

# Concepts of linked list

- 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.
- Similar to array, it is a linear collection of data elements of the **same type**.
- Different from array, data elements of linked list are generally not lined in consecutive memory space; instead they are **dispersed** in various locations

# Concepts of linked list

- 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.

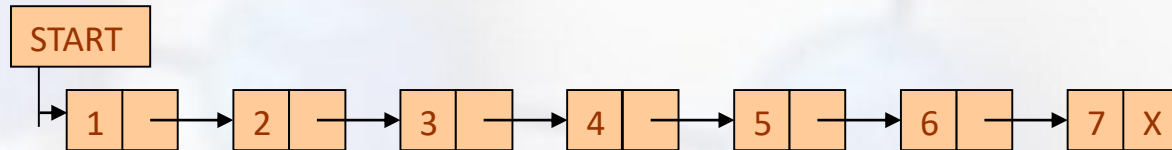
# Element of linked list

- Linked list element (node) is user defined structure data type, typically contains two parts
  - data variables
  - pointers to next elements, hold the addresses of next elements

Example:

```
struct node {  
    int data;           // data  
    struct node *next; // pointer to next element  
};
```

# 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 data part of the node which contains data may include a simple data type, an array or a structure.
- The pointer 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 NULL value.
- A linked list is defined by the pointer pointing to the first node, e.g START

# Linked List Operations

1. Create a node and linked list
2. Traversal, e.g. display all elements
3. Search node
4. Insert node  
at beginning, at end, after/before a node
5. Delete node  
at beginning, at end, after/before a node
6. Sort

# How to create nodes

A node is a structure data type, can be created in two methods, statically and dynamically.

- Static method
  - use array of structures
  - declared as globally outside functions
  - declared locally within a function
- Dynamic method (mostly used for linked list)
  - use stdlib function `malloc(size)` get memory space

```
struct node *np = (struct node*) malloc(sizeof(struct node));
```

# How to create nodes

```
struct node *np = (struct node*) malloc(sizeof(struct node));
```

At run time, OS allocates consecutive sizeof(struct node) bytes in the heap region,

return the address of the address of the first memory cell,

store the address to struct node type pointer np.

Need (struct node\*) to cast the return address to struct node pointer value.

Need use free(np) to release when deleting the node!!!  
Otherwise cause memory leaking

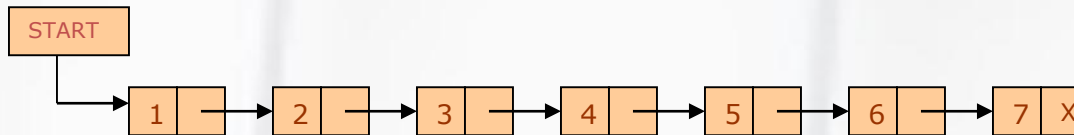
# Traversing Linked Lists

- 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.



## 2. 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.



Example:

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

# Traversal Singly Linked Lists

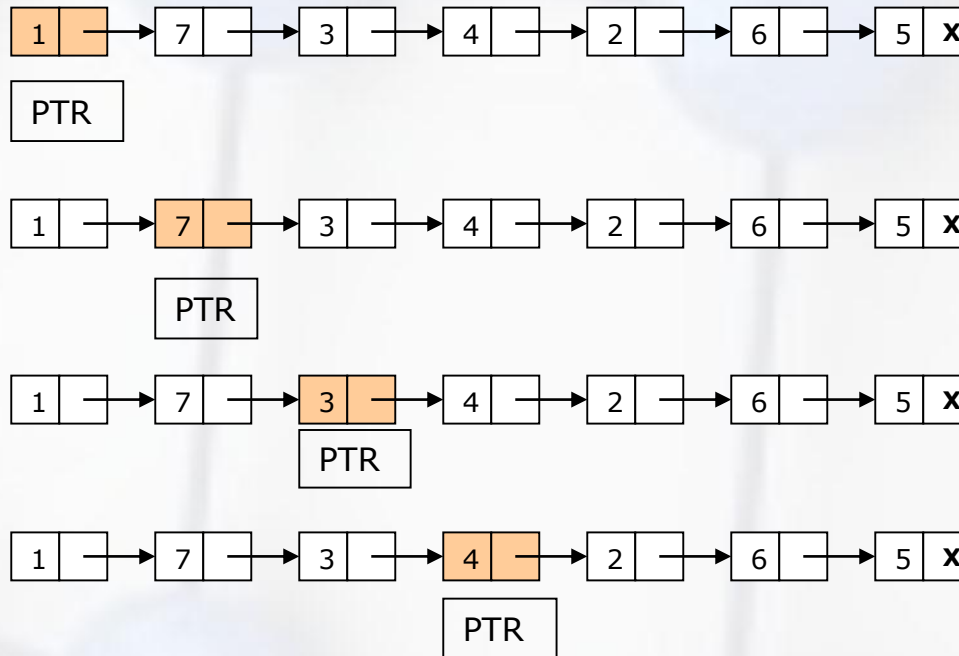
## ALGORITHM FOR TRAVERSING A LINKED LIST

```
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
```

```
void display(struct node *ptr) {
    while(ptr != NULL) {
        printf("%d ", ptr->data); // process
        ptr = ptr->next;
    }
}
```

Call as display(START);

# Searching for Val 4 in Linked List



# Searching a Linked List

## ALGORITHM TO SEARCH A LINKED LIST

Step 1: [INITIALIZE] SET PTR = START

Step 2: Repeat Step 3 while PTR != NULL

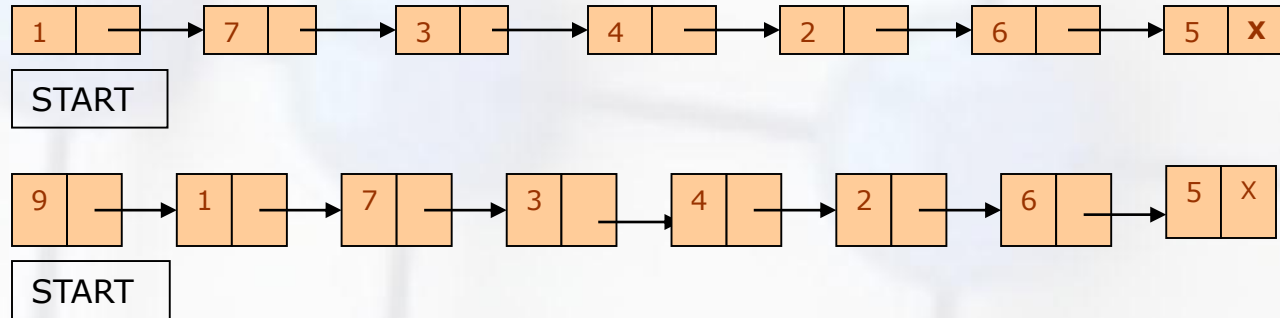
```
Step 3:   IF VAL = PTR->DATA
           SET POS = PTR
           Go To Step 5
        ELSE
           SET PTR = PTR->NEXT
        [END OF IF]
    [END OF LOOP]
```

Step 4: SET POS = NULL // not found

Step 5: EXIT // found, output POS

```
struct node* search(struct node *ptr, int num) {
    while((ptr != NULL) && (ptr->data != num)) {
        ptr = ptr->next;
    }
    return ptr;
}
// call example, search(START, 4)
```

