

Mohammad Ali Jinnah University

Chartered by Government of Sindh - Recognized by HEC

**Lab Task 8**

**Name:** Muhamad Fahad

**Id:** FA19-BSSE-0014

**Subject:** Data Structures and Algorithms Lab (CS 2511)

**Lab Title:** Linked List

**Section:** AM

**Teacher:** MUHAMMAD MUBASHIR KHAN

**Date:** Thursday, December 17, 2020

1. **Define a linked list and perform insertion, modification & deletion on that linked list.**

**Code:**

import java.util.HashSet;  
  
public class Linkedlist {  
 Node head;  
  
 static class Node{  
 private int Value;  
 private Node pointer;  
  
 Node(int data){  
 Value = data;  
 pointer = null;  
 }  
 }  
  
 static int getLenght(Linkedlist list) {  
 int i = 0;  
 Node last = list.head;  
 while (last.pointer != null) {  
 i++;  
 last = last.pointer;  
 }  
 return i;  
 }  
 static boolean isEmpty(Linkedlist list) {  
 boolean condition = true;  
  
 if (list.head == null)  
 condition = false;  
  
 return condition;  
 }  
  
 // add the element in the linked list  
 static Linkedlist insert(Linkedlist list, int data) {  
 Node new\_node = new Node(data);  
 new\_node.pointer = null;  
  
 if (!*isEmpty*(list))  
 list.head = new\_node;  
 else {  
 Node last = list.head;  
 while (last.pointer != null)  
 last = last.pointer;  
  
 last.pointer = new\_node;  
 }  
  
 return list;  
 }  
 static Linkedlist insertAtstart(Linkedlist list, int data) {  
 Node new\_node = new Node(data);  
  
 if (!(*isEmpty*(list)))  
 list.head = new\_node;  
 else {  
 new\_node.pointer = list.head;  
 list.head = new\_node;  
 }  
 return list;  
 }  
 static Linkedlist insertBykey(Linkedlist list, int data, int key) {  
 int size = *getLenght*(list);  
 Node new\_node = new Node(data);  
 Node last = null;  
 Node temp = list.head;  
  
 if(key == 0)  
 list = *insertAtstart*(list,data);  
 else if(key > size-1)  
 list = *insert*(list,data);  
 else{  
 for(int i = 0; i < key; i++)  
 temp = (last = temp).pointer;  
  
 new\_node.pointer = temp;  
 last.pointer = new\_node;  
 }  
  
 return list;  
 }  
  
 // delete the element in the linked list  
 static Linkedlist deleteByValue(Linkedlist list, int key){  
 Node currNode = list.head,  
 prev = null;  
  
 if (currNode != null && currNode.Value == key) {  
 list.head = currNode.pointer;  
 System.*out*.println(key + " found and deleted");  
 return list;  
 }  
  
 while (currNode != null && currNode.Value != key)  
 currNode = (prev = currNode).pointer;  
  
  
 if (currNode != null) {  
 prev.pointer = currNode.pointer;  
 System.*out*.println(key + " found and deleted");  
 }  
  
 if (currNode == null)  
 System.*out*.println(key + " not found");  
  
 return list;  
 }  
 static Linkedlist deleteBykey(Linkedlist list, int key){  
 int size = *getLenght*(list);  
 Node currNode = list.head,  
 prev = null;  
  
 if (size < key) {  
 System.*out*.println(key + " not Exist");  
 return list;  
 }  
  
 if (key == 0){  
 list.head = currNode.pointer;  
 System.*out*.println((currNode.pointer).Value + " found and deleted");  
 return list;  
 }  
  
 for (int i=0; i<key; i++)  
 currNode = (prev = currNode).pointer;  
  
 prev.pointer = currNode.pointer;  
 System.*out*.println(key + " found and deleted");  
 return list;  
 }  
 static Linkedlist delete(Linkedlist list){  
 return (*deleteBykey*(list,*getLenght*(list)));  
 }  
 static Linkedlist deleteFront(Linkedlist list){  
 return (*deleteBykey*(list,0));  
 }  
 static Linkedlist deleteDuplicate(Linkedlist list){  
 HashSet<Integer> hs = new HashSet<>();  
  
 Node current = list.head;  
 Node prev = null;  
 while (current != null) {  
 if (hs.contains(current.Value)) prev.pointer = current.pointer;  
 else {  
 hs.add(current.Value);  
 prev = current;  
 }  
 current = current.pointer;  
 }  
 return list;  
 }  
  
 // update the element in the linked list  
 static Linkedlist update(Linkedlist list, int index, int value){  
 Node currNode = list.head;  
 if (*getLenght*(list) < index) {  
 System.*out*.println("Index not Exist! ");  
 return list;  
 }  
  
 for (int i = 0; i < index; i++)  
 currNode = currNode.pointer;  
  
 currNode.Value = value;  
  
 return list;  
 }  
  
 // search the element in the linked list  
 static Boolean Search(Linkedlist list, int key) {  
 Node currNode = list.head;  
 Boolean condition = false;  
  
 if (currNode == null)  
 return condition;  
  
  
 while (currNode.Value != key)  
 currNode = currNode.pointer;  
  
 if (currNode != null) condition = true;  
 else System.*out*.println(key + " not Exist(404 Error)");  
  
 return condition;  
 }  
  
 //Sorting the Element in the  
 static Linkedlist sortList(Linkedlist list) {  
 Node current = list.head, index = null;  
 int temp;  
  
 if(list.head.pointer == null) {  
 return list;  
 }  
 else {  
 while(current != null) {  
 index = current.pointer;  
  
 while(index != null) {  
 if(current.Value > index.Value) {  
 temp = current.Value;  
 current.Value = index.Value;  
 index.Value = temp;  
 }  
 index = index.pointer;  
 }  
 current = current.pointer;  
 }  
 }  
 return list;  
 }  
  
