**“Experiment 1.4”**

Student Name: **SUMIT KUMAR** UID: **20BCS8226**

Branch: **CSE** Section/Group: **808-A**

Semester: **5** Date of Performance: **25-08-22**

Subject Name: **Design and Analysis of Algorithms Lab** Subject Code: **20CSP-312**

**Part 1.1**

**1. Aim/Overview of the practical:**

Code to Insert and Delete an element at the beginning and at end in Doubly Linked List.

**2. Algorithm/Flowchart (For programming based labs):**

**i. Insertion at beginning:**

START  
Step 1: If head = NULL,  
 initialize head->val = val, head->prev and head->next = NULL,  
 go to step 3.

Step 2: If head not = NULL,  
 intialize temp->val = val  
 temp->prev = NULL  
 temp->next = head  
 head->prev = temp  
 head = temp

Step 3: New node is inserted at beginning.  
END

**ii. Insertion at end:**

START  
Step 1: If head = NULL,  
 initialize head->val = val, head->prev and head->next = NULL,  
 go to step 4.

Step 2: If head not = NULL,  
 temp = head  
 while temp->next not = NULL, do temp = temp->next

Step 3: Intialize temp->next->val = val  
 temp->next->prev = temp  
 temp->next->next = NULL

Step 4: New node is inserted at end.  
END

**iii. Insertion after specific node:**

START  
Step 1: If head = NULL,  
 initialize head->val = val, head->prev and head->next = NULL,  
 go to step 4.

Step 2: If head not = NULL,  
 temp = head  
 while temp->val not = entry\_node, do temp = temp->next

Step 3: Intialize new\_node->val = val  
 temp->next->prev = new\_node  
 new\_node->next = temp->next  
 temp->next = new\_node, new\_node->prev = temp

Step 4: New node is inserted at end.  
END

**iv. Deletion at beginning:**

START  
Step 1: If head = NULL,  
 print Underflow  
 go to step 3.

Step 2: If head not = NULL,  
 head = head->next  
 head->prev = NULL

Step 3: New node is deleted at beginning, unless there is Underflow condition.  
END

**v. Deletion at end:**

START  
Step 1: If head = NULL,  
 print Underflow  
 go to step 3.

Step 2: If head not = NULL,  
 temp = head  
 while temp->next not = NULL, do node = temp->next  
 temp->prev->next = NULL

Step 3: New node is deleted at end, unless there is Underflow condition.  
END

**vi. Deletion of certain node:**

START  
Step 1: If head = NULL,  
 print Underflow  
 go to step 3.

Step 2: If head not = NULL,  
 temp = head  
 while temp->next not = req\_node, do node = temp->next  
 temp->next->prev = temp.prev  
 temp->prev->next = temp.next

Step 3: Required node is deleted, unless there is Underflow condition.  
END

**4. Steps for experiment/practical/Code:**

import java.util.\*;

class Node {

Node left, right;

int val;

Node(int val) {

this.val = val;

}

}

public class Test {

Node head;

void deleteend() {

Node node = head;

while(node.right!=null) {

node = node.right;

}

node.left.right = null;

}

void deletebeg() {

head = head.right;

head.left = null;

}

void delete(int val) {

Node node = head;

while(node.val!=val) {

node = node.right;

}

node.right.left = node.left;

node.left.right = node.right;

}

void insertbeg(int val) {

if(head==null) {

head = new Node(val);

head.left = null;

head.right = null;

return;

}

Node node = new Node(val);

node.left = null;

node.right = head;

head.left = node;

head = node;

}

void insertafter(int val, int x) {

Node node = head;

while(node.val!=val) {

node = node.right;

}

Node ins = new Node(x);

node.right.left = ins;

ins.right = node.right;

node.right = ins;

ins.left = node;

}

void insert(int val) {

if(head==null) {

head = new Node(val);

head.left = null;

head.right = null;

return;

}

Node node = head;

while(node.right!=null) {

node = node.right;

}

node.right = new Node(val);

node.right.left = node;

node.right.right = null;

}

void print() {

Node node = head;

while(node!=null) {

System.out.print(node.val+" ");

node = node.right;

}

}

public static void main(String args[]) {

Test ob = new Test();