# Inserting a Node at the Beginning



**ALGORITHM: 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**

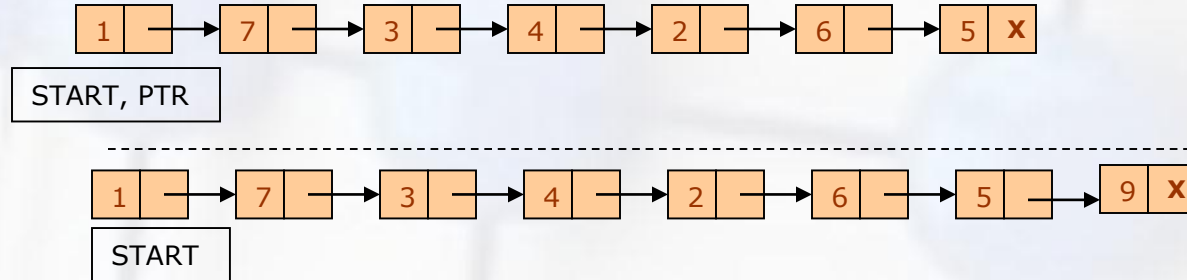
**Step 5: SET New\_Node->Next = START**

**Step 6: SET START = New\_Node**

**Step 7: EXIT**

**See example**

# Inserting a Node at the End



## 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

# Inserting a Node after Node that has Value NUM

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 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

# Deleting the First Node

Algorithm to delete the first node from the linked list

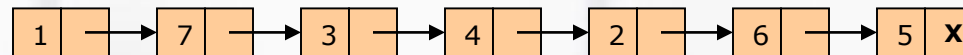
Step 1: IF START = NULL, then  
    Write UNDERFLOW  
    Go to Step 5  
[END OF IF]

Step 2: SET PTR = START

Step 3: SET START = START->NEXT

Step 4: FREE PTR

Step 5: EXIT



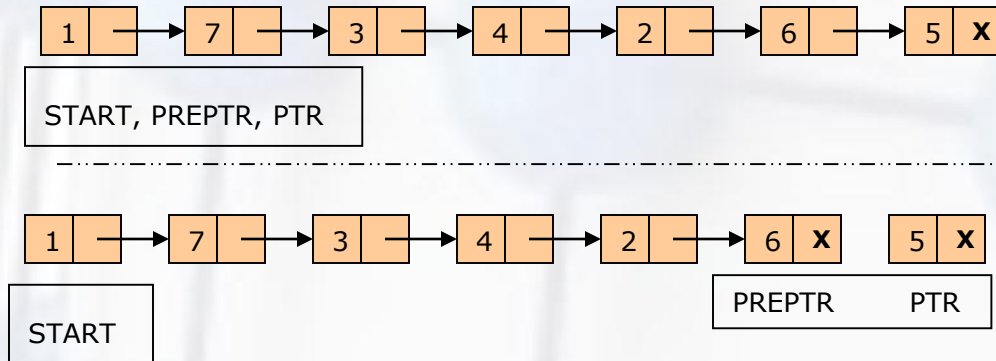
START



START



# 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 PTR = START

Step 3: Repeat Steps 4 and 5 while PTR->NEXT != NULL

Step 4: SET PREPTR = PTR

Step 5: SET PTR = PTR->NEXT

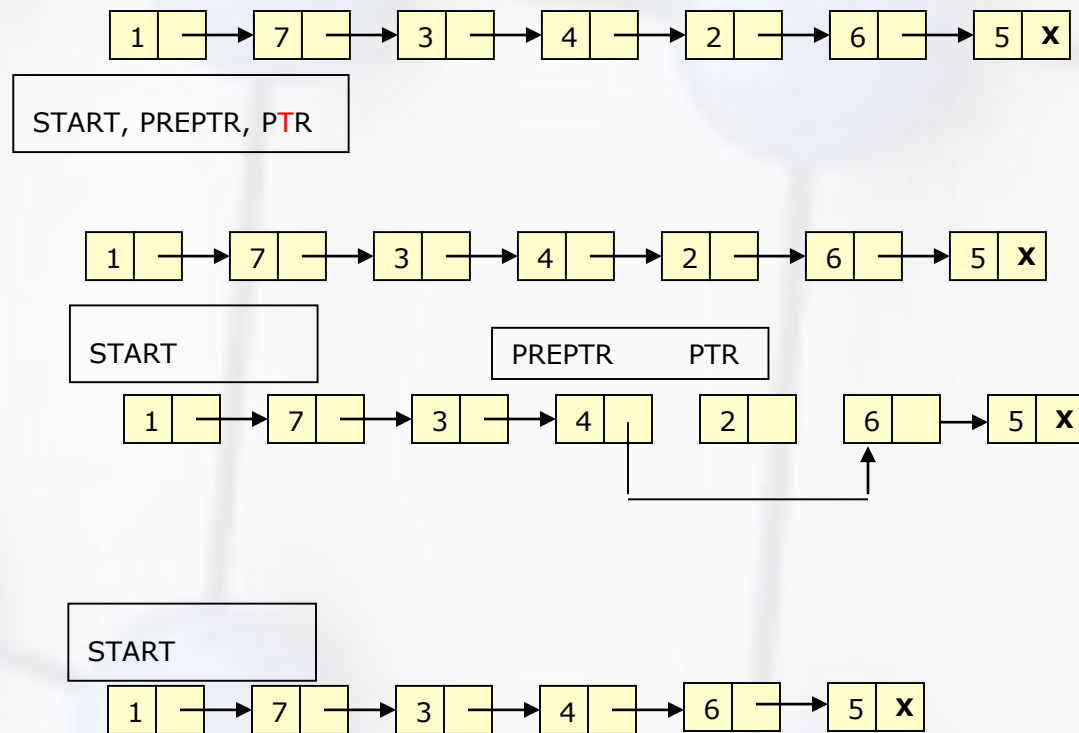
[END OF LOOP]

