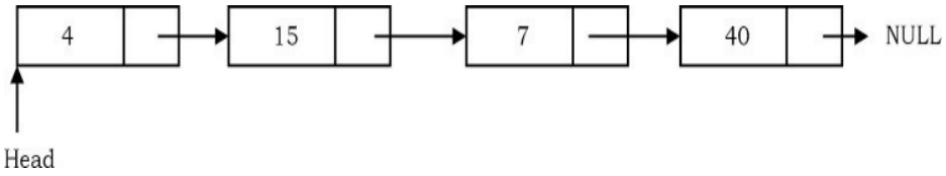
Name	**Hatim Yusuf Sawai**
UID no.	**2021300108**
Experiment No.	**3**

AIM:	**Polynomial addition**
Program	
PROBLEM STATEMENT:	Implement Polynomial addition with 1 variable using a singly linked list.
THEORY:	**What is a Linked List?** A linked list is a data structure used for storing collections of data. A linked list has the following properties: • Successive elements are connected by pointers • The last element points to NULL • Can grow or shrink in size during the execution of a program • Can be made just as long as required (until systems memory exhausts) • Does not waste memory space (but takes some extra memory for pointers). It allocates memory as the list grows. **Diagram of Structure:**  **Advantages of Linked Lists** The advantage of linked lists is that they can be expanded in constant time. To create an array, we must allocate memory for a certain number of elements. To add more elements to the array when full, we must create a new array and copy the old array into the new array. This can take a lot of time. We can prevent this by allocating lots of space initially but then we might allocate more than we need and waste memory. With a linked list, we can start with space for just one

allocated element and add on new elements easily without the need to do any copying and reallocating.

Types of Linked Lists:

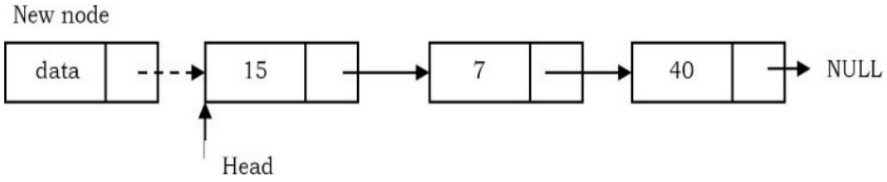
1. Singly Linked List
2. Circular Linked List
3. Doubly Linked List
4. Doubly Circular Linked List
5. Generalized Linked List

Basic Operations of Linked Lists:

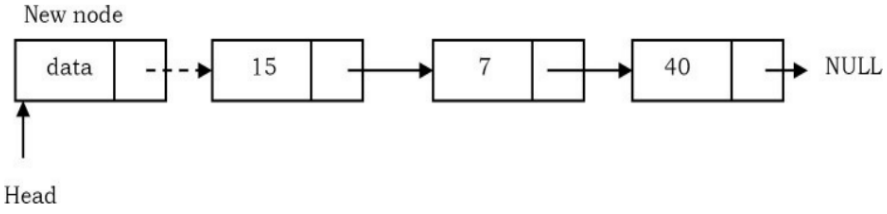
Insertion:

An element can be inserted at 3 different types of positions in a linked list:

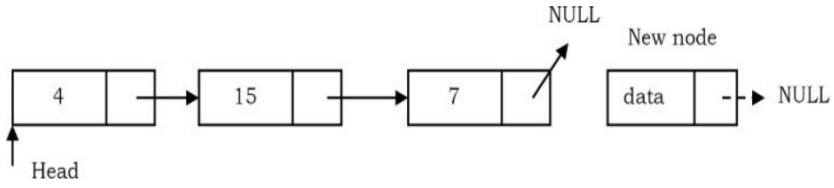
Insertion at the Beginning:



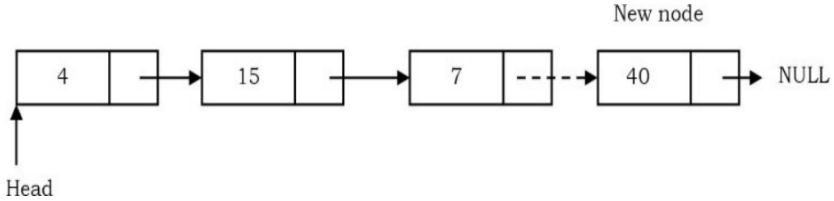
- Update head pointer to point to the new node.



