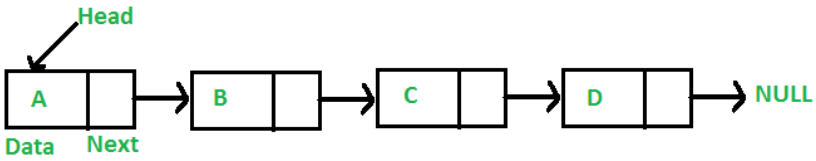
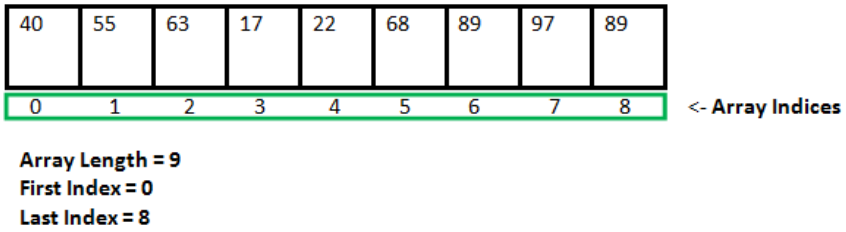
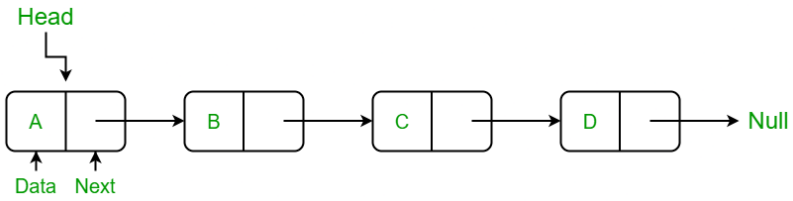
Name	Virinchi Sadashiv Shettigar
UID no.	2021300118
Experiment No.	3

AIM:	Implement a given problem statement using Linked List.
PROBLEM STATEMENT:	Polynomial Addition
THEORY:	**LINKED LIST:** 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. **LINKED LIST VS ARRAY** Array: Arrays store elements in contiguous memory locations, resulting in easily calculable addresses for the elements stored and this allows faster access to an element at a specific index. Linked List: Linked lists are less rigid in their storage structure and elements are usually not stored in contiguous locations, hence they need to be stored with additional tags giving a reference to the next element. This difference in the data storage scheme decides which data structure would be more suitable for a given situation.  Data storage scheme of an array



Linked-List representation

ARRAY	LINKED LISTS
1. Arrays are stored in contiguous location.	1. Linked lists are not stored in contiguous location.
2. Fixed in size.	2. Dynamic in size.
3. Memory is allocated at compile time.	3. Memory is allocated at run time.
4. Uses less memory than linked lists.	4. Uses more memory because it stores both data and the address of next node.
5. Elements can be accessed easily.	5. Element accessing requires the traversal of whole linked list.
6. Insertion and deletion operation takes time.	6. Insertion and deletion operation is faster.

Advantages of Linked Lists:

- The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of usage, and in practical uses, the upper limit is rarely reached.
- Inserting a new element in an array of elements is expensive because a room has to be created for the new elements and to create a room existing elements have to be shifted.

Disadvantages of Linked Lists:

- Random access is not allowed. We have to access elements sequentially starting from the first node. So, we cannot do a binary search with linked lists.
- Extra memory space for a pointer is required for each element of the list.
- Arrays have a better cache locality that can make a pretty big difference in performance.
- It takes a lot of time in traversing and changing the pointers.
- It will be confusing when we work with pointers.

