CHAPTER 4: LINKED LIST

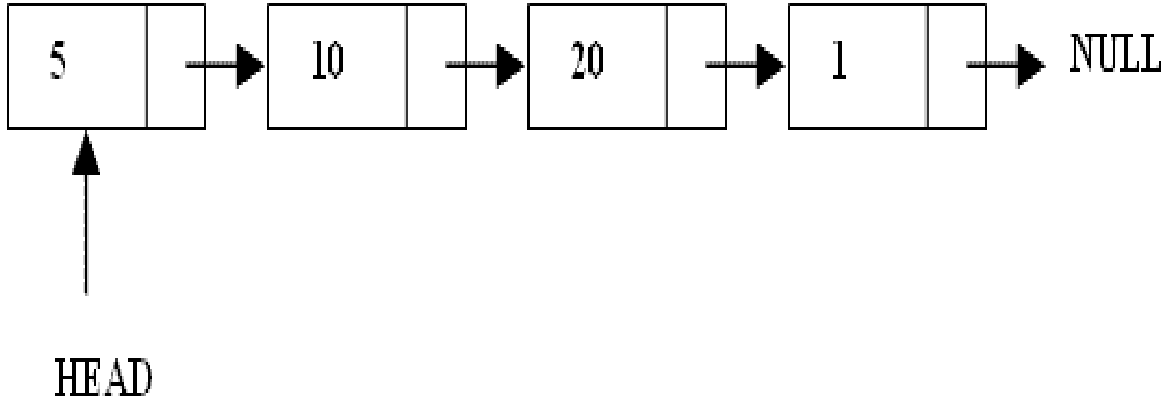
BIBHA STHAPIT
ASST. PROFESSOR
IoE, PULCHOWK CAMPUS

Array Vs Linked List

Arrays	Linked list
Fixed size: Resizing is expensive	Dynamic size
Insertions and Deletions are inefficient: Elements are usually shifted	Insertions and Deletions are efficient: No shifting
Random access i.e., efficient indexing	No random access → Not suitable for operations requiring accessing elements by index such as sorting
No memory waste if the array is full or almost full; otherwise may result in much memory waste.	Since memory is allocated dynamically(acc. to our need) there is no waste of memory.
Sequential access is faster [Reason: Elements in contiguous memory locations]	Sequential access is slow [Reason: Elements not in contiguous memory locations]

Linked List : Introduction

- A linked list is a linear data structure.
- Nodes make up linked lists.
- Nodes are structures made up of data and a pointer to another node called “next”.



Linked List : Introduction

- It is a **list or collection of data items** that can be stored in scattered locations (positions) in memory by establishing link between the items.
- To store data in scattered locations in memory we have to **make link** between one data item to another.
- So, each data item or element must have **two parts**: one is data part and another is **link/next (pointer) part**.

Linked List : Introduction

- Each data item of a linked list is called a node.
- Data part contains (holds) actual data (information) and the link part points to the next node of the list.
- To locate the list an external pointer is used that points the first node of the list.
- The link part of the last node will not point any node. That means it will be *null*.
- This type of list is called linear (one way) linked list or simply linked list.

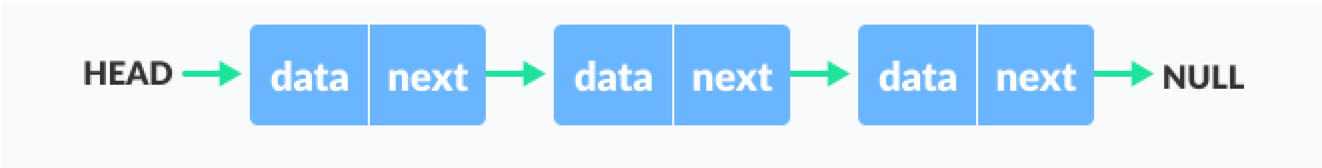
Types of linked list

- There are two basic typesof linked list

- Singly Linked list
  - Singly linear linked list
  - Singly Circular linked list

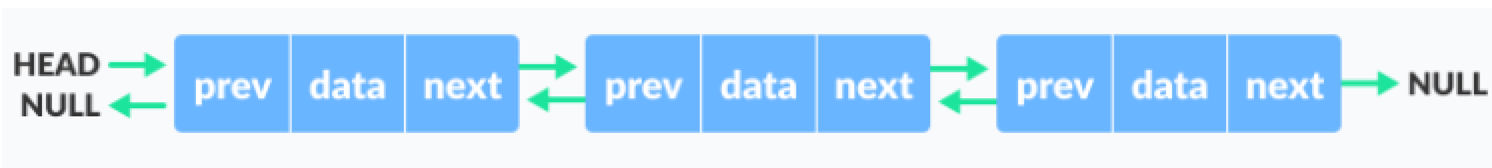
- Doubly linked list
  - Doubly linear linked list
  - Doubly Circular linked list

Types of linked list



Singly linked list

Circular linked list



Doubly linked list

Doubly circular linked list

Singly Linked List

- Each node has only one link part

- Each link part contains the address of the next node in the list

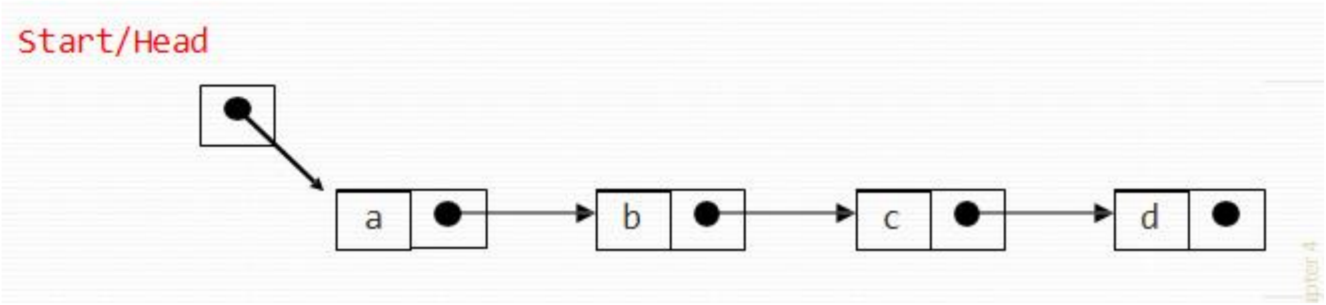
- Link part of the last node contains NULL value which signifies the end of the node

Schematic representation of SLL

- Each node contains a value(data) and a pointer to the next node in the list

- Start/Head is the header pointer which points at the first node in the list

- Here is a singly-linked list(SLL):



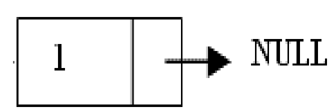
Basic Operations on a list

- Creating a List
- Inserting an element in alist
  - At the beginning / end
  - Before / After specific node
  - In the sorted list
- Deleting anelement from a list
  - From the beginning / end
  - After specific node
  - From the sorted list
- Searching a list
- Reversing a list

Creating a node

```
struct node{
        int data;              // A simple node of a linked list
        node*next;
    }*start;                   //start  points at the first node
start=NULL ;                      initialised to NULL at beginning
```

```
node* create( int num) //say num=1 is passed from main
{
   node*ptr;
   ptr= new  node;  //memory allocated dynamically
         if(ptr==NULL)
           'OVERFLOW' // no memory available
            exit(1);
         else
         {
            ptr->data=num;
            ptr->next=NULL;
            return ptr;
 }
         }
```

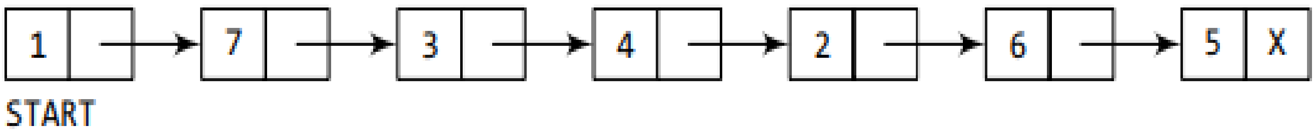


Insertion at the beginning

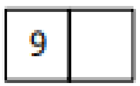
```
Step 1: IF AVAIL = NULL
              Write OVERFLOW
              Go to Step 7
         [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET START = NEW_NODE
Step 7: EXIT
```

Algorithm to insert a new node at the beginning

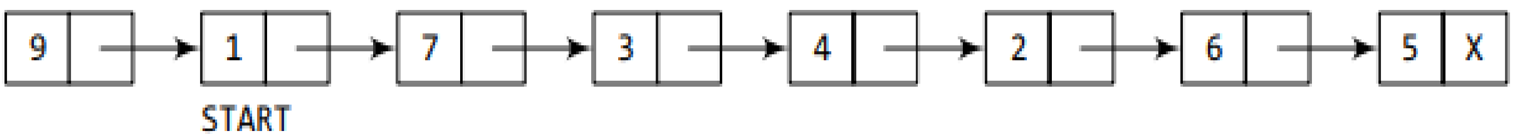
Insertion at the beginning



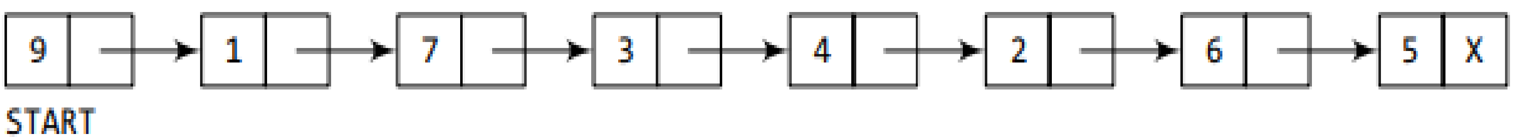
Allocate memory for the new node and initialize its DATA part to 9.



