LAB SESSION 1: SINGLY LINKED LIST

AIM: To implement polynomial arithmetic and set operations using a single linked list.

PROBLEM DEFINITION:

1. Develop a C program to implement the following:
    a. Accept two polynomials from the user
    b. Add the two polynomials
    c. Multiply the two polynomials
    d. Modify a given polynomial:
        i. Insert term
        ii. Delete term
    e. Accept the polynomials from a file
2. Develop a C program to perform the following set operations using a single linked list.

. Set Union
a. Set Intersection
b. Set Difference

THEORY: A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers as shown in the below image:

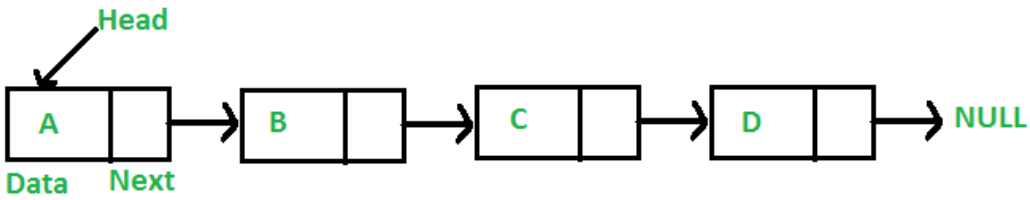

Fig. 1.1: Singly Linked List

In simple words, a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.

Advantages of Linked Lists over arrays:

- Dynamic Array.
- Ease of Insertion/Deletion.

Drawbacks of Linked Lists:

- Random access is not allowed. We have to access elements sequentially starting from the first node(head node). So we cannot do a binary search with linked lists efficiently with its default implementation.
- Extra memory space for a pointer is required with each element of the list.
- Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

Types of Linked Lists:

- Simple Linked List – In this type of linked list, one can move or traverse the linked list in only one direction

- Doubly Linked List – In this type of linked list, one can move or traverse the linked list in both directions (Forward and Backward)
- Circular Linked List – In this type of linked list, the last node of the linked list contains the link of the first/head node of the linked list in its next pointer and the first/head node contains the link of the last node of the linked list in its prev pointer

Representation of Linked Lists:

A linked list is represented by a pointer to the first node of the linked list. The first node is called the head of the linked list. If the linked list is empty, then the value of the head points to NULL.

Each node in a list consists of at least two parts:

- A Data Item (we can store integer, strings, or any type of data).
- Pointer (Or Reference) to the next node (connects one node to another) or An address of another node

In C, we can represent a node using structures.

Basic Operations on a Linked List

Insert into a Linked List

1. Insert at the beginning

1. Allocate memory for new node
2. Store data
3. Change next of new node to point to head
4. Change head to point to recently created node

2. Insert at the End

1. Allocate memory for new node
2. Store data
3. Traverse to last node
4. Change next of last node to recently created node

3. Insert at the Middle

1. Allocate memory and store data for new node
2. Traverse to node just before the required position of new node
3. Change next pointers to include new node in between

Delete from a Linked List

You can delete either from the beginning, end or from a particular position.

1. Delete from beginning

1. Point head to the second node

2. Delete from end

1. Traverse to second last element

2. Change its next pointer to null

3. Delete from middle

1. Traverse to element before the element to be deleted
2. Change next pointers to exclude the node from the chain

Search an Element on a Linked List

You can search an element on a linked list using a loop using the following steps. We are finding item on a linked list.

1. Make head as the current node.
2. Run a loop until the current node is NULL because the last element points to NULL.
3. In each iteration, check if the key of the node is equal to item. If it the key matches the item, return true otherwise return false.