|  |  |
| --- | --- |
| **Name** | **Hatim Yusuf Sawai** |
| **UID no.** | **2021300108** |
| **Experiment No.** | **4** |

|  |  |
| --- | --- |
| **AIM:** | **Doubly Linked List - insert & delete operations** |
| **Program** | |
| **PROBLEM STATEMENT:** | Perform insert & delete operations on doubly linked list. Take position as input from the user and make functions for left/right specific insertion/deletion. |
| **THEORY:** | **Doubly Linked List (DLL)**  A Doubly Linked List (DLL) contains an extra pointer, typically called the previous pointer, together with the next pointer and data which are there in the singly linked list.  **Structure:**  **Advantages:**  The advantage of a doubly linked list is that given a node in the list, we can navigate in both directions. A node in a singly linked list cannot be removed unless we have the pointer to its predecessor. But in a doubly linked list, we can delete a node even if we don’t have the previous node’s address (since each node has a left pointer pointing to the previous node and can move backwards).  **Disadvantages:**  The primary disadvantages of doubly linked lists are:  • Each node requires an extra pointer, requiring more space.  • The insertion or deletion of a node takes a bit longer (more pointer operations).  **Insertion in DLL:**  A node can be added in four ways:  **1. At the front of the DLL**  Update the right pointer of the new node to point to the current head node and make the left pointer of the new node NULL. Update the head node’s left pointer to point to the new node and make the new node as head.  **2. At the end of the DLL**  New node’s right pointer points to Null and left pointer points to the end of the list. Update right pointer of last node to point to new node.  **3. After a given node:**  **4. Before a given node:**  **Deletion in DLL:**  Similarly, deletion in DLL can be done in 4 ways:  **1. Deleting from front**  **2. Deleting from end**  **3. Deleting after given node**  **4. Deleting before given node**  **Applications of DLL:**  • Doubly linked list can be used in navigation systems where both forward and backward traversal is required.  • It can be used to implement different tree data structures.  • It can be used to implement undo/redo operations. |
| **ALGORITHM:** |  |
| **PROBLEM SOLVING:** |  |
| **PROGRAM:** | **DLLCheck.java:**  import *java*.*util*.*Scanner*;  import *dlinkedlistds*.*DLL*;  *public* *class* DLLCheck {  *public* *static* void main(String[] args) {          Scanner sc = new Scanner(System.*in*);          DLL d = new DLL();          int choice,flag,pos,data;          while(true) {              System.*out*.println(                      "Select an option:\n1.Insert at front\n2.Insert at Right\n3.Insert at left\n4.Insert at end\n5.Delete at front\n6.Delete at right\n7.Delete at left\n8.Delete at end");              choice = sc.nextInt();              switch(choice) {                  case 1:                      System.*out*.print("Enter the element to be inserted: ");                      d.insertAtFront(sc.nextInt());                      System.*out*.println("The list is: " + d.printList());                      break;                  case 2:                      System.*out*.print("Enter the element to be inserted: ");                      data = sc.nextInt();                      System.*out*.print("Enter the pos after which to insert: ");                      pos = sc.nextInt();                      if (pos == -1) {                          System.*out*.println("Element not found!");                      } else {                          d.insertAtPos(data, pos+1);                          System.*out*.println("The list is: " + d.printList());                      }                      break;                  case 3:                      System.*out*.print("Enter the element to be inserted: ");                      data = sc.nextInt();                      System.*out*.print("Enter the pos before which to insert: ");                      pos = sc.nextInt();                      if (pos == -1) {                          System.*out*.println("Element not found!");                      } else {                          d.insertAtPos(data,pos);                          System.*out*.println("The list is: " + d.printList());                      }                      break;                  case 4:                      System.*out*.print("Enter the element to be inserted: ");                      d.insertAtEnd(sc.nextInt());                      System.*out*.println("The list is: " + d.printList());                      break;                  case 5:                      System.*out*.println("Deleted element: "+d.deleteAtFront());                      System.*out*.println("The list is: " + d.printList());                      break;                  case 6:                      System.*out*.print("Enter pos after which to delete: ");                      pos = sc.nextInt();                      if(pos == -1) {                          System.