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.



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



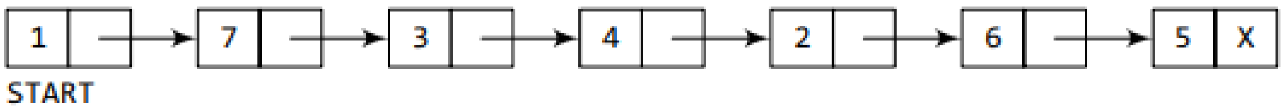
Inserting an element at the beginning of a linked list

Insertion at the end

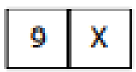
```
Step 1: IF AVAIL = NULL
              Write OVERFLOW
              Go to Step 10
         [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = NULL
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT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

Algorithm to insert a new node at the end

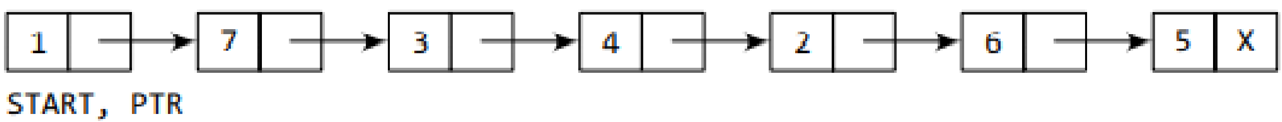
Insertion at the end



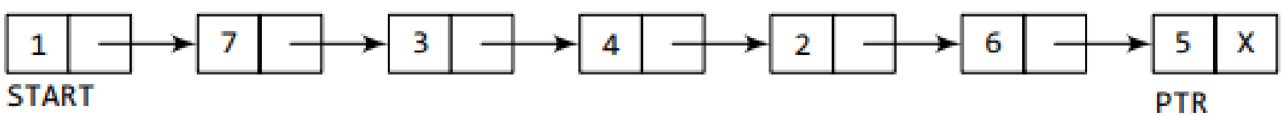
Allocate memory for the new node and initialize its DATA part to 9 and NEXT part to NULL.



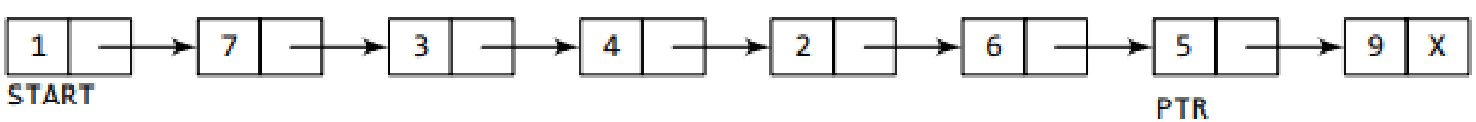
Take a pointer variable PTR which points to START.



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



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



Inserting an element at the end of a linked list

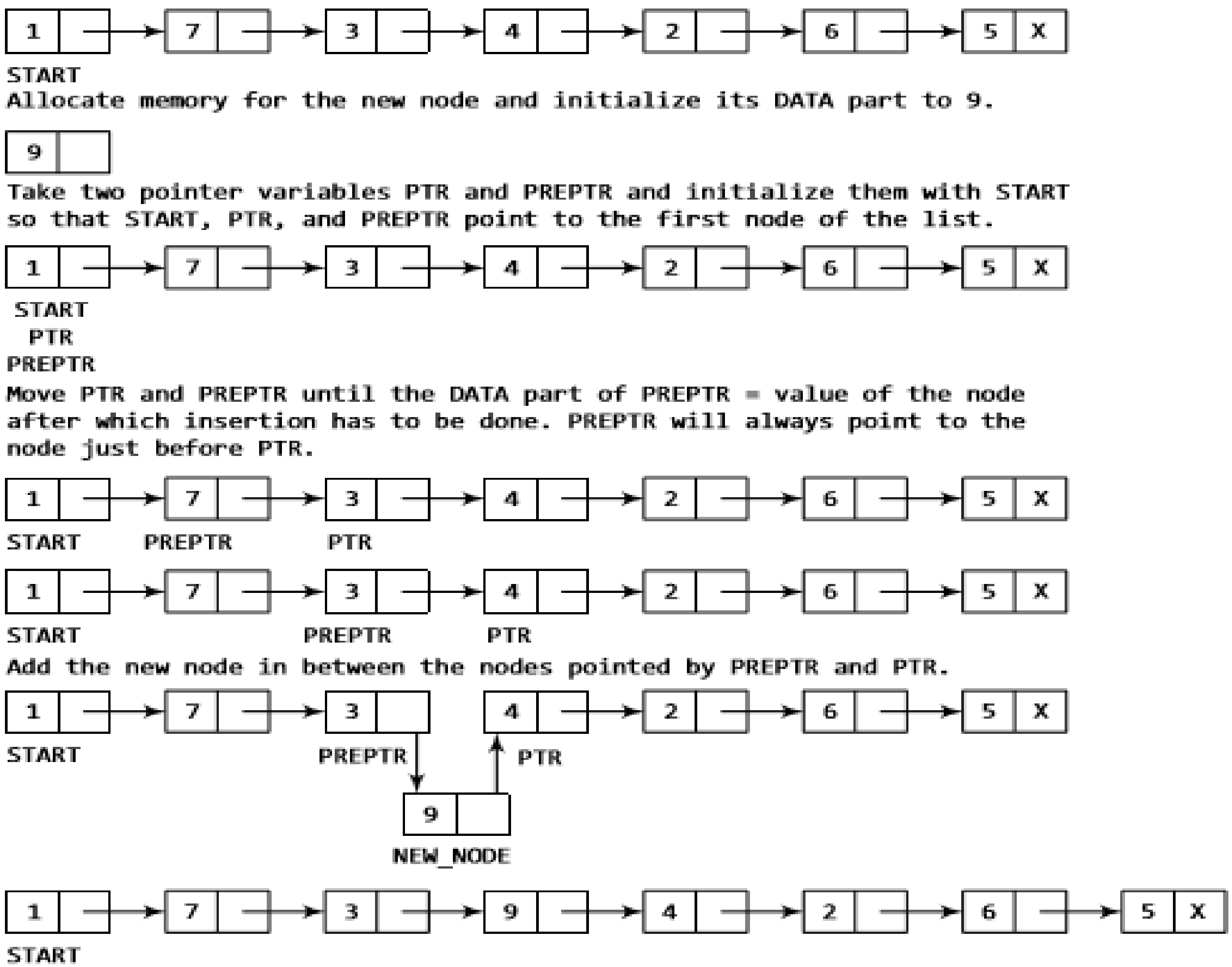
Insertion after specific node

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

Algorithm to insert a new node after a node

Chapter 4

Insertion after specific node



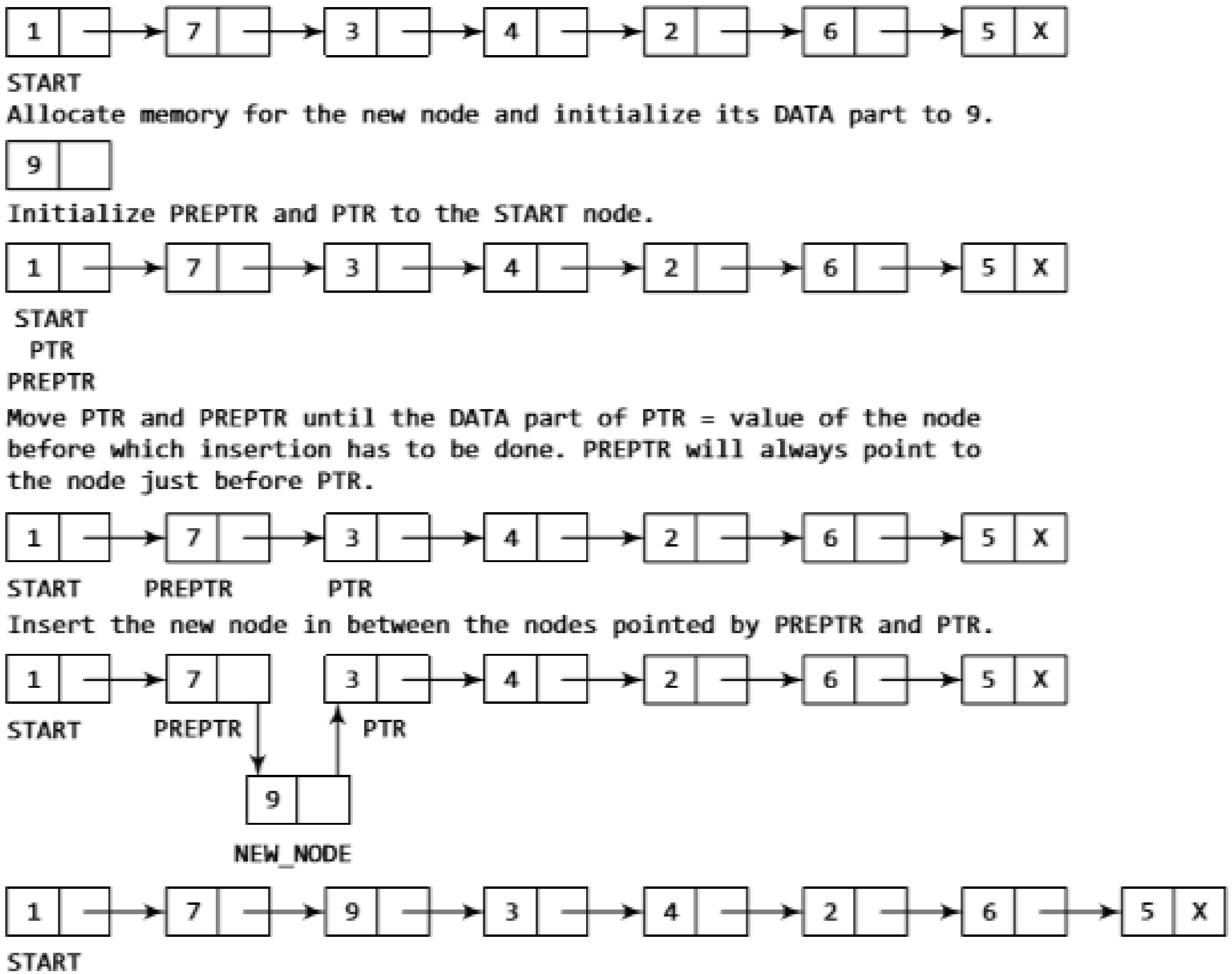
Inserting an element after a given node in a linked list

Insertion before specific node

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
       [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET NEW_NODE->DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PTR->DATA != NUM
Step 8:     SET PREPTR = PTR
Step 9:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 10: PREPTR->NEXT = NEW_NODE
Step 11: SET NEW_NODE->NEXT = PTR
Step 12: EXIT
```

