OXFORD
HIGHER EDUCATION

Data Structures Using C

Reema Thareja, Assistant Professor
Institute of Information Technology and Management
GGS IP University

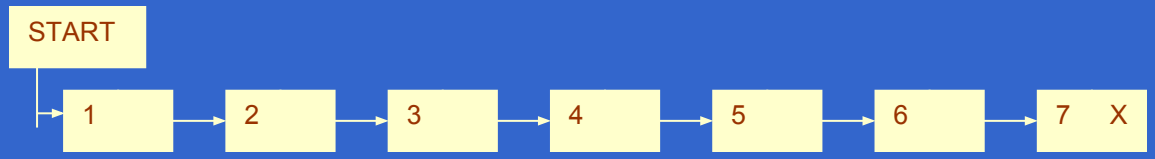
OXFORD
HIGHER EDUCATION

CHAPTER 8

LINKED LISTS

INTRODUCTION

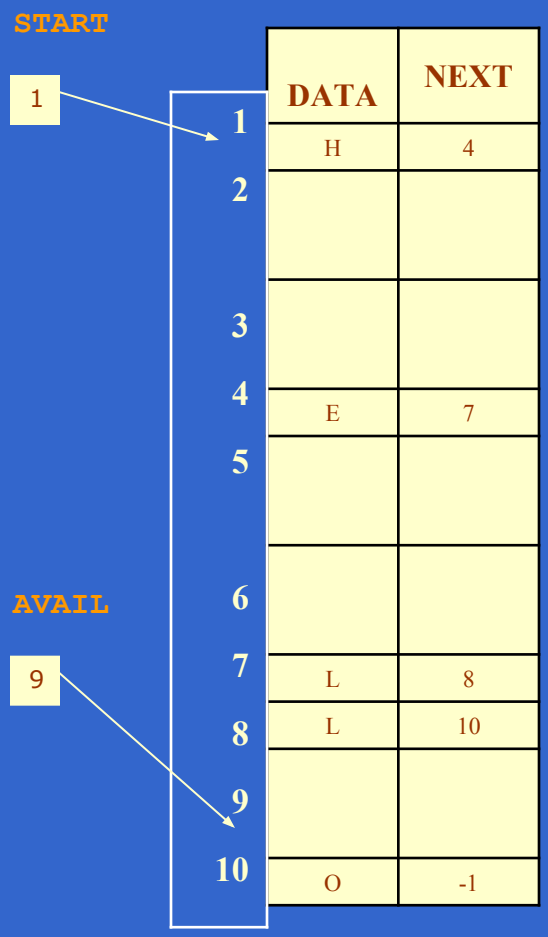
- A linked list in simple terms is a linear collection of data elements. These data elements are called nodes.
- Linked list is a data structure which in turn can be used to implement other data structures. Thus, it acts as building block to implement data structures like stacks, queues and their variations.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



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.

INTRODUCTION contd.

- Linked list contains a pointer variable, START which stores the address of the first node in the list.
- We can traverse the entire list using a single pointer variable START. The START node will contain the address of the first node; the next part of the first node will in turn store 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 will implement a linked list using the following code:
- struct node
- {
- int data;
- struct node *next;
- };



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

If we want to add a node to an already existing linked list in the memory, we will first find any free space in the memory and then use it to store the information.

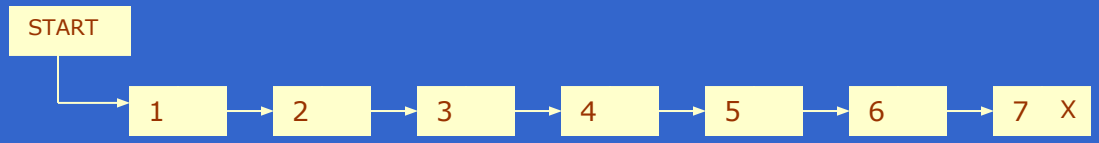
The operating system maintains a free pool which is a linked list of a of all free memory cells. It maintains a pointer variable AVAIL which stores the address of the first free space.

When you delete a node from the linked list, the operating system adds the freed memory to the free pool.

The operating system will perform this operation whenever it finds the CPU idle or whenever the programs are falling short of memory. The operating system scans through all the memory cells and mark the cells that are being used by some or the other program. Then, it collects all those cells which are not being used and add their address to the free pool so that it can be reused by the programs. This process is called garbage collection. The whole process of collecting unused memory cells (garbage collection) is transparent to the programmer.

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.