Scanner in = new Scanner(System.in);

ob.insert(1);

ob.insert(2);

ob.insert(3);

ob.print();

ob.insertbeg(4);

System.out.println();

ob.print();

ob.insertafter(2, 5);

System.out.println();

ob.print();

ob.delete(1);

System.out.println();

ob.print();

ob.deletebeg();

System.out.println();

ob.print();

ob.deleteend();

System.out.println();

ob.print();

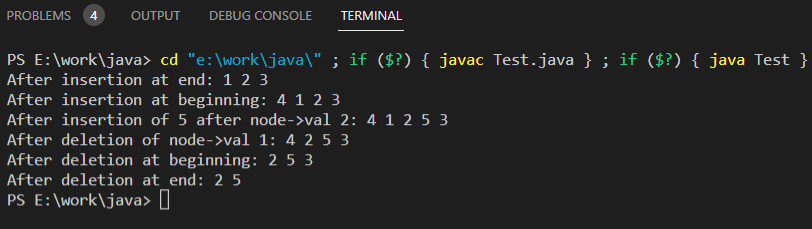
}

}

**5. Observations/Discussions/ Complexity Analysis:**

Insertion/Deletion at beginning happens in O(1) time. Insertion/Deletion at end or after will take O(n) time, but this can be done in O(1) time if we maintain a tail node, which will store the current last node in the list.

**6. Result/Output/Writing Summary:**



**Part 1.2**

**1. Aim/Overview of the practical:**

Code to Insert and Delete an element at the beginning and at end in Circular Linked List.

**2. Algorithm/Flowchart (For programming based labs):**

**i. Insertion at beginning:**

START  
Step 1: If head = NULL,  
 initialize head->val = val, head->next = head,  
 go to step 3.

Step 2: If head not = NULL,  
 temp = head->next  
 Initialize head->next->val = val  
 head->next->next = temp

Step 3: New node is inserted at beginning.  
END

**ii. Insertion at end:**

START  
Step 1: If head = NULL,  
 initialize head->val = val, head->next = head,  
 go to step 4.

Step 2: If head not = NULL,  
 temp = head  
 while temp->next not = head, do temp = temp->next

Step 3: Intialize temp->next->val = val  
 temp->next->prev = temp  
 temp->next->next = NULL

Step 4: New node is inserted at end.  
END

**iii. Deletion at beginning:**

START  
Step 1: If head = NULL,  
 print Underflow  
 go to step 2.

Step 2: If head->next = head  
 head = head->next = NULL

Step 3: If head not = NULL,  
 head->next = head->next->next

Step 4: New node is deleted at beginning, unless there is Underflow condition.  
END

**iv. Deletion at end:**

START  
Step 1: If head = NULL,  
 print Underflow  
 go to step 3.

Step 2: If head not = NULL,  
 temp = head  
 while temp->next->next not = head, do node = temp->next  
 temp->next = head

Step 3: New node is deleted at end, unless there is Underflow condition.  
END

**4. Steps for experiment/practical/Code:**

import java.util.\*;

class Node {

Node next;

int val;

Node(int val) {

this.val = val;

}

}

public class Test {

Node head;

void deletebeg() {

if(head==null) {

System.out.println("Underflow");

return;

}

else if(head.next==head) {

head = null;

head.next = null;

return;

}

head.next = head.next.next;

}

void deleteend() {

if(head==null) {

System.out.println("Underflow");

return;

}

else if(head.next==head) {

head = null;

head.next = null;

return;

}

Node node = head;

while(node.next.next!=head)

node = node.next;

node.next = head;

}

void insertend(int val) {

if(head==null) {

head = new Node(val);

head.next = head;

return;

}

Node node = head;

while(node.next!=head) {

node = node.next;

}

node.next = new Node(val);

node.next.next = head;

}

void insertbeg(int val) {

if(head==null) {

head = new Node(val);

head.next = head;

return;

}

Node temp = head.next;

head.next = new Node(val);

head.next.next = temp;

}

public void display() {

Node node = head;

while(node.next!=head) {

System.out.print(node.val+" ");

node = node.next;

}

System.out.println(node.val+"-> "+head.val+"...\n");

}

public static void main(String args[]) {

Test ob = new Test();

ob.insertend(1);

ob.insertend(2);

ob.insertend(3);

ob.insertend(4);

ob.insertend(5);

