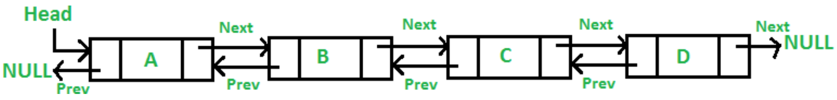
Name	Virinchi Sadashiv Shettigar
UID no.	2021300118
Experiment No.	4

AIM:	Implement a given problem statement using Doubly Linked List.
PROBLEM STATEMENT:	Insert and Delete operations at a given position

THEORY:	**DISADVANTAGES OF SINGLY LINKED LIST**
	1. It requires more space as pointers are also stored with information.
	2. Different amount of time is required to access each element.
	3. If we have to go to a particular element then we have to go through all those elements that come before that element.
	4. We cannot traverse it from the last & only from the beginning.
	5. It is not easy to sort the elements stored in the linear linked list.
	To counter some of these linked lists, we introduce the concept of a Doubly Linked List.
	
	DOUBLY LINKED LIST
	A Doubly Linked List (DLL) contains an extra pointer, typically called the previous pointer, together with the next pointer and data which are there in the singly linked list.
	 Head
	Next
	Next
	Next
	Next
	NULL
	NULL
	A
	B
	C
	D
	Prev
	Prev
	Prev
	Prev
	Syntax of a
	DLL: public
	class DLL {
	
	// Head of
	list Node
	head;
	
	// Doubly Linked list
	Node class Node {
	int data;

```
        Node
        prev;
Node next
        // Constructor to create a new node
        // next and prev is by default initialized as
        null Node(int d) { data = d; }
    }
 }
```

Advantages of DLL over the singly linked list:

1. A DLL can be traversed in both forward and backward directions.
2. The delete operation in DLL is more efficient if a pointer to the node to be deleted is given.
3. We can quickly insert a new node before a given node.
4. In a singly linked list, to delete a node, a pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using the previous pointer.

Disadvantages of DLL over the singly linked list:

1. Every node of DLL Requires extra space for a previous pointer. It is possible to implement DLL with a single pointer though (See this and this).
2. All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify previous pointers together with the next pointers. For example in the following functions for insertions at different positions, we need 1 or 2 extra steps to set the previous pointer.

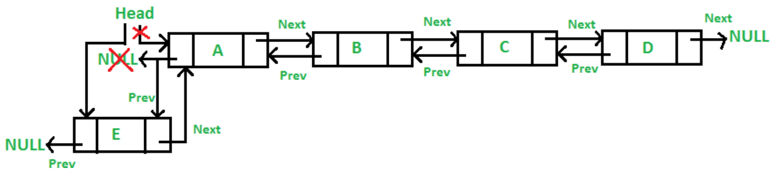
INSERTION IN DLL:

A node can be added in four ways:

1. At the front of the DLL
2. After a given node.
3. At the end of the DLL
4. Before a given node.

ADD AT THE FRONT:

The new node is always added before the head of the given Linked List. And newly added node becomes the new head of DLL. For example, if the given Linked List is 1->0->1->5 and we add an item 5 at the front, then the Linked List becomes 5->1-
>0->1->5. Let us call the function that adds at the front of the list push(). The push() must receive a pointer to the head pointer because the push must change the head pointer to point to the new node.

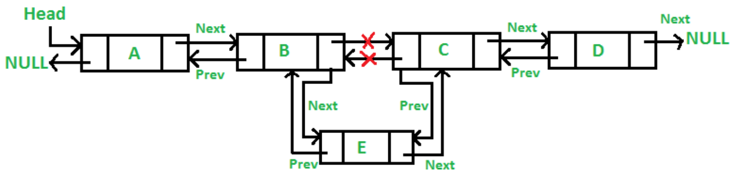


Below is the implementation of the 5 steps to insert a node at the front of the linked list:

```
public void push(int new_data)
{
    /* 1. allocate node
     * 2. put in the data */
    Node new_Node = new Node(new_data);

    /* 3. Make next of new node as head and previous as NULL
     */
    new_Node.next =
    head; new_Node.prev
    = null;

    /* 4. change prev of head node to new
    node */ if (head != null)
        head.prev = new_Node;

    /* 5. move the head to point to the new
    node */ head = new_Node;
}
```

ADD A NODE AFTER A GIVEN NODE

We are given a pointer to a node as prev_node, and the new node is inserted after the given node.

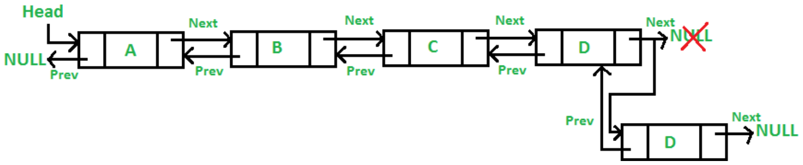


Below is the implementation of the 7 steps to insert a node after a given node in the linked list:

```
public void InsertAfter(Node prev_Node, int new_data)
{
    /*1. check if the given prev_node is
    NULL */ if (prev_Node == null) {
        System.out.println(
            "The given previous node cannot be NULL
        "); return;
    }
    /* 2. allocate node
     * 3. put in the data */
    Node new_node = new Node(new_data);
    /* 4. Make next of new node as next of
    prev_node */ new_node.next = prev_Node.next;

    /* 5. Make the next of prev_node as
    new_node */ prev_Node.next = new_node;

    /* 6. Make prev_node as previous of
    new_node */ new_node.prev = prev_Node;

    /* 7. Change previous of new_node's next
    node */ if (new_node.next != null)
        new_node.next.prev = new_node;
}
```

ADD A NODE AT THE END

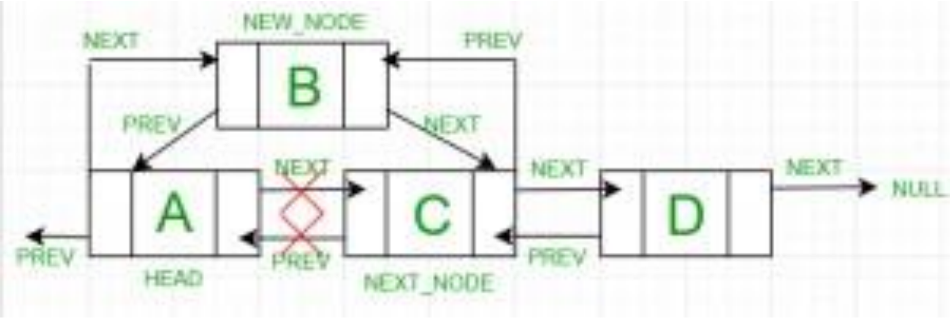
The new node is always added after the last node of the given Linked List. For example, if the given DLL is 5->1->0->1->5->2 and we add item 30 at the end, then the DLL becomes 5->1->0->1->5->2->30. Since a Linked List is typically represented by its head of it, we have to traverse the list till the end and then change the next of last node to the new node.



