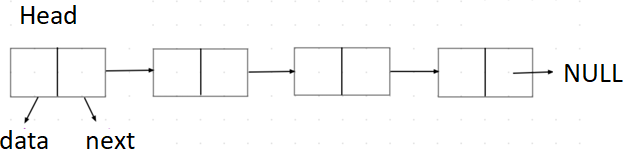
LAB SESSION 1: SINGLY LINKED LIST

**AIM:** To implement polynomial arithmetic and set operations using singly linked list

# PROBLEM DEFINITION:

1. Develop a C program to implement the following:
   1. Accept two polynomials from the user
   2. Add the two polynomials
   3. Multiply the two polynomials
   4. Modify a given polynomial:
      1. Insert term
      2. Delete term
   5. Accept the polynomials from a file
2. Develop a C program to perform the following set operations using singly linked list.
   1. Set Union
   2. Set Intersection
   3. 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:

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:

* Dynamic size: Linked lists do not have a fixed size, so you can add or remove elements as needed, without having to worry about the size of the list.
* Efficient Insertion and Deletion: Inserting or deleting elements in a linked list is fast and efficient, as you only need to modify the reference of the next node.
* Memory Efficiency: Linked lists use only as much memory as they need, so they are more

efficient with memory compared to arrays, which have a fixed size and can waste memory if not all elements are used.

# Disadvantages of Linked Lists::

* Slow Access Time: Accessing elements in a linked list can be slow, as you need to traverse the linked list to find the element you are looking for.
* Cache Inefficiency: Linked lists are cache-inefficient because the memory is not contiguous. This means that when you traverse a linked list, you are not likely to get the data you need in the cache, leading to cache misses and slow performance.
* Extra memory required: Linked lists require an extra pointer for each node, which takes up extra memory.

# Types of Linked Lists:

* Singly Linked List: It is the type of linked list in which every node contains some data and a pointer to the next node of the same data type.
* Doubly Linked List: A doubly linked list or a two-way linked list is a type of linked list that contains a pointer to the next as well as the previous node in sequence.
* Circular Linked List: A circular linked list is that in which the last node contains the pointer to the first node of the list.

# Basic operations on Linked List Insert into Linked List

1. At the front of the linked list
   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. In the middle
   1. Allocate memory for new node
   2. Store data
   3. Traverse to node just before the required position of new node
   4. Change next pointer to include new node in between
3. At the end of the linked list.
   1. Allocate memory for new node
   2. Store data
   3. Traverse to last node
   4. Change next of last node to recently created node

# Delete from a Linked List

1. From the beginning
   1. Point head to second node
2. From the middle
   1. Traverse to element before the element to be deleted
   2. Change next pointer to exclude the node
3. From the end
   1. Traverse to second last element
   2. Change its next pointer to null

