Unit-4

“Linked List”

UNIT – 4 SYLLABUS

Linked List: Introduction of Linked Lists, Realization of linked list using dynamic memory management, operations, Linked List as ADT, Introduction to Static and Dynamic Memory Allocation,

Types of Linked List: singly linked, linear and Circular Linked Lists, Doubly Linked List, Doubly Circular Linked List, Primitive Operations on Linked List-
Create, Traverse, Search, Insert, Delete, Sort, Concatenate.

Polynomial Manipulations-Polynomial
addition. Generalized Linked List (GLL) concept,

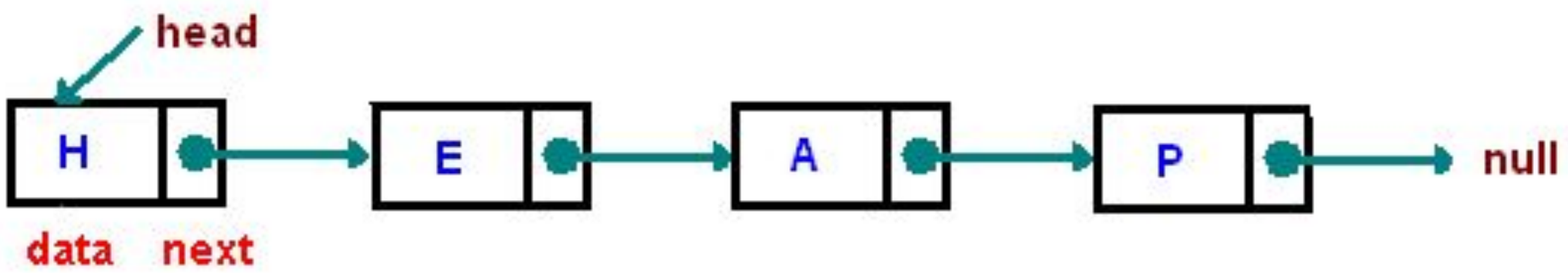
Representation of Polynomial using GLL.

INTRODUCTION OF LINKED LIST

“ Linked List is a very commonly used linear data structure which consists of group of **nodes** in a sequence.

Each node holds its own **data** and the **address of the next node** hence forming a chain like structure.”

Linked Lists are used to create trees and graphs.



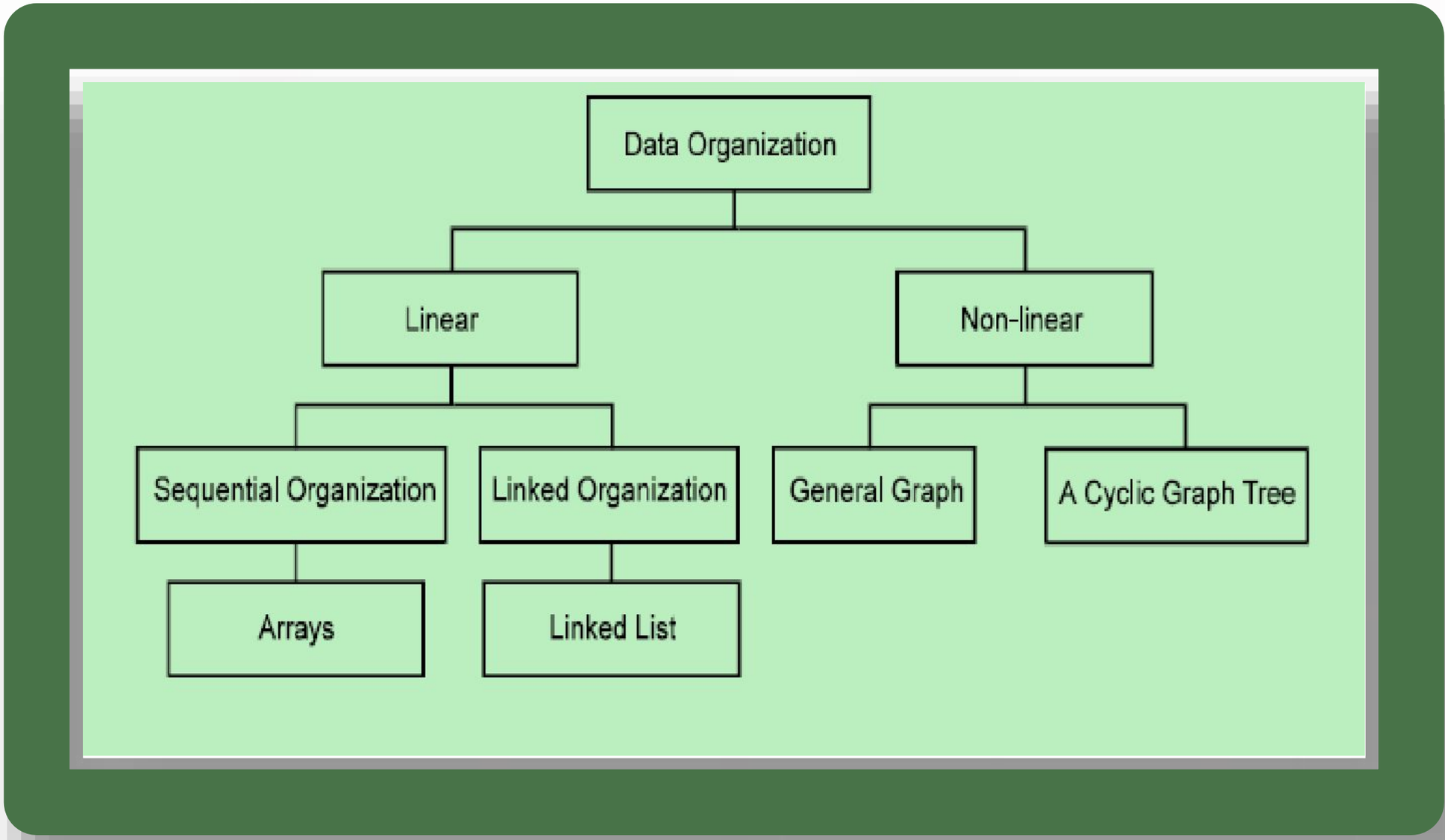


Fig: 1 Data Organization

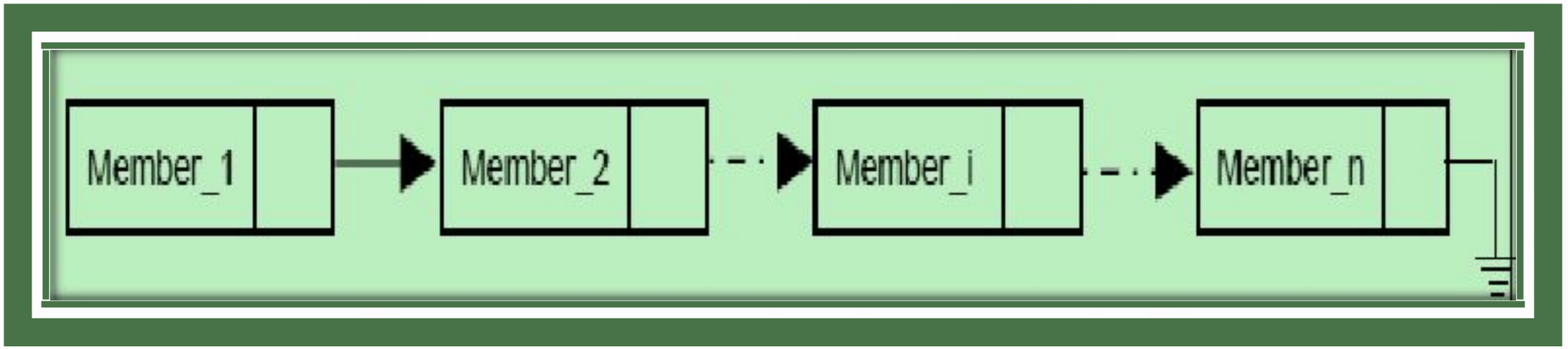


Fig: 2 (a): A linked list of nelements

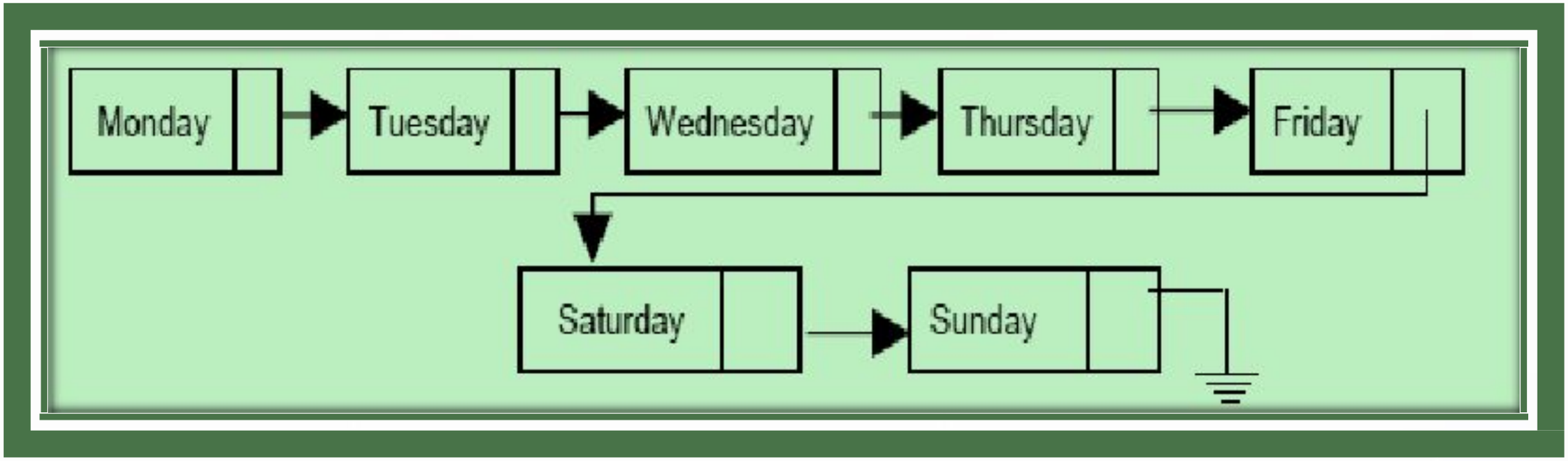


Fig: 2 (b) : A linked list ofweekdays

Advantages of Linked Lists

- They are a dynamic in nature which allocates the memory when required.
- Insertion and deletion operations can be easily implemented.
- Stacks and queues can be easily executed.
- Linked List reduces the access time.