Below is the implementation of the 7 steps to insert a node at the end of the linked list:

```
void append(int new_data)
{
    /* 1. allocate node
     * 2. put in the data */
Node new_node = new Node(new_data);

    Node last = head; /* used in step 5*/
    /* 3. This new node is going to be the last node, so
     * make next of it as
    NULL*/ new_node.next =
    null;

    /* 4. If the Linked List is empty, then make the new
     * node as
    head */ if (head
    == null) {
        new_node.prev =
        null; head =
        new_node; return;
    }

    /* 5. Else traverse till the last
    node */ while (last.next != null)
        last = last.next;

    /* 6. Change the next of last
    node */ last.next = new_node;

    /* 7. Make last node as previous of new
    node */ new_node.prev = last;
}
```

ADD A NODE BEFORE A GIVEN NODE



Follow the below steps to solve the problem:

1. Let the pointer to this given node be next_node and the data of the new node be added as new_data.

2. Check if the next_node is NULL or not. If it's NULL, return from the function because any new node can not be added before a NULL
3. Allocate memory for the new node, let it be called new_node
4. Set new_node->data = new_data

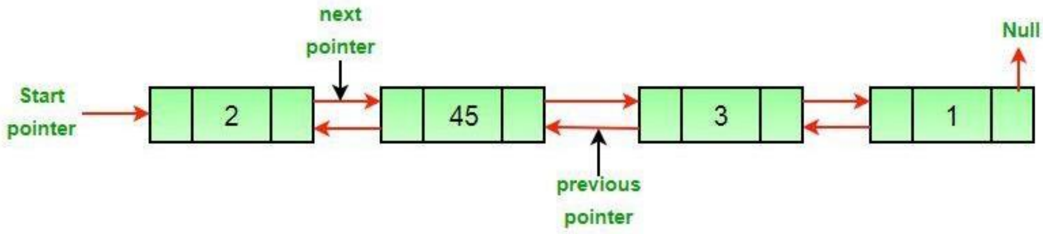
Set the previous pointer of this new_node as the previous node of the next_node, new_node->prev = next_node->prev

6. Set the previous pointer of the next_node as the new_node, next_node->prev = new_node
7. Set the next pointer of this new_node as the next_node, new_node->next = next_node;
8. If the previous node of the new_node is not NULL, then set the next pointer of this previous node as new_node, new_node->prev->next = new_node
9. Else, if the prev of new_node is NULL, it will be the new head node. So, make (*head_ref) = new_node.

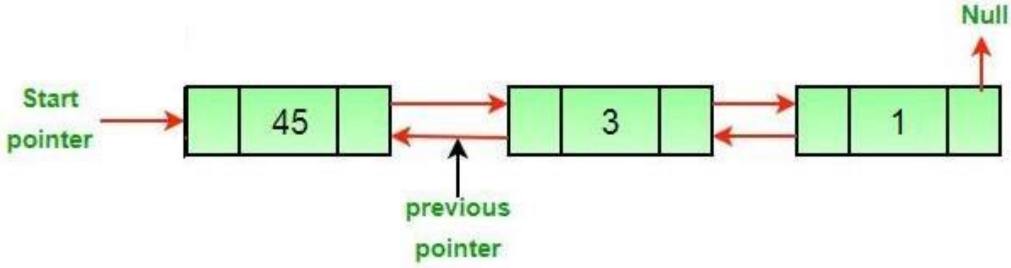
```
Below is the implementation of the steps to insert the
node: public void InsertBefore(Node next_node, int
new_data)
  {
    /*Check if the given nx_node is
    NULL*/ if (next_node == null) {
      System.out.println(
        "The given next node can not be
      NULL"); return;
    }

    // Allocate node, put in the data
    Node new_node = new Node(new_data);

    // Making prev of new node as prev of next
    node new_node.prev = next_node.prev;

    // Making prev of next node as new
    node next_node.prev = new_node;

    // Making next of new node as next
    node new_node.next = next_node;

    // Check if new node is added as
    head if (new_node.prev != null)
      new_node.prev.next =
    new_node; else
      head = new_node;
  }
```

DELETION IN DLL

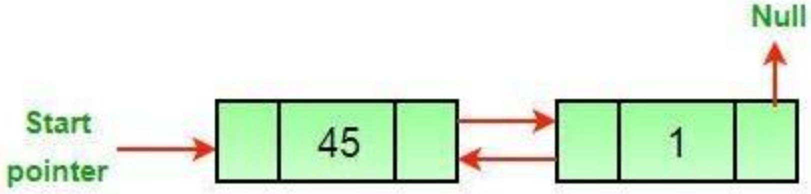
The deletion of a node in a doubly-linked list can be divided into three main categories:



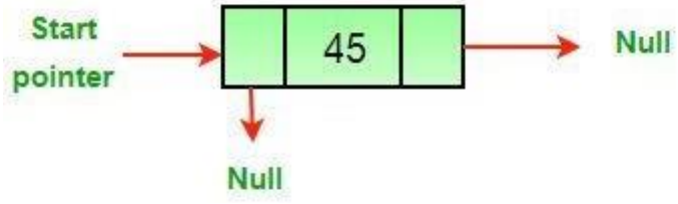
After the deletion of the head node.



After the deletion of the middle node.



After the deletion of the last node.



All three mentioned cases can be handled in two steps if the pointer of the node to be deleted and the head pointer is known.

1. If the node to be deleted is the head node then make the next node as head.
2. If a node is deleted, connect the next and previous node of the deleted node.

