Book

A Simplified Approach to Data Structures

Prof.(Dr.)Vishal Goyal, Professor, Punjabi University Patiala

Dr. Lalit Goyal, Associate Professor, DAV College, Jalandhar

Mr. Pawan Kumar, Assistant Professor, DAV College, Bhatinda

Shroff Publications and Distributors

Edition 2014

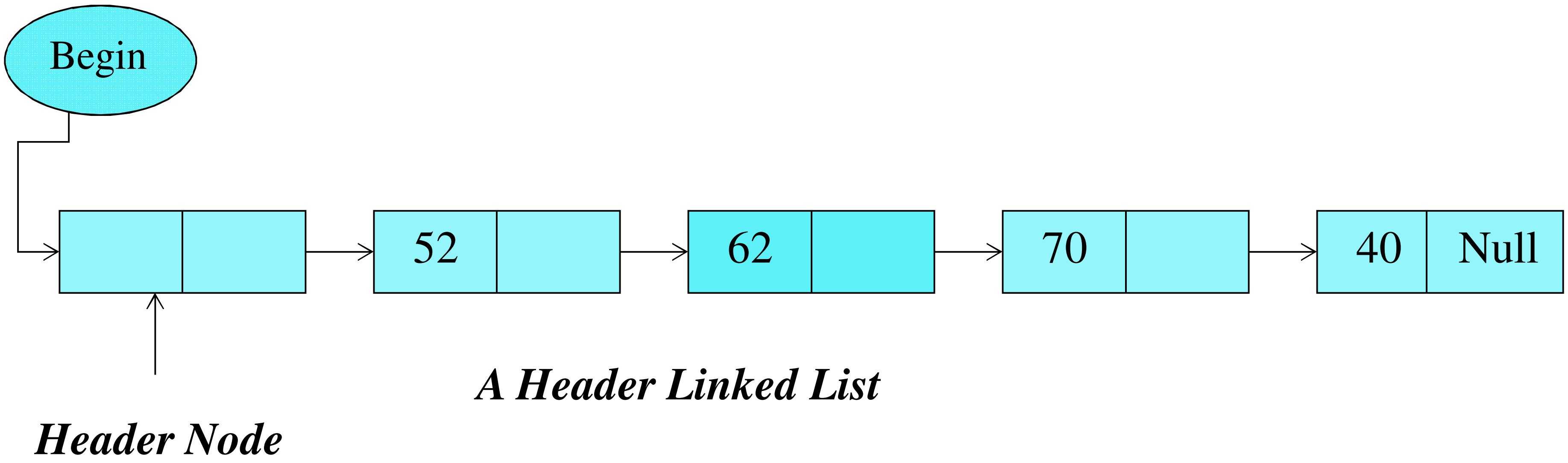
Header Linked List

CONTENTS

- Applications of the linked list
- Header linked list

Header Linked List

A header linked is a special kind of linked list which contains a special node at the beginning of the list. This special node is known as *header node* and contain important information regarding the linked list. This information may be total number of nodes in list, some description for the user like creation and modification data about, whether the data in the list is sorted and unsorted. The header linked list in shown below:



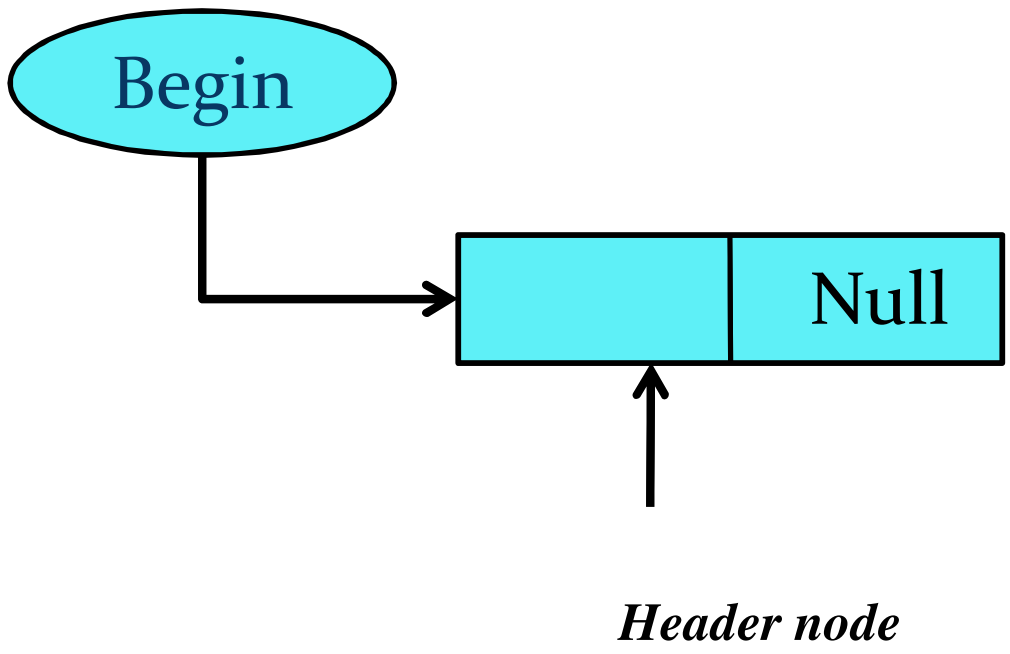
A Header Linked List

Categories of Header Linked List

- Grounded Header Linked List
- Circular Header Linked List
- Two-Way Header Linked List
- Circular Two-Way Header Linked List

Grounded Header Linked List

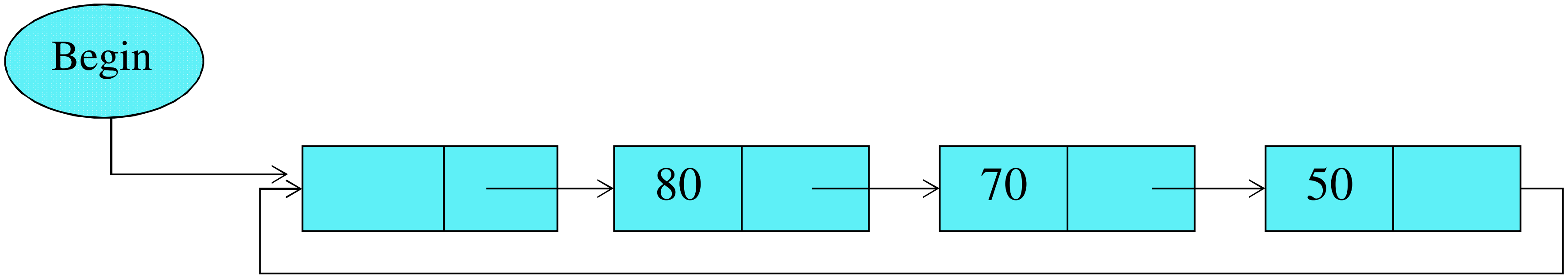
A grounded header linked list is a list in which last node of the list control the Null in its Next pointer field. Shown in figure blow:



An Empty Grouned Header Linked List

Circular Header Linked List

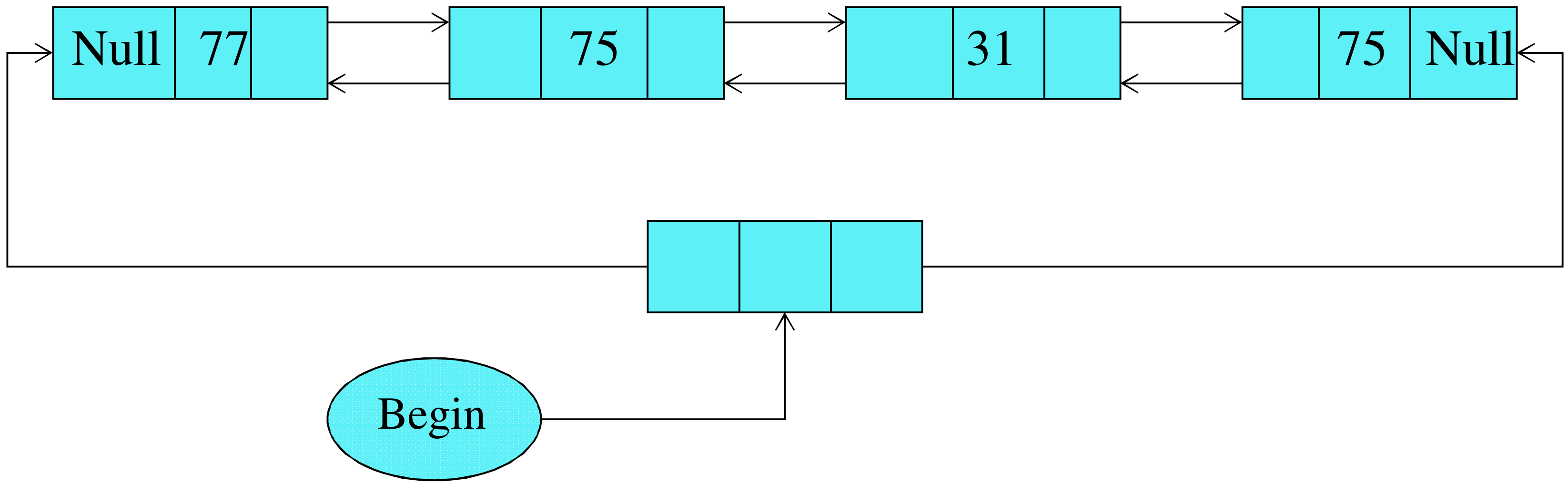
A *Circular header* linked list is a list in which last node of the list points back to the header node i.e. *Next* pointer field of the last node contains the address of the header node. Shown in figure blow:



A Circular Header Linked List

A Two –Way Header Linked List

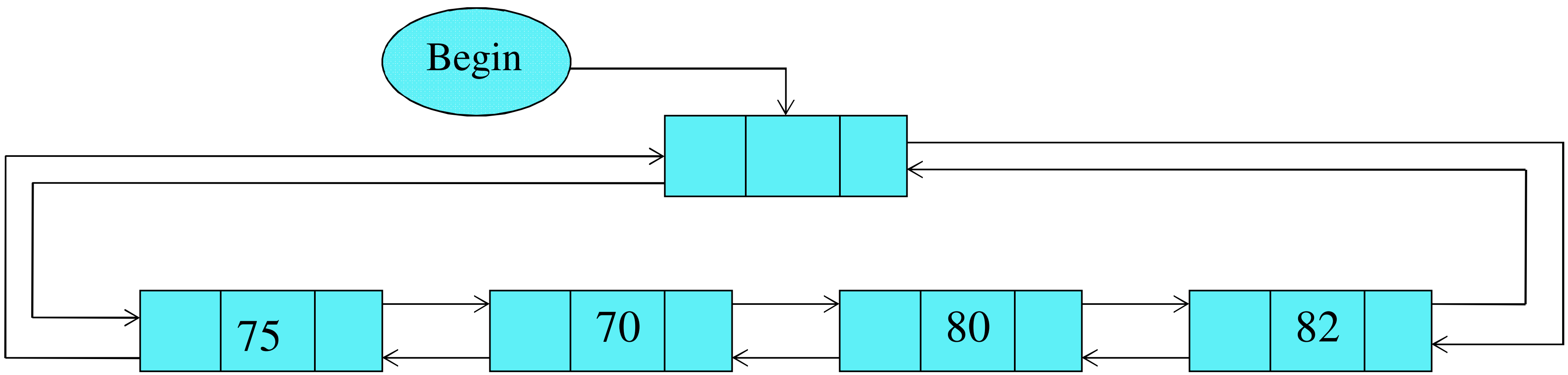
In general, a header node can be inserted in any type of linked list either ***one-way or two-way*** linked list. Shown in figure blow:



A Two-Way Header Linked List

A Two-Way Circular Header Linked List

A *two-way circular header* linked list shown in figure blow:



A Two-Way Circular Header Linked List

Operation Performed On Header Linked List

Algorithm: Traverses a circular header linked list

Step 1: If **begin→Next =Begin Then**

Printf: "Circular header linked List is Empty"

Exit

[End If]

Step 2: Set **Pointer=Begin→Next**

Step3: Repeat Step 4 and 5 While **Pointer !=Begi**n

Step 4: Process **Pointer→Info**

Step 5: Set **Pointer =Pointer→ Next**

[End loop]

Step 6: Exit

Applications Of The linked List

- To represent the Polynomials
- To represent sparse matrices
- To implement other data structures like Tree, Graph, Stack, queue etc.