Step 6: SET PREPTR->NEXT = NULL

Step 7: FREE PTR

Step 8: EXIT

# Deleting the Node After a Given Node



# 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 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]

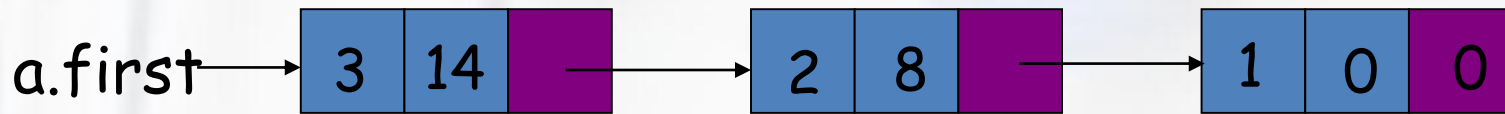
Step 7: SET TEMP = PTR->NEXT

Step 8: SET PREPTR->NEXT = TEMP->NEXT

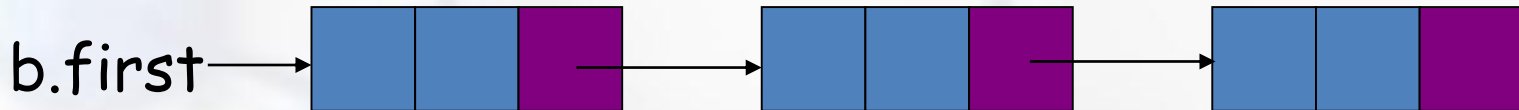
Step 9: FREE TEMP

Step 10: EXIT

# Polynomials



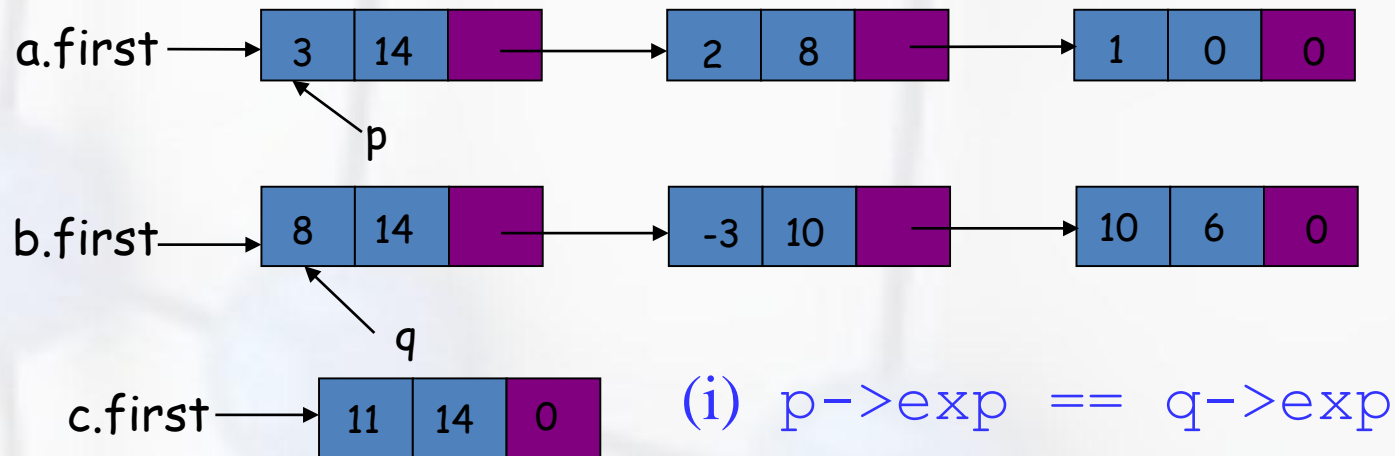
$$a = 3x^{14} + 2x^8 + 1$$



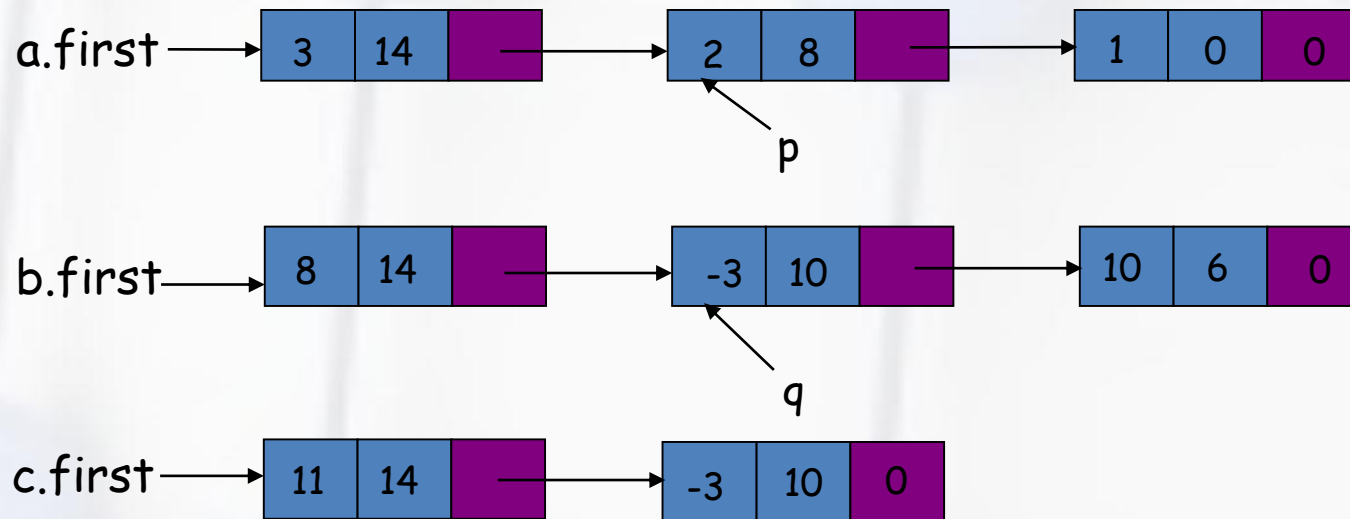
$$b = 8x^{14} - 3x^{10} + 10x^6$$

# Addition of Two Polynomials (1)

- It is an easy way to represent a polynomial by a linked list.
- Example of adding two polynomials **a** and **b**

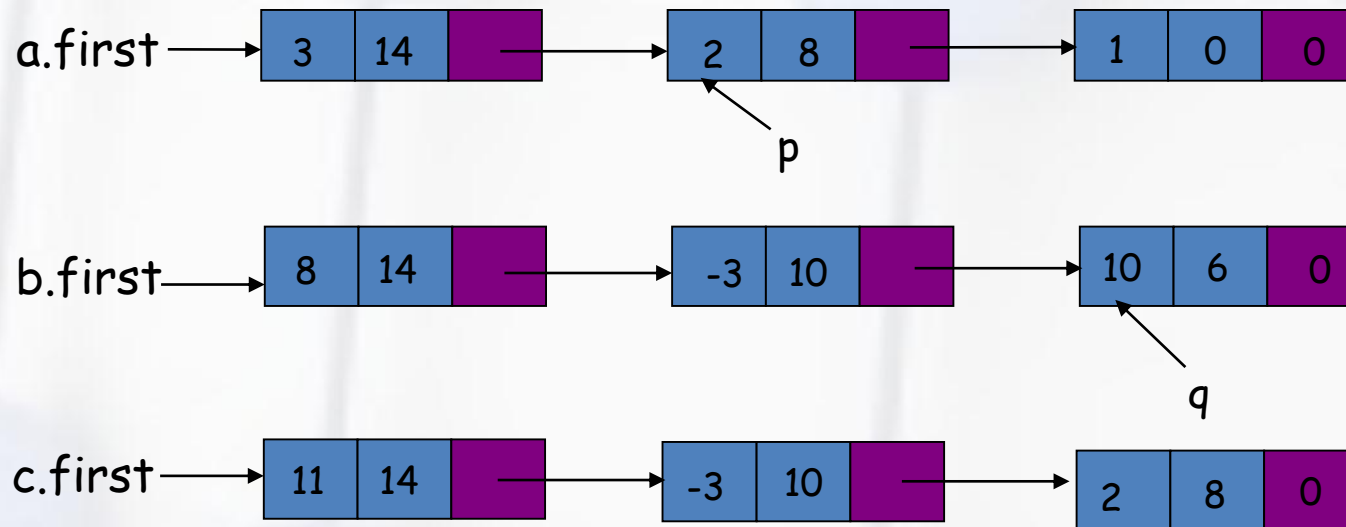


# Addition of Two Polynomials (2)



(ii)  $p \rightarrow \text{exp} < q \rightarrow \text{exp}$

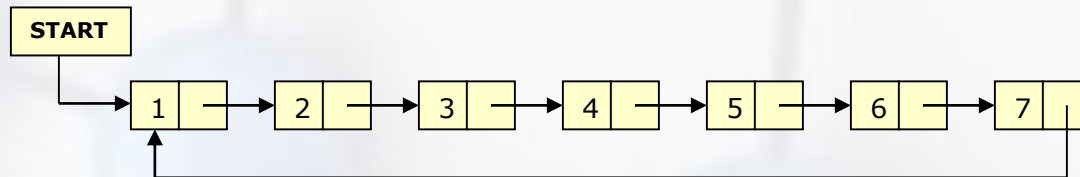
# Addition of Two Polynomials (3)



