College of computer science & mathematics

Dep. Of Computer Science

DATA STRUCTURE
هياكل بيانات

Lecture 8 :
Linked Kist

Prepared & Presented by

Mohammed B. Omar

2023 -2024

Linked list

- **A linked list is a linear collection of data elements called nodes where linear order is given by means of pointers.**

- Like arrays, Linked List is a linear data structure.
- Unlike arrays, Linked list elements are not stored at a contiguous location; the elements are linked using pointers.

- Collection of links with reference to the first.

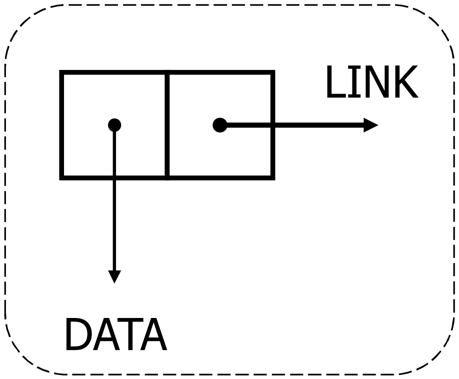
Each link has
- **part to store data**
- **link that refers to the next link in the list.**

- Data part of the link can be an integer, a character, a String or an object of any kind.

- **Each node has two parts**

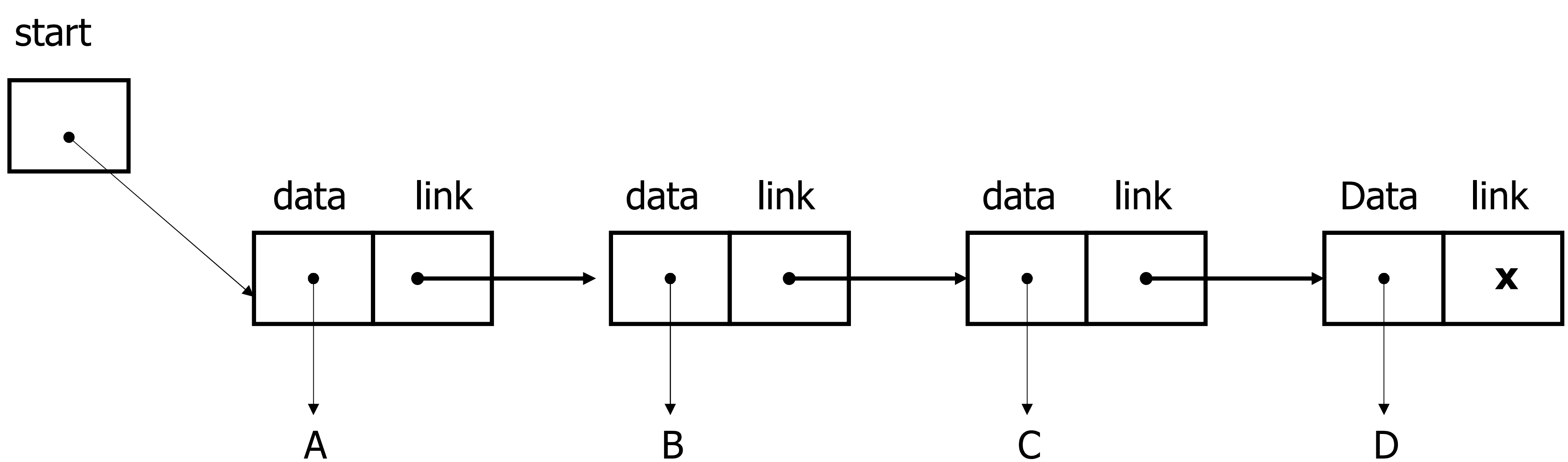
  1) Data Field – Stores data of the node
  2) Link Field – Store address of the next node (i.e. Link to the next node)

NODE :



Linked Lists

- **START is List pointer contains address of the first node in the List.**
- **All nodes are connected to each other through Link fields.**
- **Link of the last node is NULL pointer denoted by ‘X’ sign.**
- **Null pointer indicated end of the list.**



Declaration

```
struct nodeType
{
    int info;
    nodeType *link;
};
        //The variable declaration is as follows:

nodeType *head=NULL;
```

```
struct Node                        // name of node
{
    char name[20];           // Name of up to 20 letters
    int age;                 // D.O.B. would be better
    float height;            // In meters
    Node *next;              // Pointer to next node
};
    Node *start_ptr = NULL;    // Start Pointer (root)
```

ADVANTAGES AND DISADVANTAGES

- **Linked list have many *advantages* and some of them are:**

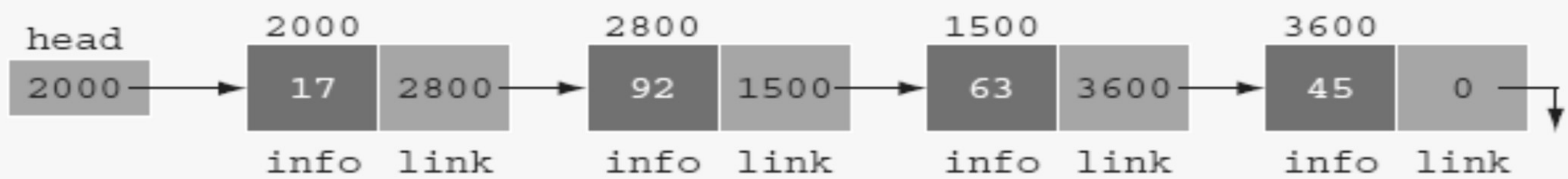
1. Linked list are **dynamic data structure**. That is, they can grow or shrink during the execution of a program.
2. **Efficient memory utilization**: In linked list (or dynamic) representation,
3. **memory is not pre-allocated**. Memory is allocated whenever it is required. And it is deallocated (or removed) when it is not needed.
4. **Insertion and deletion are easier and efficient**. Linked list provides flexibility in inserting a data item at a specified position and deletion of a data item from the given position. Many complex applications can be easily carried out with linked list.

- **Linked list has following *disadvantages***

1. More memory: to store an integer number, a node with integer **data and address** field is allocated. That is more memory space is needed.
2. Access to an arbitrary data item is little bit **cumbersome and also time consuming.**

يعد الوصول إلى عنصر بيانات أصعب ويستغرق وقت.