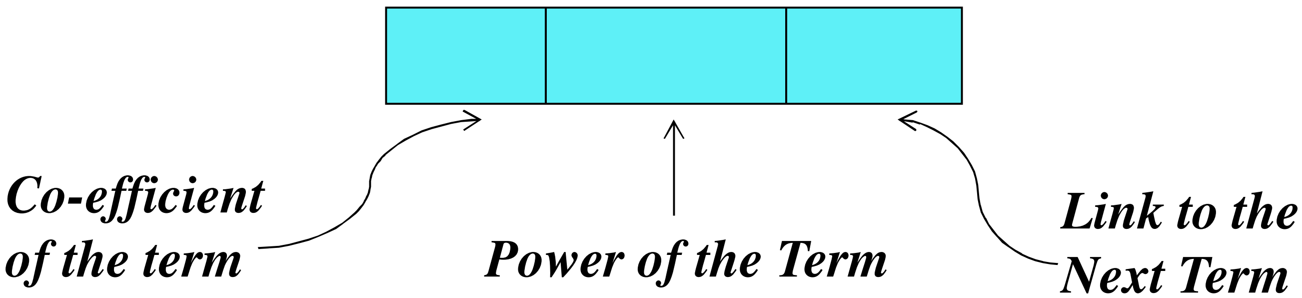
Representation of Polynomials

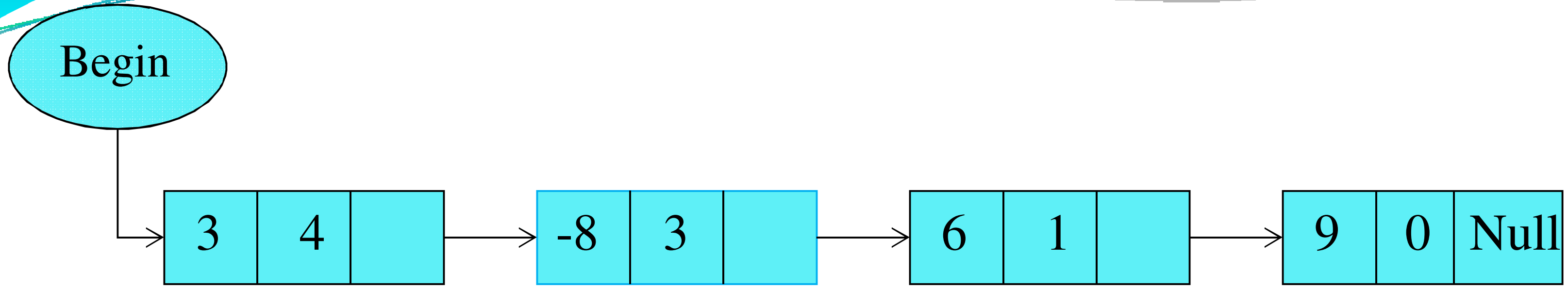
A polynomials are frequently used in both mathematical as well as scientific applications. In Mathematics, various operations are performed on polynomials, e.g. addition of two polynomials, subtraction of a polynomial form other , multiplication and division etc.
A polynomial is mathematical equation of type:

$$P1 : 3x^4 - 8x^3 + 6x + 9$$

$$P2 : 3x^3 - 4x^2 + 6x + 9$$

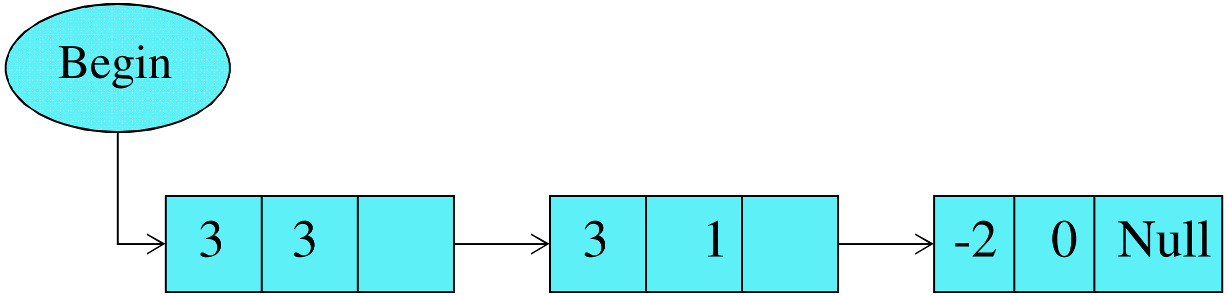
$$Subtraction(P1 - P2) : 3x^4 - 11x^3 + 4x^2 + 3x + 11$$