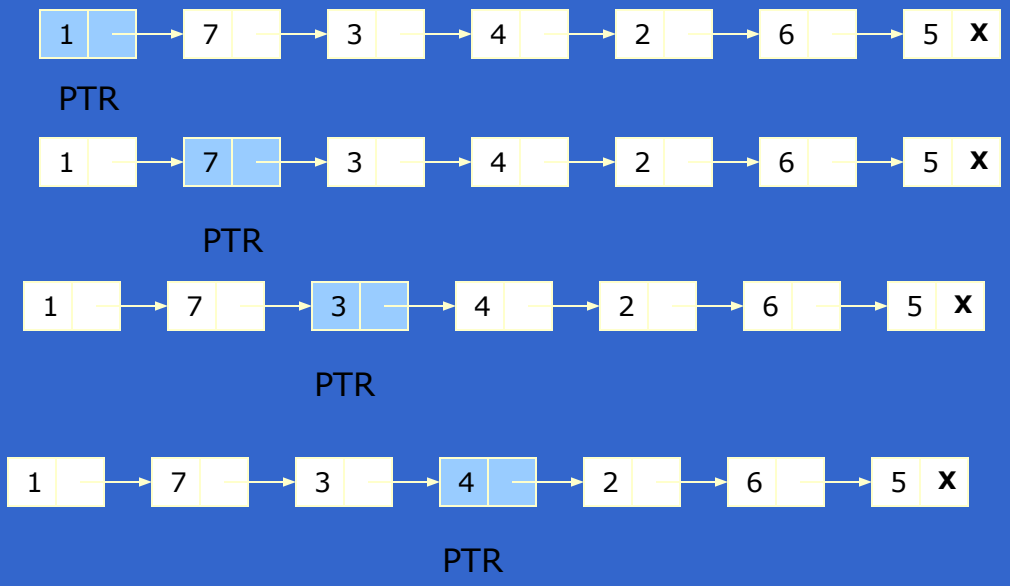



```
Algorithm for traversing a linked list

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
```

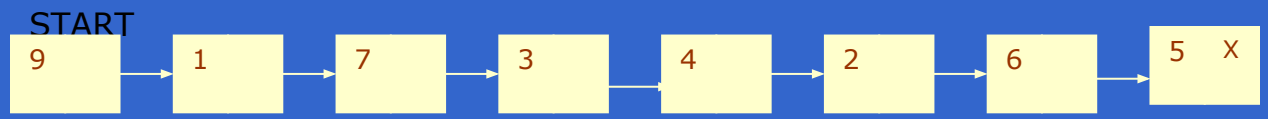
```
Algorithm to print the information stored in each node of the linked list

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

```
Algorithm to print the number of nodes in the 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: EXIT
```

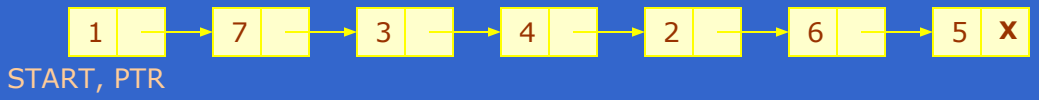
```
Algorithm to search an unsorted linked list

Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Steps 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
```

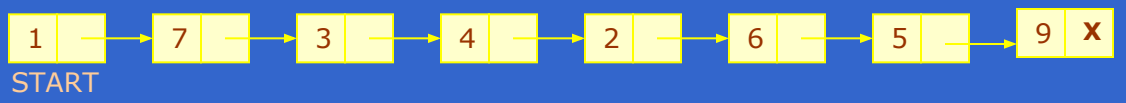


START




```
Algorithm to insert a new node in the beginning of 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 New_Node->Next = START
Step 6: SET START = New_Node
Step 7: EXIT
```





PTR

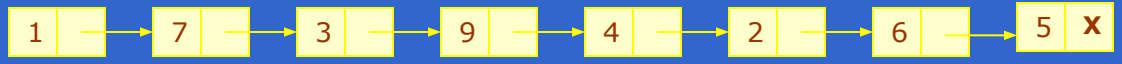
```
Algorithm to insert a new node at the end of the linked list

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

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 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

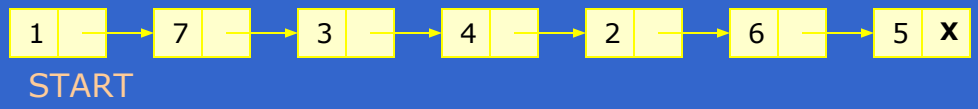
START, PTR, PREPTR

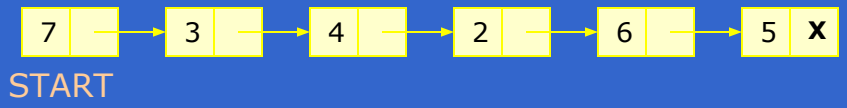
START PREPTR PTR



START

```
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 PTR = START
Step 3: SET START = START->NEXT
Step 4: FREE PTR
Step 5: EXIT
```