# PROGRAM AND OUTPUT:

**Program 1:**

# #include<stdio.h>

# #include<stdlib.h>

# #include<math.h>

# #define NEWLINE printf("\n")

# struct poly

# {

# int coeff;

# int power;

# struct poly\* link;

# };

# //linked list functions

# void display(struct poly\* start);

# struct poly\* addatbeg(struct poly\* start, int power, int coeff);

# struct poly\* addatend(struct poly\* start, int power, int coeff);

# struct poly\* insert(struct poly\* start, int coeff, int power);

# struct poly\* delete(struct poly\* start, int power);

# //impliment functions

# int menu();

# struct poly\* accept(struct poly\* start);

# struct poly\* addpoly(struct poly\* start1, struct poly\* data);

# struct poly\* multiplypoly(struct poly\* start1, struct poly\* start2);

# struct poly\* insert\_term(struct poly\* start);

# struct poly\* delete\_list(struct poly\* start);

# struct poly\* poly\_from\_file();

# int main()

# {

# struct poly \*start1 =NULL, \*start2 = NULL, \*start3 = NULL;

# struct poly\* add = NULL, \*multiply = NULL;

# \_Bool loop = trunc;

# int op;

# while(loop)

# {

# op=menu();

# switch (op)

# {

# case 1: start1 = accept(start1);

# start2 = accept(start2);

# printf("\nPolynomial 1: "); display(start1); NEWLINE;

# printf("\nPolynomial 2: "); display(start2); NEWLINE;

# NEWLINE;

# break;

# case 2:

# add = addpoly(start1,start2);

# printf("\nAddition of Polynomial 1 and 2: ");display(add);NEWLINE;

# break;

# case 3:

# multiply = multiplypoly(start1, start2);

# printf("\nMultiplication of Polynomial 1 and 2: ");display(multiply);NEWLINE;

# break;

# case 4: printf("\nSelect polynomial");

# printf("\n1. Polynomial 1 : "); display(start1);

# printf("\n2. Polynomial 2 : "); display(start2); NEWLINE;

# scanf("%d",&op);

# if(op == 1){

# start1 = insert\_term(start1);

# printf("\nPolynomial after insertion : "); display(start1); NEWLINE;

# break;

# }

# if(op == 2){

# start2 = insert\_term(start2);

# printf("\nPolynomial after insertion : "); display(start2); NEWLINE;

# break;

# }

# else printf("\nINPUT ERROR!\n");

# break;

# case 5: printf("\nSelect polynomial");

# printf("\n1. Polynomial 1 : "); display(start1);

# printf("\n2. Polynomial 2 : "); display(start2); NEWLINE;

# scanf("%d",&op);

# if(op == 1){

# printf("Enter power of term to delete : "); scanf("%d",&op);

# start1 = delete(start1, op);

# printf("\nPolynomial after deletion : "); display(start1); NEWLINE;

# break;

# }

# if(op == 2){

# printf("Enter power of term to delete : "); scanf("%d",&op);

# start2 = delete(start2, op);

# printf("\nPolynomial after deletion : "); display(start2); NEWLINE;

# break;

# }

# else printf("\nINPUT ERROR!\n");

# break;

# case 6: start3 = poly\_from\_file();

# printf("\nPolynomial from FILE : ");

# display(start3);NEWLINE;

# break;

# case 7: loop = 0;

# break;

# default:

# break;

# }

# }

# }

# // displays options and returns the selected one

# int menu(){

# int op;

# printf("\*\*\*\*\*MENU\*\*\*\*\*\n1.Accept two polynomial\n2.Add two polynomial\n");

# printf("3.Multiply two polynomials\n4.Insert a term in Polynomial\n");

# printf("5.Delete a term in Polynomial\n6.Accept polynomial from files\n7. Exit\n");

# printf("Enter your choice: "); scanf("%d",&op);

# return op;

# }

# /\* LinkedList Functions \*/

# // Takes in a pointer and displays the polymial

# void display(struct poly\* start)

# {

# struct poly\* p;

# if(start == NULL){printf("Empty list"); return;}

# p=start;

# while(p!