|  |  |
| --- | --- |
| **Name** | Vineet Parmar |
| **UID no.** | 2021300092 |
| **Experiment No.** | 3 |

|  |  |
| --- | --- |
| **AIM:** | Implement a given problem statement using Linked List. |
| **PROBLEM STATEMENT:** | To check if a given input is a palindrome or not. |
| **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  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. |
| **ALGORITHM:** | Main Class  Main function   1. Take input string s from user 2. Create 3 objects l,r and new of linked list class 3. Run a for-loop for half the length of s and keep inserting the characters of the string at the front of the node in linked list l. 4. Run a second for-loop for the other half of the length of s skipping the middle term if length is odd and keep inserting the characters of the string at the front of the node in linked list r. 5. Run a third for loop again for half the length of s and now delete at front from the l and insert the deleted element at the front in new. 6. Initialize a flag variable and set it to 0. 7. Run a last for loop and keep checking if the data at the head of the linked list is equal or not for l and r and then delete the front elements of both l and new 8. If the elements are not equal break the for loop and set flag to 1. 9. If flag equals 0, then print Palindrome, else print not a Palindrome.   Linked List class  Node class  Data Members: character node, Node next  Constructor Node(character d)  set data to d and next to null  Data Members  Node head  Void insertAtBegin(char d)   1. Create an object newNode of node class and set parameter to data 2. If head equals null, then set head to newNode 3. Else, set next of newNode to head and set head to newNode.   Char DeleteAtBegin()   1. Initialize char x and set it to 0 2. Create an object temp of Node class and set it to head. 3. If head equals null, then print List is empty 4. Else set x to data of temp 5. Set head to next of head 6. Set next of temp to null 7. Return x |
| **PROBLEM SOLVING:** |  |
| **PROGRAM:** | import java.util.\*;  class Experiment2{  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  System.out.println("Enter a string to check whether it is palindrome or not: ");  String s = sc.nextLine();  charLinkedList l = new charLinkedList();  charLinkedList r = new charLinkedList();  int flag = 0;  for(int i = 0; i < s.length()/2 ; i++)  {  l.insertAtBegin(s.charAt(i));  }  l.printList();  for(int i = (s.length() % 2 == 0 ? (s.length()/2): (s.length()/2 + 1)); i < s.length(); i++)  {  r.insertAtBegin(s.charAt(i));  }  r.printList();  charLinkedList n = new charLinkedList();  for(int i = 0; i < s.length()/2; i++)  {  char x = r.deleteAtBegin();  n.insertAtBegin(x);  }  n.printList();  for(int i = 0; i < s.length()/2; i++)  {  if(l.deleteAtBegin() == n.deleteAtBegin())  continue;  else{  flag = 1;  break;  }  }  if(flag == 0)  {  System.out.println("PALINDROME.");  }  else{  System.out.println("NOT A PALINDROME.");  }  System.exit(0);  }  }  class charLinkedList{  class Node{  char data;  Node next;  Node(char d){  data=d;  next=null;  }  }  Node head;  public void insertAtBegin(char data){  Node newNode = new Node(data);  if(head == null){  head = newNode;  }  else{  newNode.next = head;  head = newNode;  }  }  public void insertAtEnd(char data){  Node newNode = new Node(data);  if(head == null){  head = newNode;  }  else{  Node temp = head;  while(temp.next != null){  temp = temp.next;  }  temp.next = newNode;  }  }  public void insertAtPos(char data, int pos){  Node newNode = new Node(data);  if(pos == 1){  newNode.next = head;  head = newNode;  }  else{  Node temp = head;  for(int i=1; i<pos-1; i++){  temp = temp.next;  }  newNode.next = temp.next;  temp.next = newNode;  }  }  public char deleteAtBegin(){  char x = 0;  Node temp = head;  if(head == null){  System.out.println("List is empty");  }  else{  x = temp.data;  head = head.next;  temp.next = null;  }  return x;  }  public char deleteAtEnd(){  char x = 0;  if(head == null){  System.out.println("List is empty");  }  else{  Node temp = head;  while(temp.next.next != null){  temp = temp.next;  }  x = temp.data;  temp.next = null;  }  return x;  }  public char deleteAtPos(int pos){  char x = 0;  if(pos == 1){  head = head.next;  }  else{  Node temp = head;  for(int i = 1; i < pos-1; i++){  temp = temp.next;  }  x = temp.data;  temp.next = temp.next.next;  }  return x;  }  public void printList(){  Node temp = head;  while(temp != null){  System.out.print(temp.data+" ");  temp = temp.next;  }  System.out.println();  }  } |
| **OUTPUT:** | |
| **CONCLUSION:** | In this experiment, we learned about Singly Linked Lists. We saw the differences between Linked Lists and Arrays. Moving on further, we learnt about the advantages and Disadvantages of a Linked List. We also learnt about the data members and some functions of Linked List. Lastly, we implemented our knowledge on a problem statement that required us to check if a given input was a palindrome or not. |