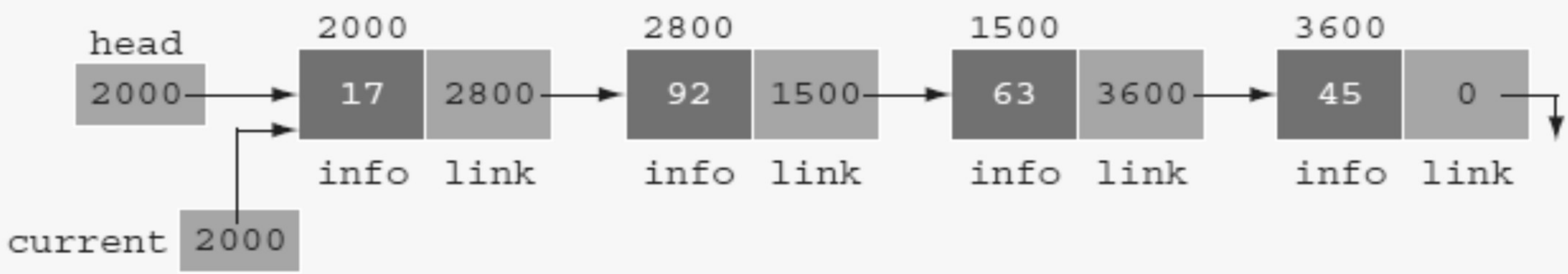
Linked Lists: some properties



	Value	Explanation
head	2000	
head->info	17	Because head is 2000 and the info of the node at location 2000 is 17
head->link	2800	
head->link->info	92	Because head->link is 2800 and the info of the node at location 2800 is 92

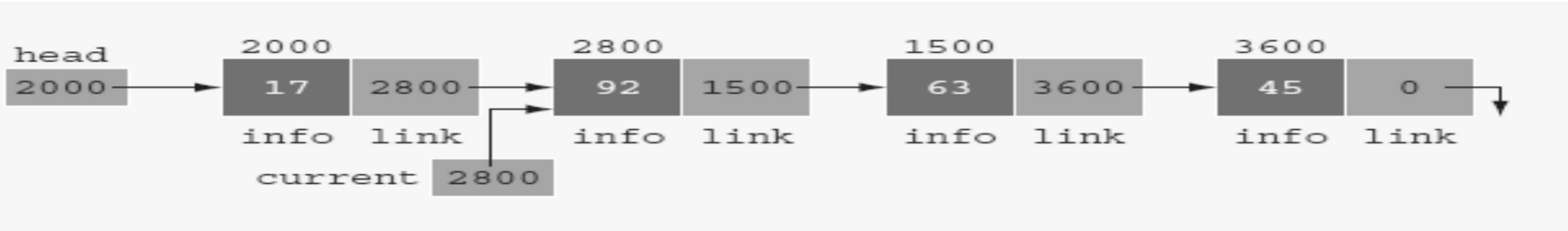
- **current = head ;**

copies the value of head into current .



	Value
Current	**2000**
Current −> info	**17**
Current −> link	**2800**
Current −> link −> info	**92**

- **current = current ->link ;**



	Value
Current	2800
Current -> info	92
Current -> link	1500
Current -> link -> info	63
head-> link ->link	1500
head->link->link->info	63
head->link->link->link	3600
head->link->link->link->info	45
current-> link ->link	3600
current->link->link->info	45
current->link->link->link	0(that is NULL)
Current ->link->link->link->info	Does not exist

The following code traverses the list:

```
current = head;
while (current != NULL)
{
    //Process current
    current = current->link;
}
```

The following code outputs the data stored in each node:

```
current = head;
while (current != NULL)
{
    cout << current->info << " ";
    current = current->link;
}
```

Basic Operations on SLL

● Insertion

1-First node

2-In beginning

3-In end

4-In middle

● Deletion

1-from beginning

2-from end

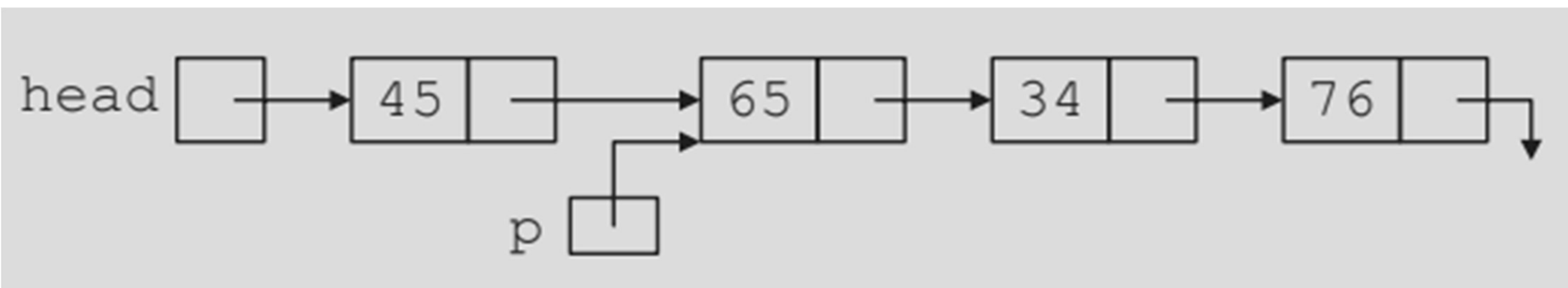
3-from middle

INSERTION

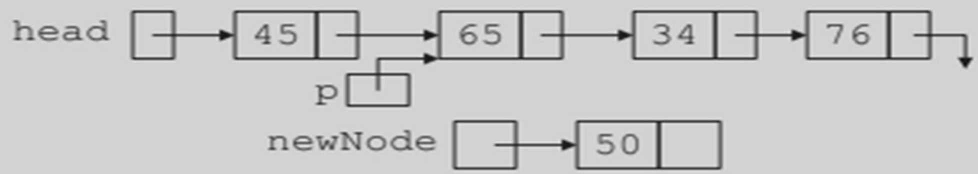
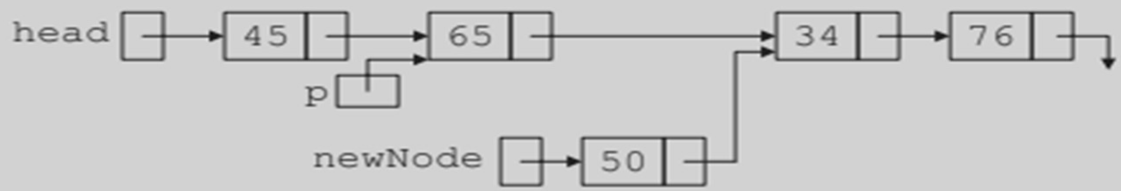
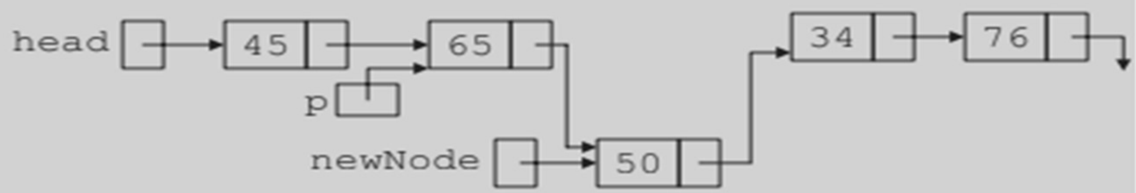
We will use the following variable declaration:

```
nodeType *head, *p, *q, *newNode;
```

Consider the linked list shown in Figure



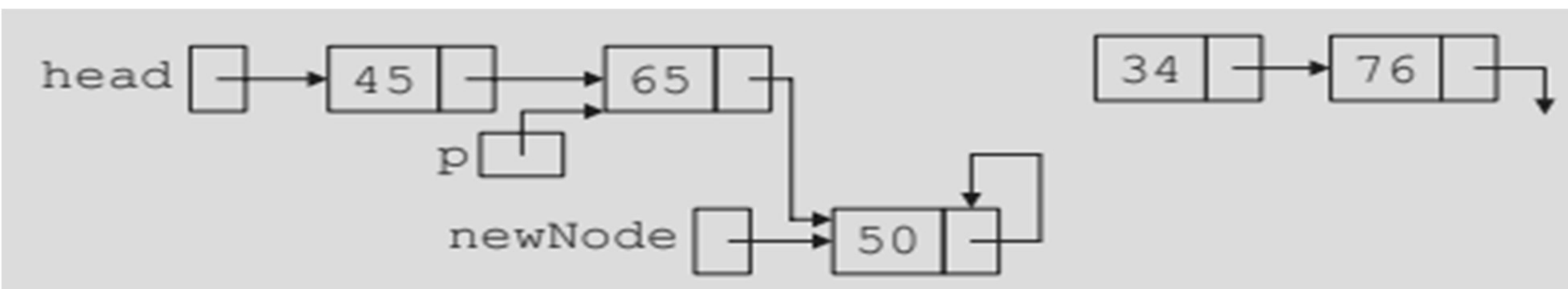
```
newNode = new nodeType;                 //create newNode

newNode->info = 50;                     //store 50 in the new node

newNode->link = p->link;

p->link = newNode;
```