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

Insertion before specific node



Inserting an element before a given node in a linked list

Insertion in a sorted list

- Here, we shall consider insertion of a node after the first node or before the last node and the data of the list are arranged in **Ascending Order**.

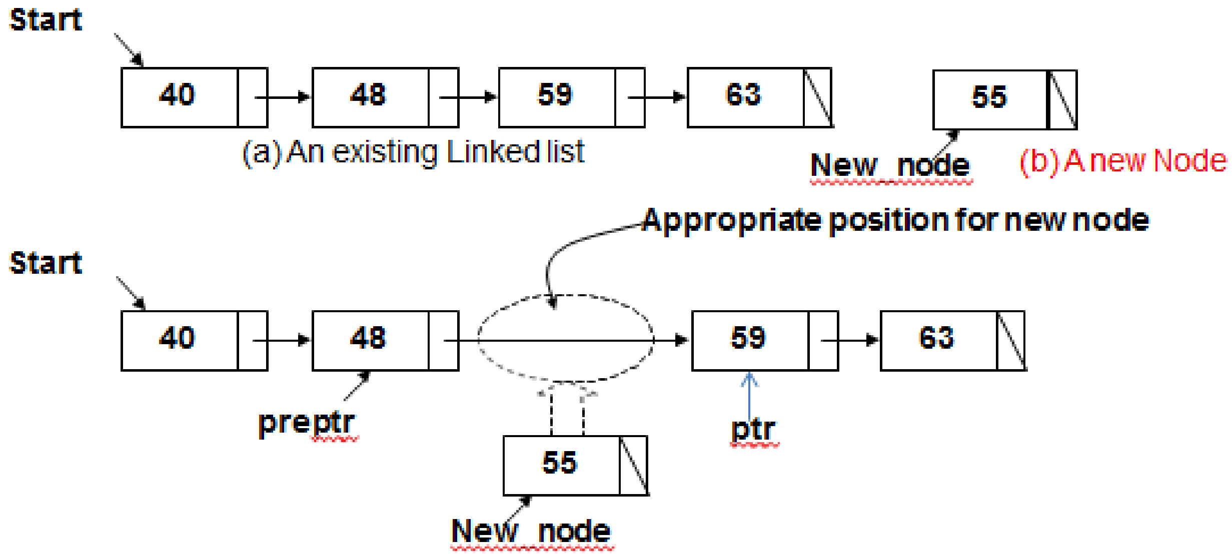
- Here we have to perform two major tasks:

1. Locate (find out) **the node after which** the new node will be inserted.

2. Insert the node by making link.

Insertion in a sorted list

- Locate the position to insert (Graphical view)



(c) Finding out appropriate position for the new node

Insertion in a sorted list

- The steps to locate the position:

1. Assign the value of external pointer to a temporary pointer (ptr = start).

2. Compare the value of the next node with the value of the new node.

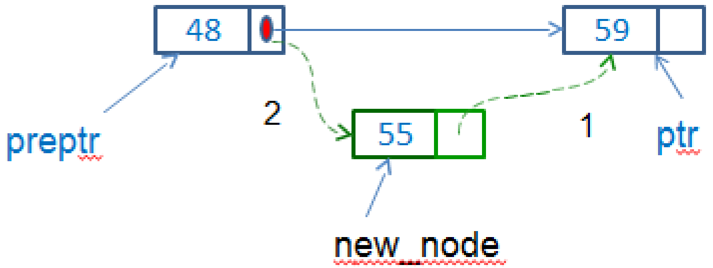
3. Traverse the temporary pointer until we find a greater node than the value of the value new node
    - ptr = ptr->next.

Insertion in a sorted list

- Insert the node by making link. The steps are:

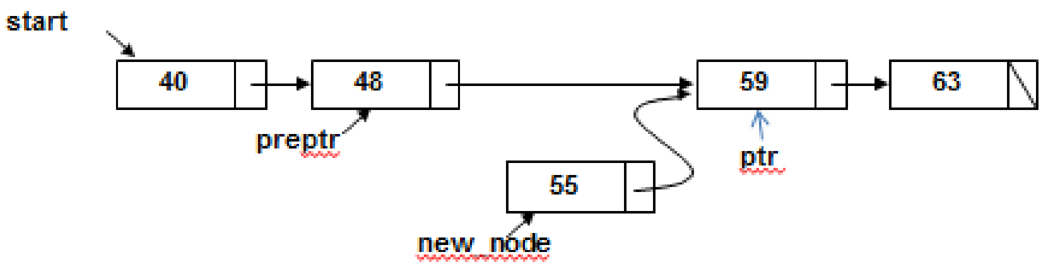
1. Point the next node by the new node.
   - (new_node->next = preptr->next).
2. Point the new node by the previous node.
   - (preptr->next = new_node).

3. We have got an updated linked list.

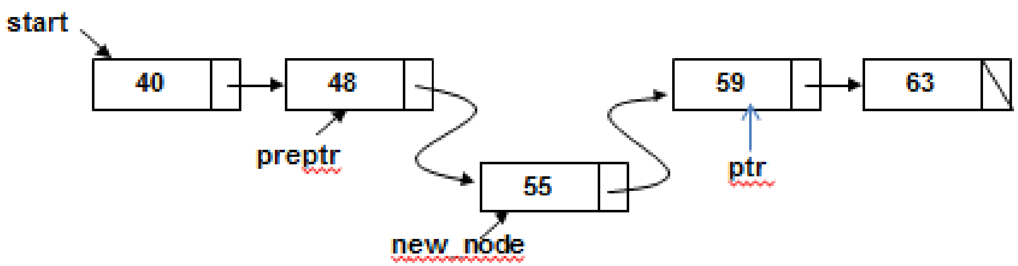


Insertion in a sorted list

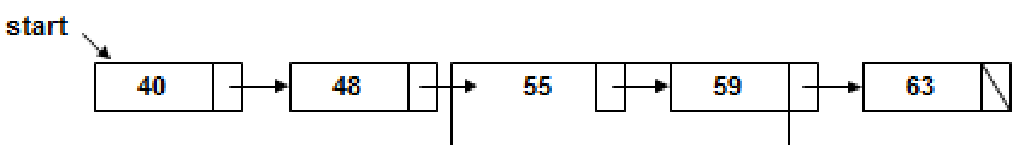
- Insert the node into a list (Graphical view)



(d) Making link between the new node and the node after the *ptr.*



(e) Making link between ptr and the new node



(f): Updated List

Insertion in a sorted list (pseudocode)

1. Input linked list (we have to use an existing list)
2. Declare pointers (start, ptr, preptr, new_node)

3. Create a new node:

```
new_node = new (node);
new_node →data = item;
new_node →next = NULL;
```

4. Locate the appropriate position for the new node

```
while (ptr→data < new_node →data)
  {
     preptr= ptr;  ptr = ptr→next;
  }
```