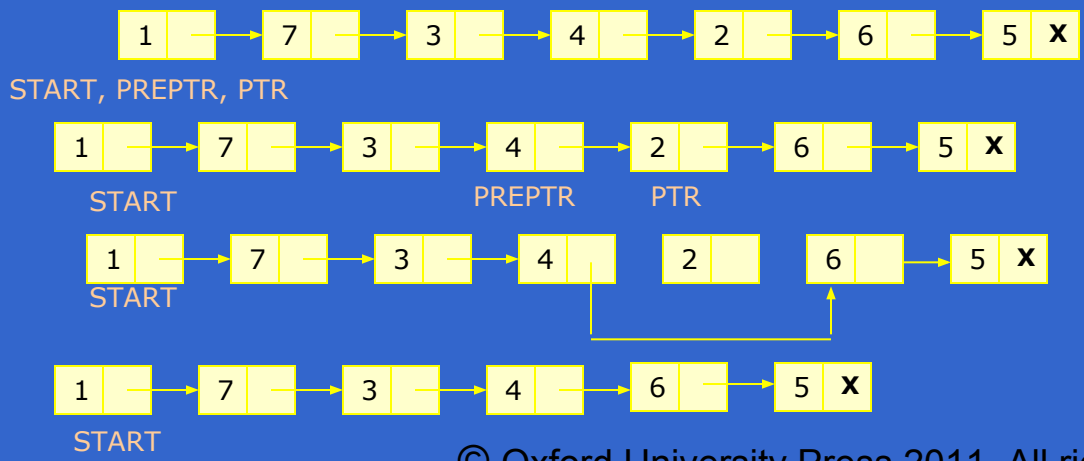


```
Algorithm to delete the last node of the linked list

Step 1: IF START = NULL, then
          Write UNDERFLOW
          Go to Step 8
     [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 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
```

START, PREPTR, PTR

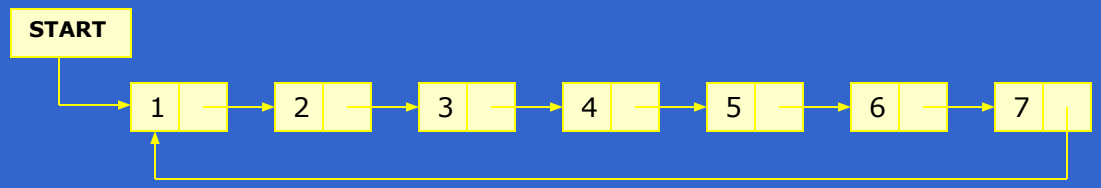
START PREPTR PTR

```
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 PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Step 5 and 6 while PRETR->DATA != NUM
Step 5:         SET PREPTR = PTR
Step 6:         SET PTR = PTR->NEXT
     [END OF LOOP]
Step7: SET TEMP = PTR->NEXT
Step 8: SET PREPTR->NEXT = TEMP->NEXT
Step 9: FREE TEMP
Step 10: EXIT
```

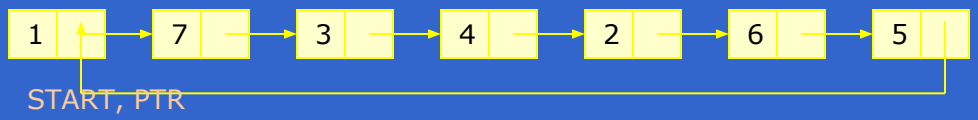


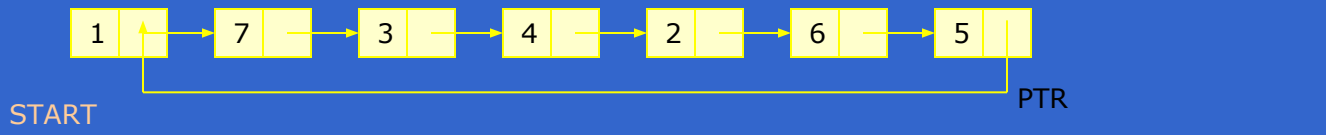
Circular Linked List

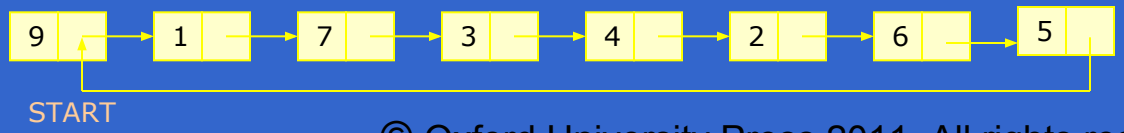
- In a circular linked list, the last node contains a pointer to the first node of the list. We can have a circular singly listed list as well as circular doubly linked list. While traversing a circular linked list, we can begin at any node and traverse the list in any direction forward or backward until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



```
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
```

