K. J. Somaiya College of Engineering, Mumbai-77
(Autonomous College Affiliated to University of Mumbai)

Batch: B1 **Roll No.: 1711072**

Experiment No. 2

Grade: AA / AB / BB / BC / CC / CD /DD

Title: Linked Lists

Objective: Implementation of polynomial operations (addition, subtraction) using linked list.

Expected Outcome of Experiment:

CO	Outcome
CO2	Use linear and non-linear data structure in domain like compiler construction, DBMS, etc.

Books/ Journals/ Websites referred:

Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein; (CLRS) "Introduction to Algorithms", Third Edition, The MIT Press.

Abstract:-

DATA NODE:

1. pow <integer>
2. coeff <integer>
3. next <node pointer>

OPERATIONS:

1. **Create polynomial**

This function is used to take input for polynomial expressions from the user. It accepts the address of resultant polynomial as parameter and has no return type.

2. **Display polynomial**

This function is used to display the polynomial entered by the user in the proper mathematical format. It accepts the node pointer to the polynomial and has no return type.

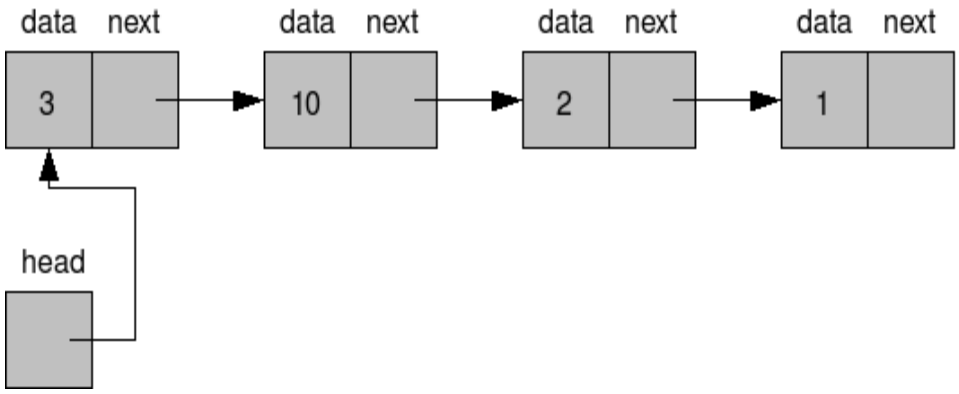
3. **Add polynomials**

This function is used to add two polynomials and then store the resultant in third polynomial (linked list). It accepts the address of third polynomial, and node pointers to first and second polynomial and has no return type.

RELATED THEORY:

- Linked list is a data structure that is free from the aforementioned restrictions. A linked list does not store its elements in consecutive memory locations and the user can add any number of elements to it.
- Unlike an array, a linked list does not allow random access of data. Elements in a linked list can be accessed only in a sequential manner. But like an array, insertions and deletions can be done at any point in the list in a constant time.
- A linked list, in simple terms, is a linear collection of data elements. These data elements are called nodes. A linked list can be perceived as a train or a sequence of nodes in which each node contains one or more data fields and a pointer to the next node.

- Linked list is a data structure which in turn can be used to implement other data structures. Thus, it acts as a building block to implement data structures such as stacks, queues, priority queues, circular queues, double ended queues, etc.
- An example of a linked list of students is represented diagrammatically below:



- The head pointer is a data-less (dummy) pointer of node type which is used to maintain address of first node.
- The left part of the node which contains data may include a simple data type, an array, or a structure.
- The right part of the node contains a pointer to the next node (or address of the next node in sequence). The last node will have no next node connected to it, so it will store a special value called NULL. A NULL pointer denotes the end of the list.
- Since in a linked list, every node contains a pointer to another node which is of the same type, it is also called a self-referential data type.
- In linked lists, nodes can be added as well as deleted one by one. While adding a node, memory occupied by one node is allocated at a time, not more, not less. While deleting a node from the linked list, the memory occupied by that node will be made free.
- Hence, a dynamic data structure (one which can grow or shrink) can be implemented without shortage or waste of memory (These are the two possible problems that may arise while using arrays). Moreover, insertion and deletion can be implemented efficiently by adjusting a couple of links without having the need to shift any element.

- The only disadvantage of linked lists is that it requires a link pointer with every node and hence requires more space. But, this space is negligible in real life applications where the data field may be a structure of 100s of bytes whereas the link pointer will consume just 2 to 4 bytes.

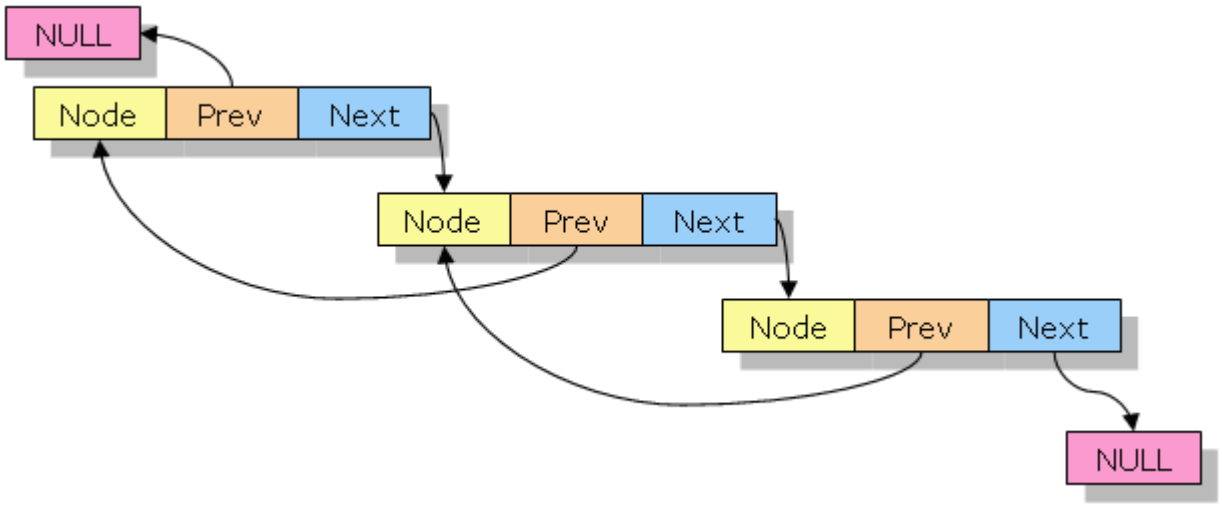
TYPES OF LINKED LISTS:

1. Singly Linked Lists

2. Doubly Linked Lists

3. Circular Singly Linked Lists

4. Circular Doubly Linked Lists



DOUBLY LINKED LIST

Implementation Details:

CODE:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct poly{
  int pow;
  int coeff;
  struct poly *next;
}node;

void create_poly(node **poly){
  int flag, coeff, pow;
  node *temp_poly;
  temp_poly=(node*)malloc(sizeof(node));
  *poly=temp_poly;
  do{
    printf("Enter coefficient: ");
    scanf("%d", &coeff);
    temp_poly->coeff=coeff;
    printf("Enter power: ");
    scanf("%d", &pow);
    temp_poly->pow=pow;
    temp_poly->next=NULL;
    printf("\nEnter more terms? (1/0): ");
    scanf("%d", &flag);
    if(flag){
      temp_poly->next=(node*)malloc(sizeof(node));
      temp_poly=temp_poly->next;
      temp_poly->next=NULL;
    }
  }while(flag);
}
void display(node *poly){
  printf("\nThe polynomial expression is: \n");
  while(poly!=NULL){
    printf("(%dx^%d)", poly->coeff, poly->pow);
    poly=poly->next;
    if(poly!=NULL)
        printf("+");
  }
}
void add_poly(node **poly3, node *poly1, node *poly2){
```

```
  node *temp_poly;
  temp_poly=(node*)malloc(sizeof(node));
  temp_poly->next=NULL;
  *poly3=temp_poly;
  while(poly1!=NULL && poly2!=NULL){
    if(poly1->pow > poly2->pow){
      temp_poly->pow=poly1->pow;
      temp_poly->coeff=poly1->coeff;
      poly1=poly1->next;
    }
    else if(poly2->pow > poly1->pow){
      temp_poly->pow=poly2->pow;
      temp_poly->coeff=poly2->coeff;
      poly2=poly2->next;
    }
    else{
      temp_poly->pow=poly1->pow;
      temp_poly->coeff=poly1->coeff + poly2->coeff;
      poly1=poly1->next;
      poly2=poly2->next;
    }
      temp_poly->next=(node*)malloc(sizeof(node));
      temp_poly=temp_poly->next;
     // temp_poly->next=NULL;
  }
  while(poly1 || poly2){
        if(poly1)
        {
            temp_poly->pow = poly1->pow;
            temp_poly->coeff = poly1->coeff;
            poly1 = poly1->next;
        }
        if(poly2)
        {
            temp_poly->pow = poly2->pow;
            temp_poly->coeff = poly2->coeff;
            poly2 = poly2->next;
        }
        temp_poly->next = (node*)malloc(sizeof(node));
        temp_poly = temp_poly->next;
        //temp_poly->next = NULL;
    }
}
```

```
void main(){
int ch;
do{
  node *poly1, *poly2, *poly3;
  printf("\nCreate first polynomial: \n");
  create_poly(&poly1);
  display(poly1);
  printf("\nCreate second polynomial: \n");
  create_poly(&poly2);
  display(poly2);
  add_poly(&poly3, poly1, poly2);
  display(poly3);
  printf("\nEnter another pair of expressions?(1/0): ");
  scanf("%d", &ch);
}while(ch);
}
```

OUTPUT SCREEN:

```
Create first polynomial:
Enter coefficient:  7
Enter power:  8

Enter more terms? (1/0):  1
Enter coefficient:  9
Enter power:  7

Enter more terms? (1/0):  1
Enter coefficient:  5
Enter power:  4

Enter more terms? (1/0):  1
Enter coefficient:  10
Enter power:  2

Enter more terms? (1/0):  0

The polynomial expression is:
(7x^8)+(9x^7)+(5x^4)+(10x^2)
Create second polynomial:
Enter coefficient:  10
Enter power:  9

Enter more terms? (1/0):  1
Enter coefficient:  10
Enter power:  8

Enter more terms? (1/0):  20
Enter coefficient:  20
Enter power:  5

Enter more terms? (1/0):  1
Enter coefficient:  10
Enter power:  4

Enter more terms? (1/0):  1
Enter coefficient:  -5
Enter power:  2

Enter more terms? (1/0):  0

The polynomial expression is:
(10x^9)+(10x^8)+(20x^5)+(10x^4)+(-5x^2)
The polynomial expression is:
(10x^9)+(17x^8)+(9x^7)+(20x^5)+(15x^4)+(5x^2)
Enter another pair of expressions?(1/0):  0
```

CONCLUSION:

The program ran successfully as we were able to add two polynomials correctly, verified using numerous edge cases and test cases.