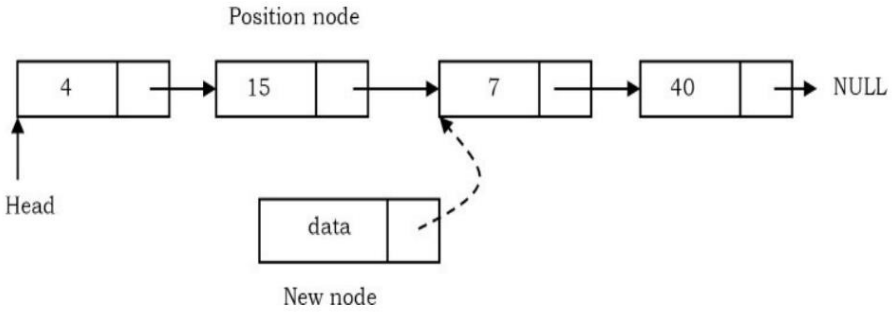
Insertion at the end:



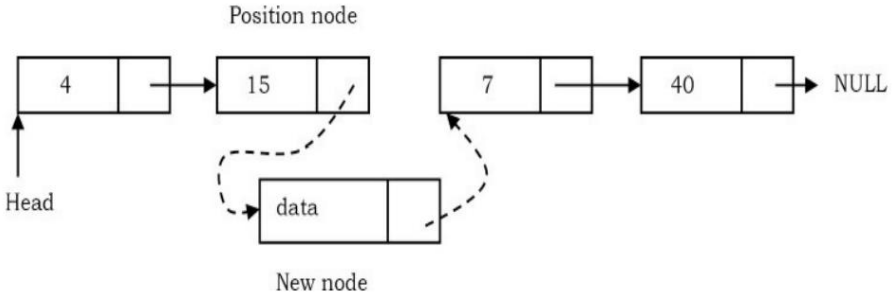
- Last nodes next pointer points to the new node.



Insertion at any place in between:



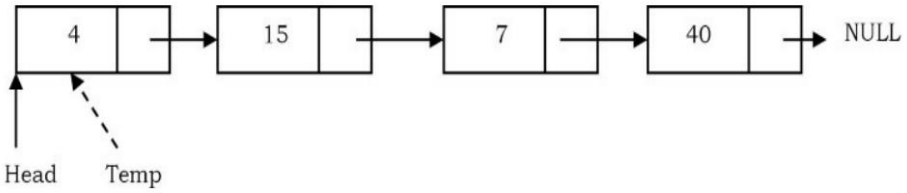
- Position node's next pointer now points to the new node.



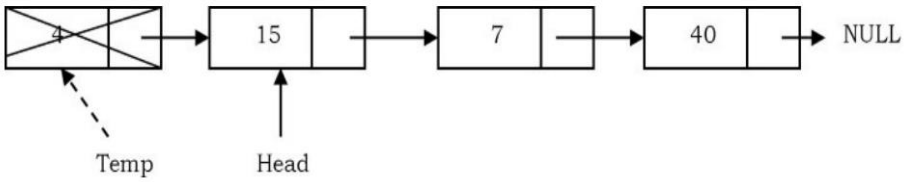
Deletion:

Similarly, an element can be deleted at 3 different types of positions:

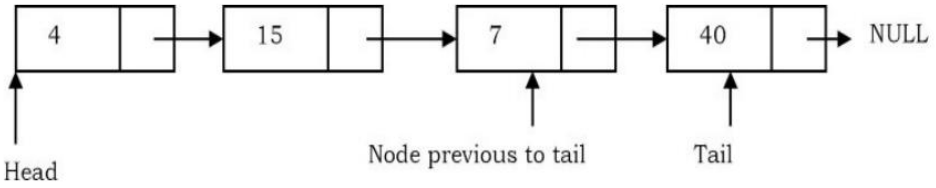
Deletion at the front:



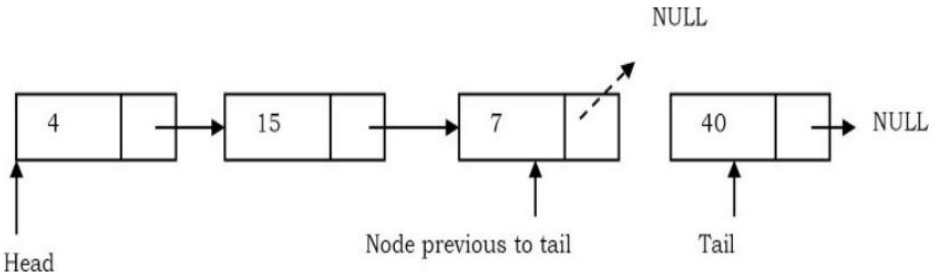
- Now, move the head nodes pointer to the next node and dispose of the temporary node.