=NULL){

# if(p != start) if(p->coeff > 0) printf("+");

# if(p->power == 0)

# printf("%d",p->coeff);

# else if(p->coeff == 1)

# printf("x^%d",p->power);

# else

# printf("%dx^%d",p->coeff,p->power);

# p=p->link;

# }

# }

# // Takes in a pointer and two other argumments coeff and power

# // Adds a node to the list and returns it address

# struct poly\* addatbeg(struct poly\* start, int coeff, int power)

# {

# struct poly\* tmp;

# tmp = (struct poly\*)malloc(sizeof(struct poly));

# tmp->coeff = coeff;

# tmp->power = power;

# tmp->link = start;

# start = tmp;

# return start;

# }

# // Adds a node to the end of the list by treversing it and return its address

# struct poly\* addatend(struct poly\* start, int coeff, int power)

# {

# struct poly\* p, \* tmp;

# p=start;

# tmp = (struct poly\*)malloc(sizeof(struct poly));

# tmp->coeff = coeff;

# tmp->power = power;

# while(p->link != NULL){

# p = p->link;

# }

# tmp->link = p->link;

# p->link = tmp;

# return start;

# }

# // combination of addatbeg and addatend

# struct poly\* insert(struct poly\* start, int coeff, int power)

# {

# if(!start){

# return addatbeg(start,coeff,power);

# }

# else{

# return addatend(start, coeff, power);

# }

# }

# struct poly\* delete(struct poly\* start, int power)

# {

# struct poly\* p = start, \*tmp;

# if(start == NULL){

# printf("\nEmpty list\n");

# return start;

# }

# if(power == start->power){

# tmp = start;

# start = start->link;

# free(tmp);

# return start;

# }

# while(p->link != NULL){

# if(p->link->power == power){

# tmp = p->link;

# p->link = tmp->link;

# free(tmp);

# return start;

# }

# p = p->link;

# }

# printf("\nTerm was not found!\n");

# }

# /\* Implementation Functions \*/

# // Function to take in a Polynomial

# struct poly\* accept(struct poly\* start)

# {

# int n;

# int coeff, power;

# printf("Enter number of terms: "); scanf("%d",&n);

# if(n==0) return start;

# for(int i=1; i<=n; i++)

# {

# printf("Enter coeff of %d term: ",i+1); scanf("%d",&coeff);

# printf("Enter power of %d term: ",i+1); scanf("%d",&power);

# start = insert(start, coeff, power);

# }

# return start;

# }

# // Function to add two polynomial

# struct poly\* addpoly(struct poly\* start1, struct poly\* start2)

# {

# struct poly \*p1 = start1, \*p2= start2, \*start3 = NULL;

# int coeff, power;

# while(p1 && p2){

# if(p1->power == p2->power){

# coeff = p1->coeff + p2->coeff;

# start3 = insert(start3, coeff, p1->power);

# p1 = p1->link; p2 = p2->link;

# }else if(p1->power > p2->power){

# start3 = insert(start3,p1->coeff, p1->power);

# p1 = p1->link;

# }else{

# start3 = insert(start3,p2->coeff, p2->power);

# p2 = p2->link;

# }

# }

# while(p1){

# start3 = insert(start3,p1->coeff,p1->power);

# p1 = p1->link;

# }

# while(p2){

# start3 = insert(start3, p2->coeff, p2->power);

# p2 = p2->link;