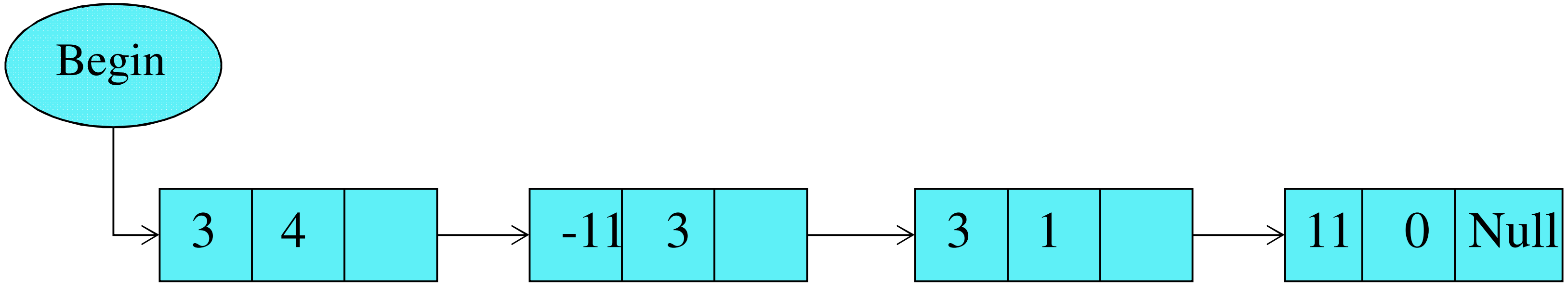
P1

Linked List Representation of the polynomials 'p1'



P2

Linked List Representation of the polynomials 'p2'



Linked List Representation of the Resultant polynomials 'p'

Algorithm :Subtraction of a polynomial

Step 1: If **P1=Null OR P2=Null** then

printf:" One or both n the polynomial are **Null**"

EXIT

[End if]

Step 2: If **Free =Null** Then

Printf" No free space available"

Exit

[End if]

Step 3: **Set P=Null**

Step 4: Allocate memory to node **New**

(Set **New=Free** And **Free =free→ Next**)

Step 5: Set **Pointer1=P1** And **Pointer2=p2**

Step 6: If **Pointer1→Pow=pointer 2→Pow** Then

Set **New→Coeff=Pointer 1→Coeff – Pointer 2→Coeff**

Set **New→Pow=Pointer 1→Pow**

Set **Pointer 1=Pointer 1→Next**

Set **Pointer 2=Pointer 2→Next**

[END IF]

Step7: If **Pointer 1→Pow>Pointer 2→Pow then**

Set **New→Coeff=Pointer 1→Coeff**

Set **New→Pow=Pointer 1→Pow**

Set **Pointer 1 =Pointer 1→Next**

Else

Set **New→Coeff=(- Pointer 2→Coeff)**

Set **New→Pow=Pointer 1→Pow**

Set **Pointer 2 =Pointer 2→Next**

[End if]

Step 8: Set **New→Next=Null**

Set **Pointer =New**

Set **P=New**

Step 9 Repeat Step10 to 13 While **Pointer1!=Null AND Pointer2!= Nul**

Step 10: If **Free =Null** Then

Printf:"No available space"

Exit

[End If]

step 11: Allocate memory to node **New**

(**set New=Free** And **Free=Free → Next**)

step12: if **Pointer1 →Pow=Pointer2 Pow** Then

Set **New →Coeff=Pointer1 →Coeff -Pointer2→Coeff**

Set **New →Pow =Pointer1 → Pow**

Set **Pointer1 =Pointer1→ Next**

Set **Pointer2=Pointer2→ Next**

Else If **Pointer1→ Pow>Pointer2 →Pow** Then

Set **New→ Coeff=Pointer1 → coeff**

Set **New→Pow=pointer1 → Pow**

Set **Pointer1=Pointer1 → Next**

Else

Set **New→ Coeff=-(Pointer2→Coeff)**

Set **New→ Pow=pointer2→Pow**

Set Pointer2=Pointer2 → Next

[End If]

step 13: Set **New→ Next=Null**

Set **Pointer→Next=New** And **Pointer=New**

[End Loop}

Step14: If **Pointer1=Null** then

repeat Step **a to g** while **Pointer2!=Null**

a if **free = Null** then

printf:" Not enough space"