COMPOSITION OF SINGLY LINKED LIST

Data Members:

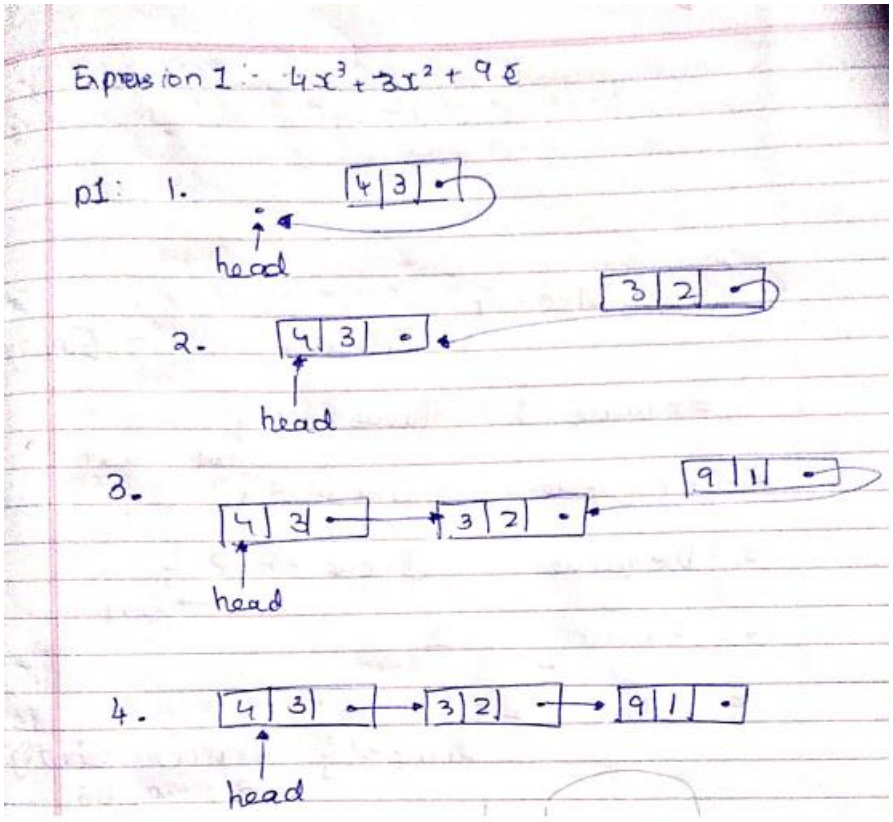
Node object head

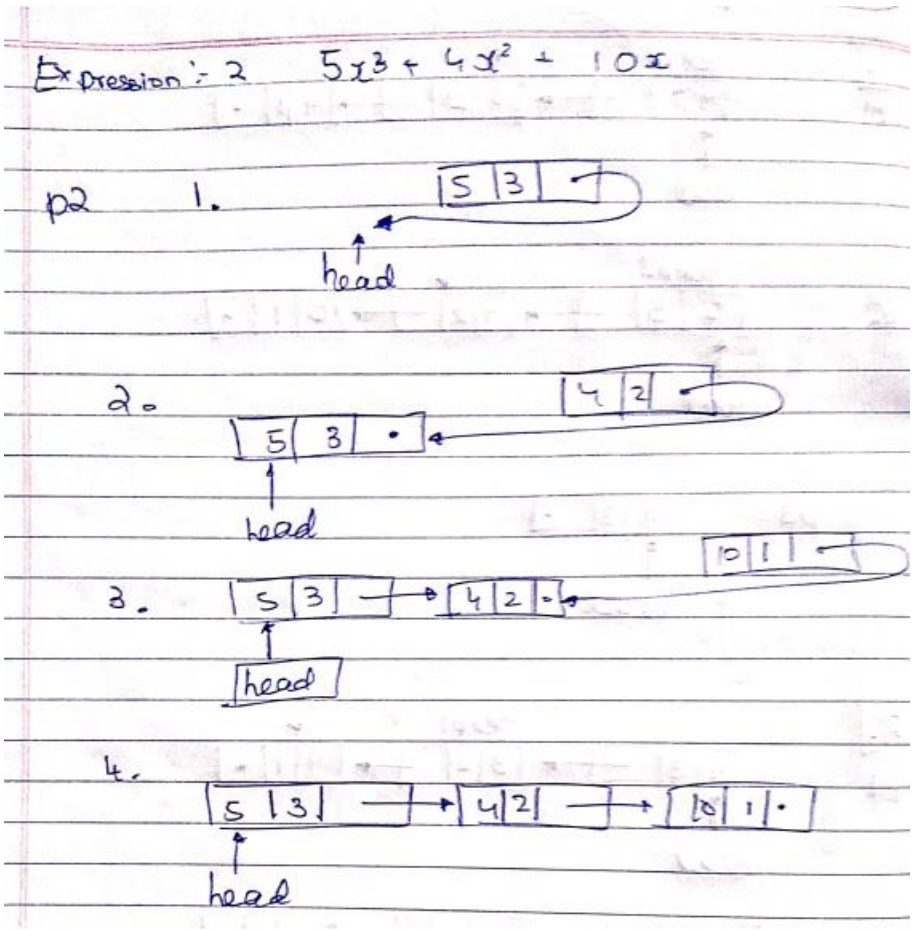
Node Class

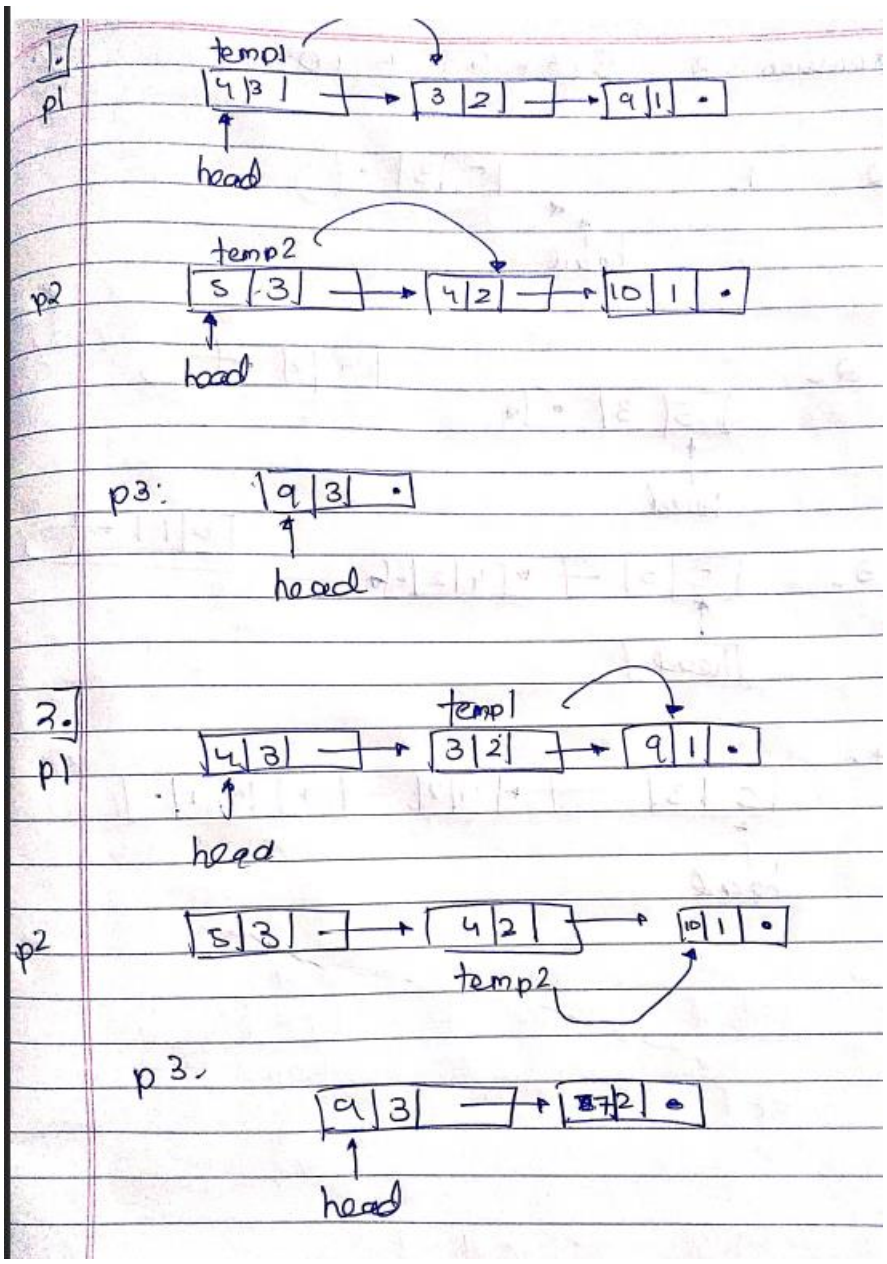
Char data

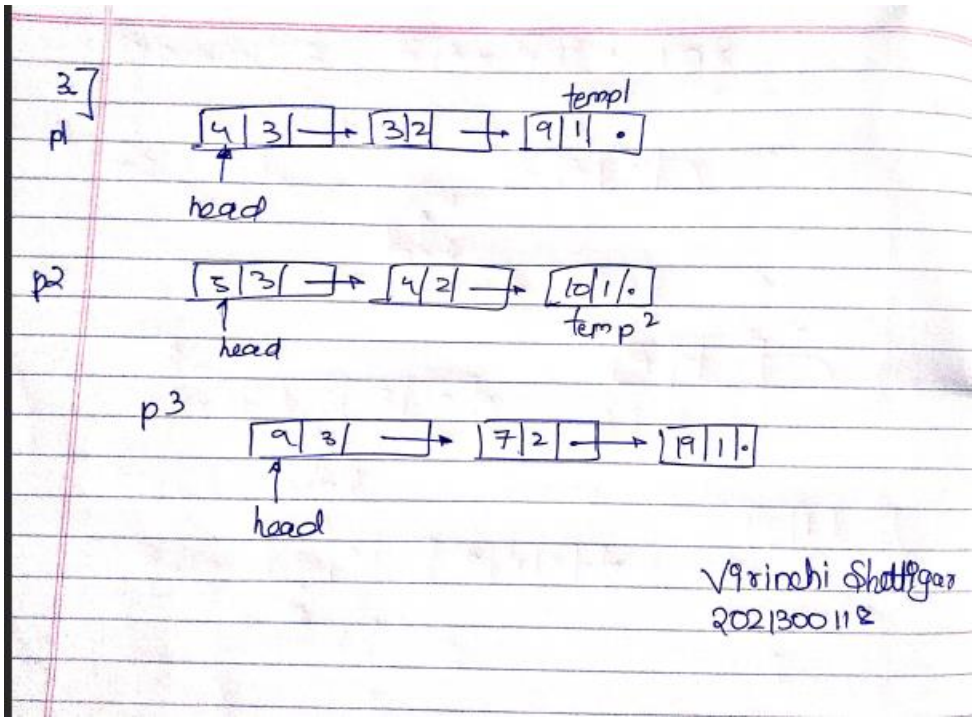
Node next

	Constructor Node (char data) This.data = data Next = null Functions: Insertatfront(): Inserts a node at the front of the list Insertatend(): Inserts a node at the end of the list Insertatpos(): Inserts a node at the position specified in the list. DeleteatFront(): Deletes the front most node of the list. DeleteatEnd(): Deletes the last most node of the list. DeleteatPos(): Deletes the node at specified position in the list. **Applications of Linked List:** 1. Implementation of stacks and queues. 2. Implementation of graphs: Adjacency list reputation of graphs is the most popular which uses a linked list to store adjacent vertices. 3. Dynamic memory allocation: We use a linked list of free blocks. 4. Maintaining a directory of names 5. Performing arithmetic operations on long integers 6. Manipulation of polynomials by storing constants in the node of the linked list 7. Representing sparse matrices
ALGORITHM:	Main Class Main function 1. Create 3 objects p1, p2 and p3 of polynomial class 2. Input no. of terms for the 1st polynomial expression (n) 3. Run a loop n times to user-input for the coefficient and exponent 4. Insert these nodes at the end of p1 5. Repeat same for p2 6. Print both the polynomial expressions 7. Create 2 temporary nodes temp1 and temp2 and assign them to the head of each polynomial 8. Run a loop till both the temporary pointers become Null 9. Check if exponent of temp1 is equal to exponent of temp2 if yes then add their coefficient and insert at the end 10. Else if the exponent of temp1 is greater than the exponent of temp2 then insert the temp1 coefficient and exponent at the end 11. Repeat the same for temp2 if the condition fails 12. Run 2 while loops checking if both the temporary pointers are not NULL 13. Insert the remaining nodes of temp1 and temp2 into p3 14. Print p3