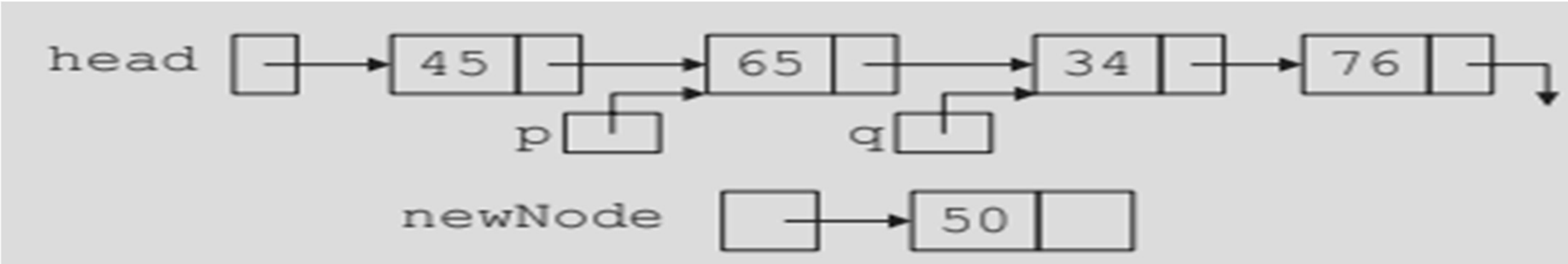
Statement	Effect
`newNode = new nodeType;`	
`newNode->info = 50;`	
`newNode->link = p->link;`	
`p->link = newNode;`	

newNode->link = p->link; //correct sequence
p->link = newNode;

Suppose that we reverse the sequence of the statements and execute the statements in the following order:

p->link = newNode; // failed sequence
newNode->link = p->link;



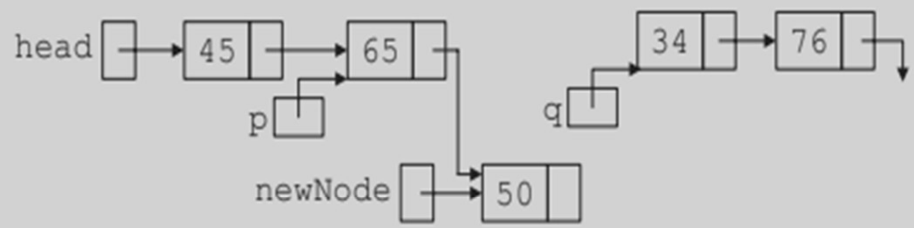
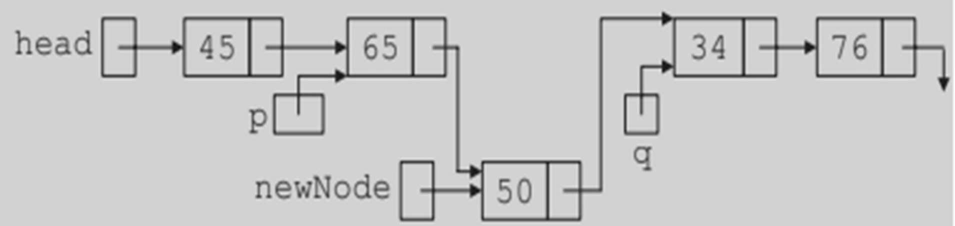


The following statements insert newNode between p and q:

```
newNode->link = q;
p->link = newNode;
```

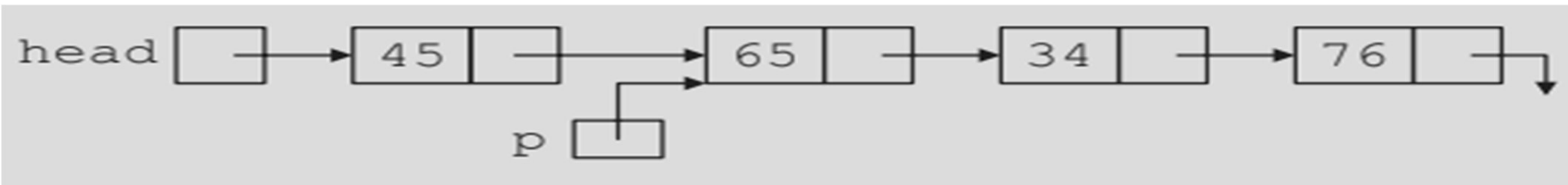
suppose that we execute the statements in the following order:

```
p->link = newNode;
newNode->link = q
```

Statement	Effect
`p->link = newNode;`	
`newNode->link = q;`	

DELETION

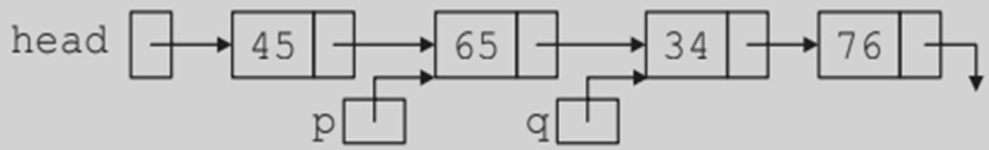
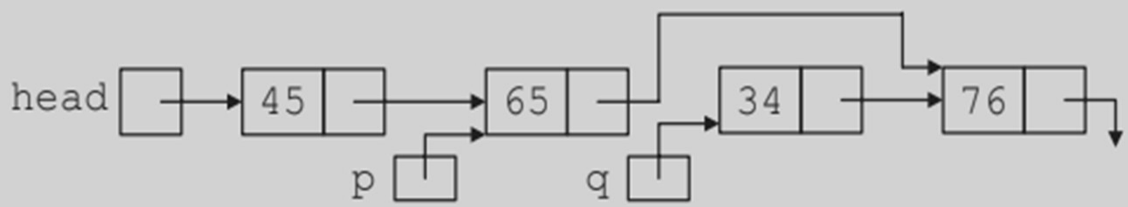
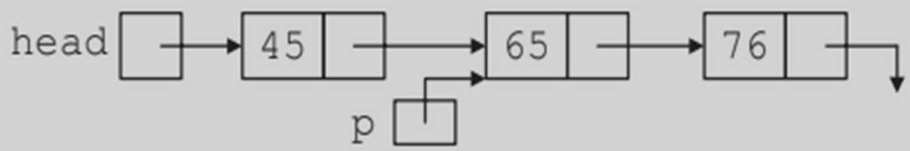
- Consider the linked list shown in Figure



- Suppose that the node with info 34 is to be deleted from the list. The following statement removes the node from the list: حذف العقدة مع بقاءها في الذاكرة بموقع محجوز
- p->link = p->link->link;