5. Insert the new node at appropriate position (by **linking previous and next node**);

```
 new_node →next = preptr→next;
  preptr→next = new_node;
```

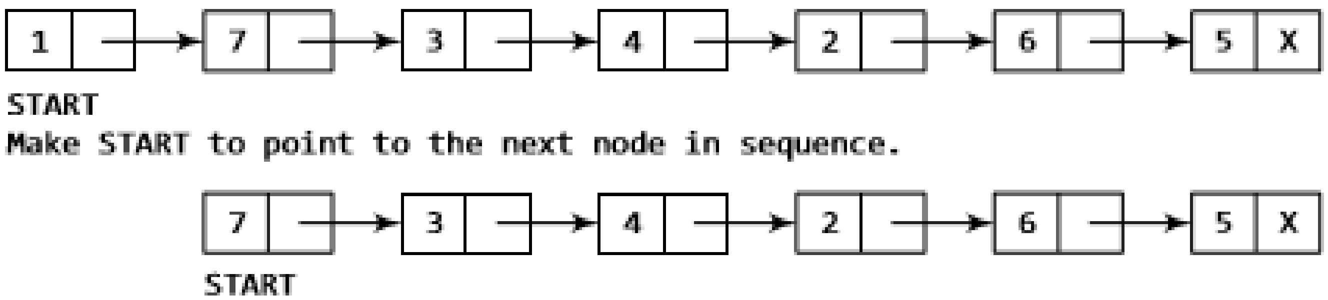
6. Output :updated linked list

Chapter 4

Deletion in linear SLL

- Deleting a node from the linked list has the following instances:
  - 1. Deletion from beginning
  - 2. Deletion from end
  - 3. Deletion after specific node
  - 4. Deletion in sorted list