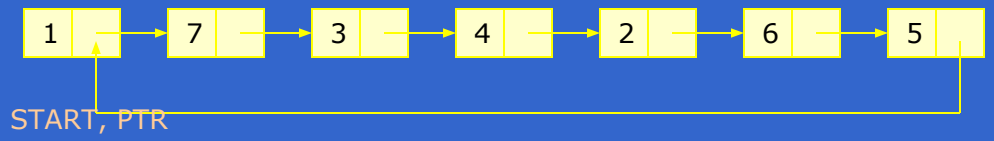


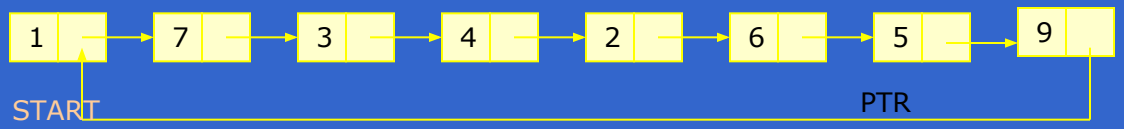




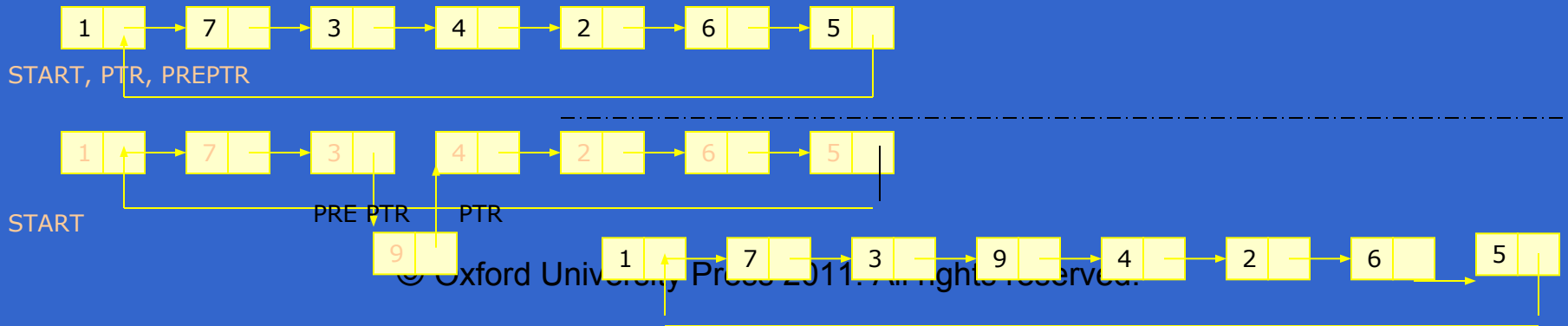
Algorithm to insert a new node at the end of the **circular** linked list

```
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
```



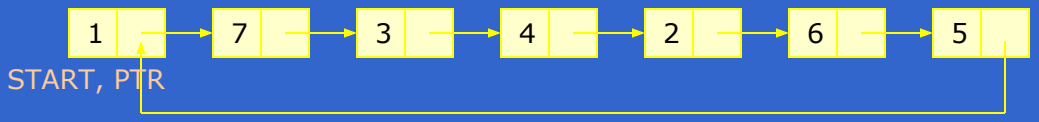


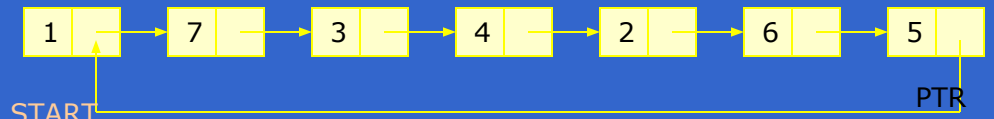
```
Algorithm to insert a new node after a node that has value NUM

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
```

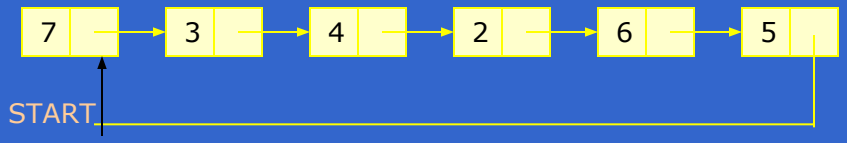


Algorithm to delete the first node from the circular linked list

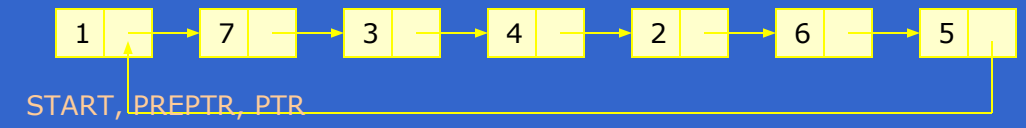
```
Step 1: IF START = NULL, then
            Write UNDERFLOW
            Go to Step 8
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR->NEXT != START
Step 4:          SET PTR = PTR->NEXT
        [END OF IF]
Step 5: SET PTR->NEXT = START->NEXT
Step 6: FREE START
Step 7: SET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

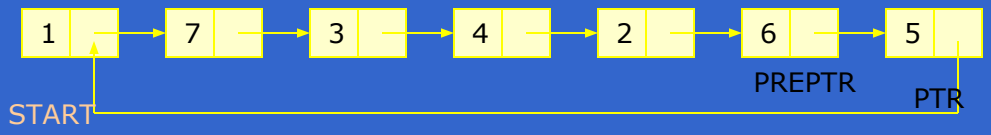


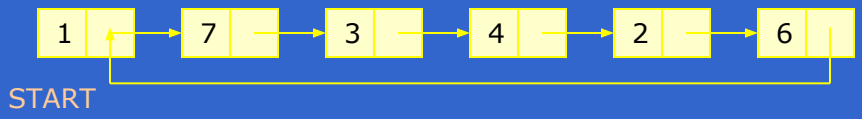