- The following statements delete the node from the list and deallocate the memory occupied by this node: الطريقة الأصح للحذف

```
q = p->link;
p->link = q->link;
delete q;
```

<table>
<tr><th>Statement</th><th>Effect</th></tr>
<tr><td><code>q = p->link;</code></td><td>
</td></tr>
<tr><td><code>p->link = q->link;</code></td><td>
</td></tr>
<tr><td><code>delete q;</code></td><td>
</td></tr>
</table>

Algorithms

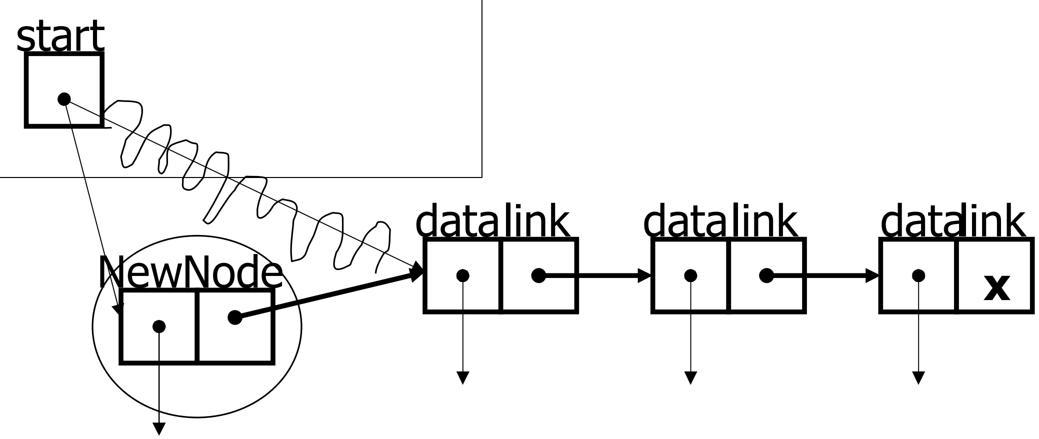
- Lets consider,

- **START** is the 1st position in Linked List
- **NewNode** is the new node to be created
- **DATA** is the element to be inserted in new node
- **POS** is the position where the new node to be inserted
- **TEMP** and **HOLD** are temporary pointers to hold the node address

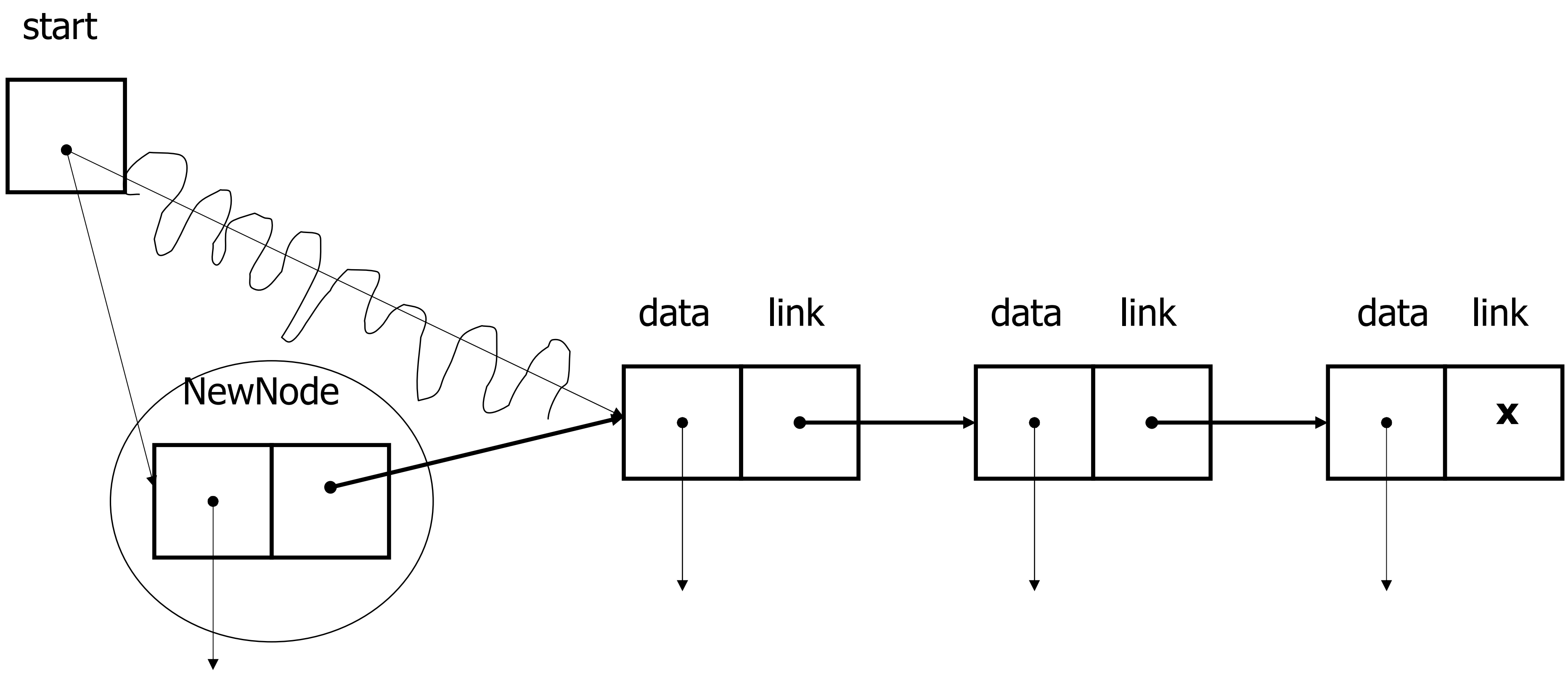
DATA STRUCTURE هياكل بيانات

Algorithm to Insert a Node at the beginning

1. Input DATA to be inserted
2. Create NewNode
3. NewNode -> DATA = DATA
4. If START is equal to NULL //list empty
   a) NewNode -> LINK = NULL
5. Else
   a) NewNode -> LINK = START
6. START = NewNode
7. Exit



