|  |
| --- |
| Exp: 07 Date: 13/09/2023  JAVA PROGRAMMING LAB 7 |

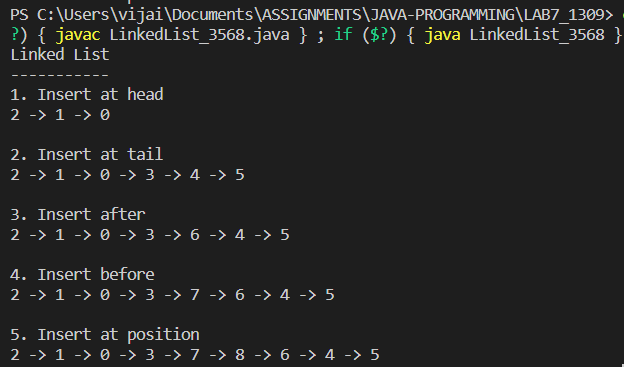
Name: VIJAI SURIA M

Reg No.: 2021503568

**Java program to implement, manipulate and applications using linked lists.**

|  |
| --- |
| import java.time.LocalDate;  import java.time.LocalTime;  import java.util.HashSet;  class Node{      int data;      Node next;      Node(){          data=0;          next=null;      }      Node(int data){          this.data=data;      }  }  class LinkedList {      Node head;      LinkedList(){          head=null;      }      public void insertHead(int data){          Node newNode = new Node(data);          if(this.head==null)              this.head=newNode;          else{              newNode.next=this.head;              this.head=newNode;          }      }      public void insertTail(int data){          Node newNode = new Node(data);          Node curr = head;          if(head==null)              head=newNode;          else{              while(curr.next!=null){                  curr=curr.next;              }              curr.next=newNode;          }      }      public void insertAfter(int data, int after){          Node newNode = new Node(data);          Node curr = head;          if(head==null){              head=newNode;              return;          }          while(curr!=null){              if(curr.data==after){                  newNode.next=curr.next;                  curr.next=newNode;                  return;              }              curr=curr.next;          }          System.out.println("Element not found");      }      public void insertBefore(int data, int before){          Node newNode = new Node(data);          Node curr = head;          if(head==null){              head=newNode;              return;          }          if(head.data==before){              newNode.next=head;              head=newNode;              return;          }          while(curr.next!=null){              if(curr.next.data==before){                  newNode.next=curr.next;                  curr.next=newNode;                  return;              }              curr=curr.next;          }          System.out.println("Element not found");      }      public void insert(int data,int pos){          Node newNode = new Node(data);          Node curr = head;          if(head==null){              head=newNode;              return;          }          if(pos==0){              newNode.next=head;              head=newNode;              return;          }          int i=0;          while(curr.next!=null){              if(i==pos-1){                  newNode.next=curr.next;                  curr.next=newNode;                  return;              }              curr=curr.next;              i++;          }          System.out.println("Position not found");      }      public void deleteHead(){          if(head==null)              System.out.println("List is empty");          else{              Node temp = head;              head=head.next;              temp.next=null;          }      }      public void deleteLast(){          if(head==null){              System.out.println("List is empty");              return;          }          Node curr=head;          if(curr.next==null){              head=null;              return;          }          while(curr.next.next!=null){              curr=curr.next;          }          curr.next=null;      }      public void delete(int data){          if(head==null){              System.out.println("List is empty");              return;          }          Node curr=head;          if(curr.data==data){              head=head.next;              return;          }          while(curr.next!=null){              if(curr.next.data==data){                  curr.next=curr.next.next;                  return;              }              curr=curr.next;          }          System.out.println("Element not found");      }      public void deleteAt(int pos){          if(head==null){              System.out.println("List is empty");              return;          }          Node curr=head;          if(pos==0){              head=head.next;              return;          }          int i=0;          while(curr.next!=null){              if(i==pos-1){                  curr.next=curr.next.next;                  return;              }              curr=curr.next;              i++;          }          System.out.println("Position not found");      }      public void display(){          Node current = head;          while(current.next!=null){              System.out.print(current.data + " -> ");              current=current.next;          }          if(current!