Algorithm:

```
Let the node to be deleted be del.
If node to be deleted is head node, then change the head pointer to next
current head.
if headnode == del then
     headnode =
     del.nextNode
Set prev of next to del, if next to del
exists. if del.nextNode != none
     del.nextNode.previousNode =
del.previousNode Set next of previous to del, if
previous to del exists. if del.previousNode !=
none
     del.previousNode.nextNode = del.next
```

APPLICATIONS OF DOUBLY LINKED LIST

1. Doubly linked list can be used in navigation systems where both forward and backward traversal is required.
2. It can be used to implement different tree data structures.
3. It can be used to implement undo/redo operations.

REAL APPLICATIONS OF DOUBLY LINKED LIST

1. Doubly linked lists are used in web page navigation in both forward and backward directions.

It can be used in games like a deck of cards.

ALGORITHM:

```
Exp4 Class
   Main function
          Main function
                   1. Create a DoublyLinkedList object obj
                   2. Display menu for menu driven program
                   3. Input choice from user
                   4. If choice is 1, input data from user and call insertAtFront(data)
                   5. If choice is 2, input data from user and call insertAtEnd(data)
                   6. If choice is 3, call deleteAtFront()
```

```
        7.  If choice is 4, call deleteAtEnd()
        8.  If choice is 5, input data and position from user and call
            insertAtLeft(data,position)
        9.  If choice is 6, input data and position from user and call
            insertAtRight(data, position)
        10. If choice is 7, input position from user and call
            deleteAtLeft(position)
        11. If choice is 8, input position from user and call
            deleteAtRight(position)
        12. Repeat steps from 2 to 11 until user chose to exit the program

doublyLinkedList class
  Node class
    Data Members: char data,Node next, Node prev
    Constructor Node(char data)
      set data to data and prev and next to null
  Data Members
    Node head

Void insertAtFront(char data)
   1. Create Node object newNode using new newNode(data)
   2. If head equals null, assign head to newNode
   3. Else, assign next of newNode to head, prev of head to newNode and
      head to newNode

Void insertAtEnd(char data)
    1. Create Node object newNode using new newNode(data)
    2. Create Node object temp and assign it to head
    3. If head equals null, assign head to newNode
    4. Else run a while loop until next of temp becomes null
    5. In this while loop, assign temp to next of temp
    6. Outside this while loop, assign next of temp to newNode and prev
       of newNode to temp.

Int deleteAtFront()
    1. Initialize char x=0.
    2. If head equals null, print list is empty.
    3. Else, set x to data of head, head becomes next of head,
    4. Check if head is not equal to null. If true, then set prev of head to null
    5. Return x

char deleteAtEnd()
     1. If head equals null, print list is empty
     2. Else, create Node object temp and set it to head.
     3. Run a while loop till next of temp equals null
     4. Set x to data of temp, set next of prev of temp to null
     5. Return x
```

Void insertAtLeft(char data,int position)

1. Initialize int c=0
2. Create Node object newNode using new newNode(data)
3. Create node object temp and assign it to head
4. Run a while loop until c does not become equal to position -2
5. In this while loop increment c and point the temp to next of temp
6. Outside the while loop, assign previous of object newNode to temp
7. Assign next of newNode to next of temp
8. The previous of next of temp is set to a newNode and next of temp to newNode.

Void insertAtRight(char data,int position)

1. Initialize int c=0
2. Create Node object newNode using new newNode(data)
3. Create node object temp and assign it to head
4. Run a while loop until c does not become equal to position -1
5. In this while loop increment c and point the temp to next of temp
6. Outside the while loop, assign previous of object newNode to temp
7. Assign next of newNode to next of temp
8. The previous of next of temp is set to a newNode and next of temp to newNode.

Void deleteAtLeft(int position)

1. Initialize int c=0
2. Create node object temp and assign it to head
3. Run a while loop until c does not become equal to position -3
4. In this while loop increment c and point the temp to next of temp
5. The previous of next of next of temp is set to a temp and next of next of temp to next of temp

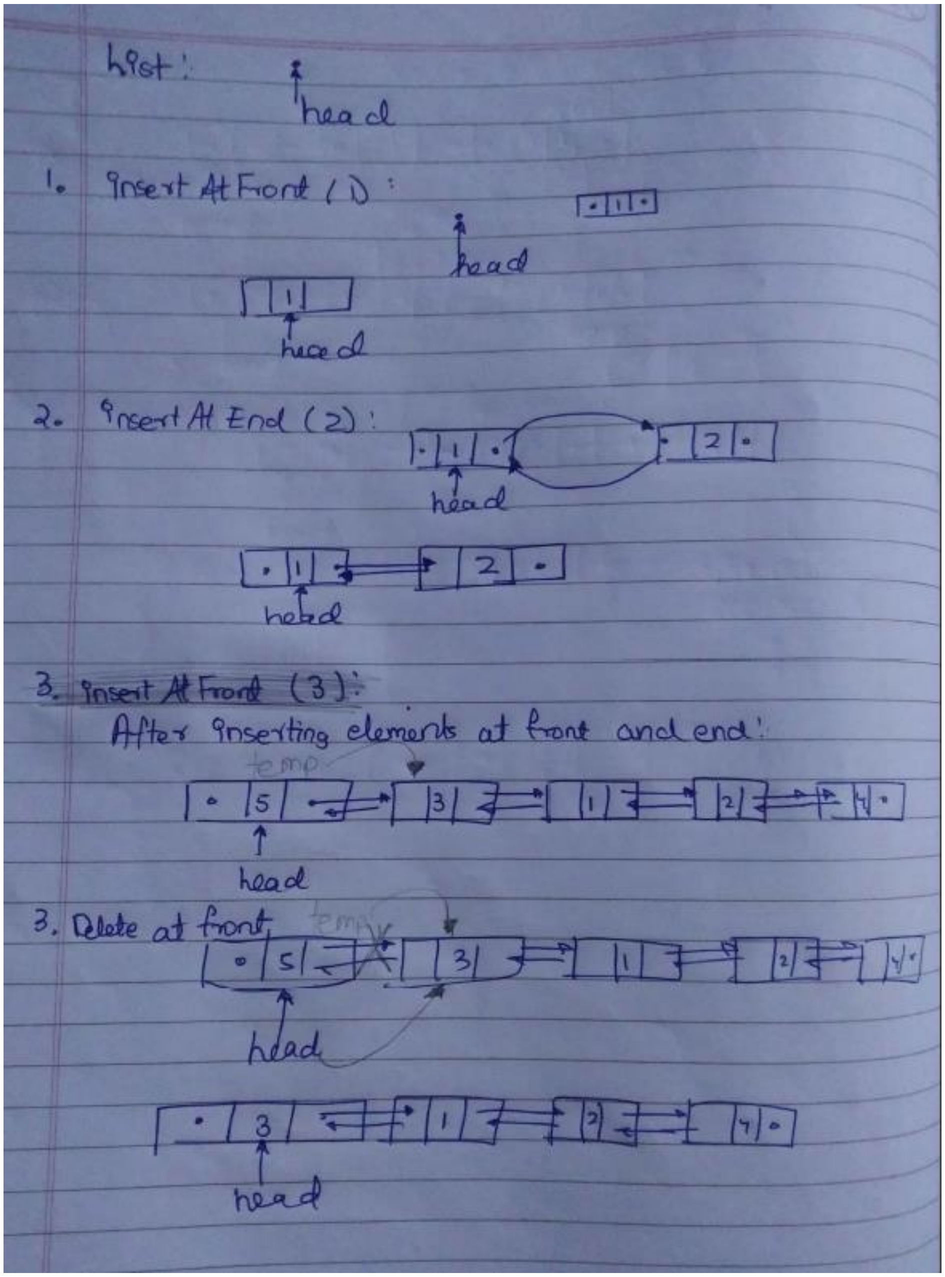
Void deleteAtRight(int position)