Disadvantages of Linked Lists

- The memory is wasted as pointers require extra memory for storage.
- No element can be accessed randomly; it has to access each node sequentially.
- Reverse Traversing is difficult in linked list.

Applications of Linked Lists

1. Linked lists are used to implement stacks, queues, graphs, etc.

Linked lists let you insertelements atthe beginning and end of the list.

In Linked Lists we don't need to know the size in advance.

TERMINOLOGIES OF LINKED LIST

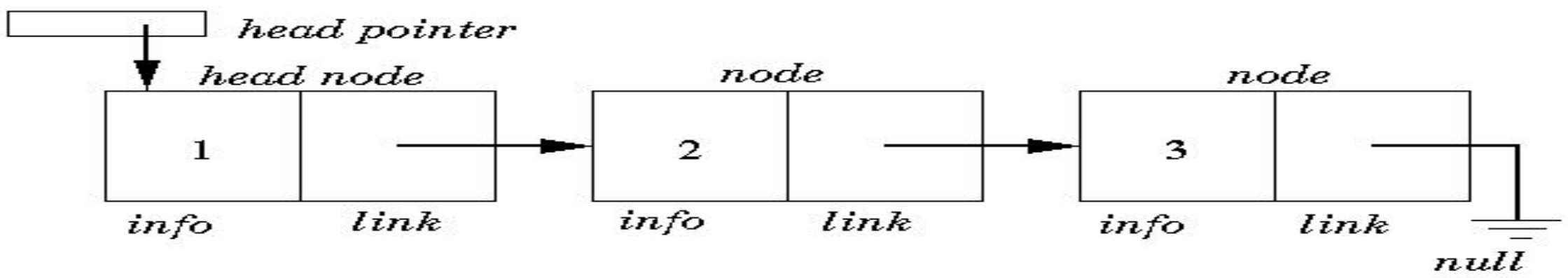
Node an item in a linked list. Each node contains a piece of list data and the location of the next node (item).

Link (of a node) the location of the next node.

head node first node in a linked list

head pointer points to the head node.

Null pointer there is no need in it (linked list is empty)



A Linked List

DIFFERENCE BETWEEN SEQUENTIAL AND LINKED ORGANIZATION

BASIS FOR COMPARISON	ARRAY	LINKED LIST
Basic	It is a consistent set of a fixed number of data items.	It is an ordered set comprising a variable number of data items.
Size	Specified during declaration.	No need to specify; grow and shrink during execution.
Storage Allocation	Element location is allocated during compile time.	Element position is assigned during run time.
Order of the elements	Stored consecutively	Stored randomly
Accessing the element	Direct or randomly accessed, i.e., Specify the array index or subscript.	Sequentially accessed, i.e., Traverse starting from the first node in the list by the pointer.
Insertion and deletion of element	Slow relatively as shifting is required.	Easier, fast and efficient.
Searching	Binary search and linear search	linear search
Memory	less	More

LINKED LIST PRIMITIVE OPERATION

Basic Operations

Following are the basic operations supported by a list.

Insertion − Adds an element at the beginning of the list.

Deletion − Deletes an element at the beginning of the list.

Display − Displays the complete list.

Search − Searches an element using the given key.

Delete − Deletes an element using the given key.

DYNAMIC MEMORY MANAGEMENT

" Dynamic Memory Allocation means memory which can be allocated or deallocated as per requirement at run time"

Importance of memory allocation :

- no need to initially occupy large amount of memory.
- Memory can be allocated or deallocated as per need.
- It avoid wastage of memory.
- We can free the memory by de-allocating it using free( ).

FUNCTION DYNAMIC MEMORY ALLOCATION

Function	Use of Function
malloc()	Allocates requested size of bytes and returns a pointer first byte of allocated space
calloc()	Allocates space for an array elements, initializes to zero and then returns a pointer to memory
free()	deallocate the previously allocated space
realloc()	Change the size of previously allocated space

COMPARE Malloc( ) AND Calloc( ) MEMORY

malloc()	calloc()
Malloc() function will create a single block of memory of size specified by the user.	Calloc() function can assign multiple blocks of memory for a variable.
Malloc function contains garbage value.	The memory block allocated by a calloc function is always initialized to zero.
Number of arguments is 2.	Number of argument is 1.
Calloc is slower than malloc.	Malloc is faster than calloc.
It is not secure as compare to calloc.	It is secure to use compared to malloc.
Time efficiency is higher than calloc().	Time efficiency is lower than malloc().
Malloc() function returns only starting address and does not make it zero.	Before allocating the address, Calloc() function returns the starting address and make it zero.
It does not perform initializes of memory.	It performs memory initialization.

Syntax:
malloc(size)

Ex:
malloc(size of (int))

Syntax:
calloc(number, size)

Representation of Linked list

Realization stands for Representation of Linked List

A linked list can be represented in two ways :

1. Dynamic representation of linked list
2. Static representation of linked list

Representation of Linked list

Dynamic Representation of linked list :

☞ **Importance of dynamic memory allocation**

- No need to initially occupy large amount of memory.
- Memory can be allocated as well as de-allocated as per necessity.
- It avoids the memory shortage as well as memory wastage.

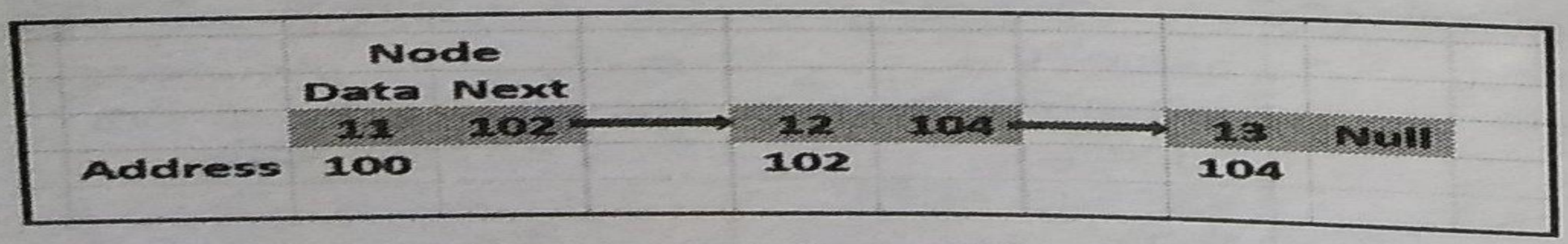


Fig. 3.6.2 : Representation of linear linked list

- In Fig. 3.6.2 we can observe that a linked list is made up of number of nodes. Every node contains two parts : data and next.

Representation of Linked list

Static Representation of linked list :

- Let LIST is linear linked list. It needs two linear arrays for memory representation. Let these linear arrays are INFO and LINK.
- INFO[K] contains the information part and LINK[K] contains the next pointer field of node K.
- A variable START is used to store the location of the beginning of the LIST and NULL is used as next pointer sentinel which indicates the end of LIST. It is shown in Fig. 3.6.3.