Deletion from beginning

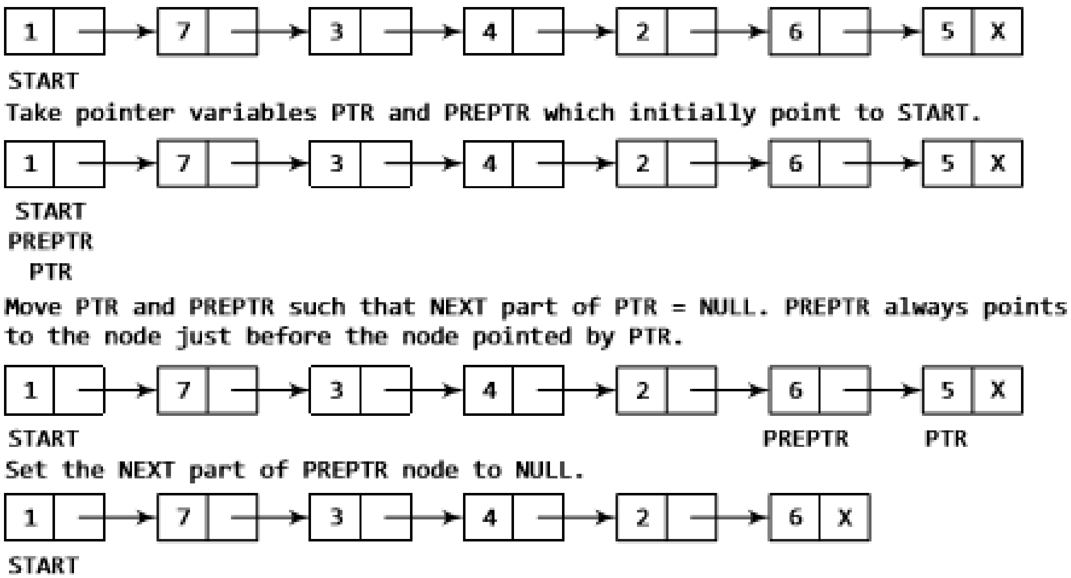


Deleting the first node of a linked list

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

Algorithm to delete the first node

Deletion from end

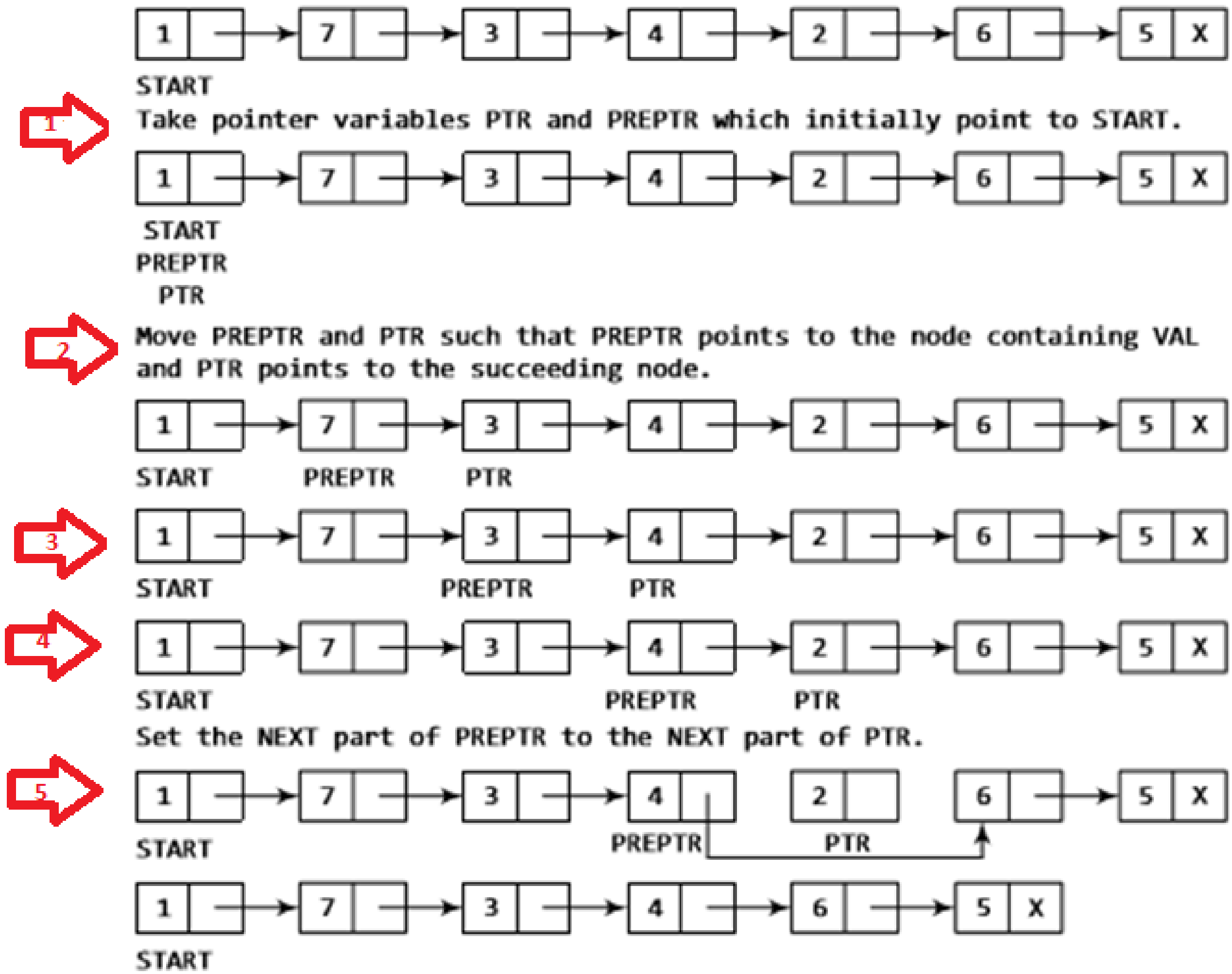


Deleting the last node of a linked list

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

Algorithm to delete the last node

Deleting after specific node



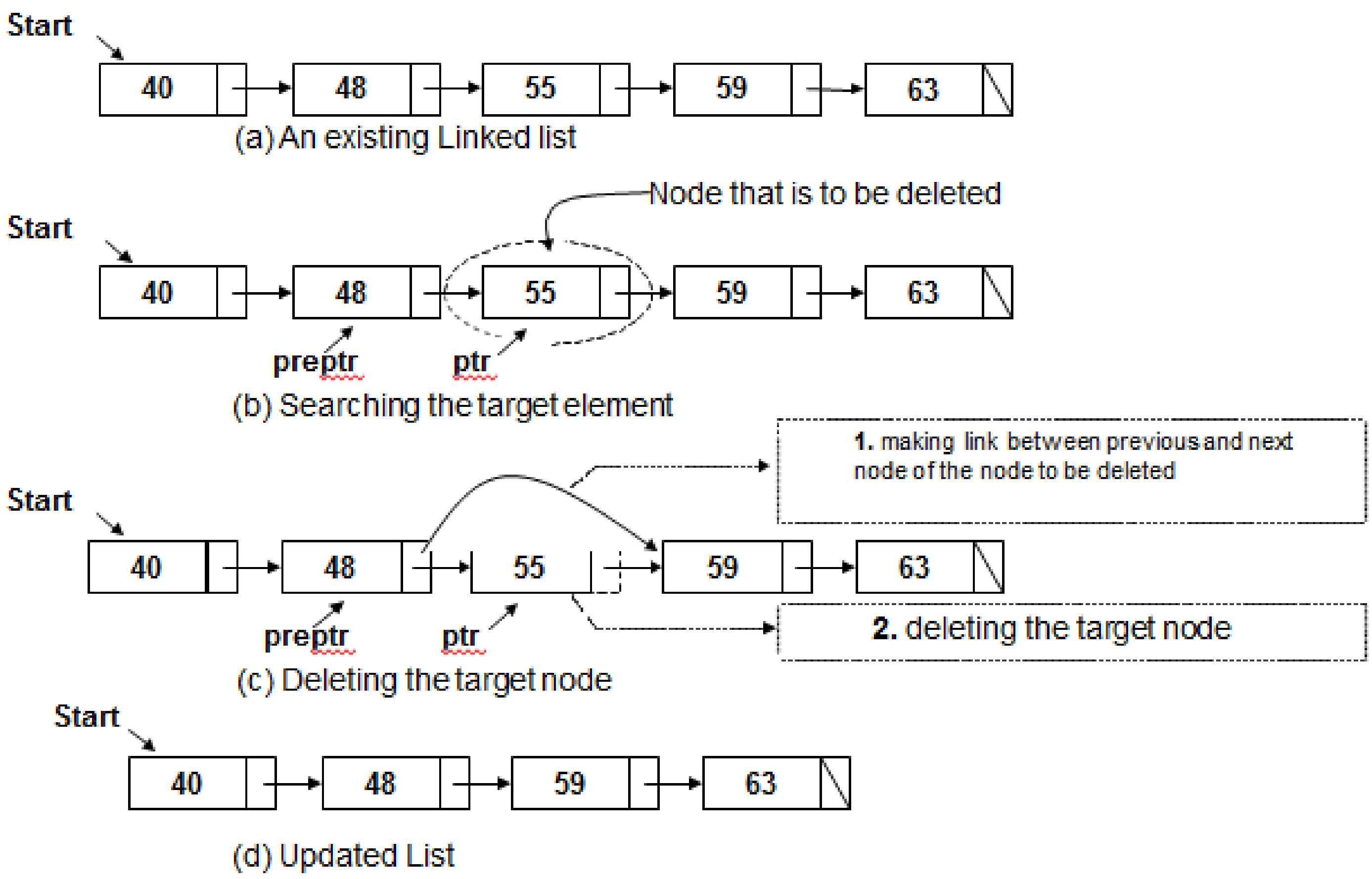
Deleting the node after a given node in a linked list

Deleting after specific node

```
Step 1: IF START = NULL
              Write UNDERFLOW
              Go to Step 10
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Steps 5 and 6 while PREPTR -> DATA != NUM
Step 5:       SET PREPTR = PTR
Step 6:       SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 7: SET TEMP = PTR
Step 8: SET PREPTR -> NEXT = PTR -> NEXT
Step 9: FREE TEMP
Step 10: EXIT
```

Algorithm to delete the node after a given node

Deletion in sorted list



Deletion in sorted list

1. Declare pointers (start, ptr, preptr)
2. Input linked list and the item (that is to be deleted)
3. Search the item to be deleted in the start: ptr = start;

   while (ptr→data != item)

   {preptr = ptr;

   ptr = ptr→next; }
4. Delete the node:

   [Make link between previous and next node of the node that is to be deleted and delete the target node]

   preptr→next=ptr→next;

   delete (ptr);
5. Output: updated linked lists

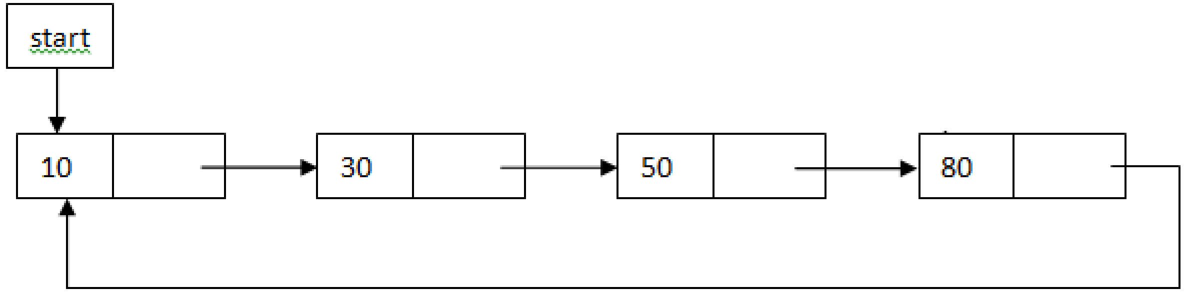
Complexity of array Vs SLL

Operation	ID-Array Complexity	Singly-linked list Complexity
Insert at beginning	O(n)	O(1)
Insert at end	O(1)	O(1) if the list has **tail** reference O(n) if the list has no **tail** reference
Insert at middle	O(n)	O(n)
Delete at beginning	O(n)	O(1)
Delete at end	O(1)	O(n)
Delete at middle	O(n): O(1) access followed by O(n) shift	O(n): O(n) search, followed by O(1) delete
Search	O(n) linear search O(log n) Binary search	O(n)
Indexing: What is the element at a given position k?	O(1)	O(n)

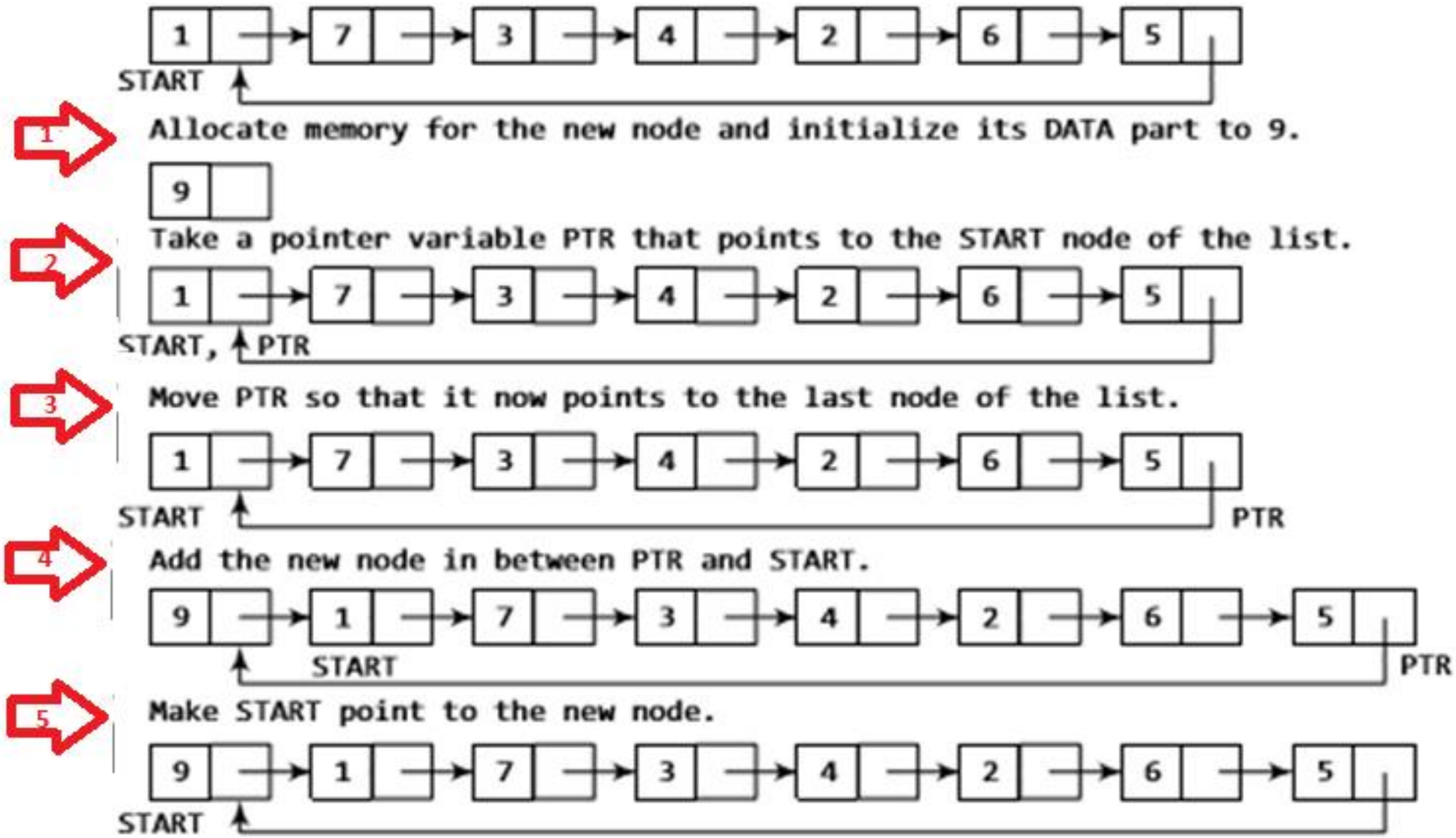
Circular linked list

- A circular linked list is a singly- linked list, in which the link field of the last node contains the address of the first node of the list instead of pointing to NULL.

