Department of Electronics & Telecommunication Engineering
Data Structures & Algorithms Lab ((DJS22EL505)

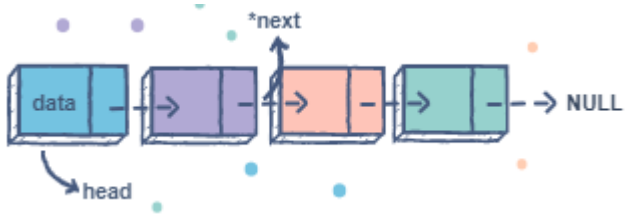
Experiment :5

Aim: To implement Singly linked list

Programming Language: C/C++/Java

Theory: A singly linked list is a type of linked list that is *unidirectional*, that is, it can be traversed in only one direction from head to the last node (tail). Each element in a linked list is called a node. A single node contains *data* and a pointer to the *next* node which helps in maintaining the structure of the list.

The first node is called the head; it points to the first node of the list and helps us access every other element in the list. The last node, also sometimes called the tail, points to *NULL* which helps us in determining when the list ends.



Declaring a Linked list :

In C language, a linked list can be implemented using structure and pointers.

```
struct node
{
int data;
struct node *next;
};
```

In Java language, a node can be created as separate class.

```
class Node
 {
  int data;
  Node next;
Node (int x)
  {
   data = x;
   next = null;
  }  }
```

Department of Electronics & Telecommunication Engineering
Data Structures & Algorithms Lab ((DJS22EL505)

Inserting a new node in a linked list

Case 1: The new node is inserted at the beginning

1. Create and allocate memory (only in C/C++) for new node.
2. Store data
3. Check if it is the first node of the linked list then set next of new node to point to null and set head to point to recently created node
4. If it is not the first node then, set next of new node to point to head and Change head to point to recently created node

Case 2: The new node is inserted at the end

1. Create and allocate memory (only in C/C++) for new node
2. Store data
3. Check if it is the first node of the linked list then set next of new node to point to null and set head to point to recently created node
4. Else, Traverse to the last node using a temp pointer which initially points to head node, set next field of last node to recently created node and next of new node to point to null value.

Case 3: The new node is inserted at specific position

1. Create and allocate memory (only in C/C++) for new node
2. Store data
3. Specify the position where the node has to be inserted.
4. Use a temp pointer to traverse to that position. Use q pointer which points to the node which is just ahead of the temp node.

5. Set next value of temp pointer to new node pointer.

6. Set next value of new node pointer to q pointer and add the node.

printing/displaying a linked list

1. Define a temp pointer which initially points to the head node. Traverse the linked list using a temp pointer till the temp becomes null and display the data values of the node

Deleting a new node from a linked list

Case 1: The first node is deleted

1. Check if head is pointing to null value it means linked list is empty. Then print "underflow" and exit from the function.
2. Else define a temp pointer, set its value to head pointer. If next field of temp is NULL value it means it is the last node of the linked list. Delete it and free the memory.
3. Else, Shift the head pointer to next node. Delete the node where temp pointer is pointing and then free the memory (only in C/C++) using free command

Case 2: The last node is deleted

1. Check if head is pointing to null value it means linked list is empty. Then print "underflow" and exit from the function.

2. Else define a temp pointer, set its value to head pointer. If next field of temp is NULL value it means it is the last node of the linked list. Delete it and free the memory.
3. Else, Traverse through the linked list using temp pointer (increment temp until the last node). Use q pointer which points to the node which is just ahead of the temp node.
4. Set next value of q pointer to null value.
5. Delete the node where temp pointer is pointing and then free (only in C/C++) the memory using free command.

Case 3: Delete the node from a given position

1. Check if head is pointing to null value it means linked list is empty. Then print "underflow" and exit from the function.
2. Specify the position of the node to be deleted
3. Use a temp pointer to traverse to the position of the node to be deleted. Use q pointer which points to the node which is just ahead of the temp node