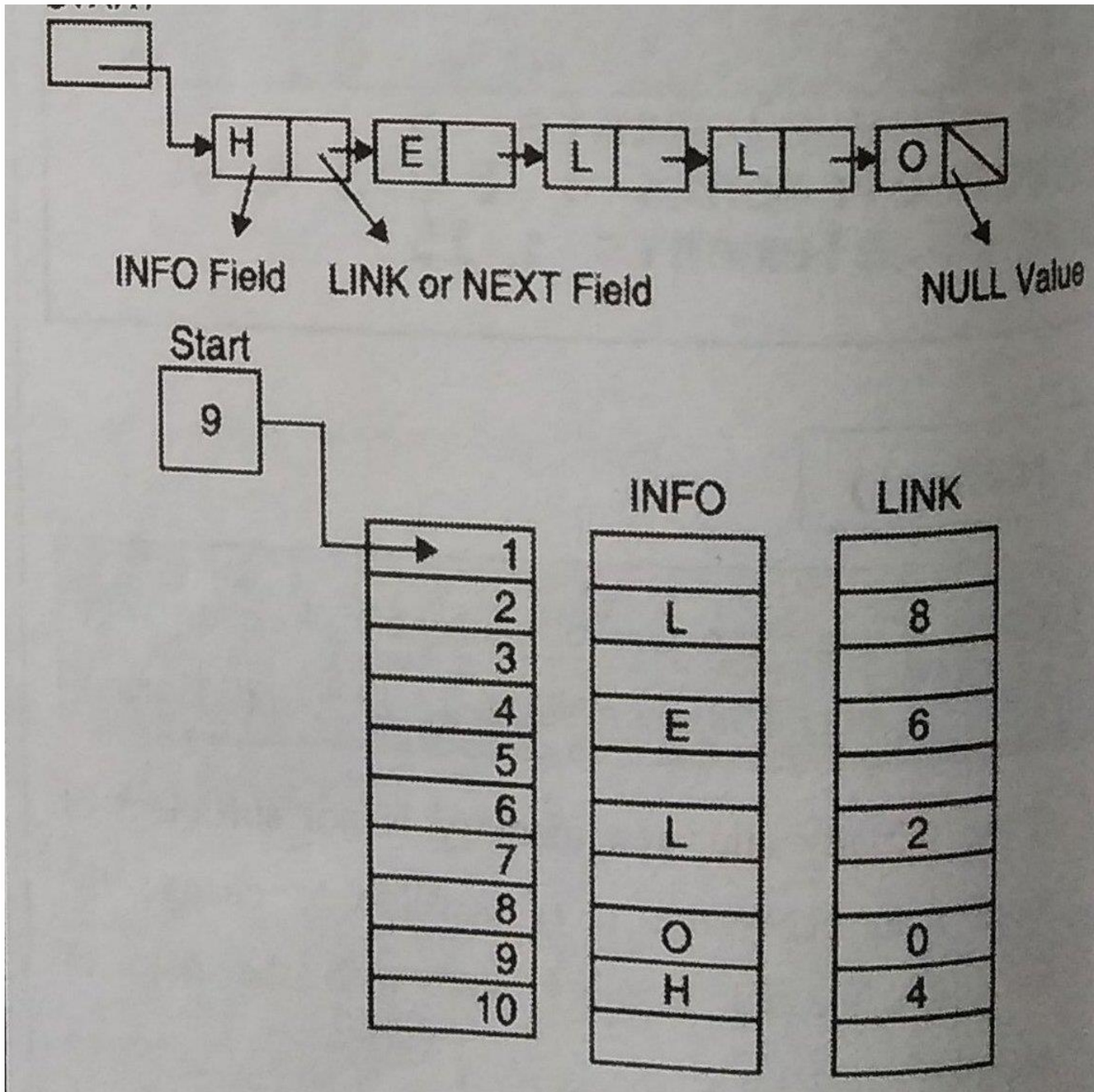


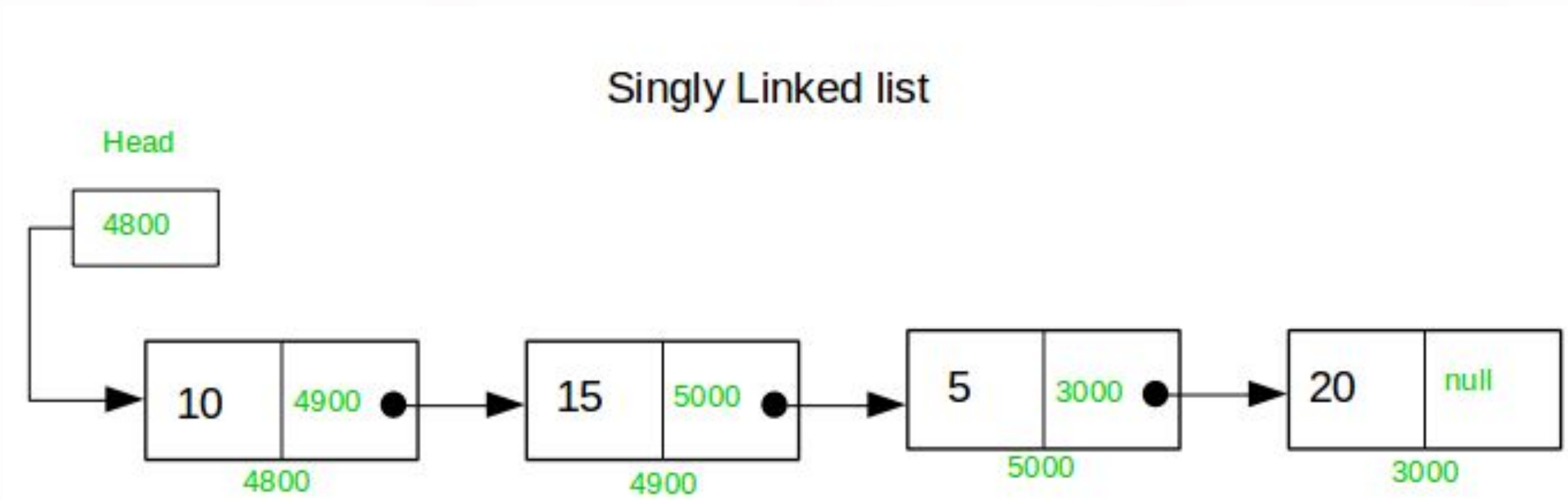
Fig. 3.6.3

TYPES OF LINKED LIST

1. Linear/Singly Linked List
2. Doubly Linked List
3. Circular Linked List
4. Doubly Circular Linked List

LINEAR/SINGLY LINKED LIST

" A linked list in which every node has one link field, to provide information about where the next node of list is, is called as ***singly linked list "***



LINEAR/SINGLY LINKED LIST

Singly linked list is a basic linked list type. Singly linked list is a collection of nodes linked together in a sequential way where each node of singly linked list contains a data field and an address field which contains the reference of the next node. Singly linked list can contain multiple data fields but should contain at least single address field pointing to its connected next node.

ADVANTAGES OF SINGLY LINKED LIST

1. Singly linked list is probably the most easiest data structure to implement.
2. Insertion and deletion of element can be done easily.
3. Insertion and deletion of elements doesn't requires movement of all elements when compared to an array.
4. Requires less memory when compared to doubly, circular or doubly circular linked list.
5. Can allocate or deallocate memory easily when required during its execution.
6. It is one of most efficient data structure to implement when traversing in one direction is required.

DISADVANTAGES OF SINGLY LINKED LIST

1. It uses more memory when compared to an array.
2. Since elements are not stored sequentially hence requires more time to access each elements of list.
3. Traversing in reverse is not possible in case of Singly linked list when compared to Doubly linked list.
4. Requires O(n) time on appending a new node to end. Which is relatively very high when compared to array or other linked list.

SINGLY LINKED LIST OPERATION

1. Creation
2. Insertion
3. Deletion
4. Searching
5. Display

C++ Program to Implement Singly Linked List

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
using namespace std;

/*  Node Declaration  */
struct node
{
    int data;
    struct node *next;
}node;
```

Operations on Linked Lists

Insert a new item

At the head of the list, or
At the tail of the list, or
Inside the list, in some designated position

Search for an item in the list

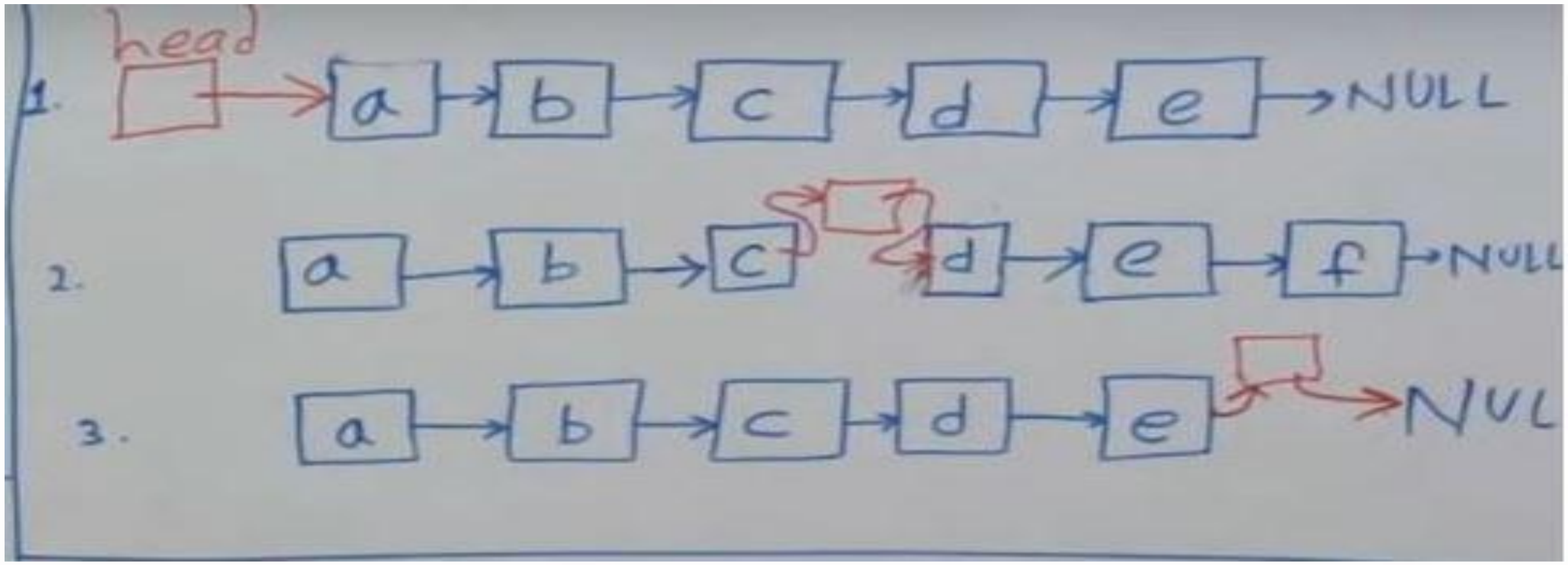
The item can be specified by position, or by some value

Delete an item from the list

Search for and locate the item, then remove the item, and finally adjust the surrounding pointers

size( );
isEmpty( )

Insert a node



Insert– At the Head

- The link value in the new item = old head_ptr
- The new value of head_ptr = newPtr

Insert – At the Tail

- The link value in the new item = NULL
- The link value of the old last item = newPtr

Insert – inside the List

- The link-value in the new item = link-value of prev item
- The new link-value of prev item = newPtr

Delete a node

Delete – the Head Item

- The new value of head_ptr = link-value of the old head item
- The old head item is deleted and its memory returned

Delete – the Tail Item

- New value of tail_ptr = link-value of the 3rd last item
- New link-value of new last item = **NULL**.

Delete – an inside Item

- New link-value of the item located before the deleted one = **the link-value of the deleted item**

Delete a node from singly linked list

Pseudo code:

```
node *delete( node *head, char d) {
 node *p, *q;
 q= haed;
 p= head     next;
if (q      data==d)
     {→
       head=p;
       delete(q);
     }
else
    {
      while (p → data!=d)
      {
        p= p →  next;
       q=q →next;
     }
if (p → next==Null)
   {
       q → next=Null;
        delete(p);
   }
else
   {
     q →  next = p →next;
        delete(p);
    }
}
 return head;
}
```

Time of the Operations

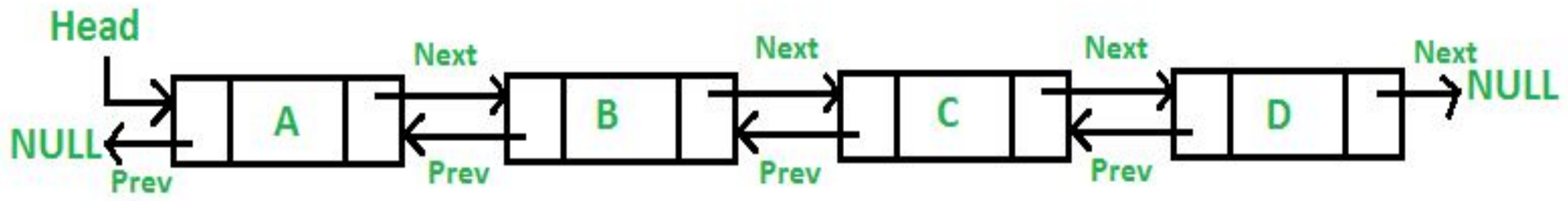
- **Time to search() is O(L) where L is the relative location of the desired item in the List.**
- **In the worst case. The time is O(n).**
- **In the average case it is O(N/2)=O(n).**
- **Time for remove() is dominated by the time for search and is thus O(n).**
- **Time for insert at head or at tail is O(1)(if tail pointer is maintained).**
- **Time for insert at other positions is dominated by search time and thus O(n).**
- **Time for size() is O(n), and time for isEmpty() is O(1)**

size() and isEmpty()

- **We need to scan the items in the list from the head_ptr to the last item marked by its link-value being NULL**
- **Count the number of items in the scan, and return the count. This is the size().**
- **Alternatively, keep a counter of the number of item, which gets updated after each insert/delete. The function size( ) returns that counter**
- **If head_ptr is NULL, isEmpty() returns true; else, it returns false.**

DOUBLY LINKED LIST

- *In **doubly linked list**, each node has two link fields to store information about who is the next and also about who is ahead of the node*
- *Hence each node has knowledge of its **successor and also its predecessor.***
- *In doubly linked list, from every node the list can be*



DOUBLY LINKED LIST

Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward and backward easily as compared to Single Linked List. Following are the important terms to understand the concept of doubly linked list.

Link − Each link of a linked list can store a data called an element.

Next − Eachlink of a linked list contains a link to the next link called Next.

Prev − Eachlink of a linked list contains a link to the previous link called Prev.

LinkedList − A Linked List contains the connection link to the first link called First and to the last link called Last.

BASIC OPERATION OF DLL

Following are the basic operations supported by a list.

Insertion − Adds an element at the beginning of the list.

Deletion − Deletes an element at the beginning of the list.

Insert Last − Adds an element at the end of the list.

Delete Last − Deletes an element from the end of the list.

Insert After − Adds an element after an item of the list.

Delete − Deletes an element from the list using the key.

Display forward − Displays the complete list in a forward manner.

Display backward − Displays the complete list in a backward manner.

ADVANTAGES OF DOUBLY LINKED LIST

)A DLL can be traversed in both forward and backward direction.

)The delete operation in DLL is more efficient if pointer to the node to be deleted is given.

3) We can quickly insert a new node before a given node.
In singly linked list, to delete a node, pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using previous pointer.

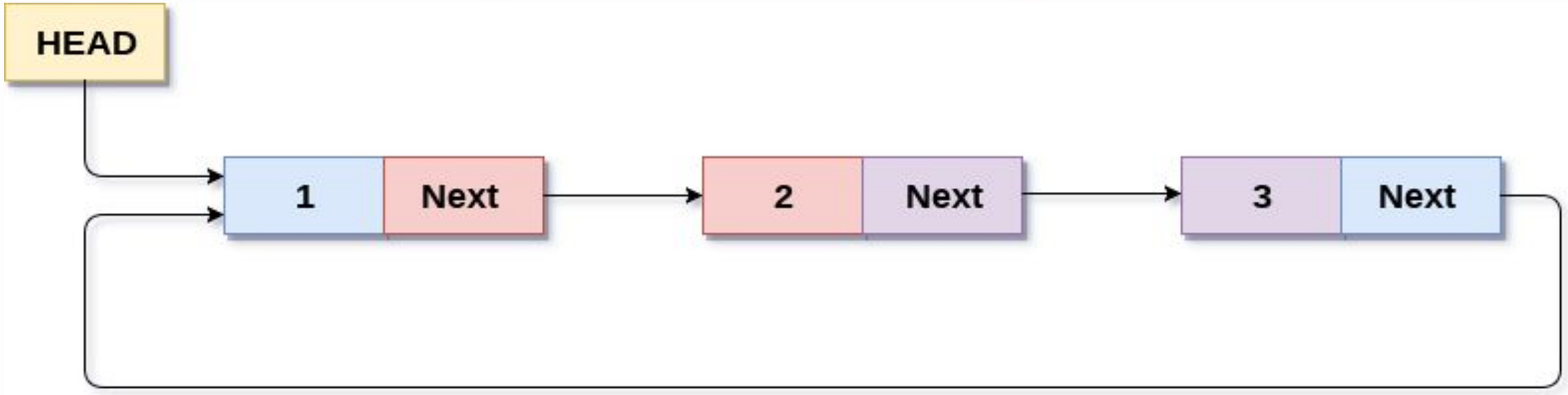
DISADVANTAGES OF DOUBLY LINKED LIST

)Every node of DLL Require extra space for an previous pointer. It is possible to implement DLL with single pointer though

)All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify previous pointers together with next pointers. For example in following functions for insertions at different positions, we need 1 or 2 extra steps to set previous pointer.

SINGLY CIRCULAR LINKED LIST

"Circular Linked List is a variation of Linked list in which the first element points to the last element and the last element points to the first element. Both Singly Linked List and Doubly Linked List can be made into a circular linked list".



Circular Singly Linked List

CIRCULAR LINKED LIST

Circular linked list are mostly used in task maintenance in operating systems. There are many examples where circular linked list are being used in computer science including browser surfing where a record of pages visited in the past by the user, is maintained in the form of circular linked lists and can be accessed again on clicking the previous button.

BASIC OPERATION OF CLL

Following are the important operations supported by a circular list.
insert − Inserts an element at the start of the list.

delete − Deletes an element from the start of the list.

display − Displays the list.

ADVANTAGES OF CIRCULAR LINKED LIST

1. Some problems are **circular** and a **circular** data structure would be more natural when used to represent it.
2. The entire **list** can be traversed starting from any node (traverse means visit every node just once)
3. fewer special cases when coding(all nodes have a node before and after it)

DISADVANTAGES OF CIRCULAR LINKED LIST

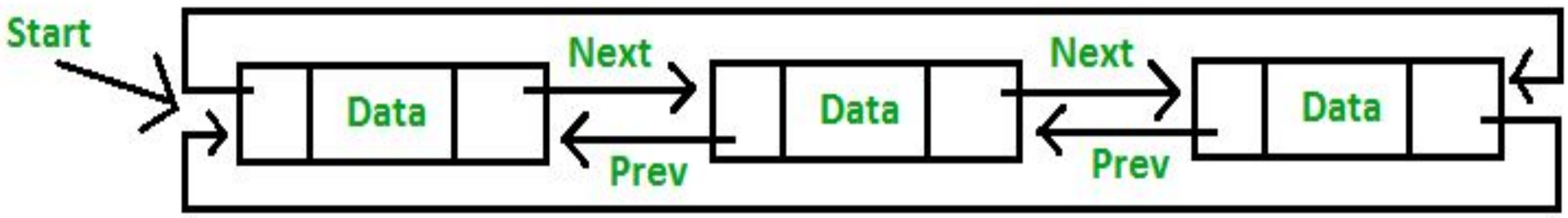
1. **Circular list** are complex as compared to **singly linked lists**.

2. Reversing of **circular list** is a complex as compared to **singly** or doubly **lists**.

3. If not traversed carefully, then we could end up in an infinite loop.

DOUBLY CIRCULAR LINKED LIST

“Circular Doubly Linked List has properties of both doubly linked list and circular linked list in which two consecutive elements are linked or connected by previous and next pointer and the last node points to first node by next pointer and also the first node points to last node by previous pointer”.



BASIC OPERATION OF DOUBLY CIRCULAR LL

SN	Operation	Description
1	Insertion at beginning	Adding a node in circular doubly linked list at the beginning.
2	Insertion at end	Adding a node in circular doubly linked list at the end.
3	Deletion at beginning	Removing a node in circular doubly linked list from beginning.
4	Deletion at end	Removing a node in circular doubly linked list at the end.

ADVANTAGES OF CIRCULAR LINKED LIST

1. List can be traversed from both the directions i.e. from head to tail or from tail to head.
2. Jumping from head to tail or from tail to head is done in constant time O(1).
3. Circular Doubly Linked Lists are used for implementation of advanced data structures like Fibonacci Heap.

DISADVANTAGES OF CIRCULAR LINKED LIST

1. It takes slightly extra memory in each node to accommodate previous pointer.
2. Lots of pointers involved while implementing or doing operations on a list. So, pointers should be handled carefully otherwise data of the list may get lost.

Applications of Circular doubly linked list

Managing songs playlist in media player applications.

Managing shopping cart in online shopping

Parameter	Singly Linked List	Doubly Linked List	Circular Linked List
Node Structure	Node contains two parts: data and link to next node.	Node contains three parts: data and links to previous and next nodes.	Node contains two parts: data and link to next node.
Traversing	Only forward traversing is allowed.	Forward and backward both traversing is allowed.	Only forward traversing is allowed but can jump from last node to first.
Memory	It uses less memory per node (single pointer).	It uses more memory per node(two pointers).	It uses less memory per node (single pointer).
Use	Singly linked list can mostly be used for stacks.	Doubly linked list can be used to implement stacks, heaps, binary trees.	Can be used to implement round robin method.
Complexity	Complexity of Insertion and Deletion at known position is O (n).	Complexity of Insertion and Deletion at known position is O (1).	At known, Position the Complexity of Insertion is O(n) an deletion is O(1).

Primitive operations on linked list

1. Create
2. Insert
3. Delete
4. Search
5. Traverse
6. Concatenate
7. Count
8. Reverse

```
#include <iostream>
#include <string.h>
using namespace std;
class SLL;
class dnode
{
    int div;
int prn;
char name[20];
dnode *next;

friend SLL;
public:
    dnode()
    {
        next = NULL;
        div = prn = 0;
    }
    dnode(int d, int p, char Name[])
        {
            div = d;
            prn = p;
            next = NULL;
            strcpy(name, Name);
        }
};
```

```
class SLL
{
public:
    dnode *head;
    dnode *insert;
    dnode *end;
    void create();
    void print();
    void insertMember();
    void deleteMember();
    void count();
    void mergeList();
    void recursionPrint(dnode *temp);
```

```
    SLL()
      {
        head = NULL;
        insert = NULL;
        end = NULL;
      }
} list1;
```

```
void SLL::create()
{
    int n, d, p;
    char Name[20];

    cout << "Enter the number of students: ";
    cin >> n;

    cout << "\nEnter the name, division and PRN of President: \n";
    cin >> Name >> d >> p;
    head = new dnode(d, p, Name);
    end = head;

    cout << "\nEnter the name, division and PRN of members: \n";
    for (int i = 1; i < n - 1; i++)
    {
        cin >> Name >> d >> p;
        end->next = new dnode(d, p, Name);
        end = end->next;
    }
    insert = end;
```