- A circular linked list has no end, so we can use two pointers *first* and *last* to point to the first and the last nodes respectively (but not necessarily).



Inserting at beginning - CLL



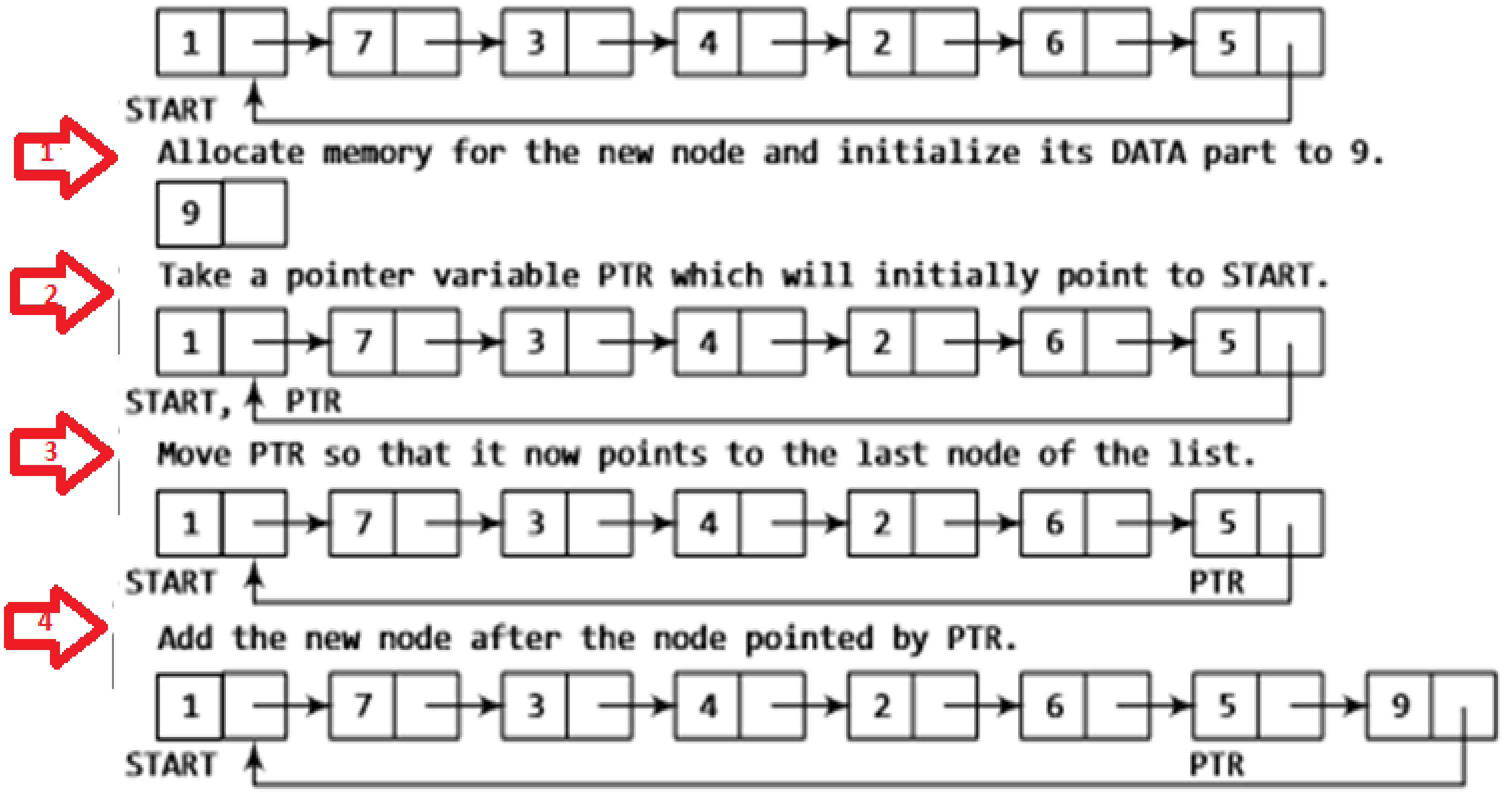
Inserting a new node at the beginning of a circular linked list

Inserting at beginning - CLL

```
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
```

Algorithm to insert a new node at the beginning

Inserting at end- CLL



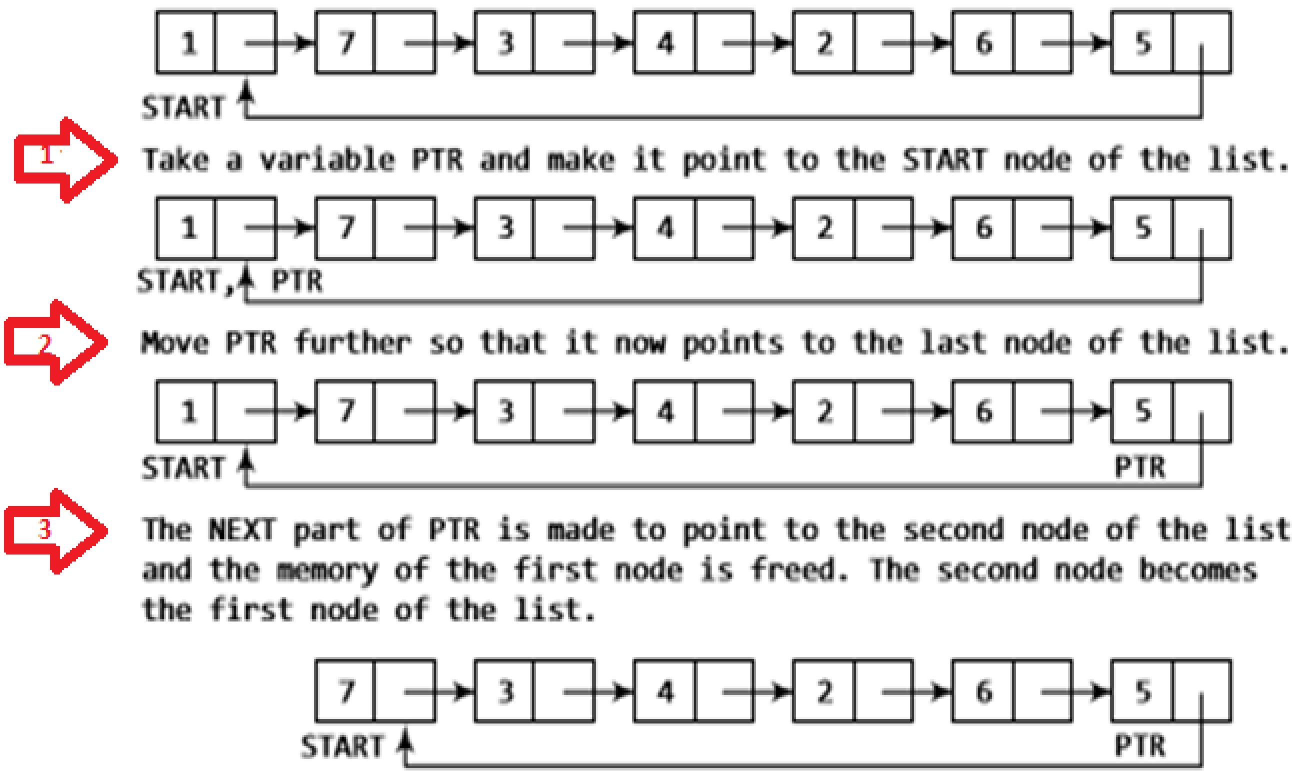
Inserting a new node at the end of a circular linked list

Inserting at end- CLL

```
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
```

