**Practical 9**

**Name: Shantanu Sethi**

**Roll no.: 163**

**Aim: To implement Circular linked list**

**Objectives:**

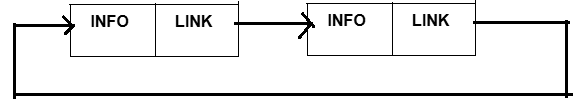
1. Learn how to implement different operations on linear linked list
   * Insert node: from front and from last
   * Delete node
   * Search list
   * Display
   * Count

**Theory:**

A linked list in which the last node of the list points to the first node in the list.

A circular list does not contain NULL pointers.

**Structure of node**

****

**Algorithm :**

**Insert from front:**

1. [ Initialize fields of new node]

Info (Node1 ) = item

2. [ Is the list is empty]

If ( first = NULL)

First = Node1

Link (Node1)= first

return (Node1)

3. [ insertion at the beginning of CLL]

move = first

temp = first

Link(Node1)= first

First= Node1

while(Link ( move) ≠ temp)

move=link(move)

link(move)= first

**Insert node form end:**

1. [ Initialize fields of new node]

info (Node1 ) = item

2. [ Is the list is empty]

If ( first = NULL)

First =Node1

link(Node1)=first

return (Node1)

3. [ find last node]

Move= first

while( link(move) ≠ first )

move=link (move)

4. [ Insert new node at last ]

Link(move)= Node1

Link(Node1 ) = first

5. return( first)

**Delete node from front:**

1. [ empty list ]

If (first = NULL)

then write (“ underflow”)

return

2. [ delete from front of CLL if CLL contain single element]

if(link(first)==first)

write (info(first ))

first=NULL

3. [initialize temp and move ]

temp=first

move=first

4. [ Delete first node]

write( info(first))

first= link(first)

5. Repeat while ( link(move) ≠ temp )

[ move to next node]

Move= link(move)

6. [ Delete Last Node ]

link(move)=first

**Delete from last node:**

1. [ empty list ]

If (first == NULL)

then write (“ underflow”)

return

2. [ Initialization ]

Move=first

3. [Find last node]

Repeat while (link(move) ≠ first)

1. [ Update predecessor Marker ]

Pred=move

2. [ move to next node]

Move=link(move)

4. [ Delete Last Node ]

write(info(move))

link(pred)=first

**Searching node in Link list:**

1. [ empty list ]

If (first=NULL)

then write (“ underflow”)

return

2. [ Initialize SAVE to first node of Link list and count to 1 ]

Save=first

C=1

3. [ count number of node by visiting every node]

Repeat while (link(save) ≠ first)

begin

C = C +1

If (info(save)=item)

begin

f= 1

return

end

save=link(save)

end

4. [ return position if element is found ]

if( f=1)

Write (“ element found”)

return ( C)

else

Write (“ element not found”)

Return

**Display node in LL:**

1. [ empty list ]

If first = NULL

then write (“ underflow”)

return

2. [ Initialize SAVE to first node of Link list ]

Save=first

3. [ Print info of every node]

Repeat while link(save) ≠ first

begin

write( info(save))

save=link(save)

End

Write (info(save))

**Count Node in CLL:**

1. [ empty list ]

If FIRST = NULL

then write (“ underflow”)

return

2. [ Initialize SAVE to first node of Link list and count to 1 ]

Save=first

C=1

3. [ count number of node by visiting every node]

Repeat while ( link(save) ≠ first)

begin

c = c +1

save=link(save)

end

4. [ return counter ]

Return ( C)

**Program:**

package Circular;

import java.util.Scanner;

public class Circular {

static class Node {

int info;

Node link;

Node(int item) {

info = item;

link = null;

}

}

Node first;

public Circular() {

first = null;

}

public void insertFront(int item) {

Node newNode = new Node(item);

if (first == null) {

first = newNode;

newNode.link = first;

} else {

Node move = first;

Node temp = first;

newNode.link = first;

first = newNode;

while (move.link != temp) {

move = move.link;

}

move.link = first;

}

}

public void insertEnd(int item) {

Node newNode = new Node(item);

if (first == null) {

first = newNode;

newNode.link = first;

} else {

Node move = first;

while (move.link != first) {

move = move.link;

}

move.link = newNode;

newNode.link = first;

}

}

public void deleteFront() {

if (first == null) {

System.out.println("Underflow: List is empty");

return;

}

if (first.link == first) {

System.out.println("Deleted: " + first.info);

first = null;

} else {

Node temp = first;

Node move = first;

System.out.println("Deleted: " + first.info);

first = first.link;

while (move.link != temp) {

move = move.link;

}

move.link = first;

}

}

public void deleteEnd() {

if (first == null) {

System.out.println("Underflow: List is empty");

return;

}

if (first.link == first) {

System.out.println("Deleted: " + first.info);

first = null;

} else {

Node move = first;

while (move.link != first) {

move = move.link;

}

System.out.println("Deleted: " + move.info);

Node pred = first;

while (pred.link != move) {

pred = pred.link;

}

pred.link = first;

}

}

public void searchNode(int item) {

if (first == null) {

System.out.println("Underflow: List is empty");

return;

}

Node save = first;

int count = 1;

boolean found = false;

do {

if (save.info == item) {

found = true;

System.out.println("Element found at position: " + count);

return;

}

save = save.link;

count++;

} while (save != first);

if (!found) {

System.out.println("Element not found");

}

}

public void displayNodes() {

if (first == null) {

System.out.println("Underflow: List is empty");

return;

}

Node save = first;

do {

System.out.print(save.info + " ");

save = save.link;

} while (save != first);

System.out.println();

}

public int countNodes() {

if (first == null) {

System.out.println("Underflow: List is empty");

return 0;