Exit

[End if]

b Allocate Memory to Node **New**

(Set **New =Free And free=Free→Next**)

c **New→ Coeff=-(Pointer2 →Coeff)**

d **New →Pow=Pointer2 → Pow**

e **Pointer2=Pointer2 → Next**

f Set **New → Next=Null**

g Set **Pointer → Next=New** And **Pointer=New**

[End Loop]

Else

Repeat Step **a to g** while **Pointer1!=Null**

a if **free =Null** then

printf:" Not enough space"

Exit

[End if]

b Allocate Memory to Node **New**

(Set **New =Free** And **Free=Free→ Next**)

c **New → Coeff=-(Pointer2 → Coeff)**

d **New → Pow=Pointer2 →Next**

e **Pointer2=Pointer2 → Next**

f Set **New→ Next=Null**

g Set **Pointer→ Next=New** And **Pointer=New**

[End Loop]

[End if]

Step 15: Exit

Storage of Sparse Array

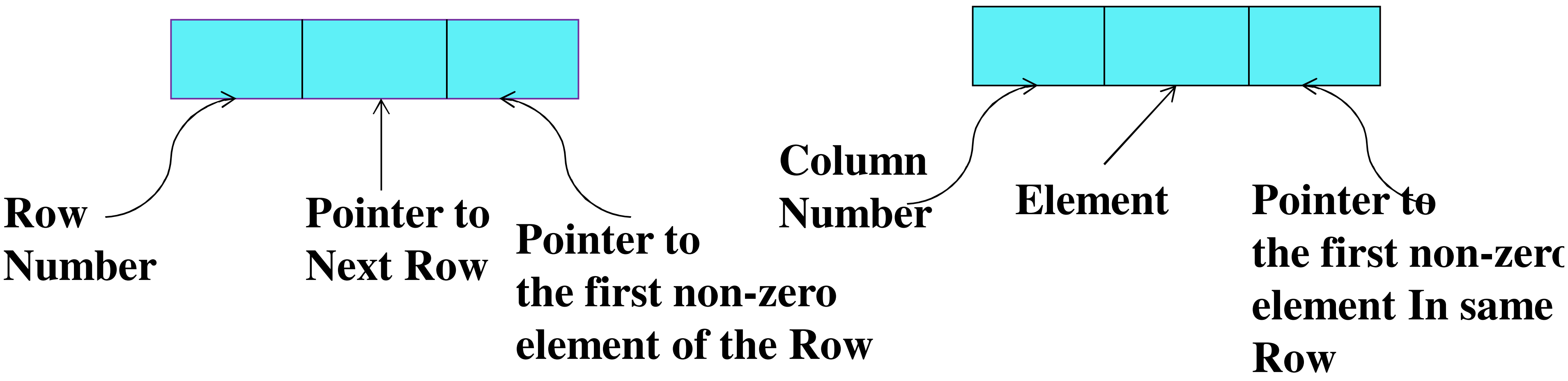
A matrices are two dimensional arrays in which elements are arranged into row and columns, a matrix of order **r** * **c** is collection of **r** * **c** elements which are arranged in **r** rows and **c** columns that is called Sparse Array.