=null)              System.out.println(current.data);      }      public void reverseDisplay(){          if(head==null)              return;          Node current = head;          reverseDisplay(current);      }      private void reverseDisplay(Node current){          if(current==null)              return;          reverseDisplay(current.next);          System.out.print(current.data + " -> ");      }      public boolean search(int data){          Node current = head;          while(current!=null){              if(current.data==data)                  return true;              current=current.next;          }          return false;      }      public int get(int pos){          Node current = head;          int i=1;          while(current!=null){              if(i==pos)                  return current.data;              current=current.next;              i++;          }          return -9999;      }      public int length(){          Node current = head;          int count=0;          while(current!=null){              count++;              current=current.next;          }          return count;      }      public int occurence(int data){          Node current = head;          int count=0;          while(current!=null){              if(current.data==data)                  count++;              current=current.next;          }          return count;      }      public void sort() {          head = mergeSort(head);      }      private Node mergeSort(Node head) {          if (head == null || head.next == null) {              return head;          }          // Split the list into two halves.          Node middle = getMiddle(head);          Node nextToMiddle = middle.next;          middle.next = null;          // Recursively sort the two halves.          Node left = mergeSort(head);          Node right = mergeSort(nextToMiddle);          // Merge the sorted halves.          return merge(left, right);      }      private Node getMiddle(Node head) {          if (head == null) {              return head;          }          Node slow = head;          Node fast = head;          while (fast.next != null && fast.next.next != null) {              slow = slow.next;              fast = fast.next.next;          }          return slow;      }      private Node merge(Node left, Node right) {          Node merged = new Node();          Node current = merged;          while (left != null && right != null) {              if (left.data < right.data) {                  current.next = left;                  left = left.next;              } else {                  current.next = right;                  right = right.next;              }              current = current.next;          }          if (left != null) {              current.next = left;          }          if (right != null) {              current.next = right;          }          return merged.next;      }      public void concat(LinkedList list){          Node current = head;          while(current.next!=null){              current=current.next;          }          current.next=list.head;      }      public void removeDuplicates(){          Node current = head;          while(current!=null){              Node temp = current;              while(temp!=null && temp.data==current.data){                  temp=temp.next;              }              current.next=temp;              current=current.next;          }      }      public void removeDuplicatesUnsorted(){          if (head == null) {              // The list is empty, no duplicates to remove.              return;          }          Node current = head;          Node previous = null;          HashSet<Integer> uniqueElements = new HashSet<>();          while (current != null) {              int data = current.data;              if (uniqueElements.contains(data)) {                  // Duplicate node found, remove it.                  previous.next = current.next;              } else {                  // Add the current element to the set of unique elements.                  uniqueElements.add(data);                  previous = current;              }              current = current.next;          }      }      public void polynomialAdd(LinkedList list){          Node current1 = head;          Node current2 = list.head;          LinkedList result = new LinkedList();          while(current1!=null && current2!