}

int count = 1;

Node save = first;

while (save.link != first) {

count++;

save = save.link;

}

return count;

}

public static void main(String[] args) {

System.out.println("Abhinav Singh--184");

Circular cll = new Circular();

Scanner scanner = new Scanner(System.in);

int choice;

int item;

do {

System.out.println("\nCircular Linked List Operations:");

System.out.println("1. Insert at Front");

System.out.println("2. Insert at End");

System.out.println("3. Delete from Front");

System.out.println("4. Delete from End");

System.out.println("5. Search Node");

System.out.println("6. Display Nodes");

System.out.println("7. Count Nodes");

System.out.println("8. Exit");

System.out.print("Enter your choice: ");

choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.print("Enter value to insert at front: ");

item = scanner.nextInt();

cll.insertFront(item);

break;

case 2:

System.out.print("Enter value to insert at end: ");

item = scanner.nextInt();

cll.insertEnd(item);

break;

case 3:

cll.deleteFront();

break;

case 4:

cll.deleteEnd();

break;

case 5:

System.out.print("Enter value to search: ");

item = scanner.nextInt();

cll.searchNode(item);

break;

case 6:

System.out.print("Circular Linked List: ");

cll.displayNodes();

break;

case 7:

System.out.println("Number of nodes: " + cll.countNodes());

break;

case 8:

System.out.println("Exiting...");

break;

default:

System.out.println("Invalid choice! Please try again.");

}

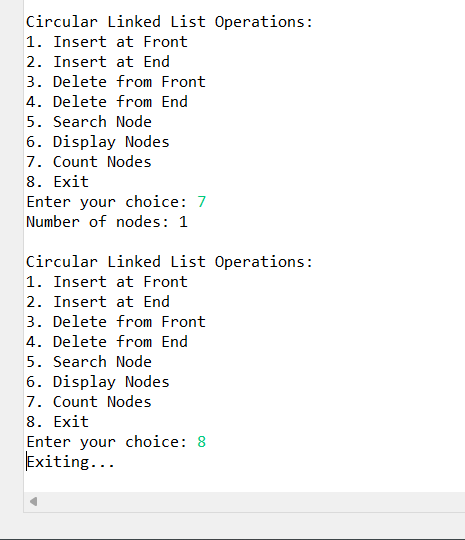
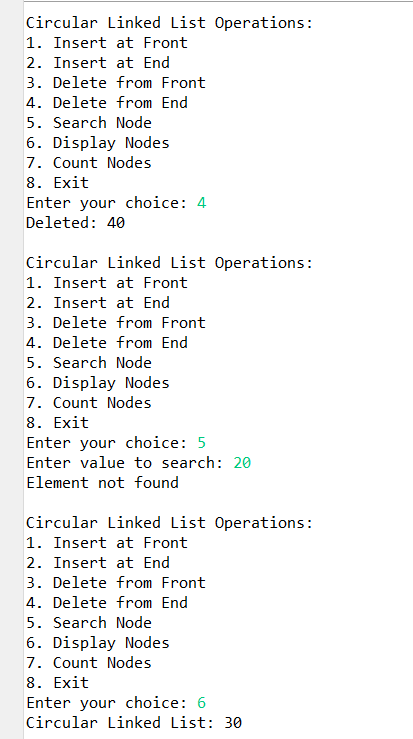
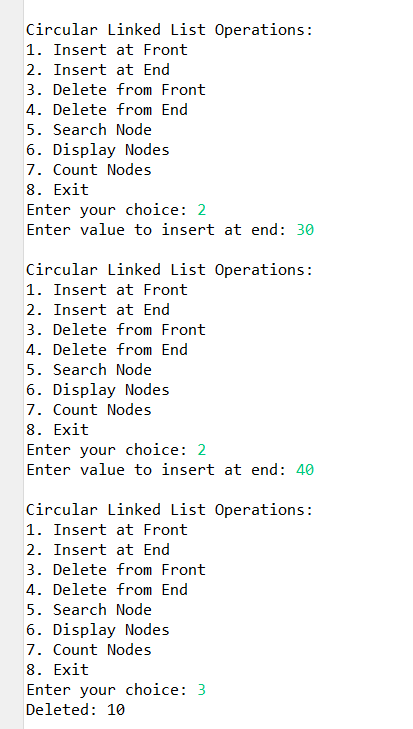
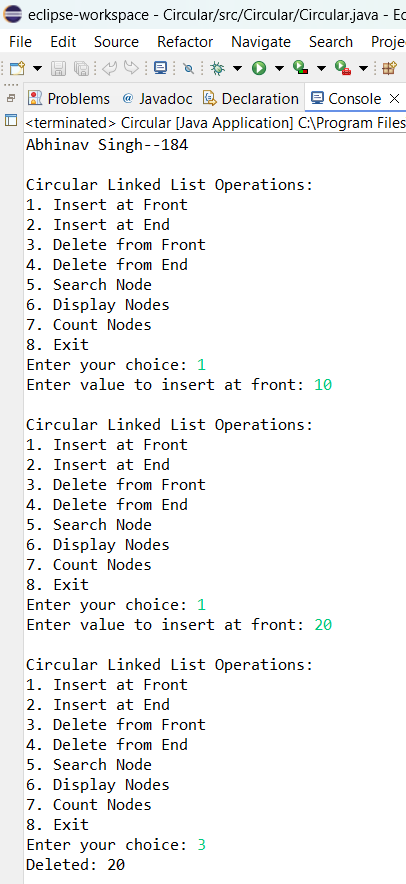
} while (choice != 8);

scanner.close();

}

}

**Output:**

****

**Conclusion:** Successfully implemented different operation on CIRCULAR linked list