System.out.print("List after inserting elements at end: ");

ob.display();

System.out.print("After inserting 6 at the beginning: ");

ob.insertbeg(6);

ob.display();

System.out.print("After deleting at beginning: ");

ob.deletebeg();

ob.display();

System.out.print("After deleting at end: ");

ob.deleteend();

ob.display();

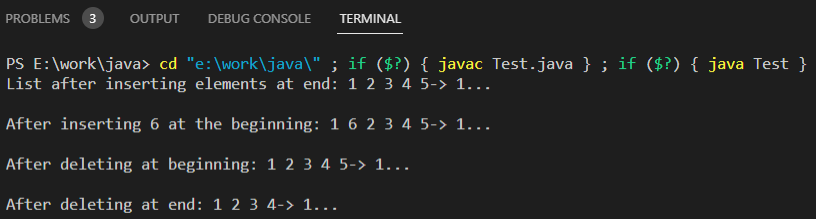
}

}

**5. Observations/Discussions/ Complexity Analysis:**

Insertion/Deletion at beginning happens in O(1) time. Insertion/Deletion at end will take O(n) time, but similar to doubly linked list, this can be done in O(1) time if we maintain a tail node, which will store the current last node in the list.

**6. Result/Output/Writing Summary:**



**Part 2**

**1. Aim:**

Code to push & pop and check Isempty, Isfull, and return top element instacks.

**2. Algorithm:**

**i. Push:**

START  
Step 1: If isFull() = true,  
 print Overflow  
 go to step 3.

Step 2: Else,  
 Initialize temp->val = val  
 temp->next = head  
 head = temp;  
 size = size + 1

Step 3: New element is pushed on top stack, unless there is overflow condition.  
END

**ii. Pop**

START  
Step 1: If isEmpty() = true,  
 print Underflow  
 go to step 3.

Step 2: Else,  
 head = head->next

Step 3: Top element is popped from stack, unless there is underflow condition.  
END

**iii. isFull**

START  
Step 1: If size = max  
 return true

Step 2: Else  
 return false  
END

**iv. isEmpty**

START  
Step 1: If size = 0  
 return true

Step 2: Else  
 return false  
END

**3. Code:**

class Node {

Node next;

int val;

Node(int val) {

this.val = val;

}

}

public class Test {

int h=0, max=5;

Node head=null;

void push(int val) {

if(h==max) {

System.out.println("Overflow");

return;

}

Node node = new Node(val);

node.next = head;

head = node;

h++;

}

void pop() {

if(h==0) {

System.out.println("Underflow");

return;

}

head = head.next;

h--;

}

int peek() {

return head.val;

}

void display() {

Node node = head;

while(node!=null) {

System.out.print(node.val+" ");

node = node.next;

}

System.out.println();

}

public static void main(String args[]) {

Test ob = new Test();

ob.push(1);

ob.push(2);

ob.push(3);

ob.push(4);

ob.push(5);

ob.display();

ob.push(6);

ob.pop();

ob.display();

ob.pop();

ob.display();

ob.pop();

ob.display();

ob.pop();

ob.display();

ob.pop();

ob.display();

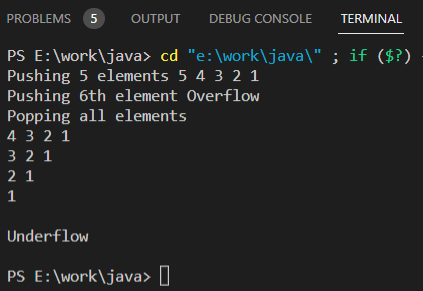
ob.pop();

ob.display();

}

}

**4. Output:**



**5. Complexity Analysis:**

All operations happen in O(1) as only 1 element is accessed whose location is always known, i.e. the first element.

**Learning outcomes (What I have learnt):**

1. Learnt about doubly and circular linked list data structures and their applications.

2. Learnt how implement linked lists in java using classes and objects in java.

3. Learnt about Stack data structure and how to implement it in a program using linked list in java.

**Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | Parameters | Marks Obtained | Maximum Marks |
| 1. |  |  |  |
| 2. |  |  |  |
| 3. |  |  |  |
|  |  |  |  |