(iii)  $p \rightarrow \text{exp} > q \rightarrow \text{exp}$

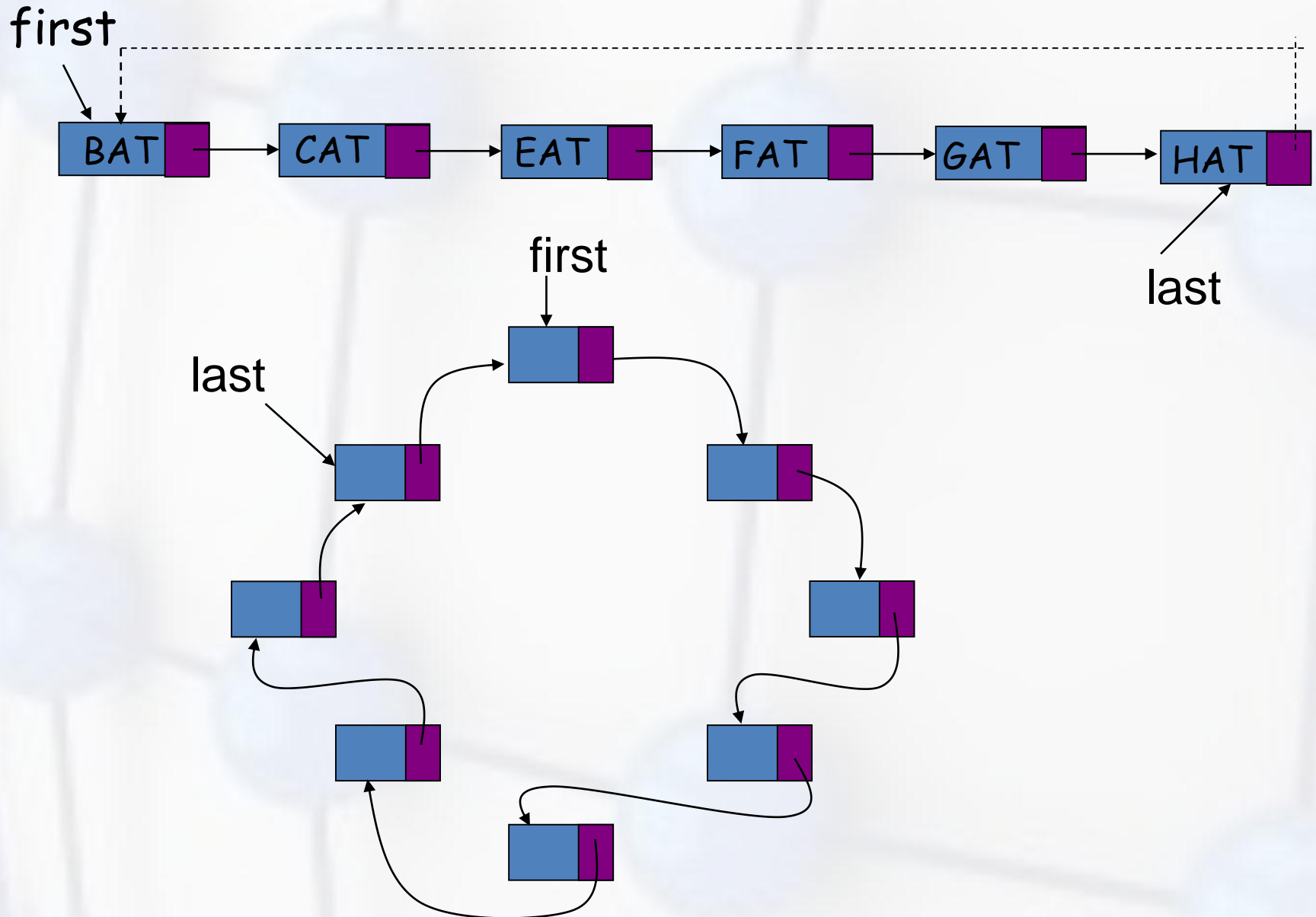
# 3. Circular Linked List

- In a circular linked list, the last node contains a pointer to the first node of the list. We can have a circular singly linked 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 List

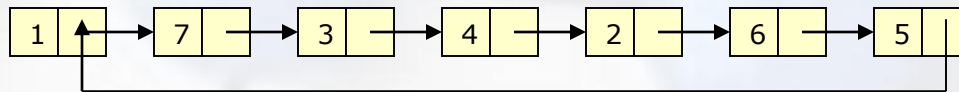


# Circular Linked List

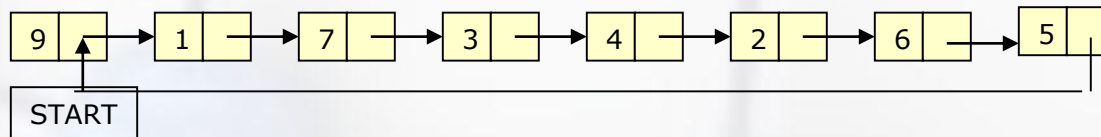
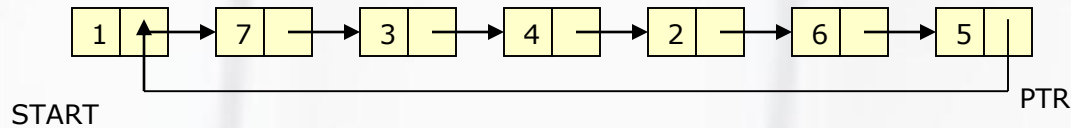
Algorithm to insert a new node in the beginning of **circular** the linked list

```
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



START, PTR



# 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

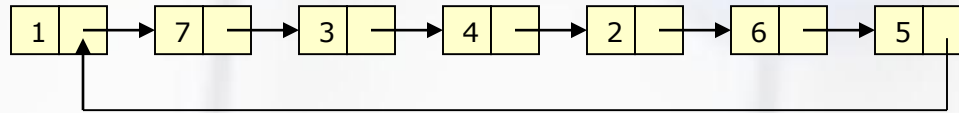
Step 8:                   SET PTR = PTR ->NEXT

[END OF LOOP]

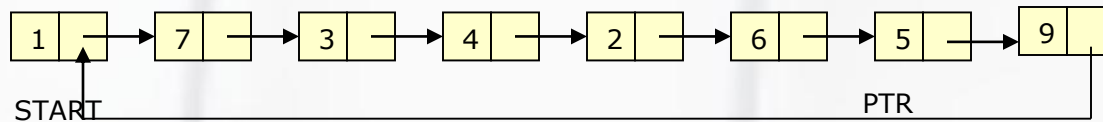
Step 9: SET PTR ->NEXT = New\_Node

Step 10: EXIT

# Circular Linked List



START, PTR



# Circular Linked List

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

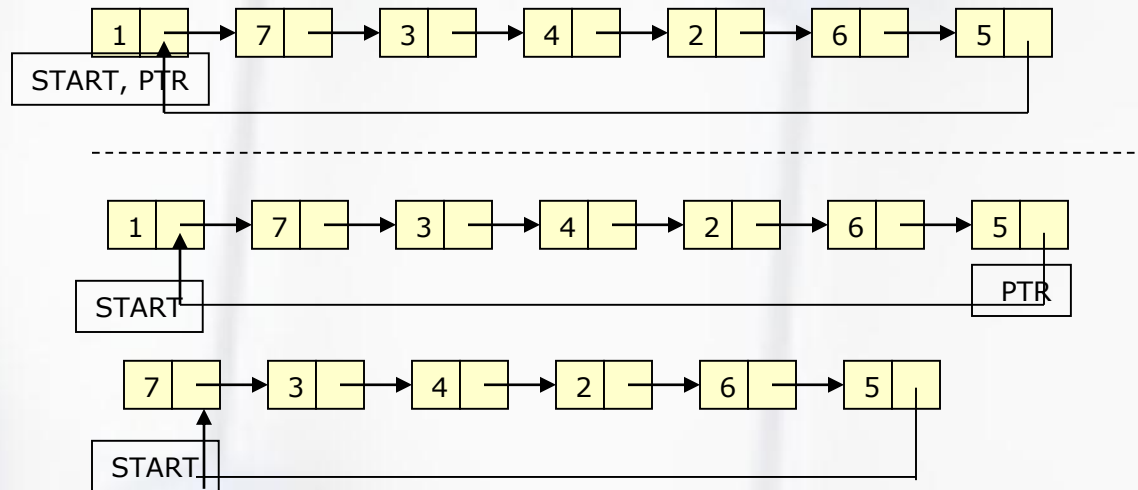
```
Step 1: IF AVAIL = NULL, then
        Write OVERFLOW
        Go to Step 12
    [END OF IF]
Step 2: SET New_Node = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET New_Node->DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Step 8 and 9 while PTR->DATA != NUM
Step 8:         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  $PTR \rightarrow NEXT \neq START$   
Step 4:                 SET  $PTR = PTR \rightarrow NEXT$   
    [END OF IF]  
Step 5: SET  $PTR \rightarrow NEXT = START \rightarrow NEXT$   
Step 6: FREE  $START$   
Step 7: SET  $START = PTR \rightarrow NEXT$   
Step 8: EXIT