*out*.println("Element not found!");                      } else {                          System.*out*.println("Deleted element: "+d.deleteAtPos(pos+1));                          System.*out*.println("The list is: " + d.printList());                      }                      break;                  case 7:                      System.*out*.print("Enter pos before which to delete: ");                      pos = sc.nextInt();                      if (pos == -1) {                          System.*out*.println("Element not found!");                      } else {                          System.*out*.println("Deleted element: " + d.deleteAtPos(pos-1));                          System.*out*.println("The list is: " + d.printList());                      }                      break;                  case 8:                      System.*out*.println("Deleted element: "+d.deleteAtEnd());                      System.*out*.println("The list is: " + d.printList());                      break;                  case 9:                      System.*out*.println("The list is: "+d.printList());                      break;                  default:                      System.*out*.println("Invalid choice!");              }              System.*out*.println("Do you want to continue?\n1. Yes\t2. No");              flag = sc.nextInt();              if (flag == 2) {                  break;              }          }          sc.close();      }  }  **DLL.java:**  package *dlinkedlistds*;  *public* *class* DLL {  *class* Node {          int data;          Node next,prev;  *public* Node(int data) {              this.*data* = data;              next = null;              prev = null;          }      }      Node head=null;      Node tail=null;      int len;  *public* void insertAtFront(int data) {          Node newNode = new Node(data);          if(head==null) {              head = newNode;              tail = newNode;          } else {              head.*prev* = newNode;              newNode.*next* = head;              head = newNode;          }          len++;      }  *public* void insertAtEnd(int data) {          Node newNode = new Node(data);          if(head==null) {              head = newNode;              tail = newNode;          } else {              tail.*next* = newNode;              newNode.*prev* = tail;              tail = newNode;          }          len++;      }  *public* void insertAtPos(int data,int pos) {          if(pos<1||pos>len+1) {              System.*out*.println("Invalid position");          } else if(pos==1) {              insertAtFront(data);          } else if(pos==len+1) {              insertAtEnd(data);          } else {              Node newNode = new Node(data);              Node temp = head;              for(int i=1;i<pos-1;i++) {                  temp = temp.*next*;              }              newNode.*next* = temp.*next*;              temp.*next*.*prev* = newNode;              temp.*next* = newNode;              newNode.*prev* = temp;              len++;          }      }  *public* int deleteAtFront() {          if(head==null) {              System.*out*.println("List is empty");              return -1;          } else {              int data = head.*data*;              head = head.*next*;              head.*prev* = null;              len--;              return data;          }      }  *public* int deleteAtEnd() {          if(head==null) {              System.*out*.println("List is empty");              return -1;          } else if(len==1) {              int data = head.*data*;              head = null;              tail = null;              len--;              return data;          } else {              int data = tail.*data*;              tail = tail.*prev*;              tail.*next* = null;              len--;              return data;          }      }  *public* int deleteAtPos(int pos) {          if(pos<1||pos>len) {              System.*out*.println("Invalid position");              return -1;          } else if(pos==1) {              return deleteAtFront();          } else if(pos==len) {              return deleteAtEnd();          } else {              Node temp = head;              for(int i=1;i<pos-1;i++) {                  temp = temp.*next*;              }              int data = temp.*next*.*data*;              temp.*next* = temp.*next*.*next*;              temp.*next*.*prev* = temp;              len--;              return data;          }      }  *public* String printList() {          Node temp = head;          String s = "head->";          while(temp!=null) {              s += temp.*data*+(temp.*next*!=null?"->":"");              temp = temp.*next*;          }          s += "<-tail";          return s;      }  } |
| **OUTPUT:**  **1. Insert at Right**  **2. Insert at Left**  **3. Delete at Right**  **4. Delete at Left** | |
| **CONCLUSION:** | In this experiment, we learned how to implement doubly linked list in java and perform insertion & deletion operation on it using position rather an data matching. |