=null){              if(current1.data==current2.data){                  result.insertTail(current1.data+current2.data);                  current1=current1.next;                  current2=current2.next;              }              else if(current1.data>current2.data){                  result.insertTail(current1.data);                  current1=current1.next;              }              else{                  result.insertTail(current2.data);                  current2=current2.next;              }          }          while(current1!=null){              result.insertTail(current1.data);              current1=current1.next;          }          while(current2!=null){              result.insertTail(current2.data);              current2=current2.next;          }          result.display();      }      public void polynomialMultiply(LinkedList list){          Node current1 = head;          Node current2 = list.head;          LinkedList result = new LinkedList();          while(current1!=null){              while(current2!=null){                  result.insertTail(current1.data\*current2.data);                  current2=current2.next;              }              current2=list.head;              current1=current1.next;          }          result.display();      }  }  public class LinkedList\_3568{      public static void main(String args[]){          System.out.println("Current Date: " + LocalDate.now());          System.out.println("Current Time: " + LocalTime.now());          System.out.println("Name: Vijai Suria M \nRegister Number: (2021503568)");          System.out.println("Linked List");          System.out.println("-----------");          // System.out.println("1. Insert at head");          //LinkedList list = new LinkedList();          // for(int i=0;i<3;i++){          //     list.insertHead(i);          // }          // list.display();          // System.out.println("\n2. Insert at tail");          // for(int i=3;i<6;i++){          //     list.insertTail(i);          // }          // list.display();          // System.out.println("\n3. Insert after");          // list.insertAfter(6, 3);          // list.display();          // System.out.println("\n4. Insert before");          // list.insertBefore(7, 6);          // list.display();          // System.out.println("\n5. Insert at position");          // list.insert(8, 5);          // list.display();          // for(int i=0;i<10;i++){          //     list.insertHead(i);          // }          // System.out.println("Before Sorting: ");          // list.display();          // list.sort();          // System.out.println("After Sorting: ");          // list.display();          // LinkedList l1 = new LinkedList();          // LinkedList l2 = new LinkedList();          // for(int i=0;i<3;i++){          //     l1.insertTail(i);          //     l2.insertTail(i+3);          // }          // System.out.println("List 1: ");          // l1.display();          // System.out.println("List 2: ");          // l2.display();          // System.out.println("Concatenated List: ");          // l1.concat(l2);          // l1.display();          LinkedList list = new LinkedList();          for(int i=0;i<5;i++){              list.insertHead(i);          }          list.insertHead(4);          list.insertHead(3);          System.out.println("List: ");          list.display();          System.out.println("\n1. Delete head");          list.deleteHead();          list.display();          System.out.println("\n2. Delete last");          list.deleteLast();          list.display();          System.out.println("\n3. Delete the value - 3");          list.delete(3);          list.display();          System.out.println("\n4. Delete at position - 2 [0-indexed]");          list.deleteAt(2);          list.display();            System.out.println("Reverse Display List: ");          list.reverseDisplay();          System.out.println("\n Length of the list: " + list.length());          System.out.println("Occurence of 4: " + list.occurence(4));          System.out.println("Occurence of 3: " + list.occurence(3));          System.out.println("Occurence of 2: " + list.occurence(2));          System.out.println("Getting the value at position 4: " + list.get(4));          System.out.println("Getting the value at position 3: " + list.get(3));          System.out.println("Finding the value: 3");          if(list.search(3))              System.out.println("Found");          else              System.out.println("Not Found");          System.out.println("Finding the value: 6");          if(list.search(6))              System.out.println("Found");          else              System.out.println("Not Found");          System.out.println("Before removing duplicates: ");          list.display();          System.out.println("After removing duplicates: ");          list.removeDuplicatesUnsorted();          list.display();      }  } |