```
Algorithm to delete the last node of the circular linked list

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
```

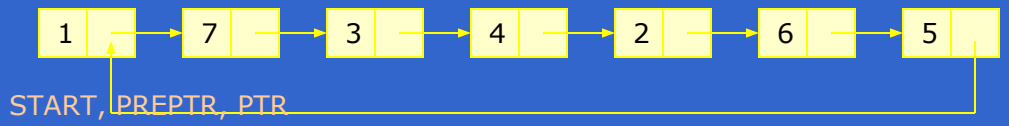


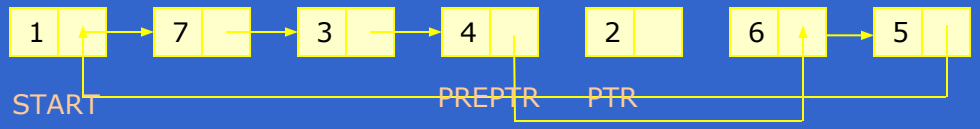


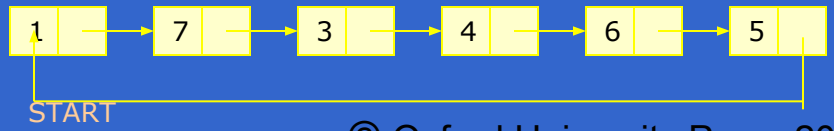


Algorithm to delete the node after a given node from the circular linked list

```
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
```

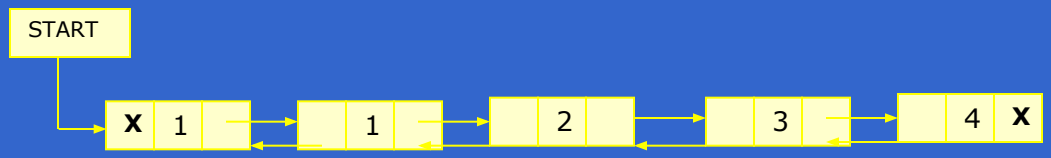






Doubly Linked List

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



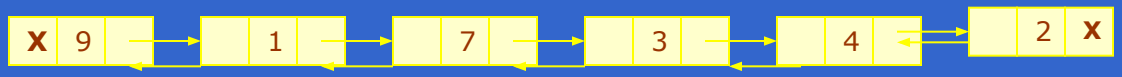
In C language, the structure of a doubly linked list is given as,

```
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.

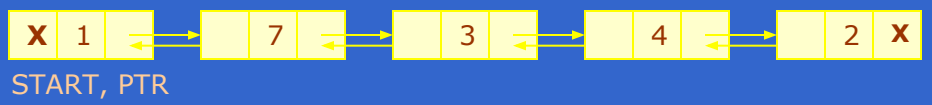
```
Algorithm to insert a new node in the beginning of the doubly linked list

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
```

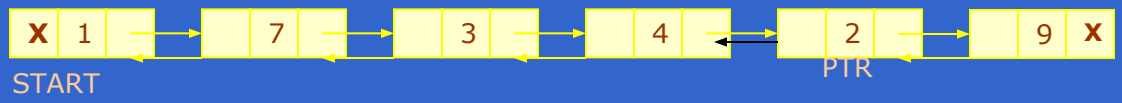
START



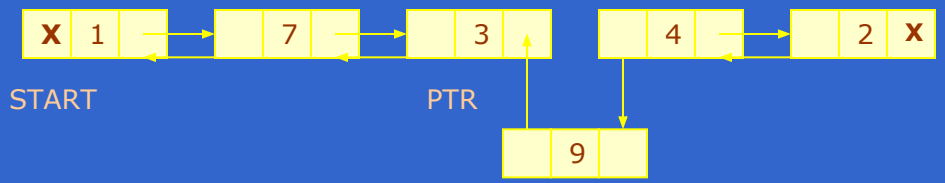
START

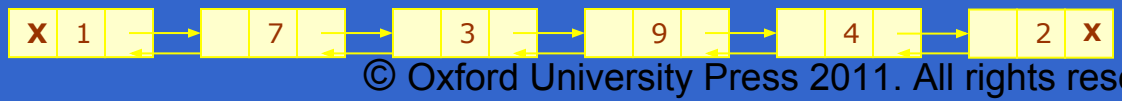
```
Algorithm to insert a new node at the end of the doubly linked list

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
```





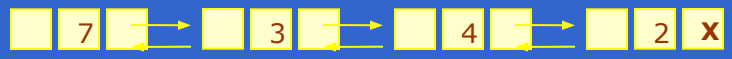
```
Algorithm to insert a new node after a node that has value NUM

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
```



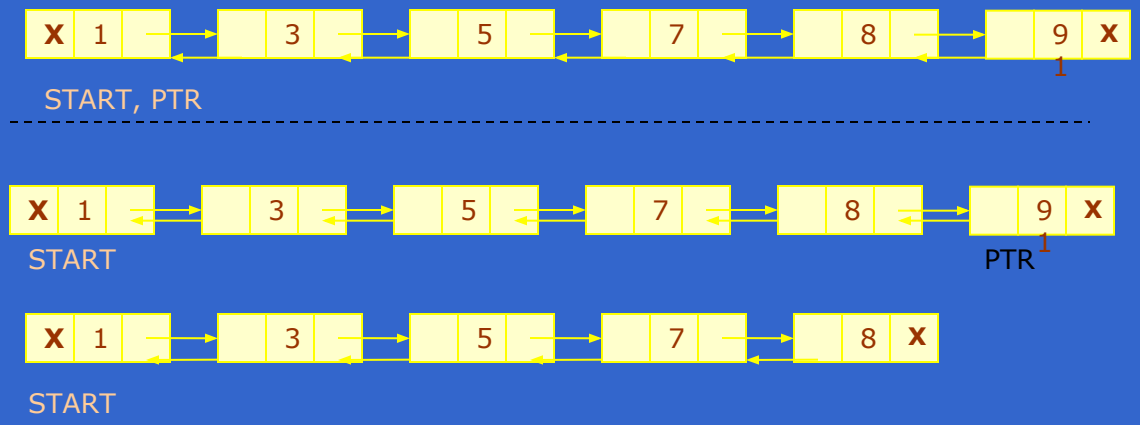


```
Algorithm to delete the first node from the doubly linked list

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
```