# Circular Linked List





# 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

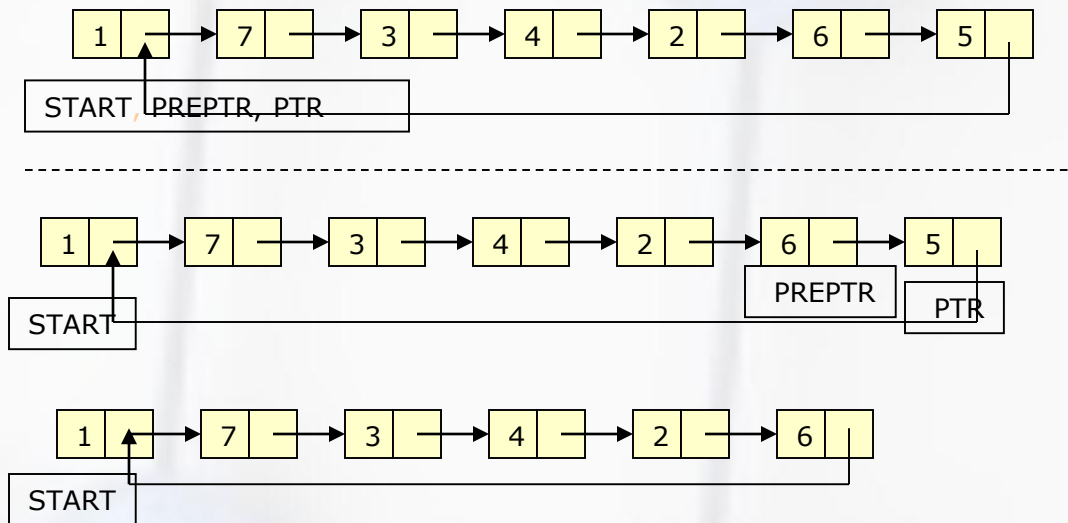
    [END OF LOOP]

Step 6: SET PREPTR->NEXT = START

Step 7: FREE PTR

Step 8: EXIT

# Circular Linked List



# 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

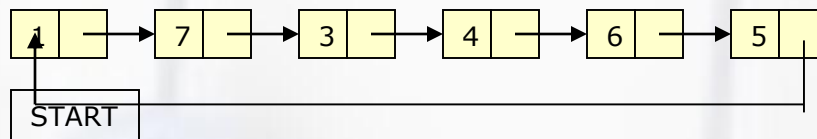
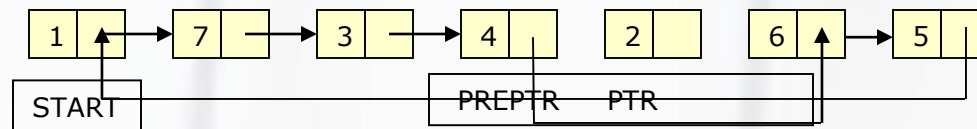
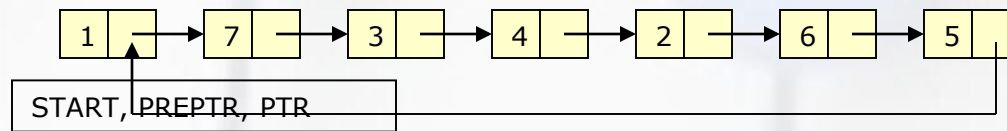
[END OF LOOP]

Step 7: SET PREPTR->NEXT = PTR->NEXT

Step 8: FREE PTR

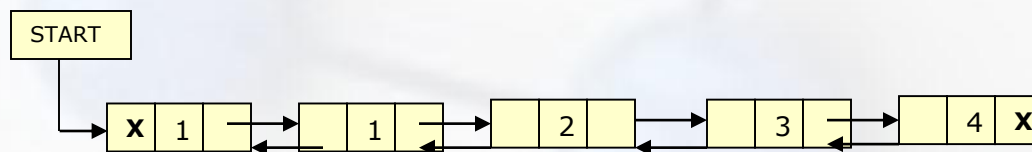
Step 9: EXIT

# Circular Linked List



# Doubly Linked List

- In a singly linked list, if we want to delete a node  $ptr$ , we have to know the preceding node of  $ptr$ . Then we have to start from the beginning of the list and to search until the node whose next (link) is  $ptr$  is found.
- To efficiently delete a node, we need to know its preceding node. Therefore, a doubly linked list is useful.
- A node in a doubly linked list has at least three fields: left, data, right.
- A header node may be used in a doubly linked list.



# 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

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

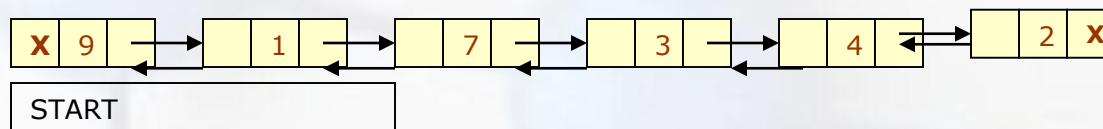
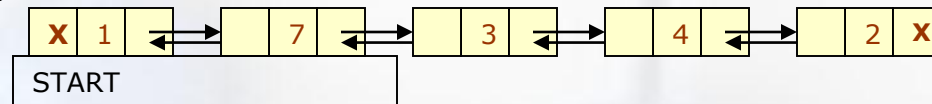
Step 4: SET New\_Node->DATA = VAL

Step 5: SET New\_Node->PREV = NULL

Step 6: SET New\_Node->Next = START

Step 7: SET START = New\_Node

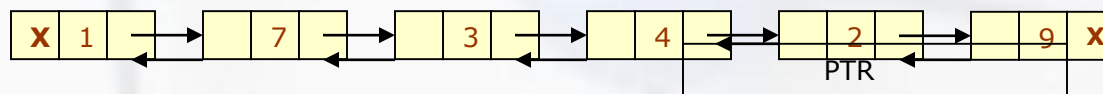
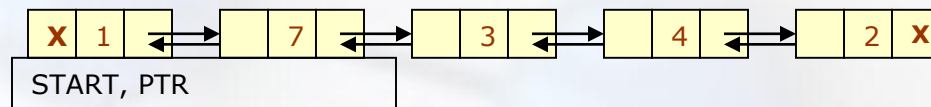
Step 8: EXIT