1. Initialize int c=0
2. Create node object temp and assign it to head
3. Run a while loop until c does not become equal to position -1
4. In this while loop increment c and point the temp to next of temp
5. The previous of next of next of temp is set to a temp and next of next of temp to next of temp

Void display

1. Create node object temp and set it to head
2. Run a while loop until next of temp becomes null
3. Inside while loop print “(temp.data)+” and set temp to next of temp
4. Print (“ “)

PROBLEM-SOLVING:



List:

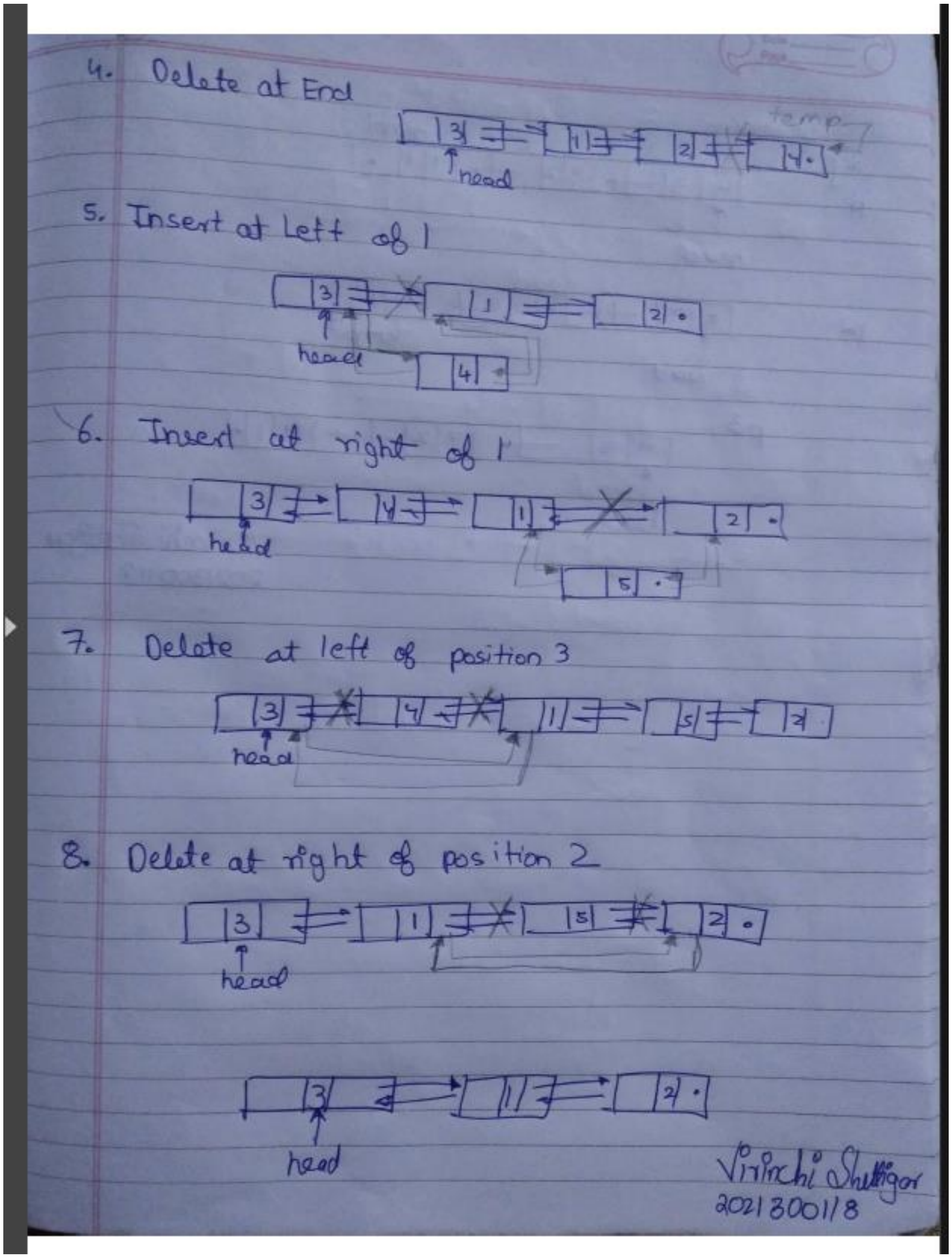
1. Insert At Front (1):

2. Insert At End (2):

3. Insert At Front (3):

After inserting elements at front and end:

3. Delete at front



4. Delete at End

5. Insert at Left of 1

6. Insert at right of 1

7. Delete at left of position 3

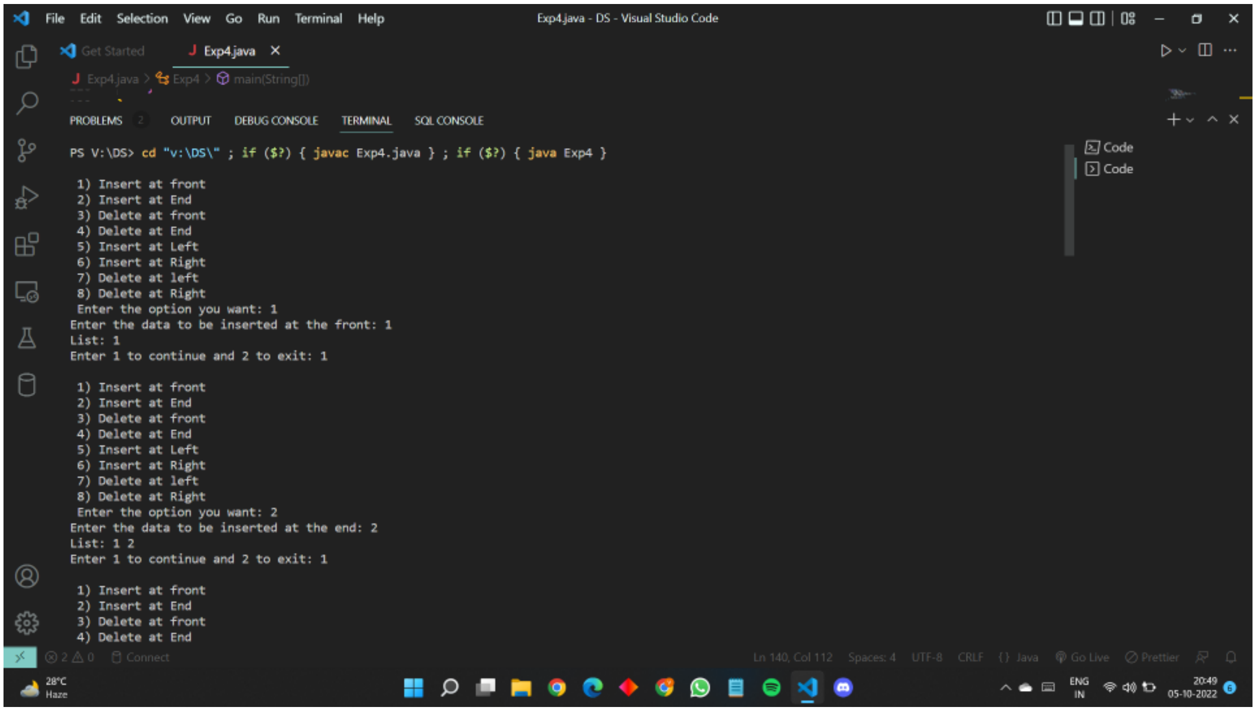
8. Delete at right of position 2

Virinchi Shettigar
2021300118

PROGRAM:

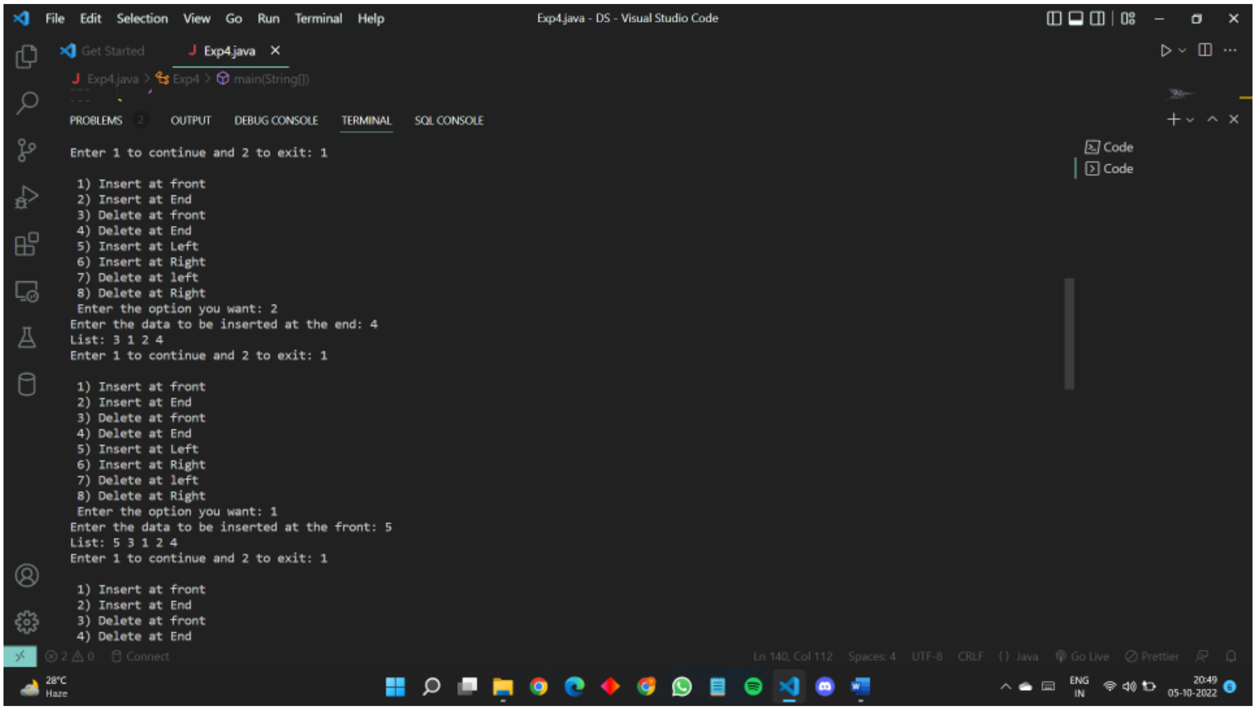
```
import java.util.*;

class doublyLinkedList {
    class Node {
        char data;
        Node next;
        Node prev;

        Node(char data) {
            this.next = null;
            this.prev = null;
            this.data = data;
        }
    }

    Node head;

    void insertAtFront(char data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
        } else {
            newNode.next = head;
            head.prev = newNode;
            head = newNode;
        }
    }

    void insertAtEnd(char data) {
        Node temp = head;
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
        } else {
            while (temp.next != null) {
                temp = temp.next;
            }
            temp.next = newNode;
            newNode.prev = temp;
            newNode.next = null;
        }
    }

    char deleteAtFront() {
        char x = 0;
        if (head == null) {
            System.out.println("List is empty");
            return 0;
        } else {
            x = head.data;
            Node temp = head;
            head = head.next;
            head.prev = null;
            temp.next = null;
            return x;
    }
}

char deleteAtEnd() {
    if (head == null) {
        System.out.println("List is empty");
        return 0;
    } else {
        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        char x = temp.data;
        temp.prev.next = null;
        return x;
    }
}

public void insertAtLeft(char data, int position) {
    int c = 0;
    Node newNode = new Node(data);
    Node temp = head;
    while (c != position - 2) {
        c++;
        temp = temp.next;
    }
    newNode.prev = temp;
    newNode.next = temp.next;
    temp.next.prev = newNode;
    temp.next = newNode;
}

public void insertAtRight(char data, int position) {
    int c = 0;
    Node newNode = new Node(data);
    Node temp = head;
    while (c != position - 1) {
        c++;
        temp = temp.next;
    }
    newNode.prev = temp;
    newNode.next = temp.next;
    temp.next.prev = newNode;
    temp.next = newNode;
}
public void deleteAtLeft(int position){
        int c=0;
        Node temp=head;
        while (c!=position-3) {
            c++;
            temp=temp.next;
        }
        temp.next.next.prev=temp;
        temp.next=temp.next.next;
    }
    public void deleteAtRight(int position){
        int c=0;
        Node temp=head;
        while (c!=position-1) {
            c++;
            temp=temp.next;
        }
        temp.next.next.prev=temp;
        temp.next=temp.next.next;
    }
    void display() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.print(" ");
    }
}

public class Exp4 {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int i;
        char num;
        doublyLinkedList d = new doublyLinkedList();
        int flag = 0;
        while (true) {
            System.out.print(
                    "\n 1)Insert at front \n 2) Insert at End \n 3)Delete at front \n 4)
Delete at End \n 5)Insert at Left \n 6) Insert at Right \n 7) Delete at left \n 8)
Delete at Right");
            System.out.print("\n Enter the option you want: ");
            int option = scan.nextInt();
            switch (option) {
                case 1:
                    System.out.print("Enter the data to be inserted at the front: ");
                    char data = scan.next().charAt(0);
                    d.insertAtFront(data);
                    System.out.print("List: ");
                    d.display();
                    break;
                case 2:
                    System.out.print("Enter the data to be inserted at the end: ");
                    data = scan.next().charAt(0);
                    d.insertAtEnd(data);
                    System.out.print("List: ");
                    d.display();
                    break;
                case 3:
                    char x = d.deleteAtFront();
                    d.display();
                    break;
                case 4:
                    x = d.deleteAtEnd();
                    d.display();
                    break;
                case 5:
                    System.out.print("Enter data to be inserted at the left: ");
                    num = scan.next().charAt(0);
                    System.out.print("Enter the position: ");
                    i = scan.nextInt();
                    d.insertAtLeft(num, i);
                    d.display();
                    break;
                case 6:
                    System.out.print("Enter data to be inserted at the right: ");
                    num = scan.next().charAt(0);
                    System.out.print("Enter the position: ");
                    i = scan.nextInt();
                    d.insertAtRight(num, i);
                    d.display();
                    break;
                case 7:
                    System.out.print("Enter position: ");
                    i=scan.nextInt();
                    d.deleteAtLeft(i);
                    d.display();
                    break;
                case 8:
                    System.out.print("Enter position: ");
                    i=scan.nextInt();
                    d.deleteAtRight(i);
                    d.display();
                    break;
                default:
                    break;
            }
            System.out.print("\nEnter 1 to continue and 2 to exit: ");
            flag = scan.nextInt();
            if (flag == 2) {
                break;
            }
        }
    }
}
```

OUTPUT:



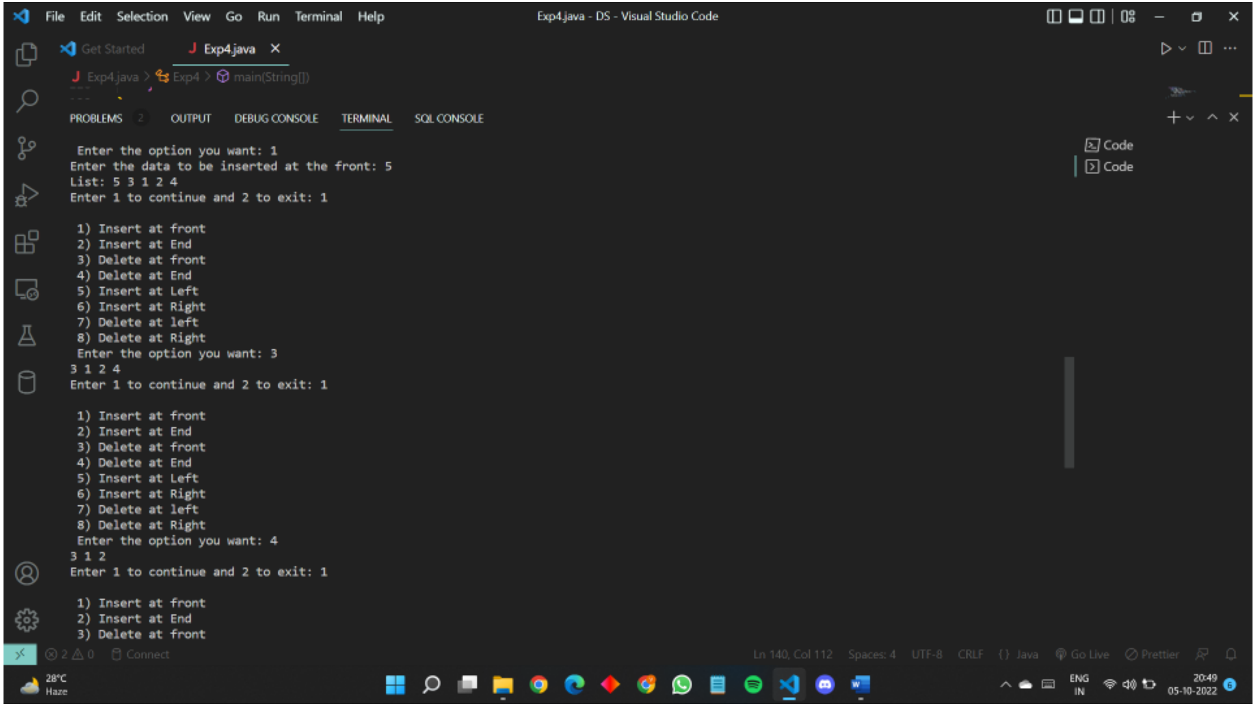
File Edit Selection View Go Run Terminal Help
Exp4.java - DS - Visual Studio Code
Get Started
Exp4.java
Exp4.java > Exp4 > main(String[])
PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
SQL CONSOLE
Code
Code
Ln 140, Col 112 Spaces: 4 UTF-8 CRLF {} Java Go Live Prettier
⊗ 2 ⚠ 0 Connect
28°C
Haze
ENG
IN
20:49
05-10-2022

```
PS V:\DS> cd "v:\DS\" ; if ($?) { javac Exp4.java } ; if ($?) { java Exp4 }

 1) Insert at front
 2) Insert at End
 3) Delete at front
 4) Delete at End
 5) Insert at Left
 6) Insert at Right
 7) Delete at left
 8) Delete at Right
 Enter the option you want: 1
Enter the data to be inserted at the front: 1
List: 1
Enter 1 to continue and 2 to exit: 1

 1) Insert at front
 2) Insert at End
 3) Delete at front
 4) Delete at End
 5) Insert at Left
 6) Insert at Right
 7) Delete at left
 8) Delete at Right
 Enter the option you want: 2
Enter the data to be inserted at the end: 2
List: 1 2
Enter 1 to continue and 2 to exit: 1

 1) Insert at front
 2) Insert at End
 3) Delete at front
 4) Delete at End
```