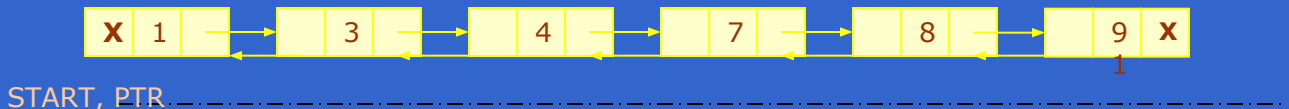
START, PTR

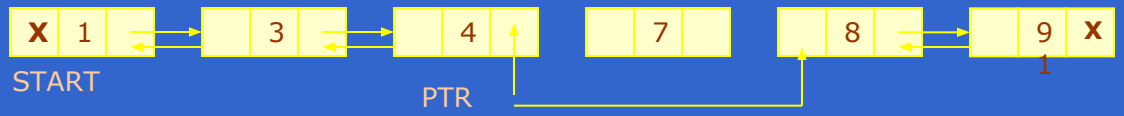


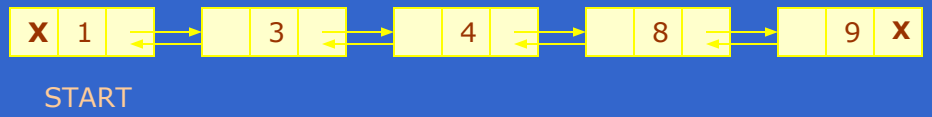
```
Algorithm to delete the last node of the doubly linked list

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
```



```
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
```

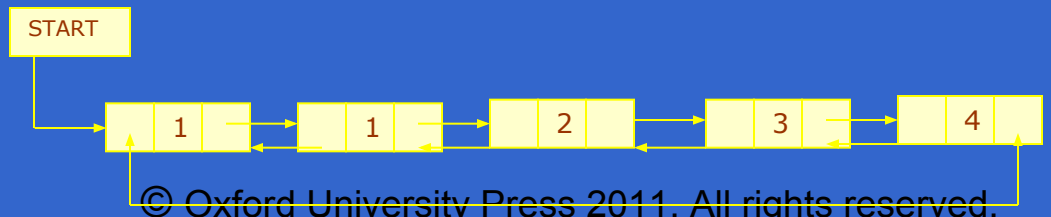






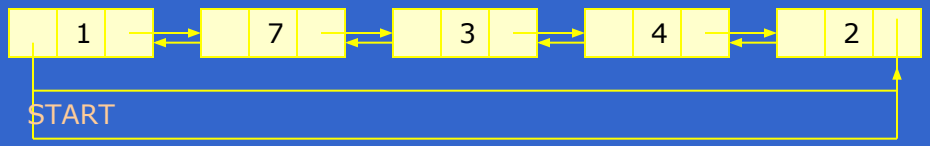
Circular Doubly Linked List

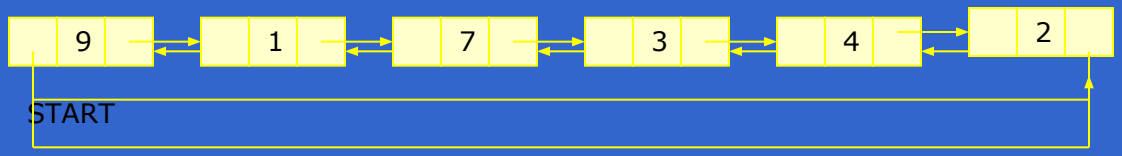
- A circular doubly linked list or a circular two way linked list is a more complex type of linked list which contains a pointer to the next as well as previous node in the sequence.
- The difference between a doubly linked and a circular doubly linked list is same as that exists between a singly linked list and a circular linked list. The circular doubly linked list does not contain NULL in the previous field of the first node and the next field of the last node. Rather, the next field of the last node stores the address of the first node of the list, i.e; START. Similarly, the previous field of the first field stores the address of the last node.
- Since a circular doubly linked list contains three parts in its structure, it calls for more space per node and for more expensive basic operations. However, it provides the ease to manipulate the elements of the list as it maintains pointers to nodes in both the directions . The main advantage of using a circular doubly linked list is that it makes searches twice as efficient.