1. Insertion:
2. Insert at the beginning.
3. Insert at the end.
4. Insert at a specific position.
5. Insert after a specific node.
6. Insert before a specific node.

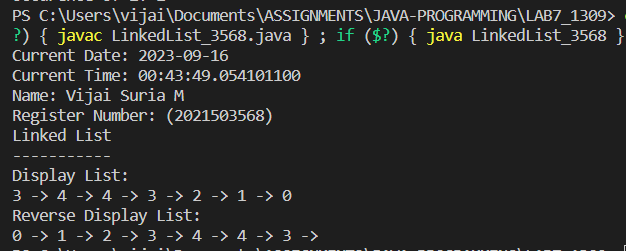


1. Deletion:
2. Delete from the beginning.
3. Delete from the end.
4. Delete a specific element by value.
5. Delete a specific element by position.

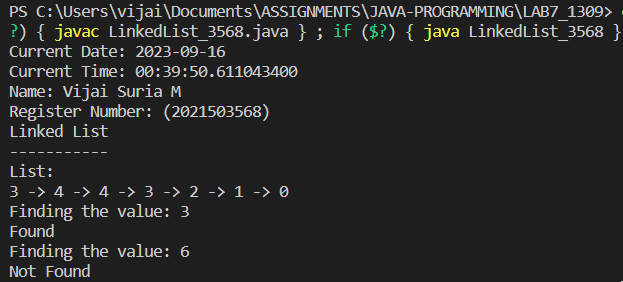
A screenshot of a computer

Description automatically generated

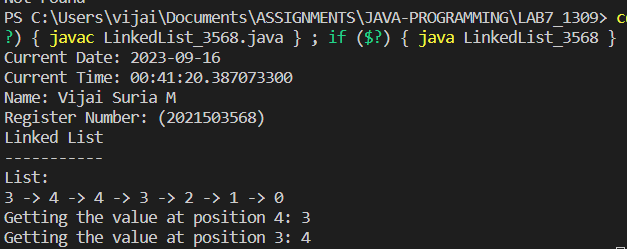
1. Traversal and Display:
2. Traverse and print the elements in the linked list.
3. Reverse and print the elements in the linked list.



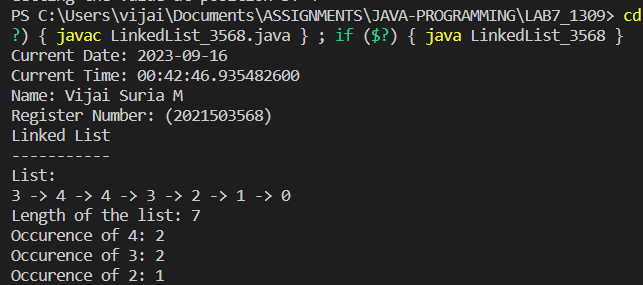
1. Search and Access:
2. Search for an element by value.



1. Access an element by position.

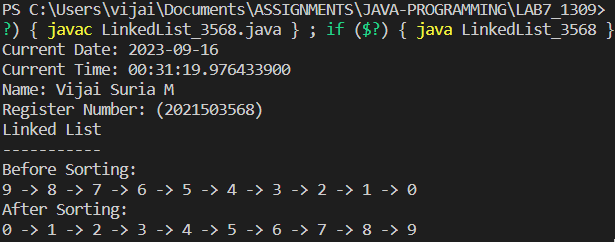


1. Length and Counting:
2. Find the length (number of nodes) of the linked list.
3. Count the occurrences of a specific value in the list.



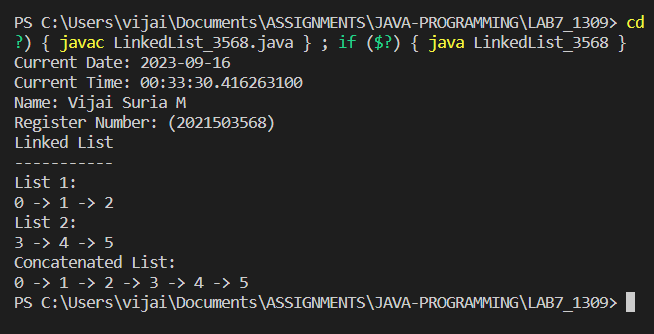
1. Sorting and Merging:

Sort the linked list (bestsort).



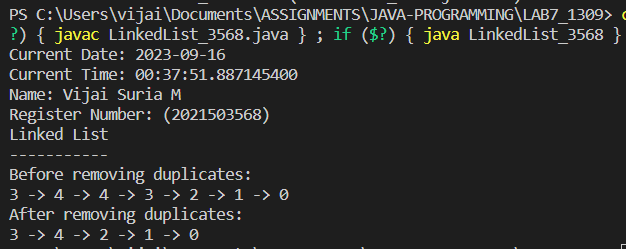
1. Concatenation:

Concatenate (combine) two linked lists together.