# }

# return start3;

# }

# // Function to multiply two polynomial

# struct poly\* multiplypoly(struct poly\* start1, struct poly\* start2)

# {

# struct poly\* p1 =start1, \*p2 = start1, \*start3 = NULL, \*tmp = NULL;

# int coeff, power;

# while(p1 != NULL){

# p2 = start2;

# tmp = NULL;

# while(p2 != NULL){

# coeff = p1->coeff \* p2->coeff;

# power = p1->power + p2->power;

# tmp = insert(tmp, coeff, power);

# p2 = p2->link;

# }

# start3 = addpoly(start3, tmp);

# p1 = p1->link;

# }

# return start3;

# }

# // Inserts a term in a polynomial

# struct poly\* insert\_term(struct poly\* start)

# {

# int coeff, power;

# struct poly\* p = start, \*tmp = NULL, \*prev = NULL;

# tmp = (struct poly\*)malloc(sizeof(struct poly));

# printf("Enter coeff of term: ");

# scanf("%d",&tmp->coeff);

# printf("Enter power of term: ");

# scanf("%d",&tmp->power);

# prev = p;

# while((p != NULL) && (p->power > tmp->power)){ //loop to make pointer stop at position of insertion

# prev = p;

# p = p->link;

# }

# if(p == start){

# tmp->link = p;

# start = tmp;

# }else if(p->power == tmp->power){

# p->coeff += tmp->coeff;

# }else{

# tmp->link = p;

# prev->link = tmp;

# }

# return start;

# }

# // Deletes the whole list

# struct poly\* delete\_list(struct poly\* start)

# {

# while(start){

# start = delete(start, start->power);

# }

# return start;

# }

# struct poly\* poly\_from\_file()

# {

# #define M\_COEFF  1

# #define M\_POWER  2

# #define M\_NCOEFF 13

# #define M\_NPOWER 23

# struct poly\* start = NULL;

# int coeff = 1, power = 1, tmp = 0, mode = 0;

# FILE \*fp;

# fp = fopen("poly.txt","r");

# char c;

# 

# while((c = fgetc(fp)) != '\n' && c != EOF){

# switch(c)

# {

# case '1':

# case '2':

# case '3':

# case '4':

# case '5':

# case '6':

# case '7':

# case '8':

# case '9':

# case '0':

# tmp = (tmp \* 10) + (c - '0') ;

# break;

# case '+':

# mode = M\_COEFF;

# break;

# case '-':

# if(mode == M\_POWER) mode = M\_NPOWER;

# else mode = M\_NCOEFF;

# break;

# case '^':

# mode = M\_POWER;

# break;

# default : break;

# }

# 

# if(mode == M\_COEFF || mode == M\_NCOEFF ){

# power \*= tmp;

# if(coeff != 0){

# start = insert(start, coeff, power);

# }

# if(mode == M\_NCOEFF)

# coeff = -1;

# else

# coeff = 1;

# tmp = 0;

# mode = 0;

# }

# 

# if(mode == M\_POWER || mode == M\_NPOWER){

# coeff \*= tmp;

# if(mode == M\_NPOWER)

# power = -1;

# else

# power = 1;

# tmp = 0;

# mode = 0;

# }

# }

# return start;

# }

**Contents of FILE : 7x^10+4x^6-5x^3**

**Output:**

\*\*\*\*\*MENU\*\*\*\*\*

1.Accept two polynomial

2.Add two polynomial

3.Multiply two polynomials

4.Insert a term in Polynomial

5.Delete a term in Polynomial

6.Accept polynomial from files

7. Exit

Enter your choice: 1

Enter number of terms: 3 4 3 5 2 -3 0 4 2 5 6 4 1 2 1 0

Enter coeff of 2 term: Enter power of 2 term: Enter coeff of 3 term: Enter power of 3 term: Enter coeff of 4 term: Enter power of 4 term: Enter number of terms: Enter coeff of 2 term: Enter power of 2 term: Enter coeff of 3 term: Enter power of 3 term: Enter coeff of 4 term: Enter power of 4 term: Enter coeff of 5 term: Enter power of 5 term:

Polynomial 1: 4x^3+5x^2-3

Polynomial 2: 2x^5+6x^4+x^2+1

\*\*\*\*\*MENU\*\*\*\*\*

1.Accept two polynomial

2.Add two polynomial

3.Multiply two polynomials

4.Insert a term in Polynomial

5.Delete a term in Polynomial

6.Accept polynomial from files

7. Exit

Enter your choice: 2

Addition of Polynomial 1 and 2: 2x^5+6x^4+4x^3+6x^2-2

\*\*\*\*\*MENU\*\*\*\*\*

1.Accept two polynomial

2.Add two polynomial

3.Multiply two polynomials

4.Insert a term in Polynomial

5.Delete a term in Polynomial

6.Accept polynomial from files

7. Exit

Enter your choice: 3

Multiplication of Polynomial 1 and 2: 8x^8+34x^7+30x^6-2x^5-13x^4+4x^3+2x^2-3

\*\*\*\*\*MENU\*\*\*\*\*

1.Accept two polynomial

2.Add two polynomial

3.Multiply two polynomials

4.Insert a term in Polynomial

5.Delete a term in Polynomial

6.Accept polynomial from files

7. Exit

Enter your choice: 4

Select polynomial

1. Polynomial 1 : 4x^3+5x^2-3

2. Polynomial 2 : 2x^5+6x^4+x^2+1

1

Enter coeff of term: 7

Enter power of term: 1

Polynomial after insertion : 4x^3+5x^2+7x^1-3

\*\*\*\*\*MENU\*\*\*\*\*

1.Accept two polynomial

2.Add two polynomial

3.Multiply two polynomials

4.Insert a term in Polynomial

5.Delete a term in Polynomial

6.Accept polynomial from files

7. Exit

Enter your choice: 5

Select polynomial

1. Polynomial 1 : 4x^3+5x^2+7x^1-3

2. Polynomial 2 : 2x^5+6x^4+x^2+1

2

Enter power of term to delete : 4

Polynomial after deletion : 2x^5+x^2+1

\*\*\*\*\*MENU\*\*\*\*\*

1.Accept two polynomial

2.Add two polynomial

3.Multiply two polynomials

4.Insert a term in Polynomial

5.Delete a term in Polynomial

6.Accept polynomial from files

7. Exit

Enter your choice: 6

Polynomial from FILE : 7x^10+4x^6-5x^3

\*\*\*\*\*MENU\*\*\*\*\*

1.Accept two polynomial

2.Add two polynomial

3.Multiply two polynomials

4.Insert a term in Polynomial

5.Delete a term in Polynomial

6.Accept polynomial from files

7. Exit

Enter your choice: 7

**Program 2:**

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

struct node

{

    int info;

    struct node\* link;

};

void display(struct node\* start);

struct node\* addatbeg(struct node\* start, int data);

struct node\* deletenode(struct node\* start, int data);

struct node\* UNION(struct node\* startA, struct node\* startB);

struct node\* INTERSECTION(struct node\* startA, struct node\* startB);

struct node\* DIFFERENCE(struct node\* start1, struct node\* start2);

int main()

{

    struct node\* startA = NULL , \* startB = NULL, \*startC = NULL;

    int n1, n2;

    int data;

    bool loop=true;

    int option;

    printf("Enter size of set A: "); scanf("%d",&n1);

    printf("Enter the elements of set A: \n");

    for(int i=0 ; i<n1; i++){

        printf("Enter element: "); scanf("%d",&data);

        startA = addatbeg(startA,data);

    }

    printf("Enter size of set B: "); scanf("%d",&n2);

    printf("Enter the elements of set B: \n");

    for(int i=0 ; i<n2; i++){

        printf("Enter element: "); scanf("%d",&data);

        startB = addatbeg(startB,data);

    }

    while(loop){

        printf("\*\*\*\*\*MENU\*\*\*\*\*\n1.Union\n2.Intersection\n3.Difference\n4.Exit\n");

        printf("Enter option: "); scanf("%d",&option);

        switch(option)

        {

            case 1: startC = UNION(startA, startB);

                    display(startC);

                    break;

            case 2: startC = INTERSECTION(startA, startB);

                    display(startC);

                    break;

            case 3: {

                    printf("1. A - B\n2. B - A\nEnter option: ");

                    scanf("%d",&option);

                    switch(option){

                    case 1: startC = DIFFERENCE(startA,startB);

                            display(startC);

                            break;

                    case 2: startC = DIFFERENCE(startB,startA);

                            display(startC);

                            break;

                    default: printf("\nInput error!\n"); break;

                    }

                }

                break;

            case 4: loop = false; break;

            default : printf("\nEnter correct option!\n"); continue;

        }

    }

    return 0;

}

//linked list functions

struct node\* addatbeg(struct node\* start, int data)

{

    struct node\* tmp = (struct node\*)malloc(sizeof(struct node));

    if(tmp == NULL) return start;

    tmp->info = data;

    tmp->link = start;

    start = tmp;

    return start;

}

void display(struct node\* start)