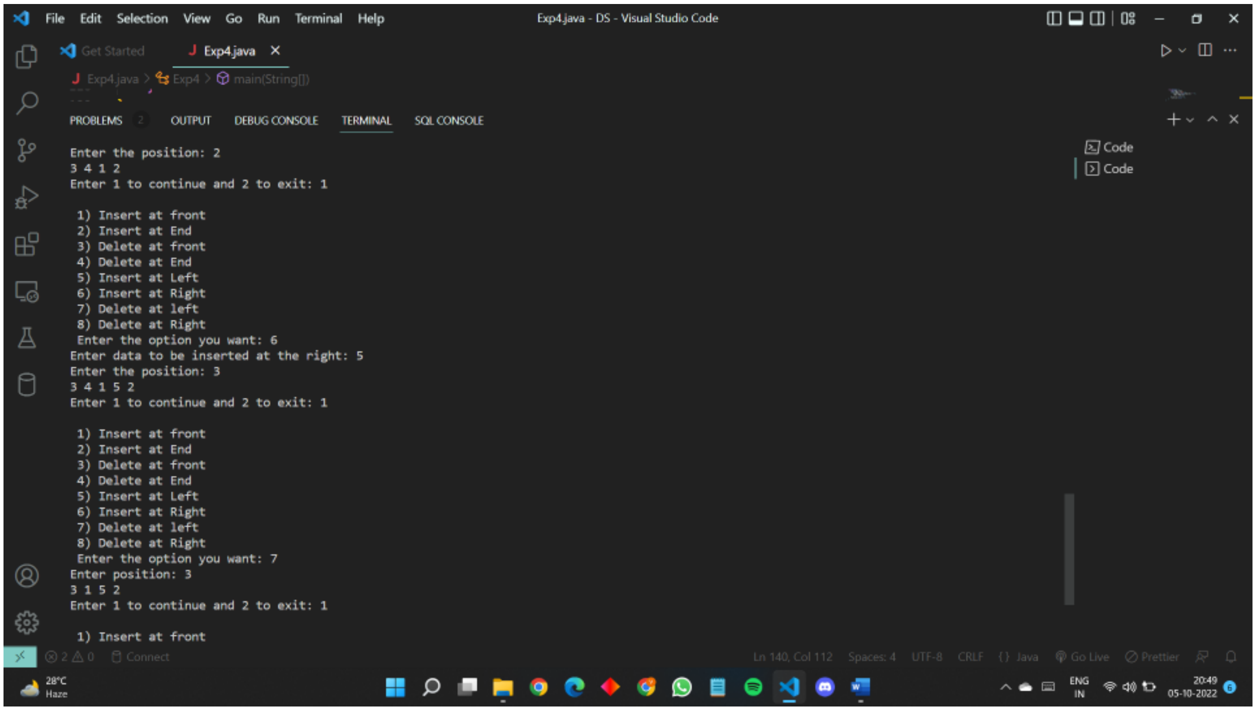
File Edit Selection View Go Run Terminal Help
Exp4.java - DS - Visual Studio Code
Get Started
Exp4.java
Exp4.java > Exp4 > main(String[])
PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
SQL CONSOLE
Code
Code
Ln 140, Col 112 Spaces: 4 UTF-8 CRLF {} Java Go Live Prettier
28°C Haze
ENG IN
20:49 05-10-2022

```
Enter 1 to continue and 2 to exit: 1

 1) Insert at front
 2) Insert at End
 3) Delete at front
 4) Delete at End
 5) Insert at Left
 6) Insert at Right
 7) Delete at left
 8) Delete at Right
 Enter the option you want: 2
Enter the data to be inserted at the end: 4
List: 3 1 2 4
Enter 1 to continue and 2 to exit: 1

 1) Insert at front
 2) Insert at End
 3) Delete at front
 4) Delete at End
 5) Insert at Left
 6) Insert at Right
 7) Delete at left
 8) Delete at Right
 Enter the option you want: 1
Enter the data to be inserted at the front: 5
List: 5 3 1 2 4
Enter 1 to continue and 2 to exit: 1

 1) Insert at front
 2) Insert at End
 3) Delete at front
 4) Delete at End
```