	Polynomial class Node class Data Members: int Coeff, int expo, Node next Constructor Node(int Coeff, int expo) set coeff to coeff and expo to expo and next to null Data Members Node head Void insertAtEnd(int Coeff, int expo) 1. Create an object newNode of node class and set parameter to coeff and expo 2. If head equals null, then set head to newNode 3. Else, create a node object temp set it to head 4. Run a while loop until next of temp becomes NULL 5. Inside that while loop set temp to next of temp 6. Set next of temp to newNode Void printList 1. Create node object temp and set it to head 2. Run a while loop until next of temp becomes null 3. Inside while loop print "(coeff of temp) with x^ (expo of temp)+" and set temp to next of temp 4. Print "(coeff of temp) with x^ (expo of temp)"
PROBLEM-SOLVING:	


Expression : 2 5x3 + 4x2 + 10x
p2 1.
5 3
head
2.
4 2
5 3
head
10 1
3.
5 3
4 2
head
4.
5 3
4 2
10 1
head


1.
p1
temp1
4 3
3 2
9 1
head
temp2
p2
5 3
4 2
10 1
head
p3:
9 3
head
2.
p1
temp1
4 3
3 2
9 1
head
p2
5 3
4 2
10 1
temp2
p3.
9 3
head



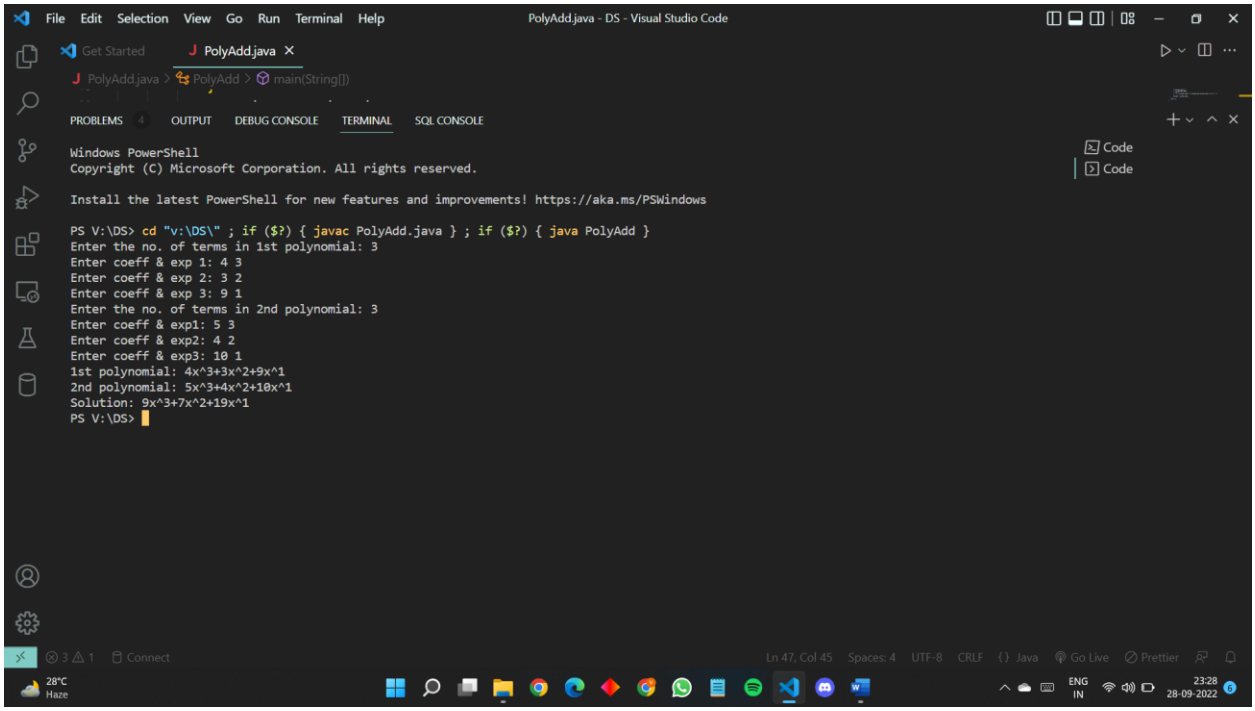
PROGRAM:

```
import java.util.Scanner;
class Polynomial {
    class Node {
        int coeff;
        int exp;
        Node next;
        Node(int coeff, int exp) {
            this.coeff = coeff;
            this.exp = exp;
            next = null;
        }
    }
    Node head;
    int getCoeff(Node node) {
        return node.coeff;
    }
    int getExp(Node node) {
        return node.exp;
    }
    public void insertAtEnd(int coeff, int exp) {
        Node newNode = new Node(coeff, exp);
        Node current = head;
        if (head == null) {
            head = newNode;
        } else {
            while (current.next != null) {
                current = current.next;
            }
            current.next = newNode;
```

```
        }
    }
    public String printList() {
        String s = "";
        Node current = head;
        while (current != null) {
            if(current.exp==0){
                s+=current.coeff;
            } else {
                s += current.coeff+"x^"+current.exp+(current.next!=null?"+":"");
            }