```
cout << "\nEnter the name, division and PRN of SECRETARY: \n";
    cin >> Name >> d >> p;
    end->next = new dnode(d, p, Name);
    end = end->next;
    end->next = NULL;
}
void SLL::insertMember()
{
    int d, p;
    char Name[20];
      cout << "\nEnter the name, division and PRN of member to INSERT: \n";
    cin >> Name >> d >> p;
    insert->next = new dnode(d, p, Name);
    insert = insert->next;
    insert->next = end;
}
```

```
void SLL::deleteMember()
{
    int searchPRN;
    dnode *remove;
      cout << "Enter the PRN of member to delete: ";
    cin >> searchPRN;
if (searchPRN == head->prn)
{
    remove = head;
    head = head->next;
    delete remove;
}
```

```
 else
    {
for (dnode *temp = head; temp != insert; temp = temp->next)
  {
   if(temp->next->prn == searchPRN)
    {
     if (temp->next == insert)
     insert = temp;
     remove = temp->next;
     temp->next = temp->next->next;
            delete remove;

  if (temp == insert)
            break;
       }
    }
}
```

```
 if (searchPRN == end->prn)
    {
       remove = end;
       insert->next = NULL;
       end = insert;
       delete remove;
    }
}

void SLL::count()
{
    int count = 0;
     for (dnode *temp = head; temp != NULL; temp = temp->next)
        count++;
    cout << "\nTotal no. of students : " << count;
}
```

```
void SLL::print()
{
   cout << "\n\nNAME\tDIV\tPRN\n";
   for (dnode *temp = head; temp != NULL; temp = temp->next)
      cout << temp->name << "\t" << temp->div << "\t" << temp->prn << "\n";
   cout << "\nPresident is: " << head->name;
   cout << "\nSecretary is: " << end->name << "\n";
}
```

```
void SLL::mergeList()
{
    SLL list2;
    cout << "\nEnter the contents for second
list: \n";
    list2.create();
    list1.end->next = list2.head;
    list1.end = list2.end;
}
```

```
void SLL::recursionPrint(dnode *temp)
{
    if (temp == NULL)
        return;
    else
        recursionPrint(temp->next);
        cout << temp->name << "\t" << temp->div
        << "\t" << temp->prn << "\n";
    }
```