1. Duplicate Removal:

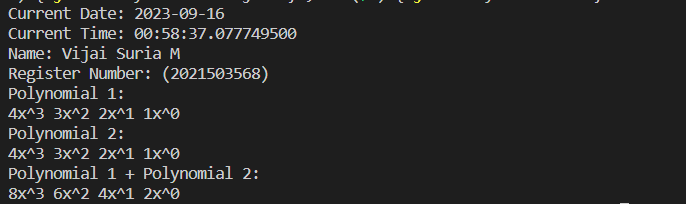
Remove duplicate elements from a linked list.



1. Polynomial Representation:

|  |
| --- |
| import java.time.LocalDate;  import java.time.LocalTime;  class Node{        int exponent;      int coefficient;      Node next;      Node(){          exponent=0;          coefficient=0;          next=null;      }      Node(int exponent,int coefficient){          this.exponent=exponent;          this.coefficient=coefficient;          next=null;      }  }  class Polynomial {      Node head;      Polynomial(){          head=null;        }        void insert(int exponent,int coefficient){          Node temp=new Node(exponent,coefficient);          if(head==null){              head=temp;          }          else{              Node ptr=head;              while(ptr.next!=null){                  ptr=ptr.next;              }              ptr.next=temp;          }      }      void display(){          Node ptr=head;          while(ptr!=null){              System.out.print(ptr.coefficient+"x^"+ptr.exponent+" ");              ptr=ptr.next;          }          System.out.println();      }      void add(Polynomial p){          Node ptr1=head;          Node ptr2=p.head;          while(ptr1!=null && ptr2!=null){              if(ptr1.exponent==ptr2.exponent){                  ptr1.coefficient+=ptr2.coefficient;                  ptr1=ptr1.next;                  ptr2=ptr2.next;              }              else if(ptr1.exponent>ptr2.exponent){                  ptr1=ptr1.next;              }              else{                  Node temp=new Node(ptr2.exponent,ptr2.coefficient);                  temp.next=ptr1.next;                  ptr1.next=temp;                  ptr1=ptr1.next;                  ptr2=ptr2.next;              }          }          while(ptr2!=null){              Node temp=new Node(ptr2.exponent,ptr2.coefficient);              temp.next=ptr1.next;              ptr1.next=temp;              ptr1=ptr1.next;              ptr2=ptr2.next;          }      }      void multiply(Polynomial p){          Node ptr1=head;          Node ptr2=p.head;          Polynomial result=new Polynomial();          while(ptr1!=null){              while(ptr2!=null){                  result.insert(ptr1.exponent+ptr2.exponent,ptr1.coefficient\*ptr2.coefficient);                  ptr2=ptr2.next;              }              ptr1=ptr1.next;              ptr2=p.head;          }          result.display();      }  }  public class Polynomial3568 {      public static void main(String args[]){          System.out.println("Current Date: " + LocalDate.now());          System.out.println("Current Time: " + LocalTime.now());          System.out.println("Name: Vijai Suria M \nRegister Number: (2021503568)");          Polynomial p1=new Polynomial();          Polynomial p2=new Polynomial();          p1.insert(3, 4);          p1.insert(2, 3);          p1.insert(1, 2);          p1.insert(0, 1);          p2.insert(3, 4);          p2.insert(2, 3);          p2.insert(1, 2);          p2.insert(0, 1);          System.out.println("Polynomial 1:");          p1.display();          System.out.println("Polynomial 2:");          p2.display();          System.out.println("Polynomial 1 \* Polynomial 2:");          p1.multiply(p2);          p1.display();      }  } |

1. Implement polynomial addition using linked lists.



1. Implement polynomial multiplication using linked lists.