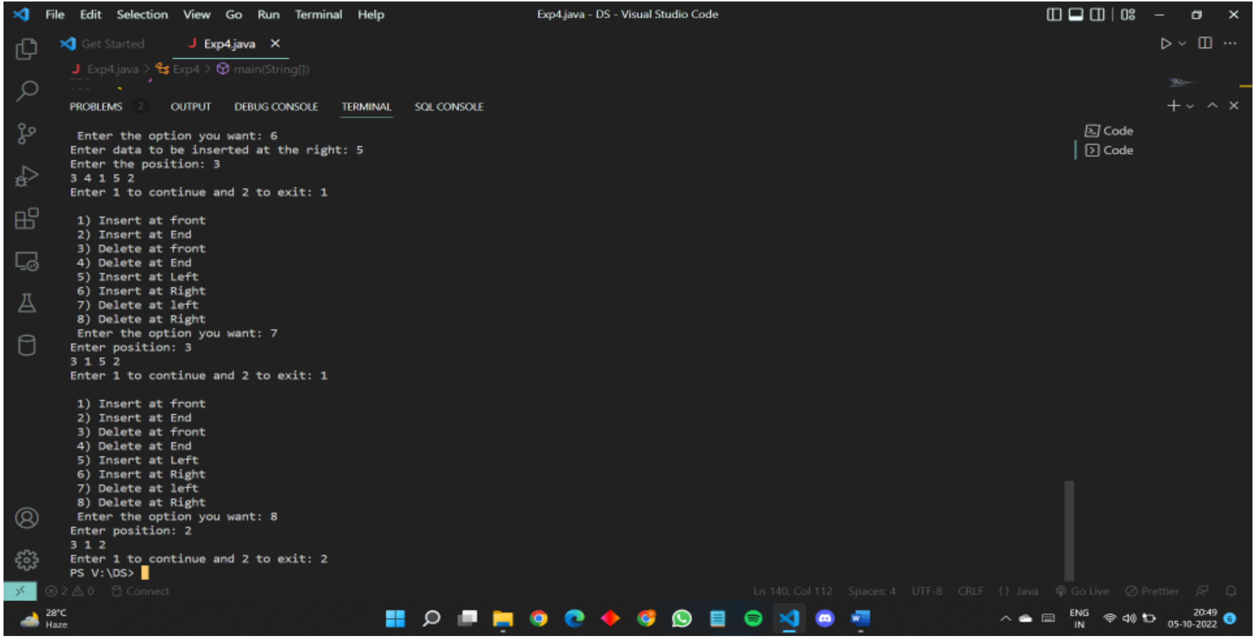


File Edit Selection View Go Run Terminal Help
Exp4.java - DS - Visual Studio Code
Get Started
Exp4.java
Exp4.java > Exp4 > main(String[])
PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
SQL CONSOLE
Code
Code
Ln 140, Col 112 Spaces: 4 UTF-8 CRLF {} Java Go Live Prettier
28°C Haze
ENG IN
20:49 05-10-2022

```
 Enter the option you want: 1
Enter the data to be inserted at the front: 5
List: 5 3 1 2 4
Enter 1 to continue and 2 to exit: 1

 1) Insert at front
 2) Insert at End
 3) Delete at front
 4) Delete at End
 5) Insert at Left
 6) Insert at Right
 7) Delete at left
 8) Delete at Right
 Enter the option you want: 3
3 1 2 4
Enter 1 to continue and 2 to exit: 1

 1) Insert at front
 2) Insert at End
 3) Delete at front
 4) Delete at End
 5) Insert at Left
 6) Insert at Right
 7) Delete at left
 8) Delete at Right
 Enter the option you want: 4
3 1 2
Enter 1 to continue and 2 to exit: 1

 1) Insert at front
 2) Insert at End
 3) Delete at front
```


File Edit Selection View Go Run Terminal Help
Exp4.java - DS - Visual Studio Code
Get Started
Exp4.java
Exp4.java > Exp4 > main(String[])
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE
Code
Code
Ln 140, Col 112 Spaces: 4 UTF-8 CRLF {} Java Go Live Prettier
28°C Haze
ENG IN
20:49 05-10-2022

```
Enter the position: 2
3 4 1 2
Enter 1 to continue and 2 to exit: 1

 1) Insert at front
 2) Insert at End
 3) Delete at front
 4) Delete at End
 5) Insert at Left
 6) Insert at Right
 7) Delete at left
 8) Delete at Right
 Enter the option you want: 6
Enter data to be inserted at the right: 5
Enter the position: 3
3 4 1 5 2
Enter 1 to continue and 2 to exit: 1

 1) Insert at front
 2) Insert at End
 3) Delete at front
 4) Delete at End
 5) Insert at Left
 6) Insert at Right
 7) Delete at left
 8) Delete at Right
 Enter the option you want: 7
Enter position: 3
3 1 5 2
Enter 1 to continue and 2 to exit: 1

 1) Insert at front
```


File Edit Selection View Go Run Terminal Help
Exp4.java - DS - Visual Studio Code
Get Started
Exp4.java
Exp4.java > Exp4 > main(String[])
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE
Code
Code
Ln 140, Col 112 Spaces: 4 UTF-8 CRLF {} Java Go Live Prettier
28°C Haze
ENG IN
20:49 05-10-2022

```
 Enter the option you want: 6
Enter data to be inserted at the right: 5
Enter the position: 3
3 4 1 5 2
Enter 1 to continue and 2 to exit: 1

 1) Insert at front
 2) Insert at End
 3) Delete at front
 4) Delete at End
 5) Insert at Left
 6) Insert at Right
 7) Delete at left
 8) Delete at Right
 Enter the option you want: 7
Enter position: 3
3 1 5 2
Enter 1 to continue and 2 to exit: 1

 1) Insert at front
 2) Insert at End
 3) Delete at front
 4) Delete at End
 5) Insert at Left
 6) Insert at Right
 7) Delete at left
 8) Delete at Right
 Enter the option you want: 8
Enter position: 2
3 1 2
Enter 1 to continue and 2 to exit: 2
PS V:\DS>
```

CONCLUSION:	In this experiment, we learned about the Doubly Linked List, and its advantages and disadvantages over singly linked list. I also learned about the advantages and dis-advantages of a Doubly Linked List. Then we implemented a menu-driven program for insertion and deletion as per needs of users