```
int main()
{
   int choice;
   do
   {
        cout << "\n1. Create List. \n2. Insert Member \n3. Delete Member \n4. Print List \n5. Merge List
\n6. Reverse List \n7. Count \n8. Exit";
       cout << "\n\nEnter your choice: ";
       cin >> choice;
 switch (choice)
 {
 case 1: list1.create();
            break;
case 2:list1.insertMember();
    break;
case 3: list1.deleteMember();
   break;
```

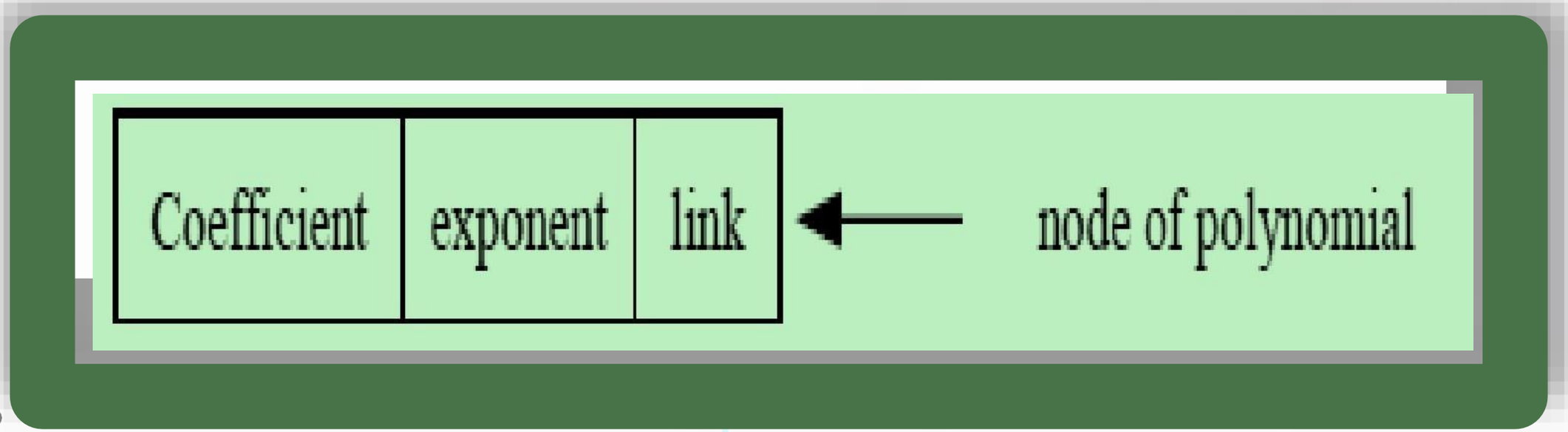
```
case 4:  list1.print();
            break;
case 5:  list1.mergeList();
          break;
 case 6:cout << "\n\nNAME\tDIV\tPRN\n";
          list1.recursionPrint(list1.head);
          break;
case 7: list1.count();
          break;
 case 8: return 0;
default: cout << "Invalid Choice !";
          break;
        }
    } while (choice != 0);
    return 0;
}
```

POLYNOMIAL MANIPULATIONS

POLYNOMIAL MANIPULATIONS

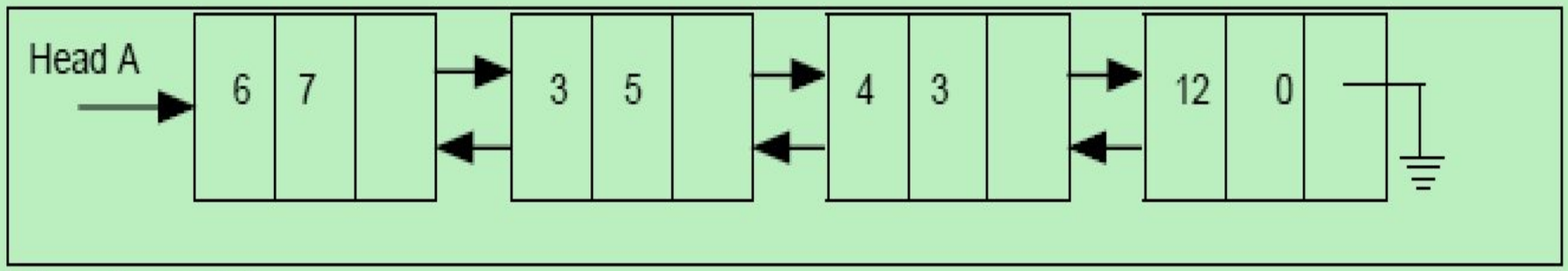
A polynomial p(x) is the expression in variable x which is in the form $(ax^n + bx^{n-1} + ... + jx + k)$, where a, b, c ...., k fall in the category of real numbers and 'n' is non negative integer, which is called the degree of polynomial.

❖*A node will have 3 fields, which represent the coefficient and exponent of a term and a pointer to the next term*



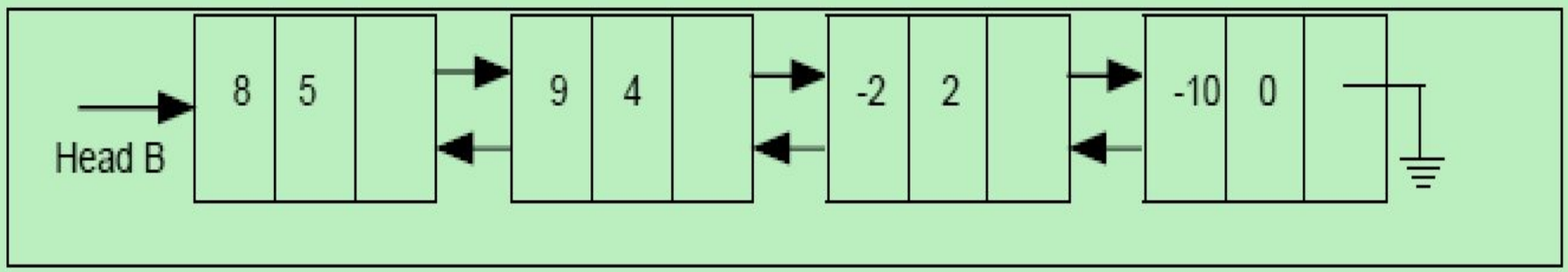
POLYNOMIAL MANIPULATIONS

- *E.x. For instance, the polynomial, say A = $6x^7 + 3x^5 + 4x^3 +$ 12 would be stored as :*



while polynomial

B = $8x^5 + 9x^4 - 2x^2 - 10 = 0$ will be represented as



Operation on Polynomial

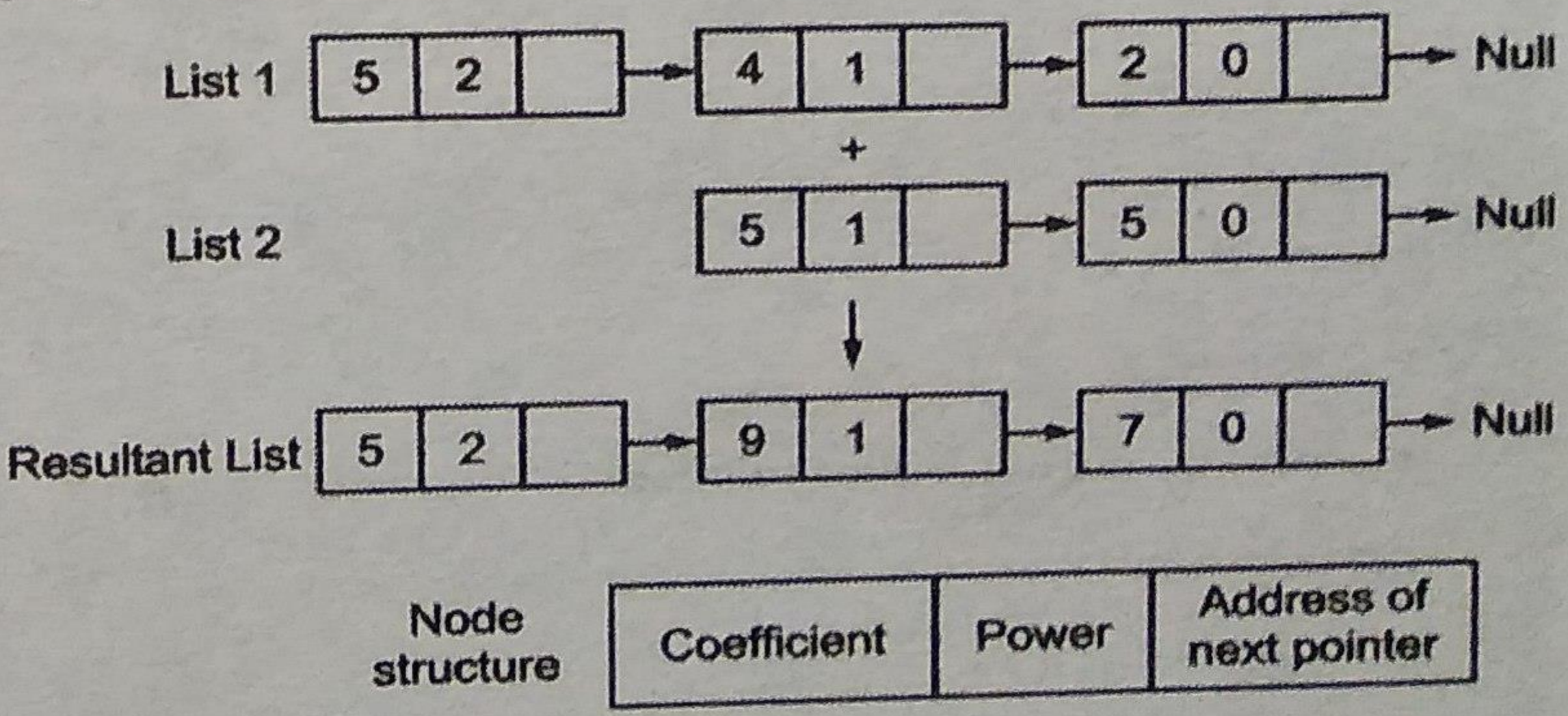
- ***Polynomial evaluation***
- ***Polynomial addition***
- ***Multiplication of two polynomials sparse of matrix using Representation linked list***
- ***Linked list implementation of the stack***
- ***Generalized linked list***

ADDITION OF POLYNOMIAL

POLYNOMIAL

3.13.1 Polynomial Addition

Fig. 3.13.2 illustrates concept of addition of polynomials :



MULTIPLICATION OF POLYNOMIAL

3.13.2 Multiplication of Two Polynomials using Linked List

$3x^2 + 5x + 6$ .....Poly 1(First polynomial)

$6x + 8$ ..... Poly 2(Second polynomial)

1) First multiply poly1 with the first term of poly2.

$18x^3 + 30x^2 + 36x.$

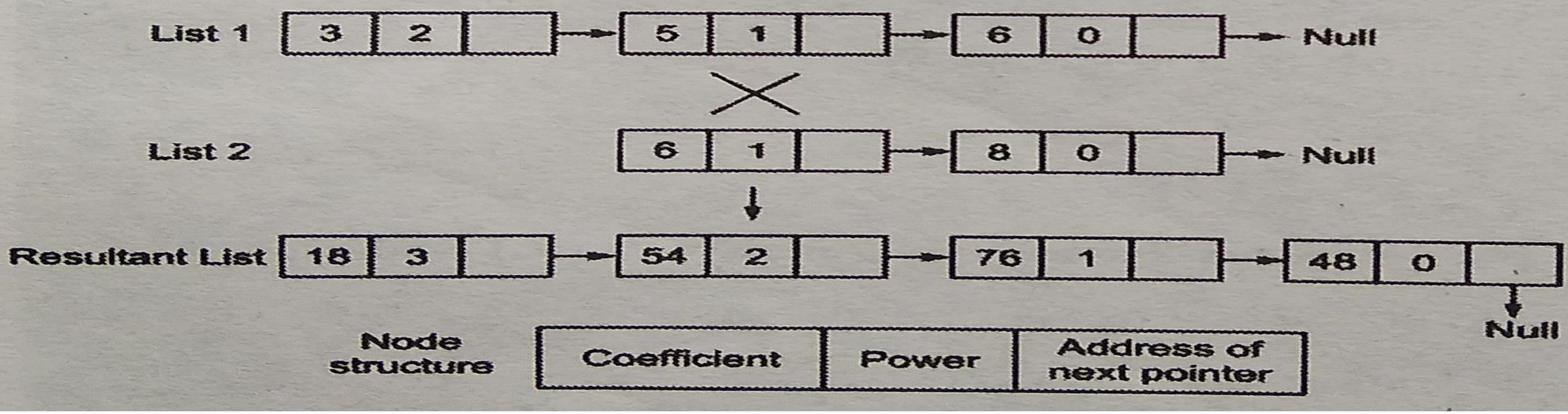
2) Multiply poly1 with the second term of poly2

$24x^2 + 40x + 48.$

3) Finally calculate poly3