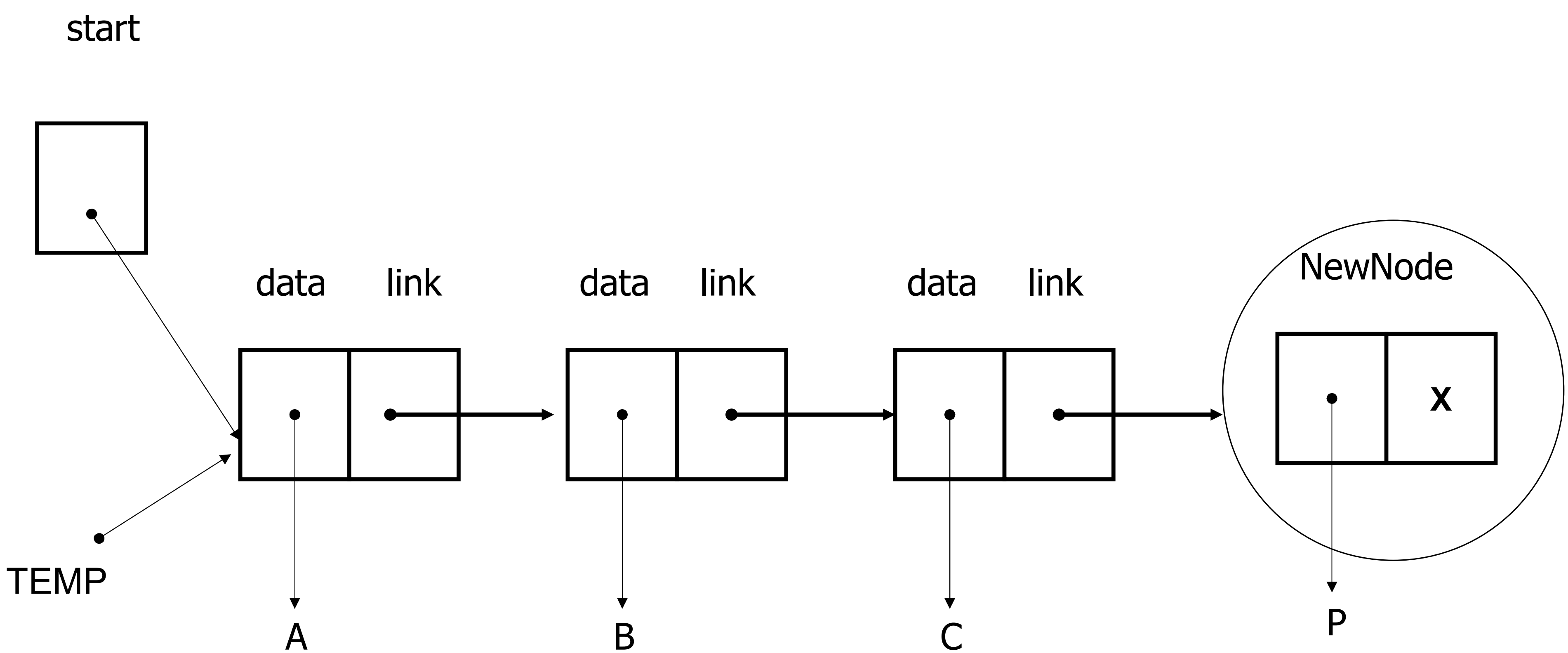
Insert a Node at the beginning



Algorithm to Insert a Node at the end

1. Input DATA to be inserted
2. Create NewNode
3. NewNode -> DATA = DATA
4. **NewNode -> LINK = NULL**
5. If START is equal to NULL
    a) START = NewNode
6. Else
    a) TEMP = START
    b) while (TEMP -> LINK not equal to NULL)
    i) TEMP = TEMP -> LINK
7. TEMP -> Link = NewNode
8. Exit

Insert a Node at the end

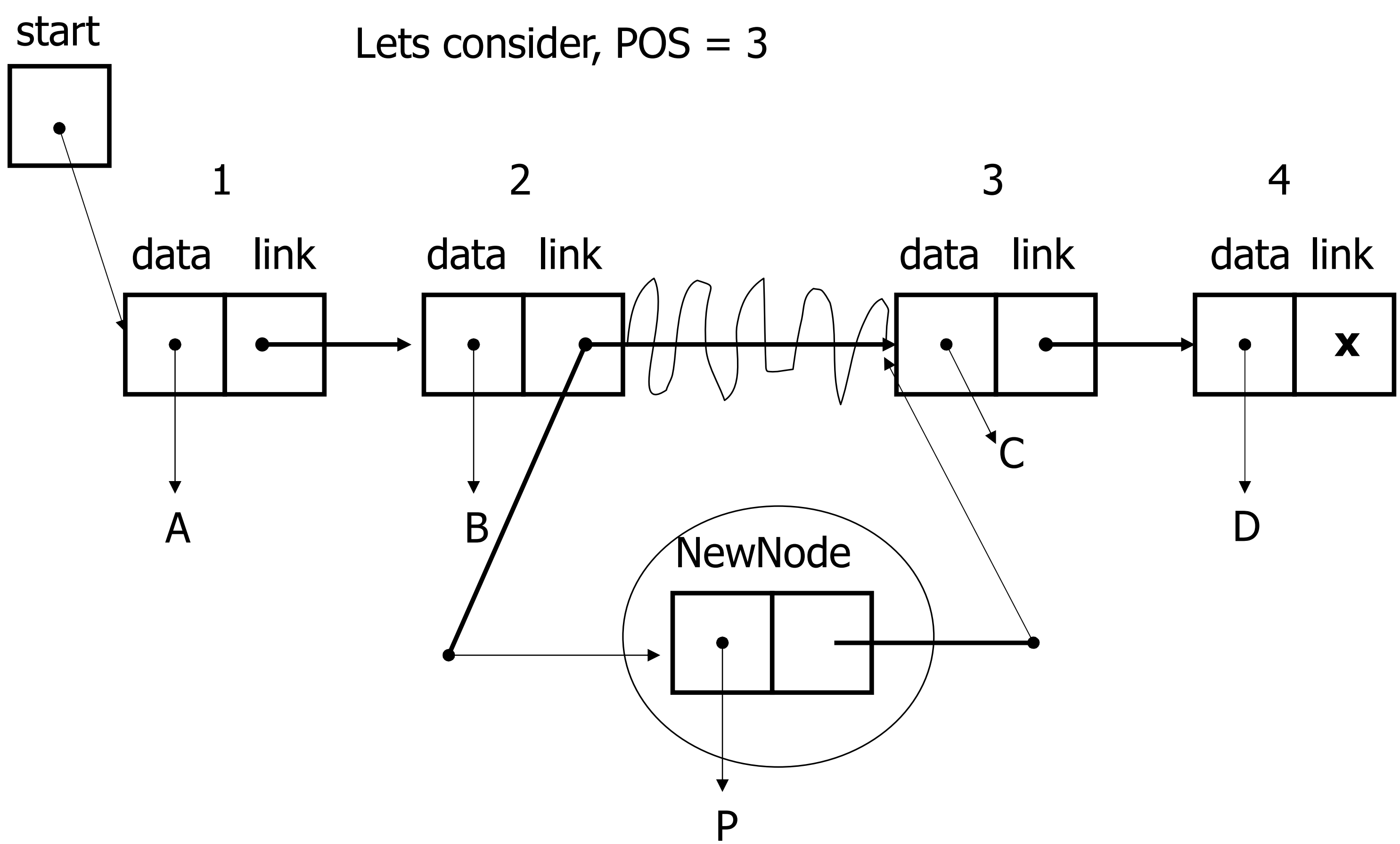


Algorithm to Insert a Node at any specified position

1. Input DATA to be inserted and POS, the position to be inserted.
2. Initialize TEMP = START and K=1
3. Repeat step 3 while (K is less than POS)
    a) TEMP = TEMP -> LINK
        b) If TEMP -> LINK = NULL
            i) Exit
        c) K = K + 1
4. Create a Newnode
5. Newnode -> DATA = DATA
6. Newnode -> LINK = TEMP -> LINK
7. TEMP -> LINK = NewNode
8. Exit