Algorithm to insert a new node at the end

Deletion from beginning - CLL



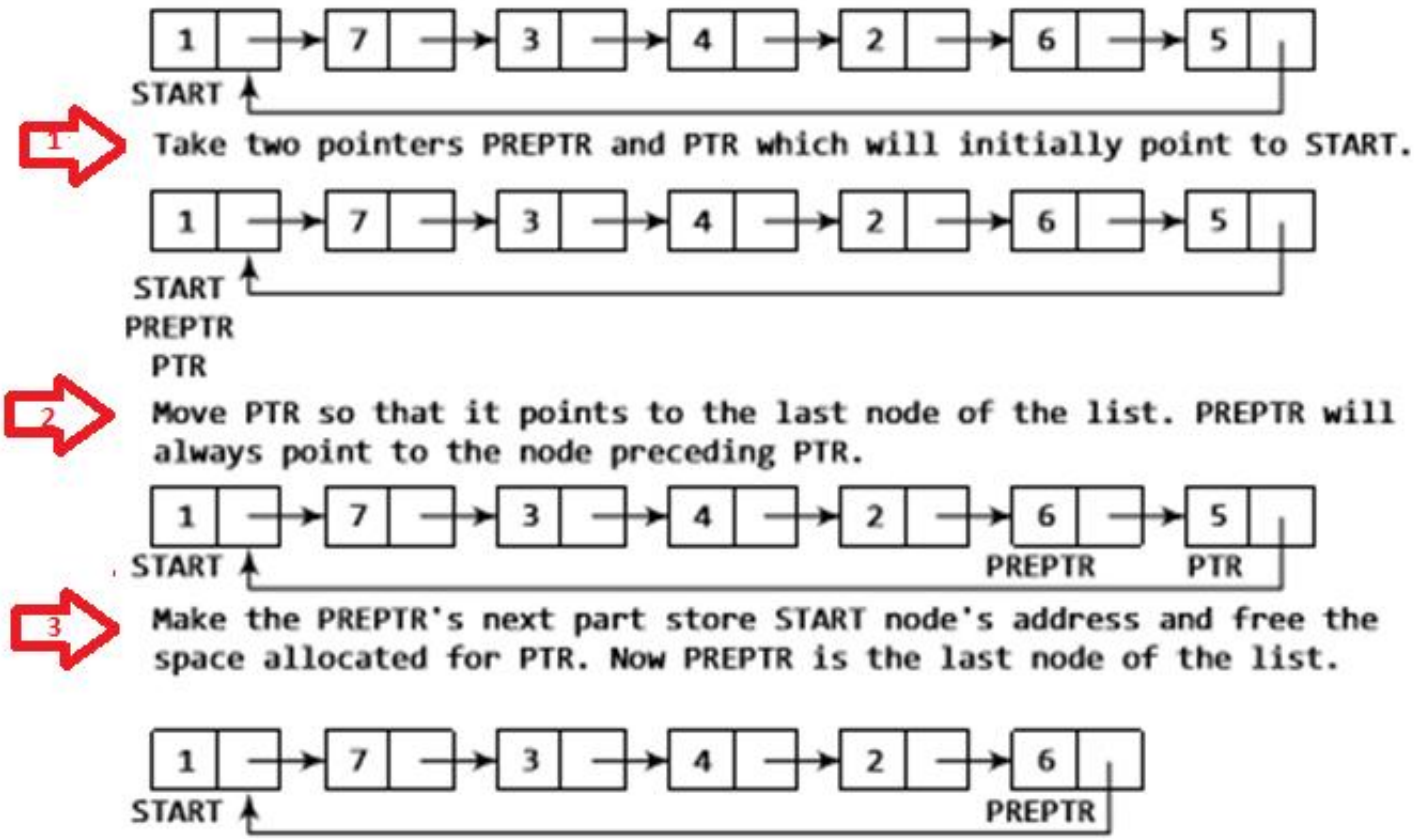
Deleting the first node from a circular linked list

Deletion from beginning - CLL

```
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
```

Algorithm to delete the first node

Deletion from end- CLL



Deleting the last node from a circular linked list

Deletion from end- CLL

```
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
```

Algorithm to delete the last node

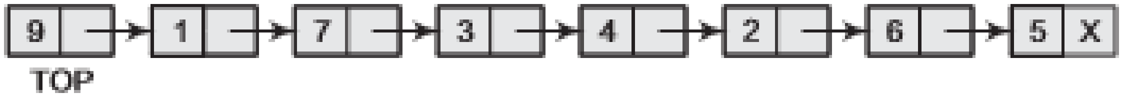
Stack as linked list

- The storage requirement of linked representation of the stack with n elements is O(n), and the typical time requirement for the operations is O(1).

- In a linked stack, every node has two parts—one that stores data and another that stores the address of the next node.

- The START pointer of the linked list is used as TOP.

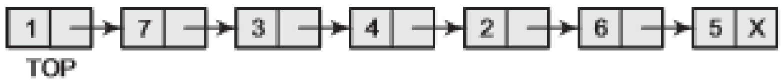
- All insertions and deletions are done at the node pointed by TOP. If TOP = NULL, then it indicates that the stack is empty.



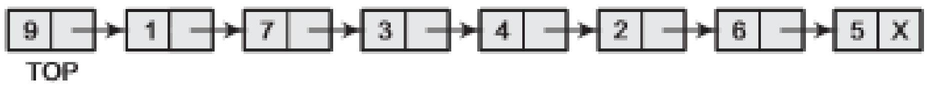
Linked stack

Operations on linked stack

- **Push operation**



Linked stack



Linked stack after inserting a new node

Step 1: Allocate memory for the NEW_NODE

Step 2: SET NEW_NODE DATA = VAL

Step 3: IF TOP = NULL

SET NEW_NODE-> NEXT = NULL

SET TOP = NEW_NODE

ELSE

SET NEW_NODE ->NEXT = TOP

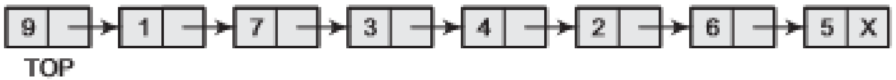
SET TOP = NEW_NODE

[END OF IF]

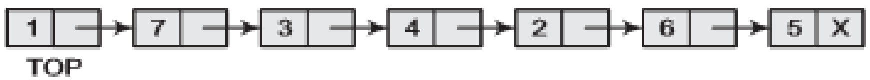
Step 4: END

Operations on linked stack

- **Pop operation**



Linked stack



Linked stack after deletion of the topmost element

Step 1: IF TOP = NULL

PRINT UNDERFLOW

Goto Step 5

[END OF IF]

Step 2: SET PTR = TOP

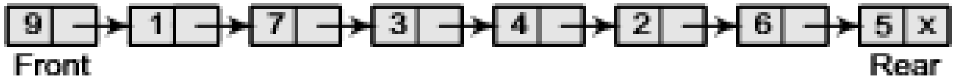
Step 3: SET TOP = TOP ->NEXT

Step 4: FREE PTR

Step 5: END

Queue as linked list

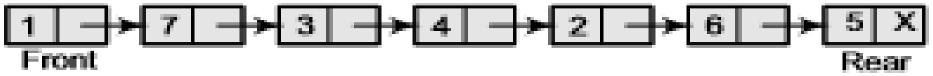
- The storage requirement of linked representation of a queue with n elements is O(n) and the typical time requirement for operations is O(1).
- In a linked queue, every element has two parts, one that stores the data and another that stores the address of the next element.
- The START pointer of the linked list is used as FRONT. Here, we will also use another pointer called REAR, which will store the address of the last element in the queue.
- All insertions will be done at the rear end and all the deletions will be done at the front end.
- If FRONT = REAR = NULL, then it indicates that the queue is empty.



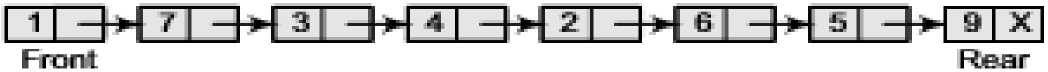
Linked queue

Operations on linked queue

- **Enqueue operation**



Linked queue

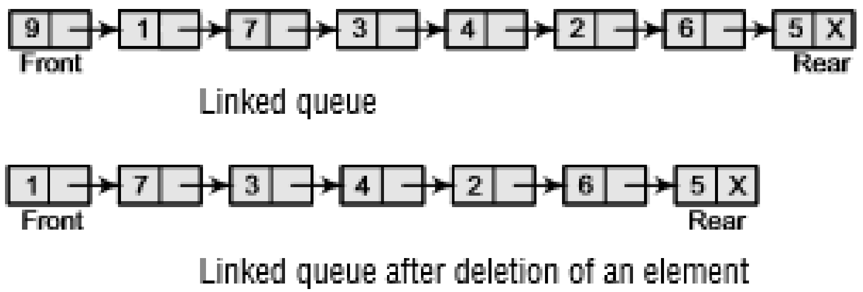


Linked queue after inserting a new node

Step 1: Allocate memory for the new node 'PTR'
Step 2: SET PTR-> DATA = VAL
Step 3: IF FRONT = NULL
    SET FRONT = REAR = PTR
    SET FRONT-> NEXT = REAR-> NEXT = NULL
  ELSE
    SET REAR-> NEXT = PTR
    SET REAR = PTR
    SET REAR-> NEXT = NULL
  [END OF IF]
Step 4: END

Operations on linked queue

- **Dequeue operation**



Step 1: IF FRONT = NULL

Write Underflow

Go to Step 5

[END OF IF]

Step 2: SET PTR = FRONT

Step 3: SET FRONT = FRONT -> NEXT

Step 4: FREE PTR

Step 5: END

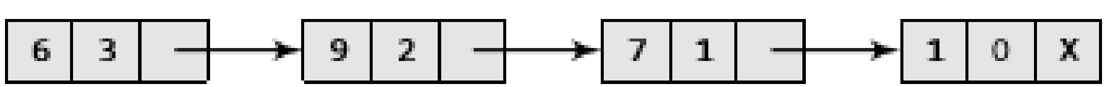
Application of linked list

Addition of two polynomials:

- Linked lists are widely used to represent and manipulate polynomials. Polynomials are the expressions containing number of terms with nonzero coefficients and exponents.
    - $p(X) = a_n x_n^{e} + a_{n-1} x_{n-1}^{e} + \dots\dots\dots + a_1 x_1^{e} + a$