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

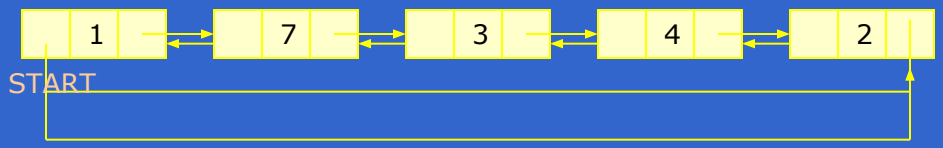
Step 1: IF AVAIL = NULL, then
Write OVERFLOW
Go to Step 12
[END OF IF]
Step 2: SET New_Node = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET New_Node->DATA = VAL
Step 6: SET START->PREV->NEXT = new_node;
Step 7: SET New_Node->PREV = START->PREV;
Step 8: SET START->PREV= new_Node;
Step 9: SET new_node->next = START;
Step 10: SET START = New_Node
Step 11: EXIT

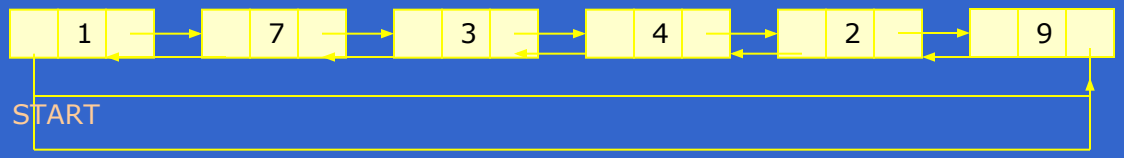




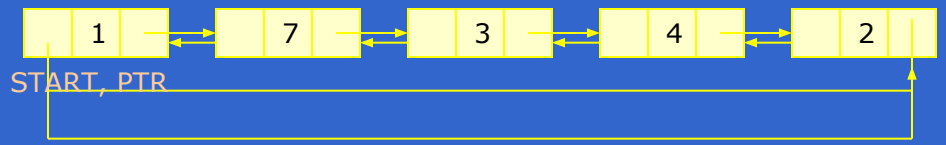
Algorithm to insert a new node at the end of the **circular doubly** linked list

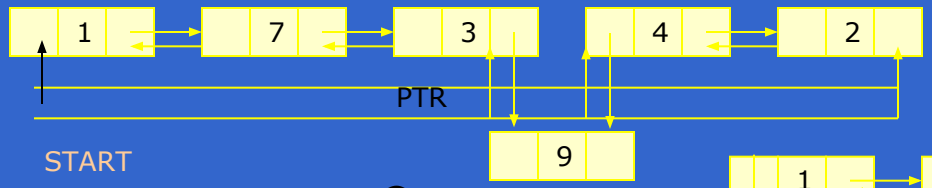
```
Step 1: IF AVAIL = NULL, then
          Write OVERFLOW
          Go to Step 11
     [END OF IF]
Step 2: SET New_Node = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET New_Node->DATA = VAL
Step 5: SET New_Node->Next = START
Step 6: SET New_Node->PREV = START->PREV
Step 7: EXIT
```

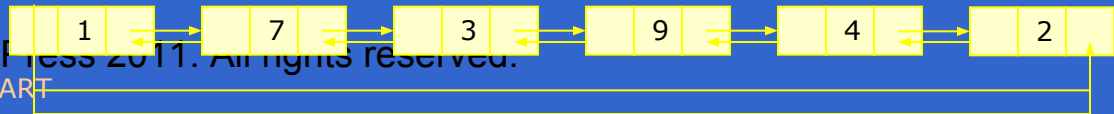




```
Algorithm to insert a new node after a node that has value NUM

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 PTR->NEXT->PREV = New_Node
Step 9: SET New_Node->PREV = PTR
Step 10: SET PTR->NEXT = New_Node
Step 11: EXIT
```

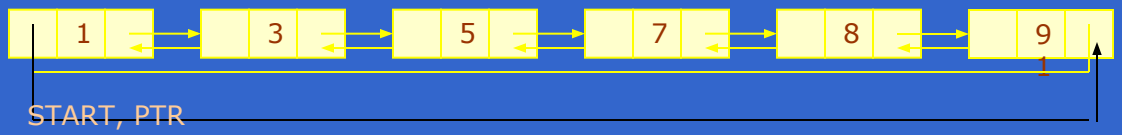


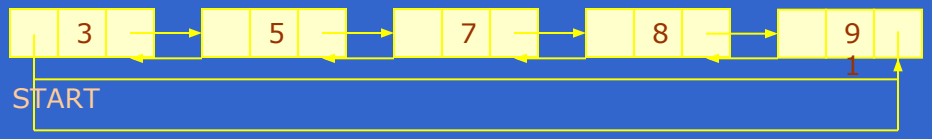




Algorithm to delete the first node from the **circular doubly** linked list

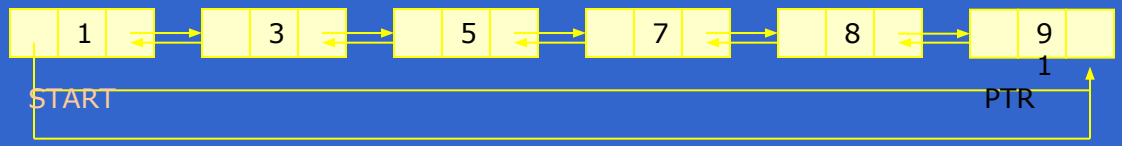
```
Step 1: IF START = NULL, then
          Write UNDERFLOW
          Go to Step 8
     [END OF IF]
Step 2: SET PTR = START
Step 3: SET PTR->PREV=>NEXT= PTR->NEXT
Step 4: SET PTR->NEXT->PREV = PTR->PREV
Step 5: SET START = START->NEXT
Step 6: FREE PTR
Step 7: EXIT
```

