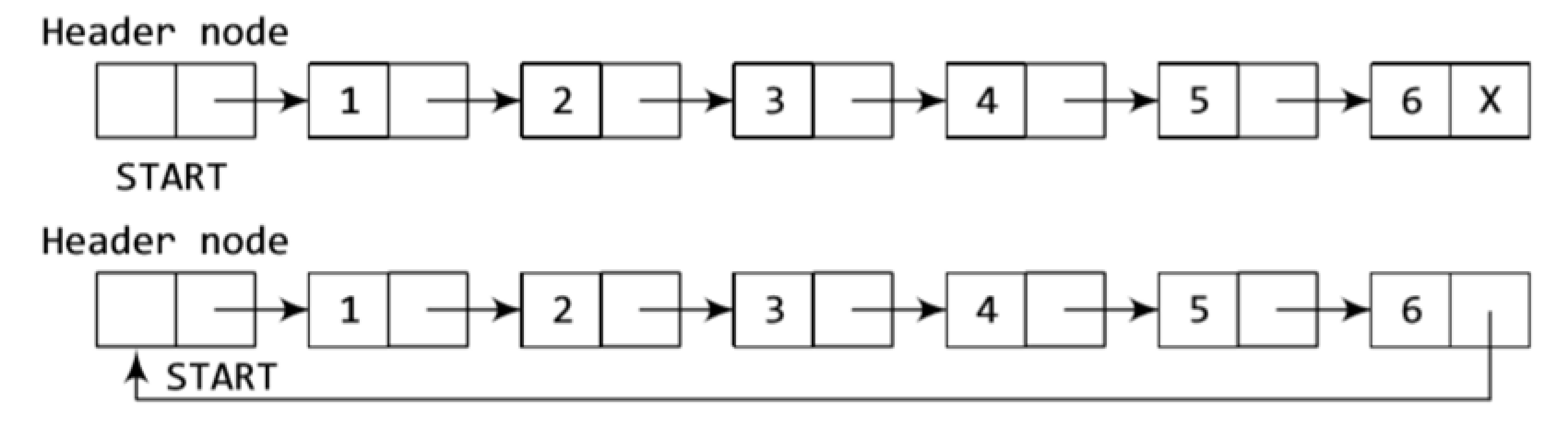


Figure 6.55 Circular doubly linked list

Refer classroom demo programs for the algorithms and implementation of Circular Doubly Linked List.

Header Linked List

Linked list with a header node.
- A node called header node is dedicatedly used as the starting node of the linked list. Header node points to the first data node. Rest is same as normal linked list.
- In a circular Header Lined list, last data node points to the header node.
- Header linked list makes the implementation of algorithm simpler, but at the cost extra memory needed for maintaining a header node.


Header node
1
2
3
4
5
6
X
START
Header node
1
2
3
4
5
6
START