Insert a Node at middle position

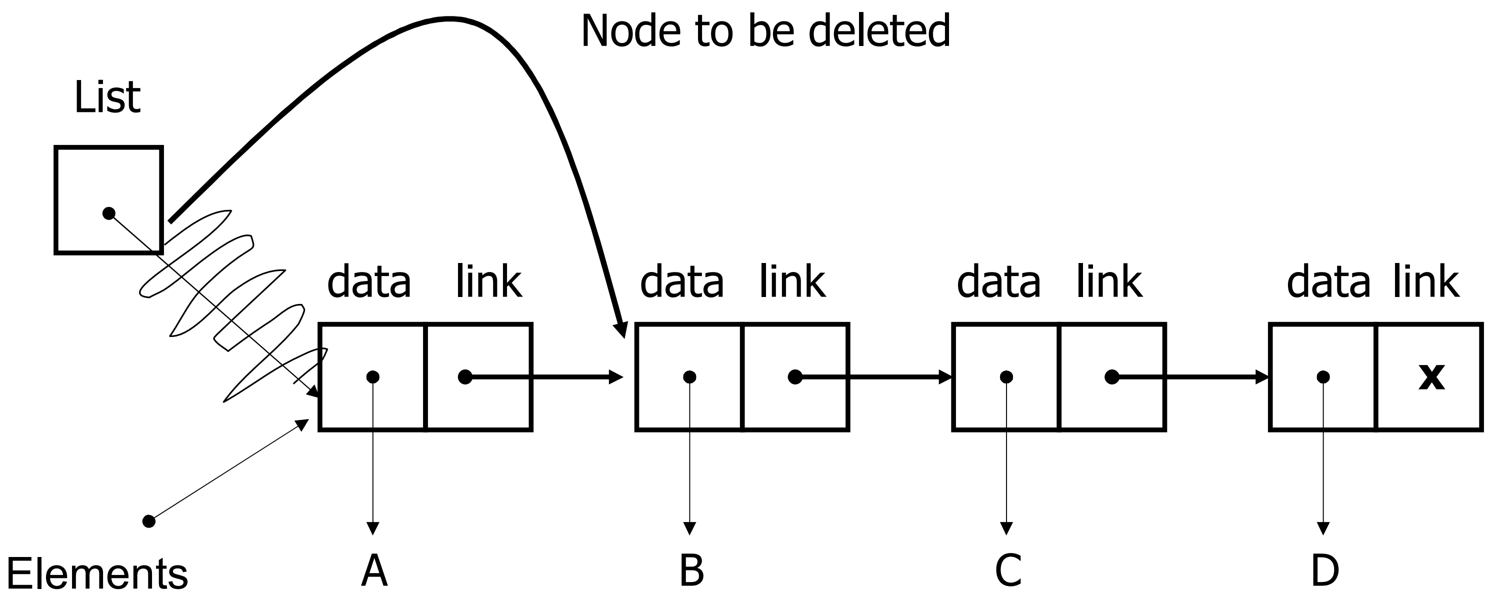
Lets consider, POS = 3



Algorithm to Delete the first Node

```
void DeleteFront(void)
   {
      Linked_List *Element;
       if(List==NULL)
       cout<<"\t\t* You can not Delete, Linked List is Empty *\n";
        else {
                 Element = List;
                 List = List ->next;
                 delete Element;
                 }
    }
```

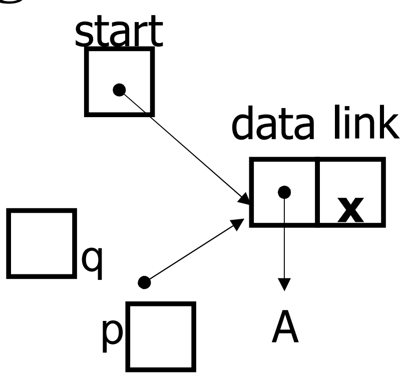
Algorithm to Delete the first Node

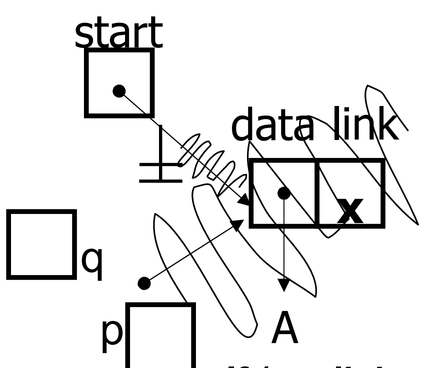


Algorithm to Delete the last Node

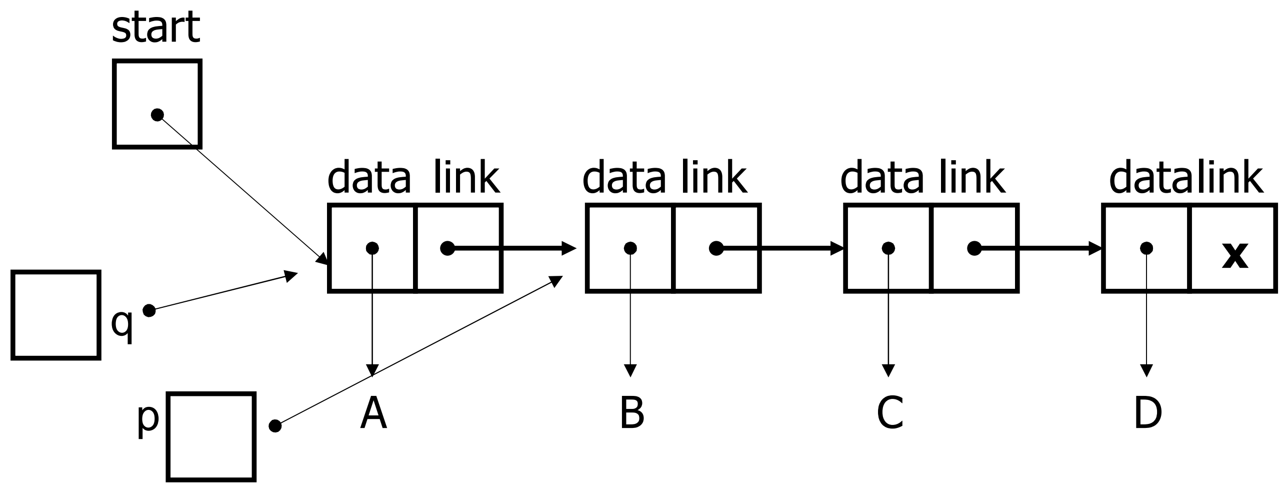
```
void delete_last()
{
  Linked_List *p, *q;
  p=start;
  If (p-> link ==NULL)
    {
      delete p;
      start=NULL;
     }
    else
      {
       while(p-> link != NULL)
           {
             q=p;
              p= p-> link ;
           }
      q-> link=NULL;
      delete p;
      }    }
```

Algorithm to Delete the last Node






```
If (p-> link ==NULL)
    {
      delete p;
      start=NULL;
     }
```