# 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
```



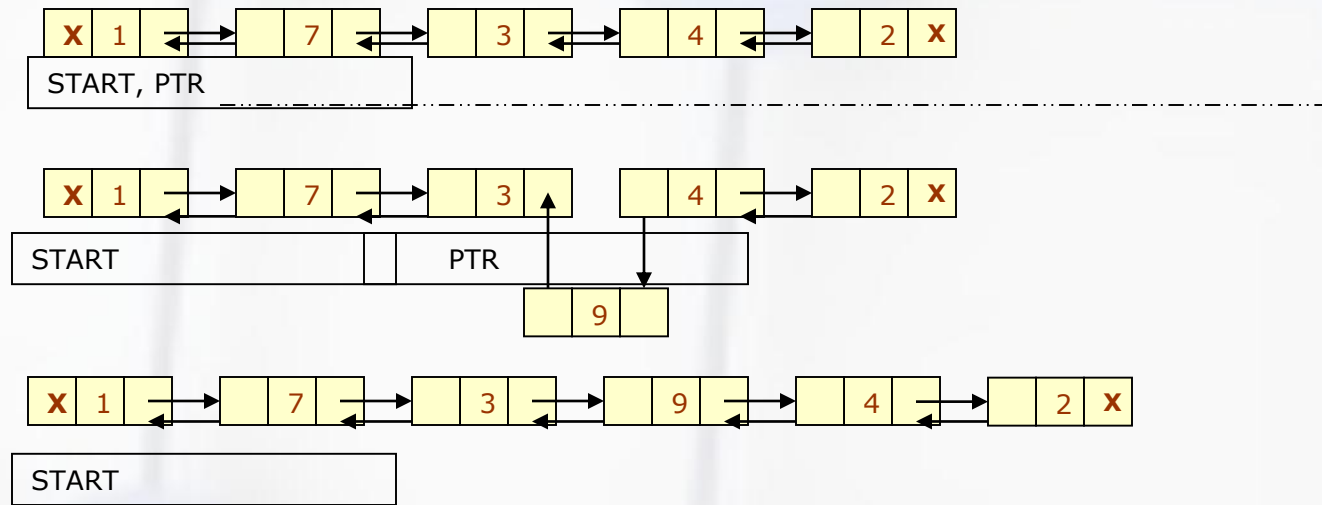


# Doubly Linked List

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



# 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]

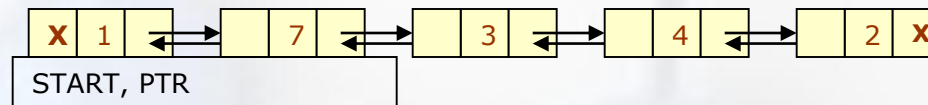
Step 2: SET PTR = START

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

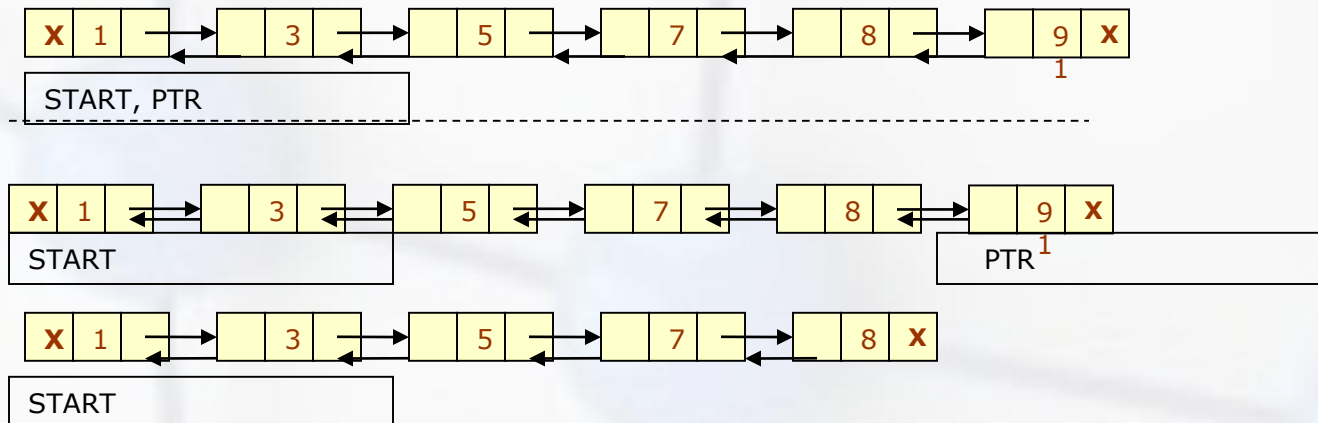
Step 4: SET PTR = PTR->NEXT

[END OF LOOP]

Step 5: SET PTR->PREV->NEXT = NULL

Step 6: FREE PTR

Step 7: EXIT

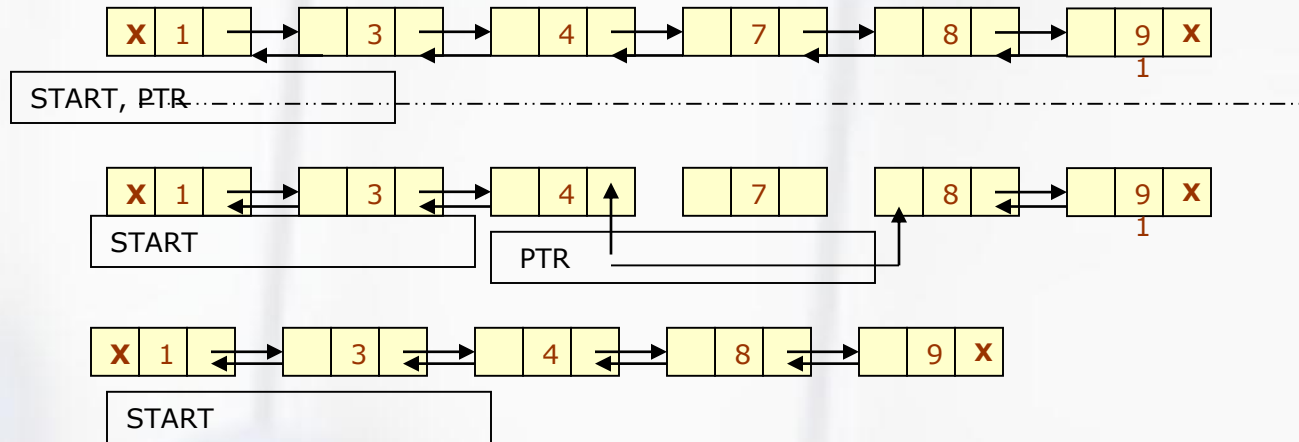


# Doubly Linked List

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
```

# Doubly Linked List

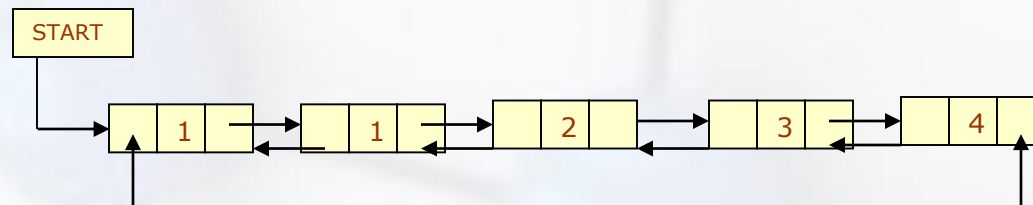


# 5. Circular Doubly Linked List

- 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.

# Circular Doubly Linked List

- 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.