 //Merge Two linked list in the element  
 static Linkedlist Merge(Linkedlist list1,Linkedlist list2){  
 Linkedlist list = new Linkedlist();  
 int l1 = *getLenght*(list1),l2 = *getLenght*(list2);  
 Node current = list1.head;  
  
 for (int i = 0; i <= (l1+l2)+1; i++) {  
 list.*insert*(list,current.Value);  
 if (l1 != i) current = current.pointer;  
 else current = list2.head;  
 }  
  
 return list;  
 }  
  
 //count the odd and even nodes  
 static int countOdd(Linkedlist list){  
 int count = 0;  
 Node current = list.head;  
 while (current.pointer != null){  
 if (current.Value % 2 == 0)  
 count++;  
 current = current.pointer;  
 }  
 return count;  
 }  
 static int countEven(Linkedlist list){  
 int count = 0;  
 Node current = list.head;  
 while (current.pointer != null){  
 if (current.Value % 2 != 0)  
 count++;  
 current = current.pointer;  
 }  
 return count;  
 }  
  
  
  
 //Display methods  
 static String Display(Linkedlist list){  
 Node currNode = list.head;  
 String display = "LinkedList: {";  
  
 while (currNode != null) {  
 display += currNode.Value + ", ";  
 currNode = currNode.pointer;  
 }  
 display += "\b\b};";  
 return display;  
 }  
 static String Displayreverse (Linkedlist list){  
 Node currNode = list.head;  
 String display = "}";  
  
 while (currNode != null) {  
 display += currNode.Value + " ";  
 currNode = currNode.pointer;  
 }  
 display += "{";  
  
 display = "LinkedList reverse: " +(new StringBuilder(display)).reverse();  
 return display;  
 }  
  
}

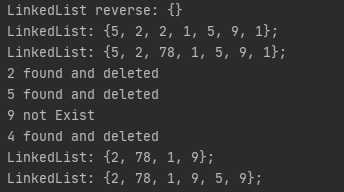
**Main Question File:**

public class PracticeQuestion {  
 public static void main(String[] args) {  
 Linkedlist list = new Linkedlist();  
 System.*out*.println(list.*Displayreverse*(list)); //Q2  
  
 list.*insert*(list, 1); // Q3  
 list.*insertAtstart*(list, 2); //Q4  
 list.*insertBykey*(list, 5,2); //Q5 at any postion or mid  
 list.*insertBykey*(list, 9,2);list.*insert*(list, 1); // Q3  
 list.*insertAtstart*(list, 2); //Q4  
 list.*insertBykey*(list, 5,0); //Q5 at any postion or mid  
  
 System.*out*.println(list.*Display*(list));  
  
 list.*update*(list, 2,78); //Q5 at any postion or mid  
  
 System.*out*.println(list.*Display*(list));  
  
 list.*deleteFront*(list);  
 list.*deleteByValue*(list,5);  
 list.*deleteBykey*(list,9);  
 list.*delete*(list);  
 System.*out*.println(list.*Display*(list));  
  
 list.*insert*(list, 5);  
 list.*insert*(list, 9);  
 System.*out*.println(list.*Display*(list));

}

}

**Output:**

****

1. **Define a linked list and delete all the duplicate values from that list.**

**Code**

static Linkedlist deleteDuplicate(Linkedlist list){  
 HashSet<Integer> hs = new HashSet<>();  
  
 Node current = list.head;  
 Node prev = null;  
 while (current != null) {  
 if (hs.contains(current.Value)) prev.pointer = current.pointer;  
 else {  
 hs.add(current.Value);  
 prev = current;  
 }  
 current = current.pointer;  
 }  
 return list;  
}

The function used

And main class

public class PracticeQuestion {  
 public static void main(String[] args) {  
 Linkedlist list = new Linkedlist();  
 System.*out*.println(list.*Displayreverse*(list)); //Q2  
  
 list.*insert*(list, 1); // Q3  
 list.*insertAtstart*(list, 2); //Q4  
 list.*insertBykey*(list, 5,2); //Q5 at any postion or mid  
 list.*insertBykey*(list, 9,2);list.*insert*(list, 1); // Q3  
 list.*insertAtstart*(list, 2); //Q4  
 list.*insertBykey*(list, 5,0); //Q5 at any postion or mid  
  
 System.*out*.println(list.*Display*(list));  
  
 list.*update*(list, 2,78); //Q5 at any postion or mid  
  
 System.*out*.println(list.*Display*(list));  
  
 list.*deleteFront*(list);  
 list.*deleteByValue*(list,5);  
 list.*deleteBykey*(list,9);  
 list.*delete*(list);  
 System.*out*.println(list.*Display*(list));  
  
 list.*insert*(list, 5);  
 list.*insert*(list, 9);  
 System.*out*.println(list.*Display*(list));  
  
 list.*deleteDuplicate*(list);  
 System.*out*.println(list.*Display*(list));  
  
 }  
}