$= 18x^3 + 30x^2 + 36x + 24x^2 + 40x + 48.$

$= 18x^3 + 54x^2 + 76x + 48$



GENERALIZED LINKED LIST

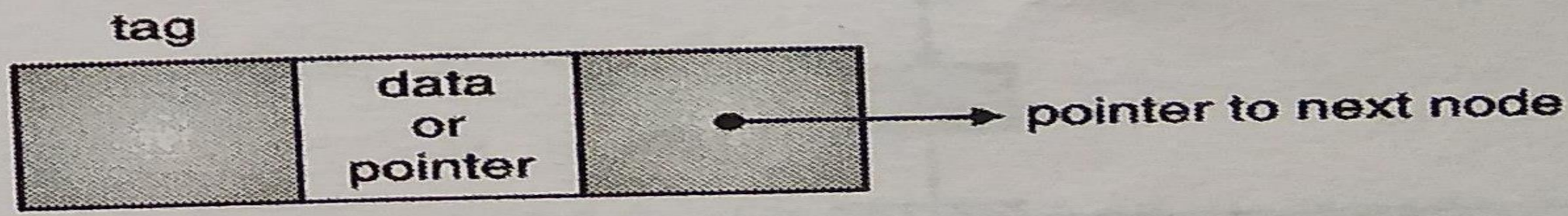
- "A *generalized list*, *A,* is a finite sequence of *n* >= 0 elements, $a_1, a_2, a_3, \ldots a_n$, such that a_i are either atoms or list of atoms.
  Thus $A=(a_1, a_2, a_3, \ldots a_n)$ n is total no. of nodes

☞ **Representation of Generalized List**

– The generalized list can be represented same as of simple linked list, but there is need of an extra field known as *tag* is there in every node to indicate that whether the element is atom or sub list.

– The *tag* field contains value either 0 or 1 . The value 1 indicated that the element is sub list while value 0 indicates that the element is atom.

– When the node represents the sub list, then it stores the address of starting node of the sub list.

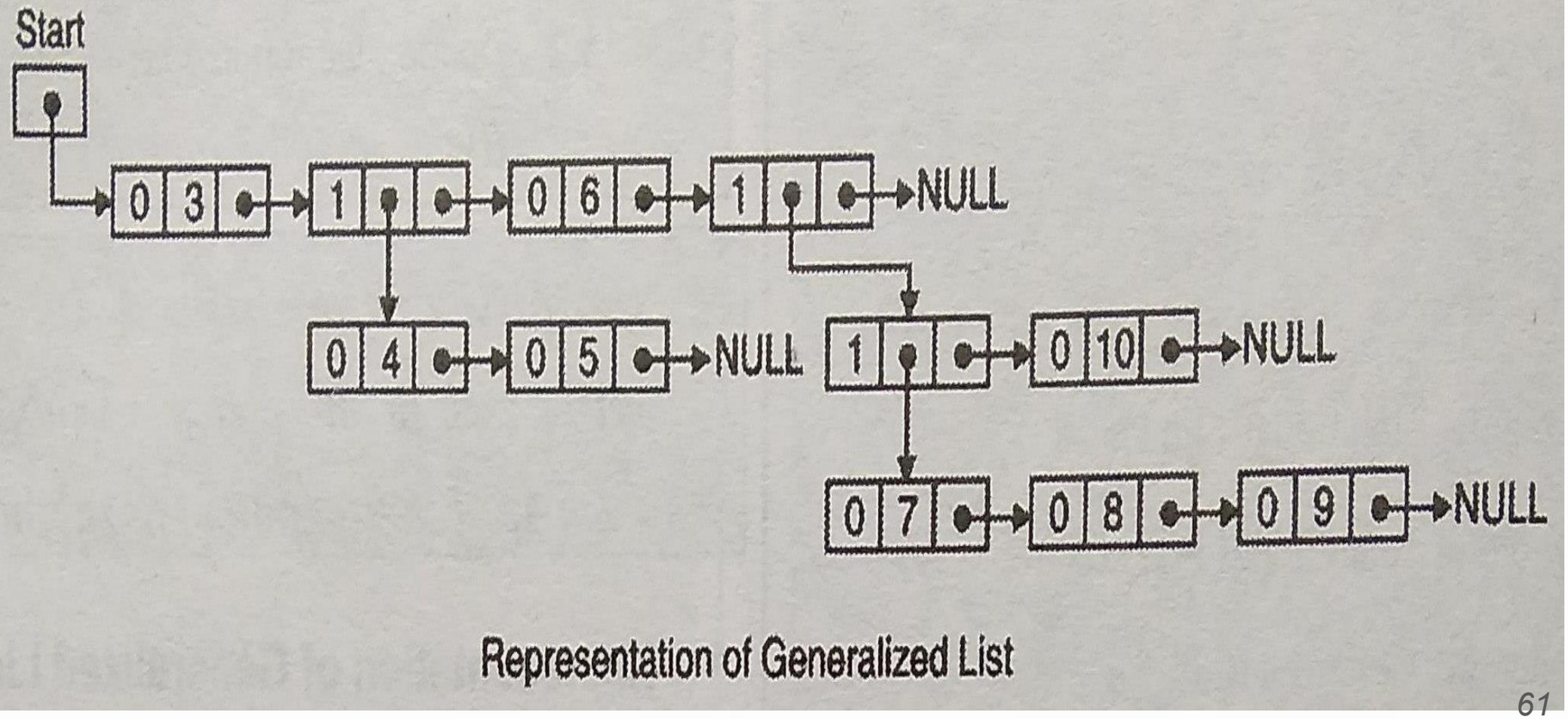


Node of a Generalized List

GENERALIZED LINKED LIST

3.14.1 Representation of Sets using GLL

- Consider a list {3, {4, 5} , 6 , {{7, 8, 9} , 10}}
- Here we can observe that the second and fourth are sub list while first and third are atoms.
- It can be represented as follows :

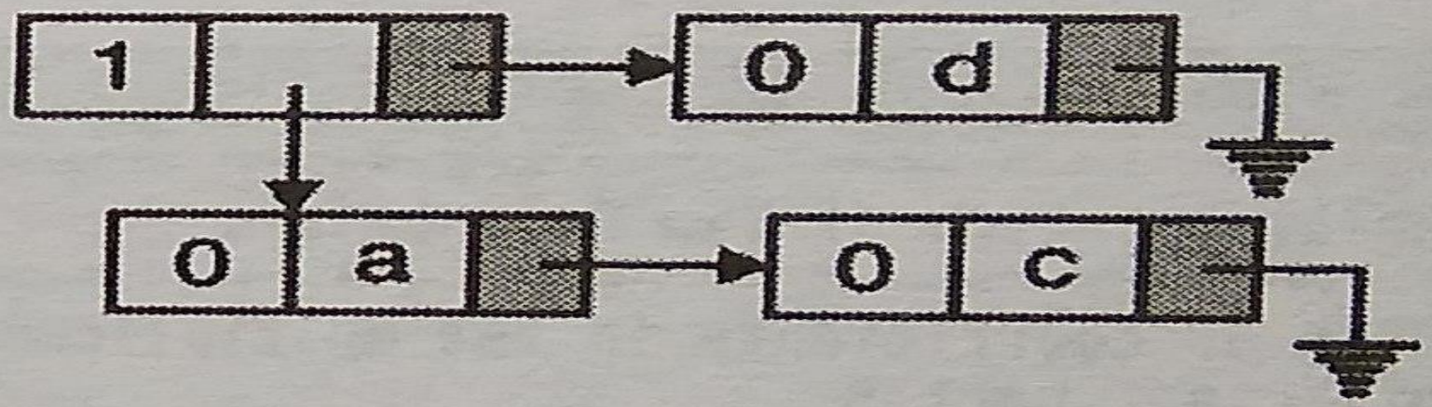


Representation of Generalized List

GENERALIZED LINKED LIST

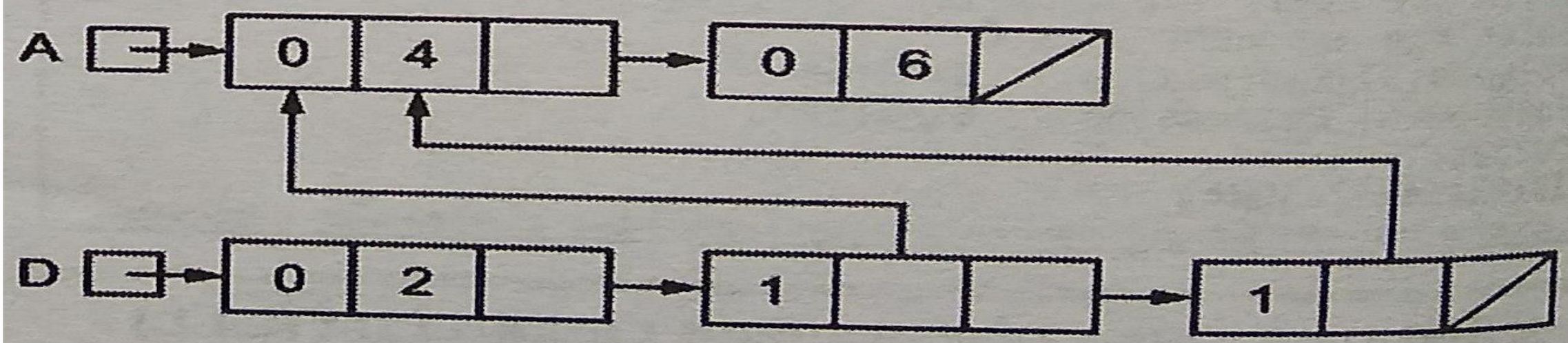
Q. 3.14.5 Draw GLL for following list.

L = ((a, c), d) **(4 Marks)**



Q. 3.14.6 Draw GLL for following list.

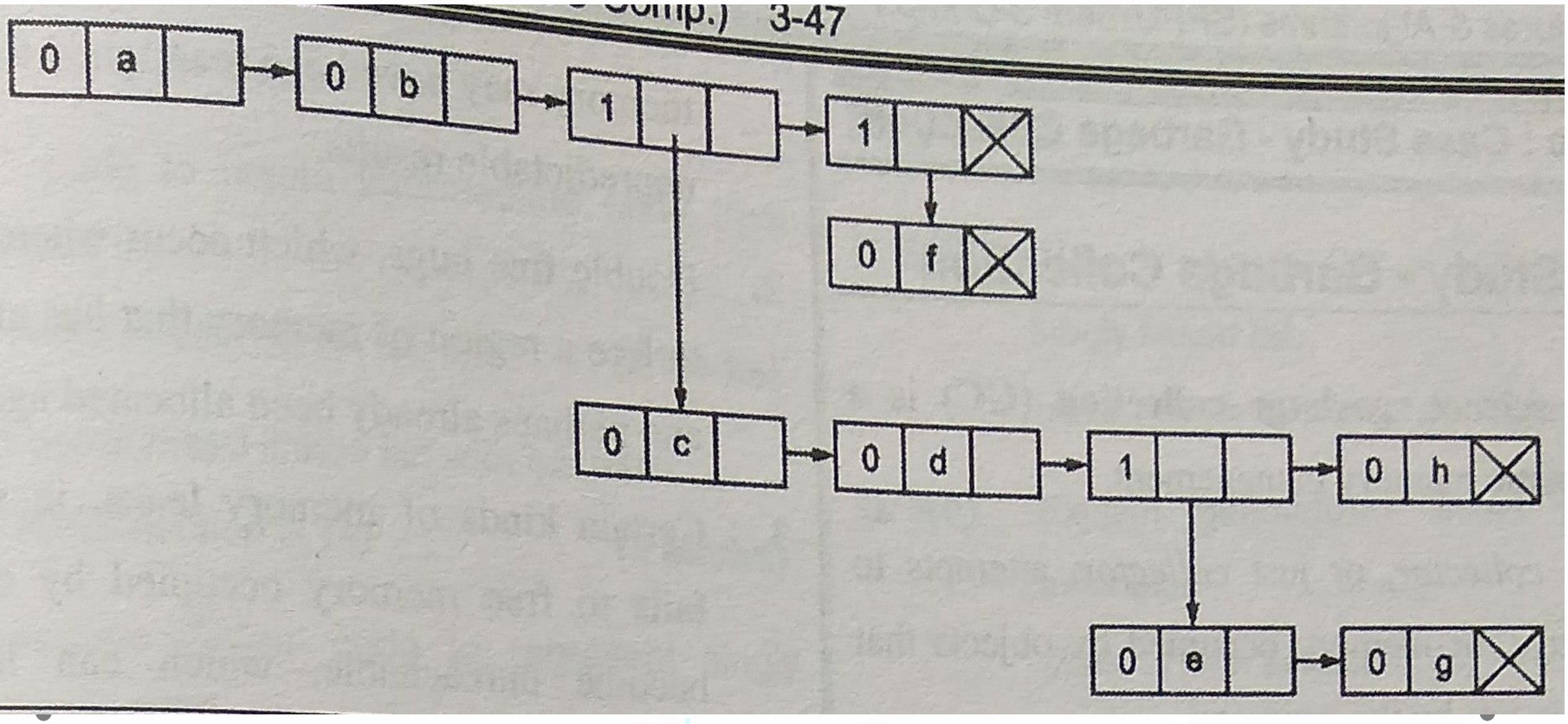
L = (2, (4,6), (4,6)) **(4 Marks)**



GENERALIZED LINKED LIST

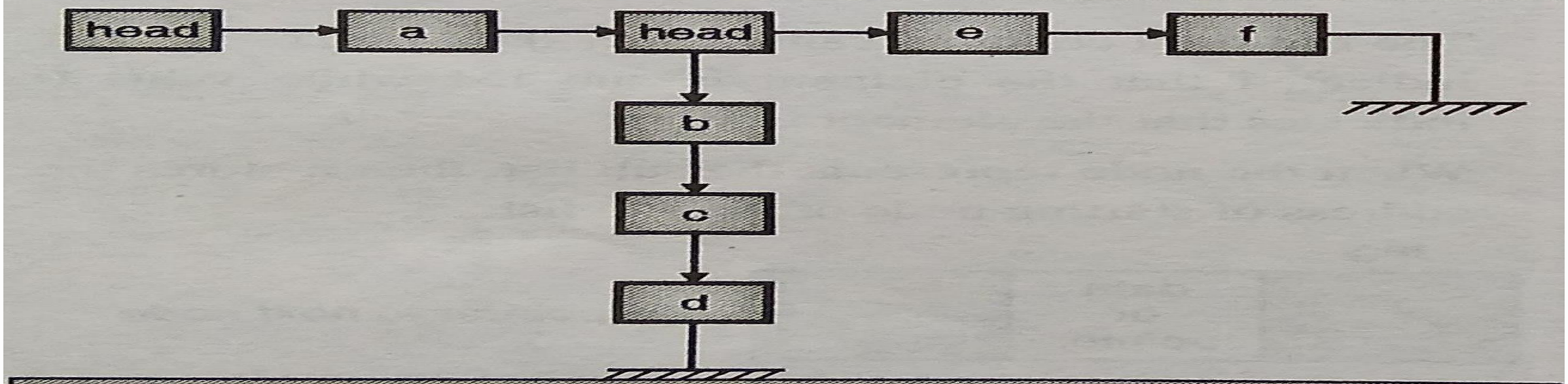
Q. 3.14.7 Represent the following set by using generalized linked List :

(a, b, (c, d, (e, g), h) (f)). **May 17, 3 Marks**

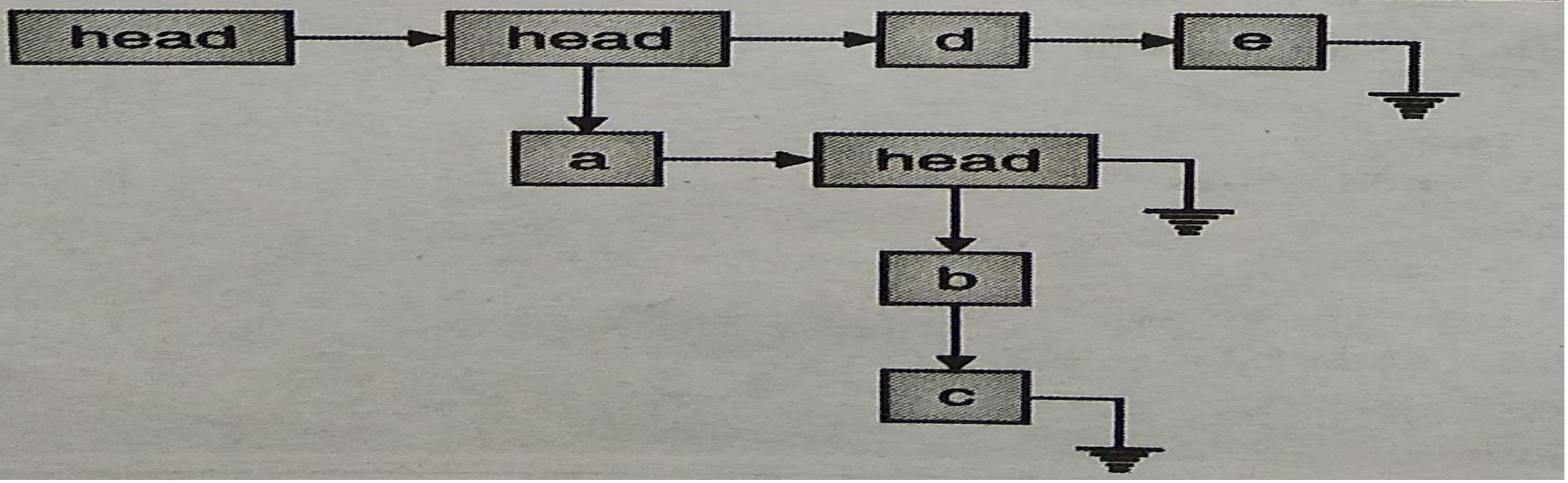


GENERALIZED LINKED LIST

Q. 3.14.3 Represent generalized linked list for the following expression, diagrammatically
G = (a (b, c, d), e, f). **(4 Marks)**



Q. 3.14.4 Write a node structure for generalized linked list. Draw GLL for ((a, (b, c)), d ,e). **(4 Marks)**



Representation of Polynomial using GLL

6.9 Representation of Polynomial using Generalized List

A polynomial can be represented using a generalized list.

A polynomial in a single variable can be represented using a simple list.

$$p(x) = 5x^6 + 9x^2 + 3x + 7$$