# Circular Doubly Linked List

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;

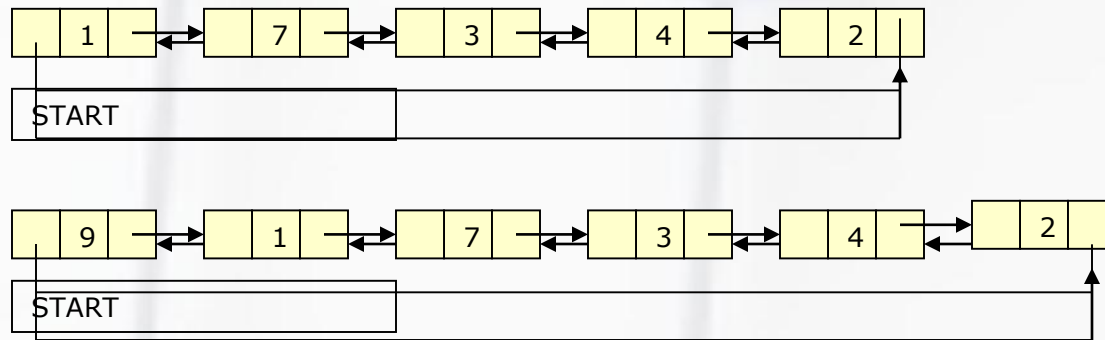
Step 8: SET START->PREV = new\_Node;

Step 9: SET new\_node->next = START;

Step 10: SET START = New\_Node

Step 11: EXIT

# Circular Doubly Linked List



# Circular Doubly Linked List

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

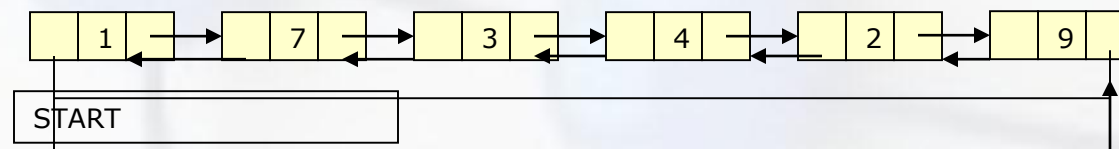
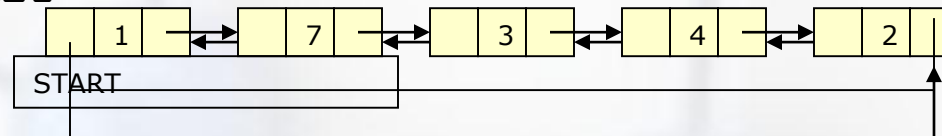
Step 3: SET AVAIL = AVAIL->NEXT

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

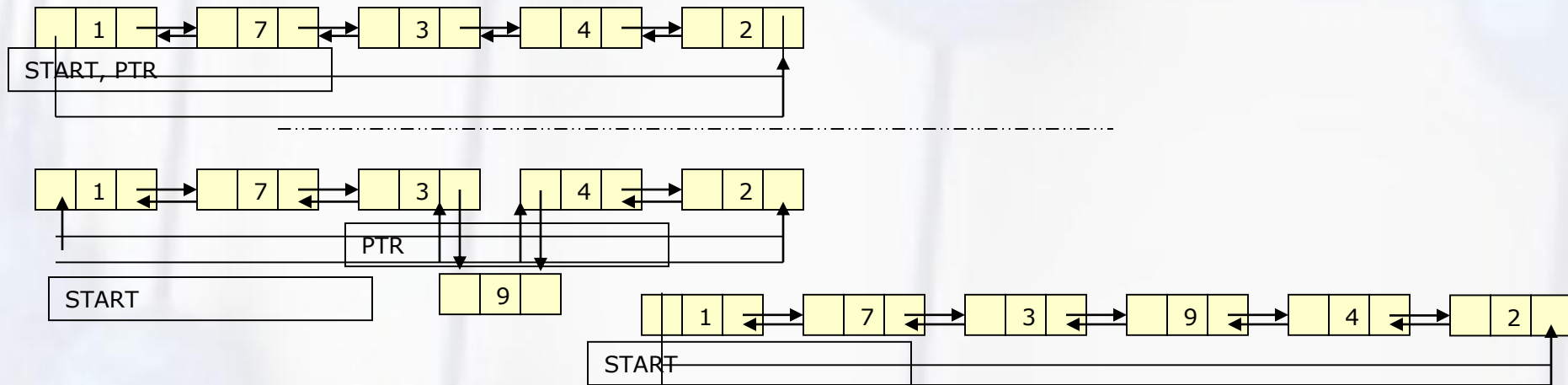


# Circular Doubly Linked List

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

# Circular Doubly Linked List



# Circular Doubly Linked List

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  $PTR = START$

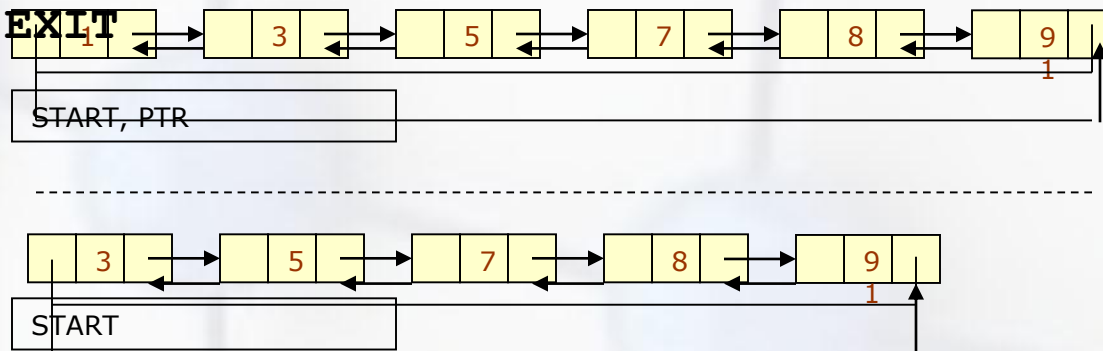
Step 3: SET  $PTR \rightarrow PREV \rightarrow NEXT = PTR \rightarrow NEXT$

Step 4: SET  $PTR \rightarrow NEXT \rightarrow PREV = PTR \rightarrow PREV$

Step 5: SET  $START = START \rightarrow NEXT$

Step 6: FREE PTR

Step 7: **EXIT**



# Circular Doubly Linked List

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]

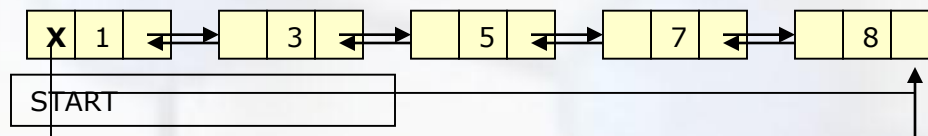
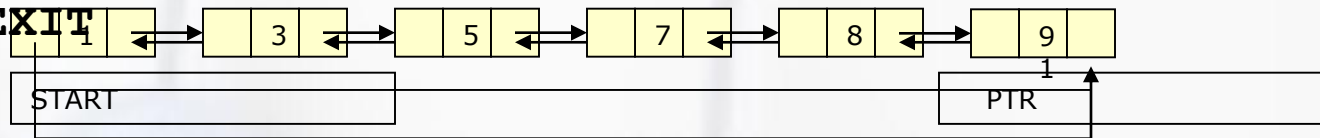
Step 2: SET  $PTR = START \rightarrow PREV$