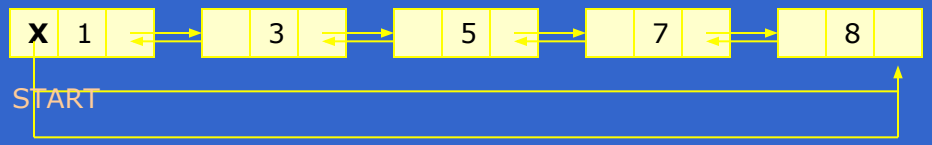




Algorithm to delete the last node of the **circular doubly** linked list

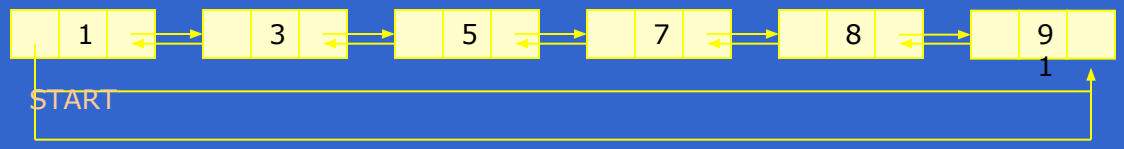
```
Step 1: IF START = NULL, then
          Write UNDERFLOW
          Go to Step 8
      [END OF IF]
Step 2: SET PTR = START->PREV
Step 5: SET PTR->PREV->NEXT = START
Step 6: SET START->PREV = PTR->PREV
Step 7: FREE PTR
Step 8: EXIT
```

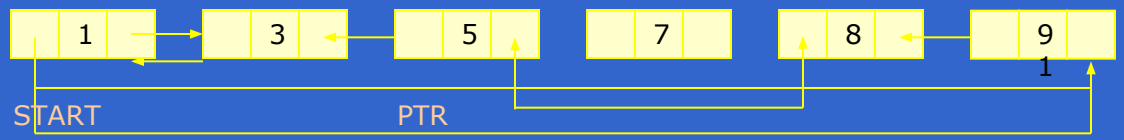


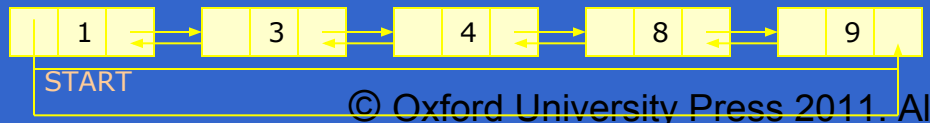


Algorithm to delete the node after a given node from the **circular doubly linked list**

```
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
```

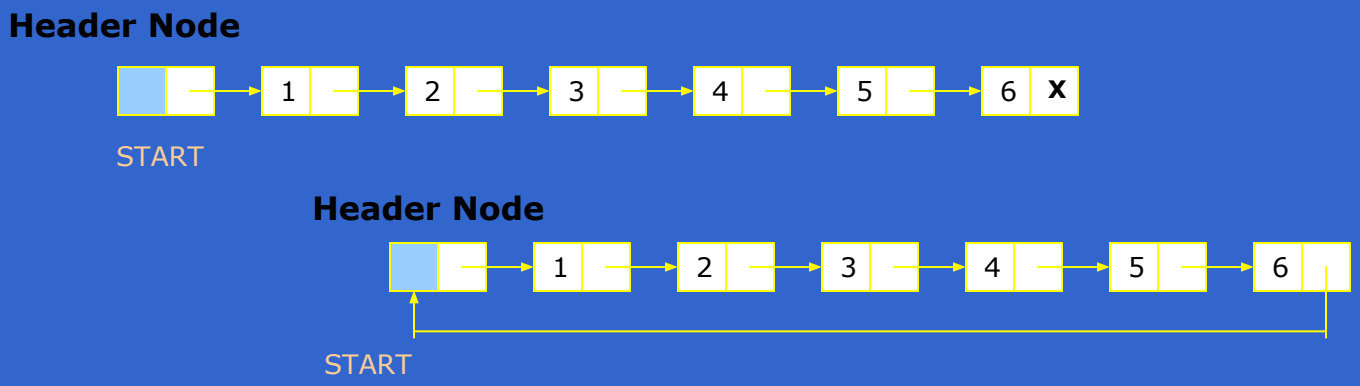






Header Linked List

- A header linked list is a special type of linked list which contains a header node at the beginning of the list. So, in a header linked list START will not point to the first node of the list but START will contain the address of the header node. There are basically two variants of a header linked list-
- Grounded header linked list which stores NULL in the next field of the last node
- Circular header linked list which stores the address of the header node in the next field of the last node. Here, the header node will denote the end of the list.



```
Algorithm to traverse a Circular Header Linked List

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

```
Algorithm to insert a new node after a given node

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