The main problem in the array representation of sparse array is that, it requires a lot of data movement while insertion and deletion of elements. This data movement can be avoided if linked representation is used to store the sparse array. Consider a two dimensional array **A** of order **5*5** as shown below**:**

$$A = \begin{bmatrix} 4 & 0 & 0 & 2 & 0 \\ 0 & 0 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 6 \\ 7 & 3 & 0 & 0 & 0 \end{bmatrix}$$

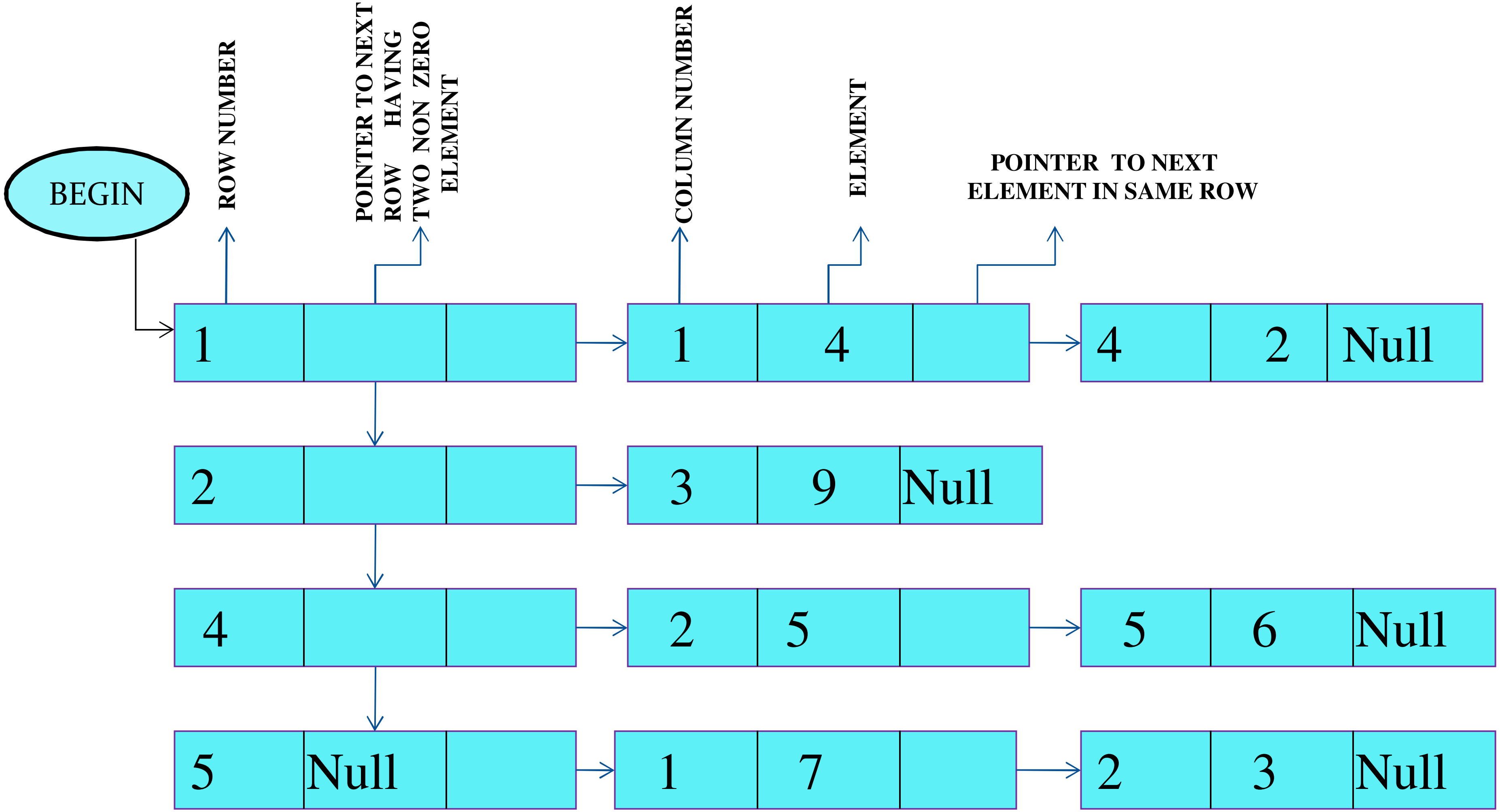
A Sparse Matrix A of order 5*5

Representing a *two dimensional array*, the structure of the nodes in the linked list will be as shown below:



Structure of Nodes used for representing the sparse Matrix

LINKED LIST REPRESENTATION OF SPARSE MATRIX 'A'



Implementing Other Data Structures

Linked lists are frequently used to implement various linear and non-linear data structure like stack, queue, tree and graph. The use of linked lists to implement these data structure will be discussed in the subsequent chapters.