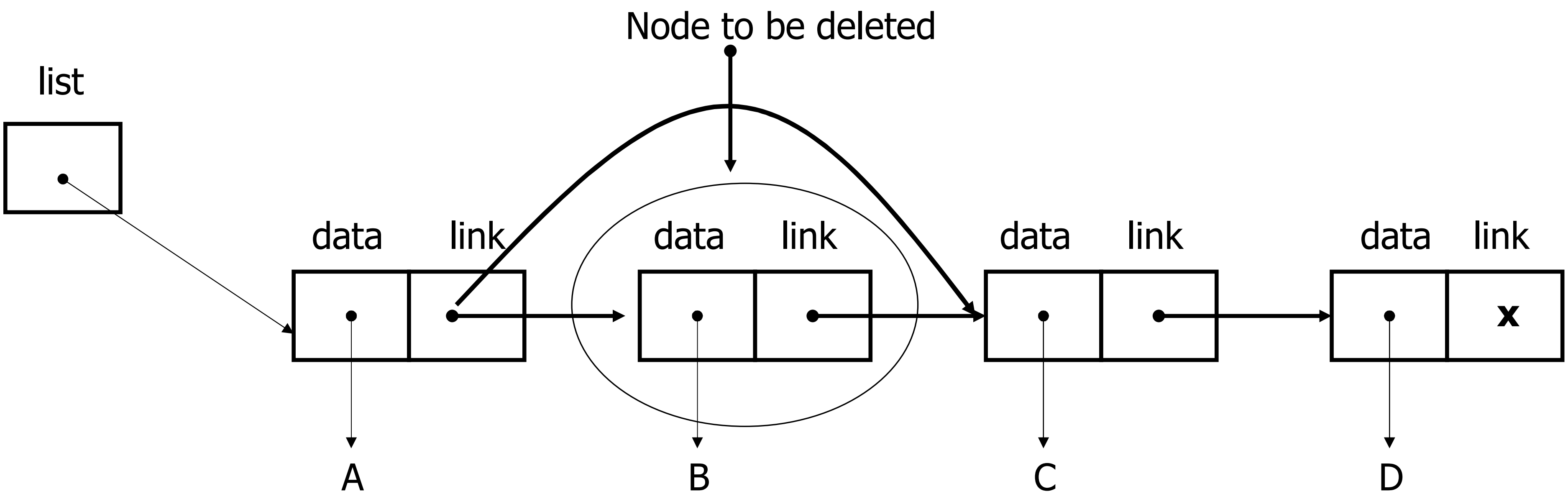



```
while(p-> link != NULL)
        {
          q=p;
           p= p-> link ;
        }
    q-> link=NULL;
    delete p;
```

Algorithm to Delete a Node with a specified value

```
void DeleteVal( int value)
{
 Linked_List *p, *q;
 p=List;
 while (p-> next != NULL )
        {
           if(p->number==value)
             {
              q=p->next;
              delete p;
              p=q;
              }
            else {

               cout<<" the Value not exist";

            }
          p=p->next;
         }
 }
```

Algorithm to Delete a Node



Algorithm to sorting a list

```
void SortAsc(void)
{
Linked_List *p,*q;
if(List == NULL)
    cout<<"\t\t* Linked List is Empty *\n";
    else {
         p = List;
         q = List;
         while(p->next!=NULL)
               {
                 q=p->next;
                 while(q!=NULL)
                        {
                          if(q->number < p->number)
                             {
                               int s;
                               s=p->number;
```

```
      p->number = q->number;
      q->number = s;
       }
   q= q->next;
}
P =p ->next;
}
}
}
```

Algorithm for Searching a Node

Suppose START is the address of the first node in the linked list and DATA is the information to be searched.

1. Input the DATA to be searched.

2. Initialize TEMP = START and Pos =1

3. Repeat the step 4,5 and 6 until (TEMP is equal to NULL) // (while temp!=null)

4. If (TEMP -> DATA is equal to DATA)

(a) Display "The DATA found at POS "

(b) Exit

5. TEMP = TEMP -> LINK

6. POS = POS + 1

7. If (TEMP is equal to NULL) // the data if not find in all list

(a) Display " The DATA is not found in the list"

8. Exit.

Algorithm for Displaying all Nodes

- **Suppose List is the address of the first node in the linked list.**

1. **If (START is equal to NULL)**

   (a) Display “The List is Empty”

   (b) Exit

2. **Initialize TEMP = START**
3. **Repeat the Step 4 and 5 until (TEMP == NULL)**
4. **Display TEMP -> DATA**
5. **TEMP = TEMP -> LINK**
6. **Exit**

Singly Linked Lists and Arrays

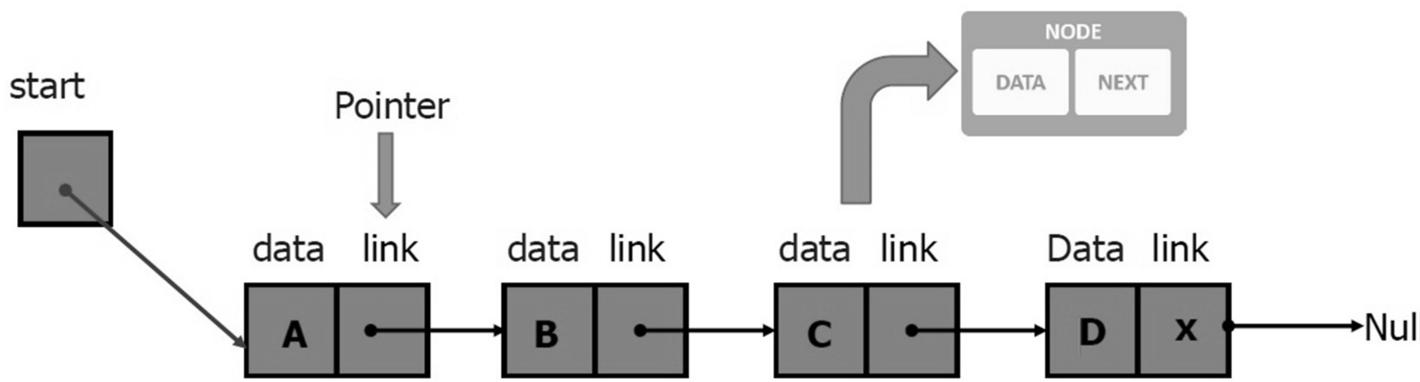
Singly linked list	Array
Elements are stored in linear order, accessible with links.	Elements are stored in linear order, accessible with an index.
Do not have a fixed size.	Have a fixed size.
Cannot access the previous element directly.	Can access the previous element easily.

Types of Linked List

- One-way linked list
- Two way or doubly linked list
- Circular linked list
- Header linked list

1- One Way Linked List (Singly Linked Lists)

One-way linked list is <u>most simple</u> list among all linked lists. It contains data part or info part and address part or link field. Address part link to the next node in sequence of nodes. <u>It can be traversed only in one direction that is forward direction.</u> One-way linked list takes less memory because it <u>has only one pointer or address part.</u>