{

    struct node\* p;

    if(start == NULL){

        printf("\nEmpty list!\n");

        return;

    }

    p = start;

    while(p!=NULL){

        printf("%d\t",p->info);

        p = p->link;

    }

    printf("\n");

}

struct node\* deletenode(struct node\* start, int data)

{

    struct node\* p, \*tmp;

    if(start == NULL){

        printf("\nEmpty list!\n");

        return start;

    }

    if(start->info == data){

        tmp = start;

        start = start->link;

        free(tmp);

        return start;

    }

    p = start;

    while(p->link != NULL){

        if(p->info == data){

            tmp = p->link;

            p->link = tmp->link;

            free(tmp);

            return start;

        }

        p = p->link;

    }

    printf("\nData not found!\n");

    return start;

}

//Custom funtions

struct node\* UNION(struct node\* startA, struct node\* startB)

{

    struct node \*p1, \*p2, \*p3, \*startC = NULL;

    bool flag;

    p1 = startA;

    p2 = startB;

    while(p1 != NULL){

        startC = addatbeg(startC, p1->info);

        p1 = p1->link;

    }

    while(p2 != NULL){

        p3 = startC;

        flag = false;

        while(p3 != NULL){

            if(p2->info == p3->info){

                flag = true;

                break;

            }

            p3 = p3->link;

        }

        if(!flag){

            startC = addatbeg(startC, p2->info);

        }

        p2 = p2->link;

    }

    return startC;

}

struct node\* INTERSECTION(struct node\* startA, struct node\* startB)

{

    struct node \*p1, \*p2, \*startC = NULL;

    p1 = startA;

    p2 = startB;

    while(p1 != NULL){

        p2 = startB;

        while(p2 != NULL){

            if(p1->info == p2->info){

                startC = addatbeg(startC, p2->info);

                break;

            }

            p2 = p2->link;

        }

        p1 = p1->link;

    }

    return startC;

}

struct node\* DIFFERENCE(struct node\* start1, struct node\* start2)

{

    struct node\* p1, \*p2, \*start3 = NULL;

    p1 = start1;

    p2 = start2;

    while(p1 != NULL){

        p2 = start2;

        while(p2 != NULL){

            if(p1->info == p2->info){

                break;

            }

            p2 = p2->link;

        }

        if(p2 == NULL){

            start3 = addatbeg(start3,p1->info);

        }

        p1 = p1->link;

    }

    return start3;

}

**Ouptput:**

Enter size of set A: 5

Enter the elements of set A:

Enter element: 1

Enter element: 2

Enter element: 3

Enter element: 4

Enter element: 5

Enter size of set B: 7

Enter the elements of set B:

Enter element: 2

Enter element: 4

Enter element: 5

Enter element: 3

Enter element: 1

Enter element: 8

Enter element: 9

\*\*\*\*\*MENU\*\*\*\*\*

1.Union

2.Intersection

3.Difference

4.Exit

Enter option: 1

8 9 1 2 3 4 5

\*\*\*\*\*MENU\*\*\*\*\*

1.Union

2.Intersection

3.Difference

4.Exit

Enter option: 2

1 2 3 4 5

\*\*\*\*\*MENU\*\*\*\*\*

1.Union

2.Intersection

3.Difference

4.Exit

Enter option: 3

1. A - B

2. B - A

Enter option: 1

Empty list!

\*\*\*\*\*MENU\*\*\*\*\*

1.Union

2.Intersection

3.Difference

4.Exit

Enter option: 3

1. A - B

2. B - A

Enter option: 2

8 9

\*\*\*\*\*MENU\*\*\*\*\*

1.Union

2.Intersection

3.Difference

4.Exit

4

**CONCLUSION:**  The project's implementation of polynomial arithmetic and set operations through a singly linked list highlights the intersection of mathematical concepts and programming. This project showcases the practical application of linked lists and also highlights the importance of data structures in solving real-world problems.