Step 5: SET  $PTR \rightarrow PREV \rightarrow NEXT = START$

Step 6: SET  $START \rightarrow PREV = PTR \rightarrow PREV$

Step 7: FREE PTR

Step 8: EXIT



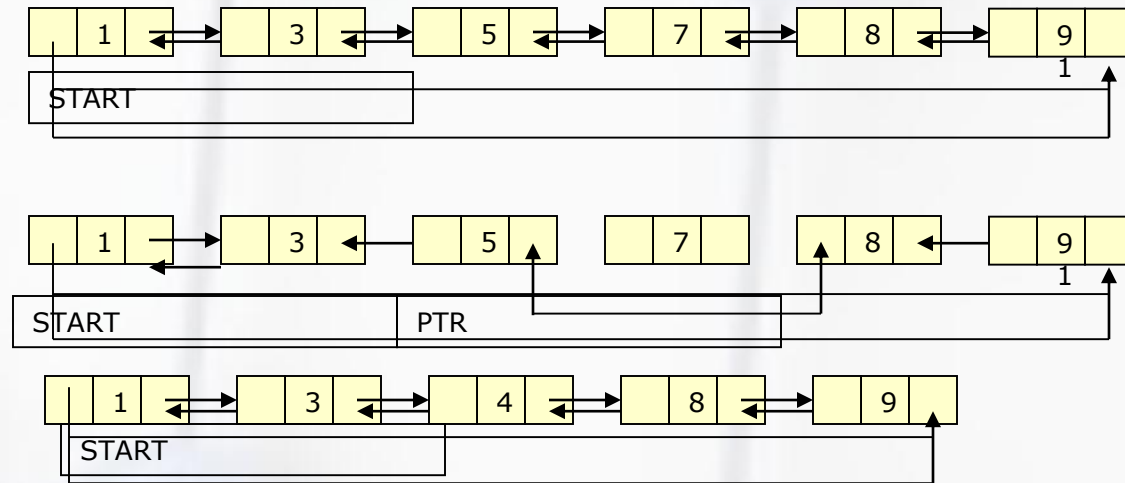
# Circular Doubly Linked List

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
```



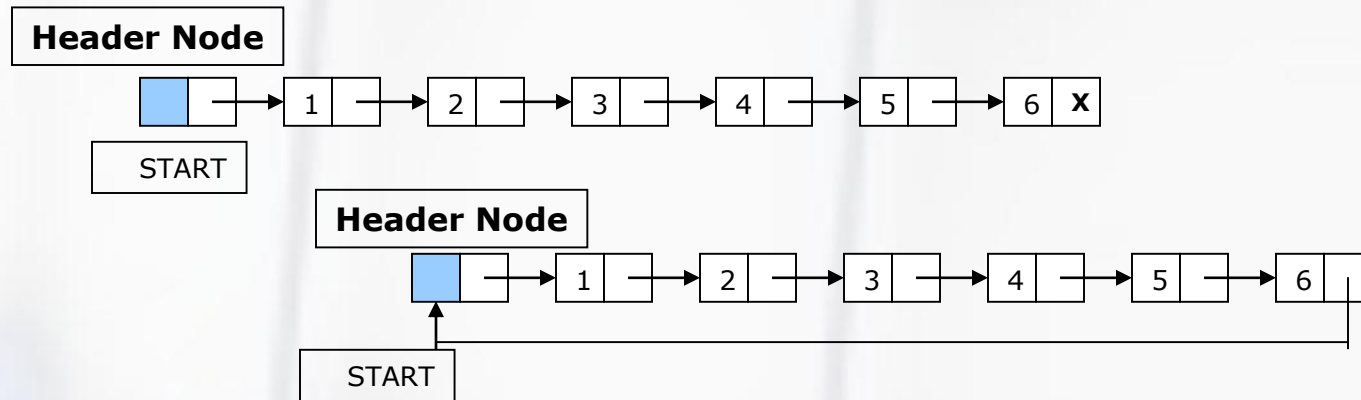
# Circular Doubly Linked List



# Circular Doubly 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.

# Circular Doubly Linked List



# Circular Doubly Linked 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

# Circular Doubly Linked List

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
```

# Josephus problem

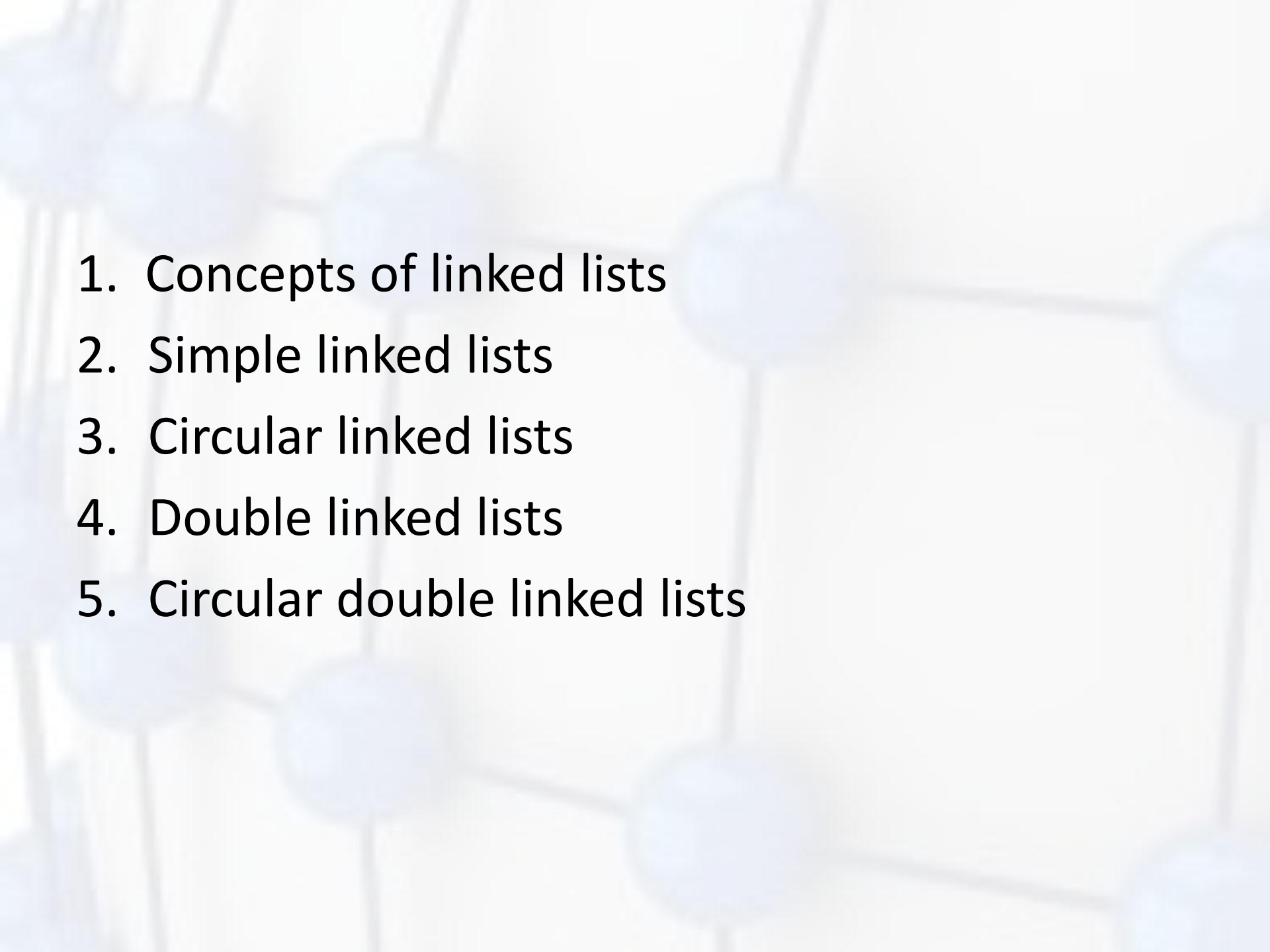
- A set of players are present in a circle
- Counting from a given player, every 'nth' player is considered 'out and eliminated from the game ' and eliminated from the game
- Counting starts again from the next person after the removed player, and the next 'nth' player is removed.
- The game continues until only one player remains, who is the winner ?

# Algorithm Josephus problem

1. Obtain the initial player list
2. Go to starting player. Start count of 1.
3. Increment count, go to next player. If player-list end is reached, go to list beginning.
4. If  $\text{count} < n$ , go back to step 3
5. If  $\text{count} = n$ , remove player from list. Set  $\text{count} = 1$  from next player. Go back to Step 3.
6. If next player is same as current player, declare winner.

## Implementation:

- 2D Arrays to hold player names OR Circular lists
- Circular lists are an easier implementation for Step 3
- Step 5: easier with doubly linked circular lists

- 
- A faint background diagram showing a network of interconnected nodes and lines, resembling a molecular structure or a data network.
1. Concepts of linked lists
  2. Simple linked lists
  3. Circular linked lists
  4. Double linked lists
  5. Circular double linked lists