4. Set next value of q pointer to next value of temp pointer

5. Delete the temp node and free the memory

Code:

```
import java.util.Scanner;
class Node  {
int data;
Node next;
public Node(int data)  {
this.data = data;
this.next = null;
        }
}

public class LinkedList  {
Node head = null;
```

```
public void insertAtBeginning(int data)  {
Node newNode = new Node(data);
newNode.next = head;
head = newNode;
System.out.println("Inserted " + data + " at the beginning");
}

public void insertAtEnd(int 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;
            }
System.out.println("Inserted " + data + " at the end");
}

public void insertAtPosition(int data, int position)  {
Node newNode = new Node(data);
if (position == 1)   {
newNode.next = head;
head = newNode;
}  else   {
Node temp = head;
int count = 1;
while (temp != null && count < position - 1)   {
temp = temp.next;
count++;
}
```

```
if (temp == null) {
System.out.println("Position out of bounds");
} else {
newNode.next = temp.next;
temp.next = newNode;
System.out.println("Inserted " + data + " at position " + position);
}
}
}

public void deleteFirst() {
if (head == null) {
System.out.println("List is empty");
} else {
System.out.println("Deleted " + head.data + " from the beginning");
head = head.next;
}
}

public void deleteLast() {
if (head == null) {
System.out.println("List is empty");
} else if (head.next == null) {
System.out.println("Deleted " + head.data + " from the end");
head = null;
} else {
Node temp = head;
while (temp.next.next != null) {
temp = temp.next;
}
System.out.println("Deleted " + temp.next.data + " from the end");
```

```
temp.next = null;
}
}

public void deleteAtPosition(int position) {
if (head == null) {
System.out.println("List is empty");
} else if (position == 1) {
System.out.println("Deleted " + head.data + " from position " + position);
head = head.next;
} else {
Node temp = head;
int count = 1;
while (temp != null && count < position - 1) {
temp = temp.next;
count++;
}
if (temp == null || temp.next == null) {
System.out.println("Position out of bounds");
} else {
System.out.println("Deleted " + temp.next.data + " from position " + position);
temp.next = temp.next.next;
}
}
}

public void display() {
if (head == null) { System.out.println("List is empty"); }
else {
Node temp = head;
System.out.println("Elements in the list:");
```

```
while (temp != null)  {
System.out.print(temp.data + " -> ");
temp = temp.next;
}
System.out.println("null");
}
}
public static void main(String[] args)
{
System.out.println("Jay and Janay");
LinkedList list = new LinkedList();
Scanner sc = new Scanner(System.in);
while (true)  {
System.out.println("1. Insert at Beginning\n2. Insert at End\n3. Insert at Position\n4. Delete First\n5. Delete Last\n6. Delete at Position\n7. Display\n8. Exit");
int choice = sc.nextInt();
switch (choice)  {
case 1:
System.out.println("Enter value to insert at the beginning:");
int valueBegin = sc.nextInt();
list.insertAtBeginning(valueBegin);
break;
case 2:
System.out.println("Enter value to insert at the end:");
int valueEnd = sc.nextInt();
list.insertAtEnd(valueEnd);
break;
case 3:
System.out.println("Enter value to insert:");
int valuePos = sc.nextInt();
System.out.println("Enter position to insert:");
```

```
int positionInsert = sc.nextInt();
list.insertAtPosition(valuePos, positionInsert);
break;
case 4:
list.deleteFirst();
break;
case 5:
list.deleteLast();
break;
case 6:
System.out.println("Enter position to delete:");
int positionDelete = sc.nextInt();
list.deleteAtPosition(positionDelete);
break;
case 7:
list.display();
break;
case 8:
sc.close();
System.out.println("Exit");
return;
default:
System.out.println("Invalid");
}
}
}
}
```

Department of Electronics & Telecommunication Engineering
Data Structures & Algorithms Lab ((DJS22EL505)

```
Jay and Janay
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete at Position
7. Display
8. Exit
1
Enter value to insert at the beginning:
12
Inserted 12 at the beginning
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete at Position
7. Display
8. Exit
2
Enter value to insert at the end:
13
Inserted 13 at the end
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete at Position
7. Display
8. Exit
2
Enter value to insert at the end:
15
Inserted 15 at the end
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete at Position
7. Display
8. Exit
```

```
3
Enter value to insert:
33
Enter position to insert:
2
Inserted 33 at position 2
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete at Position
7. Display
8. Exit
7
Elements in the list:
12 -> 33 -> 13 -> 15 -> null
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete at Position
7. Display
8. Exit
4
Deleted 12 from the beginning
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete at Position
7. Display
8. Exit
5
Deleted 15 from the end
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete at Position
7. Display
8. Exit
7
```

```
Elements in the list:
33 -> 13 -> null
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete at Position
7. Display
8. Exit
6
Enter position to delete:
1
Deleted 33 from position 1
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete at Position
7. Display
8. Exit
7
Elements in the list:
13 -> null
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete Last
6. Delete at Position
7. Display
8. Exit
8
Exit
```