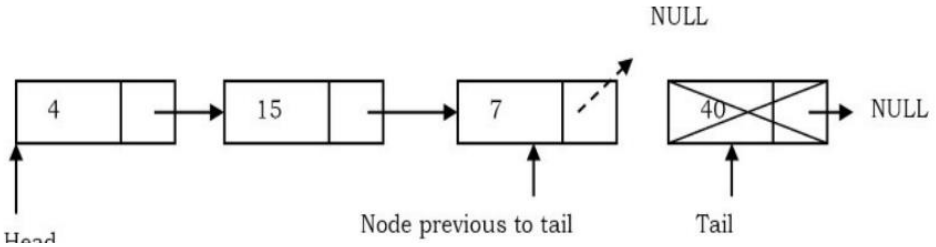
Deletion at End:



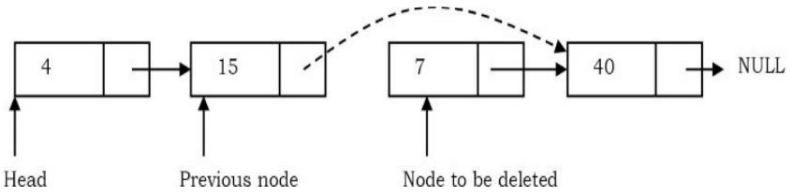
- Update previous node's next pointer with NULL.



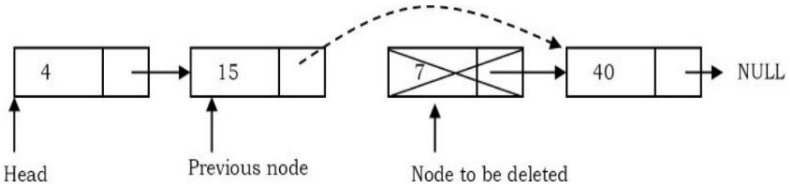
- Dispose of the tail node.

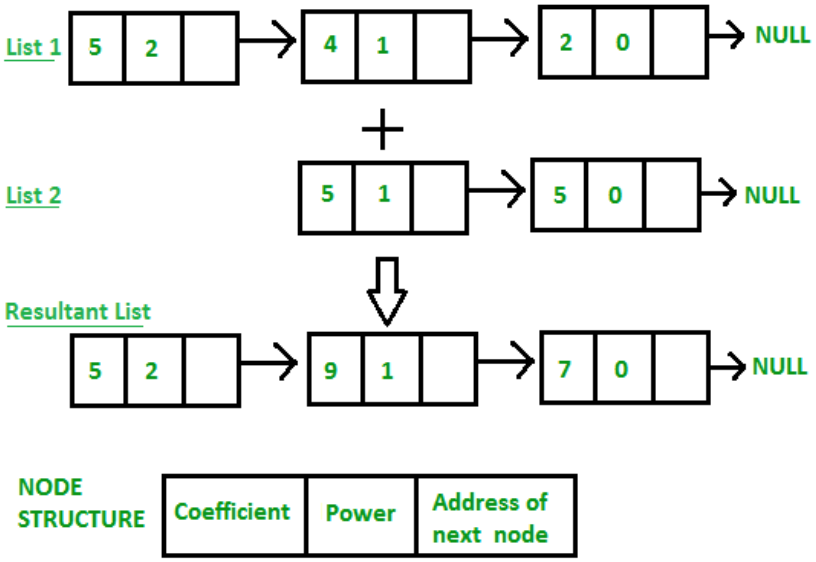


Deletion from any position in between:



- Dispose of the current node to be deleted.



	Applications of Singly Linked List: **Polynomial Addition:** A linked list can be used to add 2 polynomial expressions with one variable. **Example:** **$(5x^3 + 4x^2 + 2)+(5x^1 - 5) = 5x^3 + 4x^2 + 5x^1 - 3$** 
ALGORITHM:	**Class Polynomial:** **Subclass Node:** **Members:** int coeff,exp Node next **Constructor:** this.exp = exp this.coeff = coeff next = null **InsertAtEnd Method:** Initialize newnode of type Node Node current = head If head==null: head = newnode else while current.next is not null do current = current.next

current.next = newnode

PrintList Method:

Initialize String s

Traverse LinkedList and add all coeffs & exp to the string with proper formatting

Class PolyAdd:

Initialize 3 objects of Polynomial class: p1,p2,p3

Int n,m.,coeff,exp

Input 1st polynomial from user and store in p1

Input 2nd polynomial from user and store in p2

Initialize 2 Node pointers: temp1 & temp2

While temp1 != null and temp2 != null:

If temp1.exp == temp2.exp

Add both coeff and store res,exp in p3

Else if temp1.exp > temp2.exp:

Add temp1(coeff,exp) to p3

temp1 = temp1.next

Else:

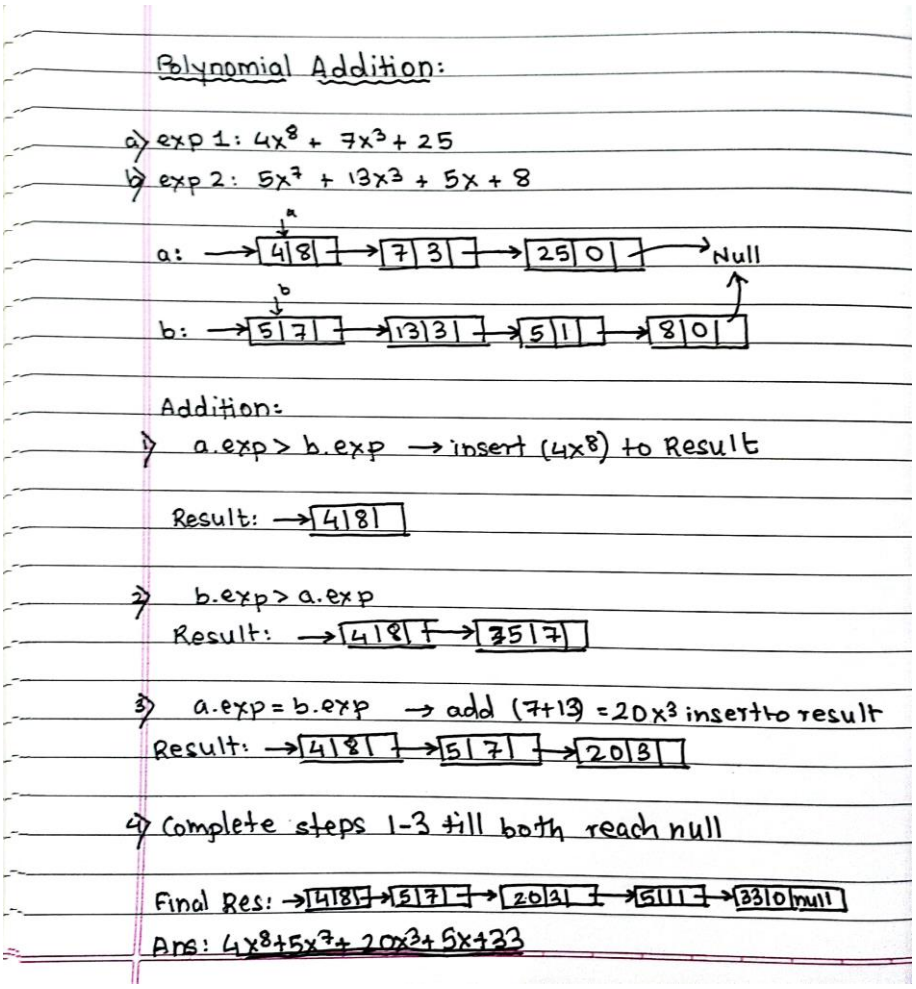
Add temp2(coeff,exp) to p3

Temp2 = temp2.next

Print p3 as solution

End

PROBLEM SOLVING:



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:

```
PS D:\Data Structures\Exp3> cd "d:\Data Structures\Exp3\"
Enter the no. of terms in 1st polynomial: 4
Enter coeff & exp: 41 6
Enter coeff & exp: 25 4
Enter coeff & exp: 14 3
Enter coeff & exp: 25 0
Enter the no. of terms in 2nd polynomial: 4
Enter coeff & exp: 51 7
Enter coeff & exp: 13 6
Enter coeff & exp: 12 3
Enter coeff & exp: 14 2
1st polynomial: 41x^6+25x^4+14x^3+25
2nd polynomial: 51x^7+13x^6+12x^3+14x^2
Solution: 51x^7+54x^6+25x^4+26x^3+14x^2+25
PS D:\Data Structures\Exp3>
```

CONCLUSION:

In this experiment, we learnt how to implement the polynomial addition of 2 polynomial expressions (1 variable) using a singly linked list data structure in java