Each term of a polynomial can be represented using a 2-tuple, (power, coefficient).

Thus $5x^6 + 9x^2 + 3x + 7$ can be represented as a list of tuples :

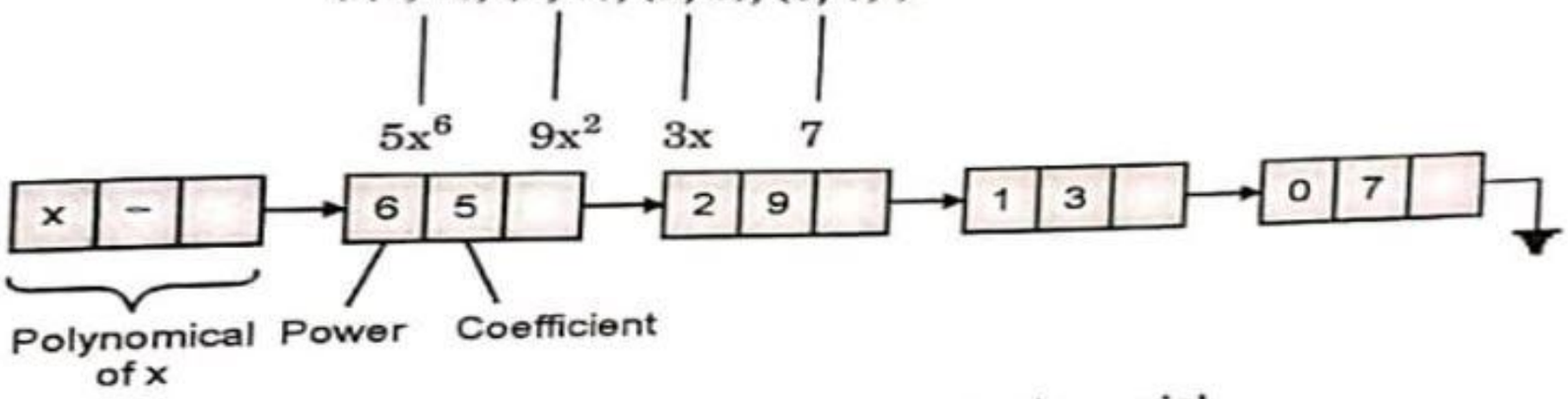


Fig. 6.9.1 : A single variable polynomial

A node consists of three fields :

1. Variable name or power
2. Coefficient
3. Address of the next node

A polynomial in two variables

Let us consider a polynomial as given below : $y^3 (5x^2 + 9x) + y(5x + 6) + (9x^3 + 7)$

In the above polynomial, coefficient of y^3 is another polynomial, $5x^2 + 9x$.

Similarly coefficient of y is $(5x + 6)$ and coefficient of y^0 is $(9x^3 + 7)$.

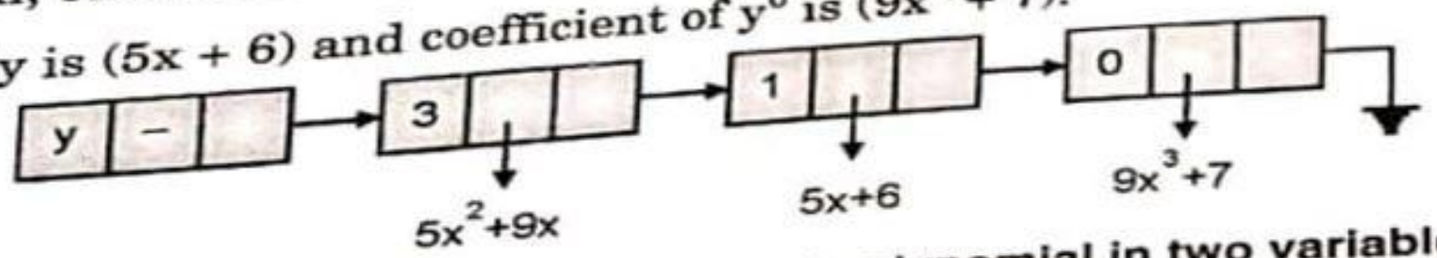
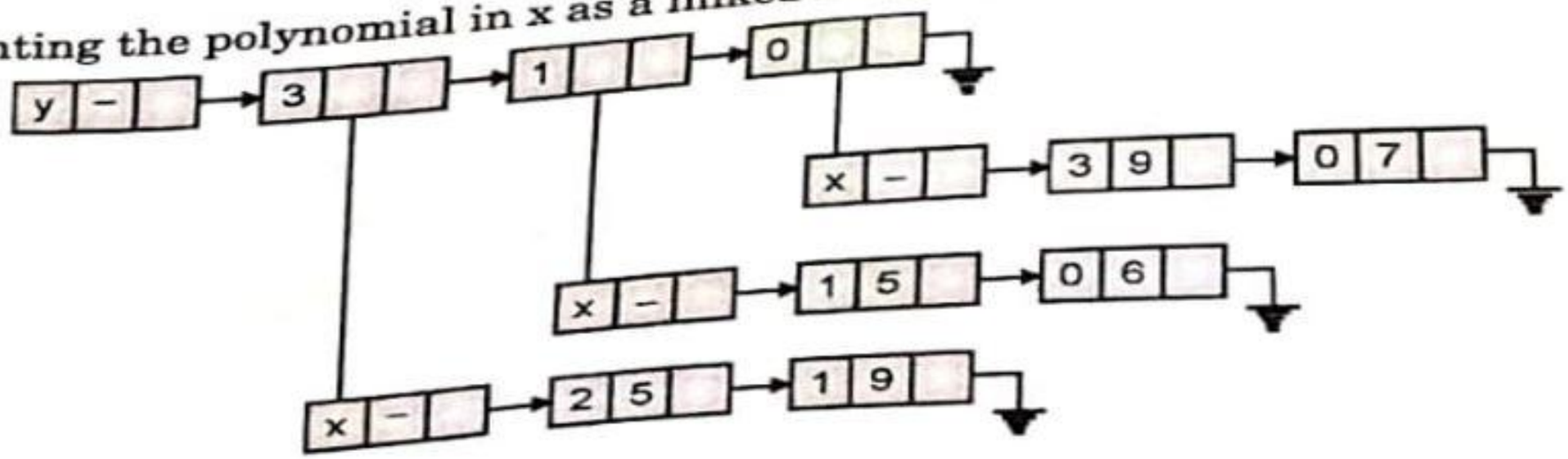


Fig. 6.9.2 : A linked representation of polynomial in two variables. Coefficient field of the node contains address of a polynomial

Further, representing the polynomial in x as a linked list, we get.



GARBAGE COLLECTION

- garbage collection is the process of collecting all unused nodes and returning them to available space.
- This process is carried out in essentially two phases. In the first phase, known as the marking phase, all nodes in use are marked. In the second phase all unmarked nodes are returned to the available space list.
- In this case, the second phase requires only the examination of each node to see whether or not it has been marked.

GARBAGE COLLECTION

- If there aretotal of n nodes, then the second phase of garbage collection can be carried out in $O(n)$ steps.
- In this situation it is only the first or marking phase that is of any interest in designing an algorithm. When variable size nodes are in use, it is desirable to compact memory so that all free nodes form a contiguous block of memory.
- The garbage collection algorithm works in two steps:
- 1) Mark
- 2) Sweep

ADVANTAGES OF GARBAGE COLLECTION

Advantages of Garbage Collection

- GC eliminates the need for the programmer to deallocate memory blocks explicitly
- Garbage collection helps ensure program integrity.
- Garbage collection can also dramatically simplify programs.

DISADVANTAGES OF GARBAGE COLLECTION

Disadvantages of Garbage Collection

- Garbage collection adds an overhead that can affect program performance.
- GC requires extra memory.
- Programmers have less control over the scheduling of CPU time.

THANK YOU !!!!!