            current = current.next;
        }
        return s;
    }
}
public class PolyAdd {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Polynomial p1 = new Polynomial();
        Polynomial p2 = new Polynomial();
        Polynomial p3 = new Polynomial();
        int n,m,coeff,exp;
        System.out.print("Enter the no. of terms in 1st polynomial: ");
        n = sc.nextInt();
        for(int i=0;i<n;i++) {
            System.out.print("Enter coeff & exp: ");
            coeff = sc.nextInt();
            exp = sc.nextInt();
            p1.insertAtEnd(coeff, exp);
        }
        System.out.print("Enter the no. of terms in 2nd polynomial: ");
        m = sc.nextInt();
        for(int i=0;i<m;i++) {
            System.out.print("Enter coeff & exp: ");
            coeff = sc.nextInt();
            exp = sc.nextInt();
            p2.insertAtEnd(coeff, exp);
        }
        System.out.println("1st polynomial: "+p1.printList());
        System.out.println("2nd polynomial: "+p2.printList());
        Polynomial.Node temp1 = p1.head;
        Polynomial.Node temp2 = p2.head;
        while (temp1 != null && temp2 != null) {
            if (temp1.exp == temp2.exp) {
                p3.insertAtEnd(temp1.coeff + temp2.coeff, temp1.exp);
                temp1 = temp1.next;
                temp2 = temp2.next;
            } else if (temp1.exp > temp2.exp) {
                p3.insertAtEnd(temp1.coeff, temp1.exp);
                temp1 = temp1.next;
            } else {
```

```
            p3.insertAtEnd(temp2.coeff, temp2.exp);
            temp2 = temp2.next;
        }
    }
    while(temp1!=null) {
        p3.insertAtEnd(temp1.coeff, temp1.exp);
        temp1 = temp1.next;
    }
    while(temp2!=null) {
        p3.insertAtEnd(temp2.coeff, temp2.exp);
        temp2 = temp2.next;
    }
    System.out.println("Solution: "+p3.printList());
    sc.close();
  }
}
```

OUTPUT:



CONCLUSION:	In this experiment, I learned about Singly Linked Lists. We saw the differences between Linked Lists and Arrays. I also learned about the advantages and dis-advantages of a Linked List. Using linked list we executed our program of addition of two polynomials