- In the linked list representation of polynomials, each term is considered as a node. Such a node contains three fields.
    - Coefficient field
    - Exponent field
    - Link field

- Consider a polynomial $6x^3 + 9x^2 + 7x + 1$:



Linked representation of a polynomial

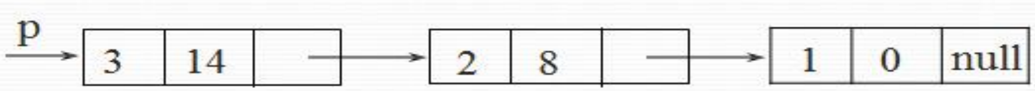
Addition of two polynomials

1. Read the number of terms in the first polynomial, **P**.
2. Read the coefficient and exponents of the first polynomial **P**.
3. Read the number of terms in the second polynomial **Q**.
4. Read the coefficient and exponents of the second polynomial **Q**.
5. Set the temporary pointers **p** and **q** to traverse the two polynomials respectively.
6. Compare the exponents of the two polynomials starting from the first nodes.
   a. If both the exponents are equal then add the coefficients and store it in the resultant linked list **R**. Move both pointers to the next nodes.
   b. If the exponent of the current term of **P** is less than the exponent of the current term of **Q**, then the current term of **Q** is added to the resultant linked list **R** and the pointer **q** is moved to the next node.
   c. If the exponent of the current term of **P** is greater than the exponent of the current term of **Q**, then the current term of **P** is added to the resultant linked list **R** and the pointer **p** is moved to the next node.
   d. Append the remaining nodes of either of the polynomials to the resultant linked list **R**.

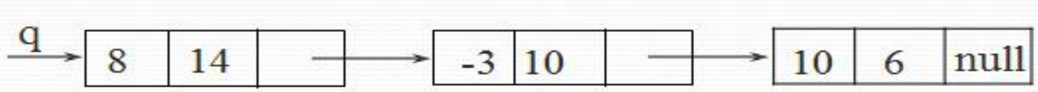
In summary:

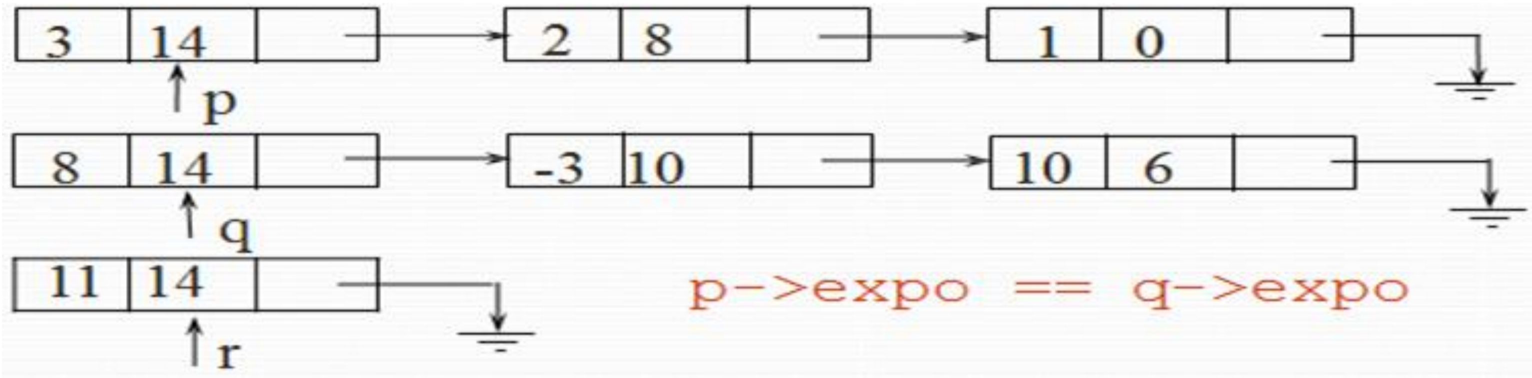
- To do this, we have to break the process down to cases:
- Case 1: exponent of p > exponent of q
  - Copy node of p to end of r .
  - p = p->next
- Case 2: exponent of p < exponent of q
  - Copy node of q to end of r .
  - q= q->next
- Case 3: exponent of p = exponent of q
  - Create a new node in r with the same exponent and with the sum of the coefficients of p and q.
  - p = p->next and q = q->next

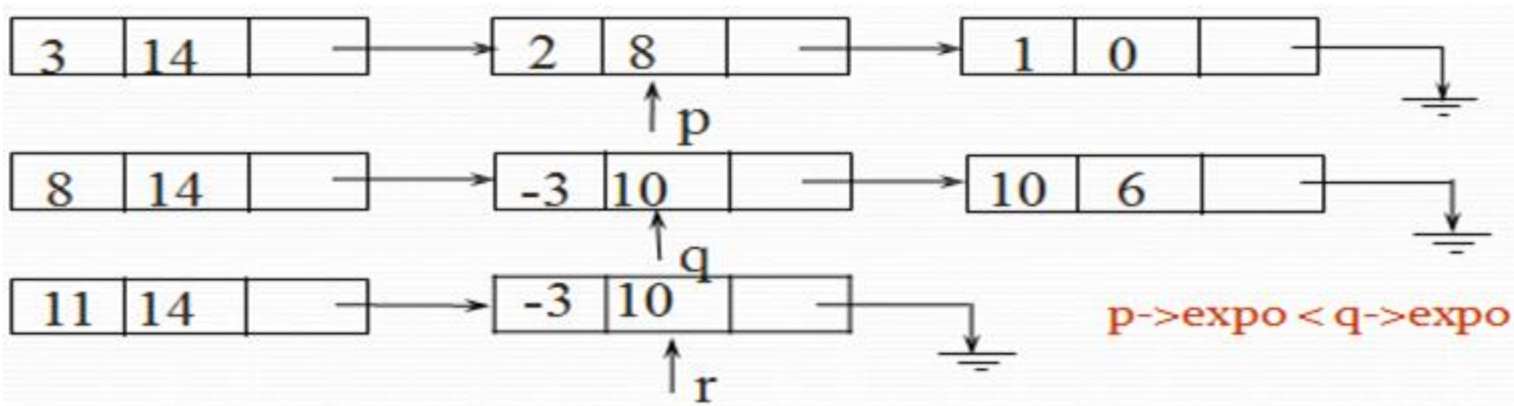
$$P = 3x^{14} + 2x^{8} + 1$$

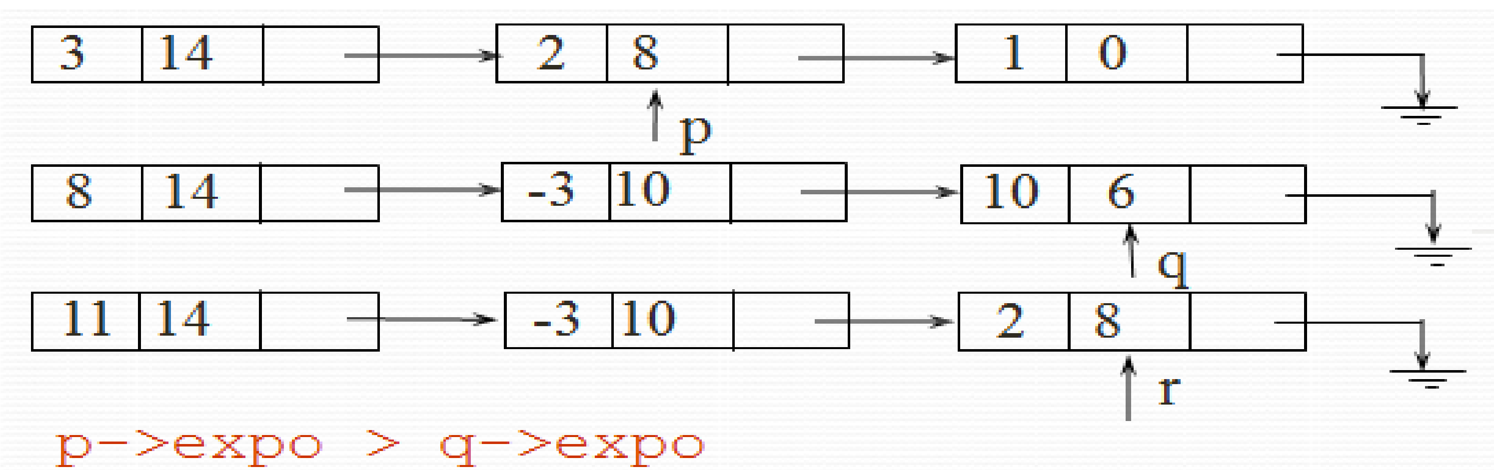


$$Q = 8x^{14} - 3x^{10} + 10x^{6}$$




3 14
2 8
1 0
p
8 14
-3 10
10 6
q
11 14
r
p->expo == q->expo


3 14
2 8
1 0
p
8 14
-3 10
10 6
q
11 14
-3 10
r
p->expo < q->expo


3 14
2 8
1 0
p
8 14
-3 10
10 6
q
11 14
-3 10
2 8
r
p->expo > q->expo

THANK YOU!