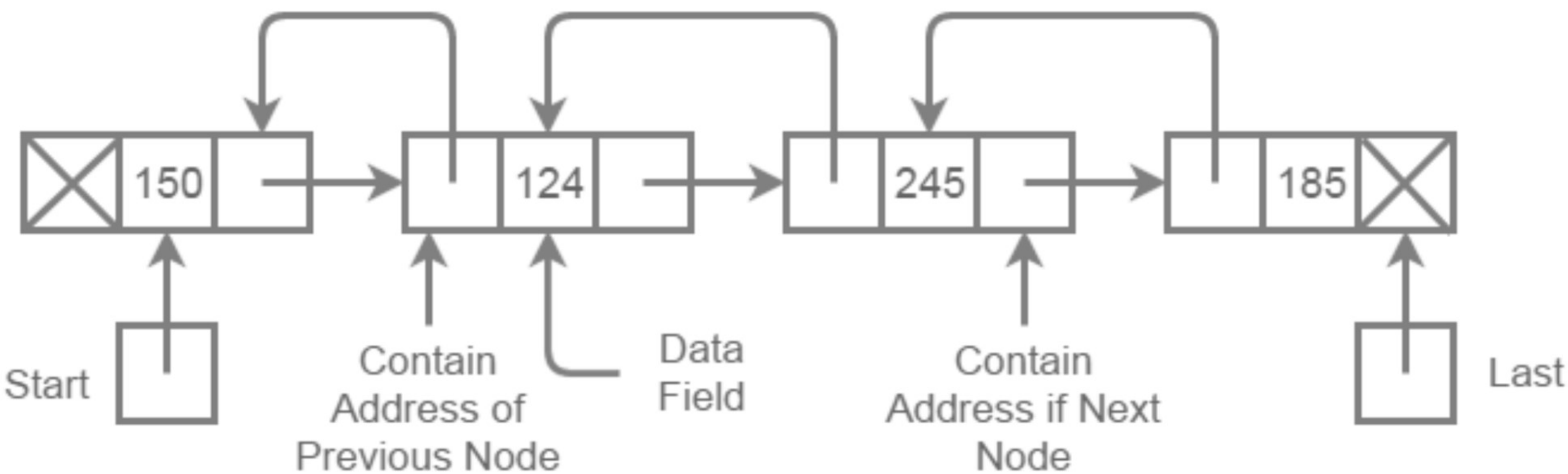


2- Two Way Linked List in Data Structure (Doubly Linked List)

A two-way linked list is also known as doubly linked list. Each node in two-way linked list divided into three parts. Which is data part and two link fields. Data part contains the info or data of the node. One link field is used for forward direction which contains the address of its next node, and second field is used for backward direction which contains the address of its previous node.

Link Field 2	Data Field	Link Field 1

As compare to one-way linked list, two-way linked list can traversed in reverse direction with help of backward link field. Sorting data as two-way link require more time and more time, and <u>now we have two pointer variables START and LAST, which contains the address of first node and last node.</u>



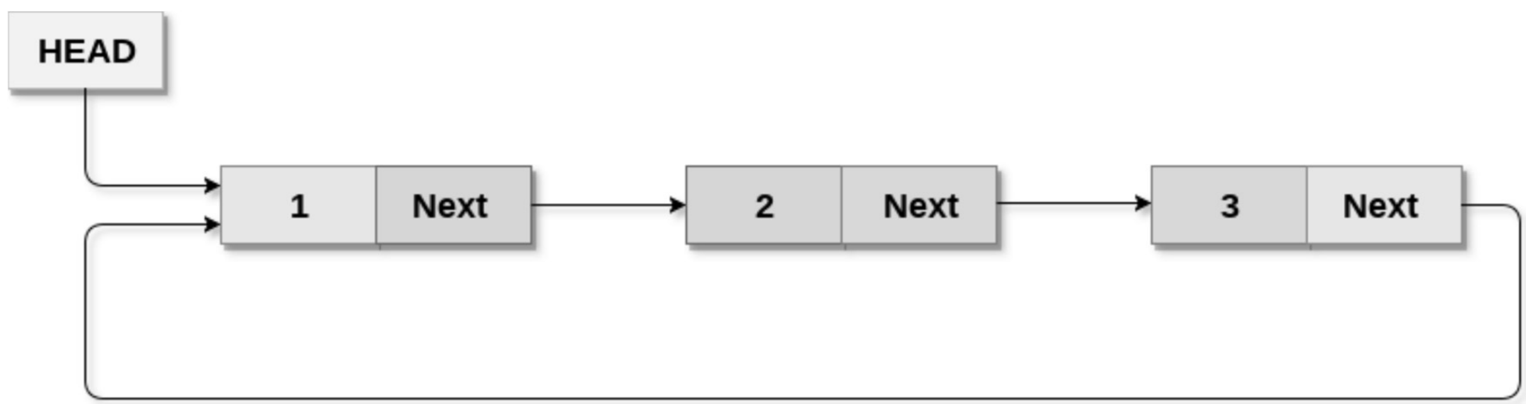
Declaration

```
Struct dnode
{
Struct dnode *back;
int data;
Struct dnode *forw;
}
```

In declaration data represent the data field and back and forw represent the two link fields which contain the address of forward and backward node.

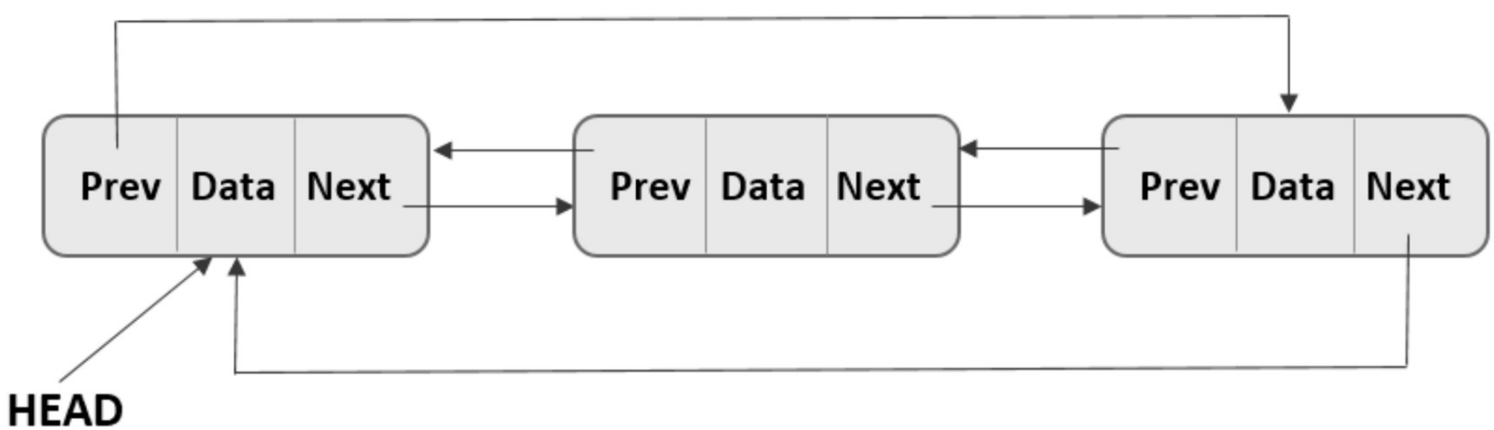
3- Circular Linked List

A circular linked list is that in which the last node contains the pointer to the first node of the list.



4- Doubly Circular Linked List

A Doubly Circular linked list or a circular two-way linked list is a more complex type of linked list that contains a pointer to the next as well as the previous node in the sequence. The difference between the doubly linked and circular doubly list is the same as that 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.



Difference between Single Linked list and Double Linked List in Data Structure

Single Linked List	**Double Linked List**
Single linked list also known as one way list	Double linked list is also known as two way list
Each node is divided into two part. **Data Field-** contain the Data of node **Link Field-** Contain the address of next node	Each node is divided into three part: **Data field-** Contain the data of node. **Forward Link Field:** Contain the address of next node **Backward Link Field-** Contain the address of previous node
It can traversed only forward direction	It can be traversed both forward and backward direction.
Single linked list use less memory and less space.	Double linked list use more memory and more space because of two pointer
If we need to save memory in need to update node values frequently and searching is not required, we can use singly linked list	If we need faster performance in searching and memory is not a limitation we use Doubly Linked List

Thank You

&

Good luck