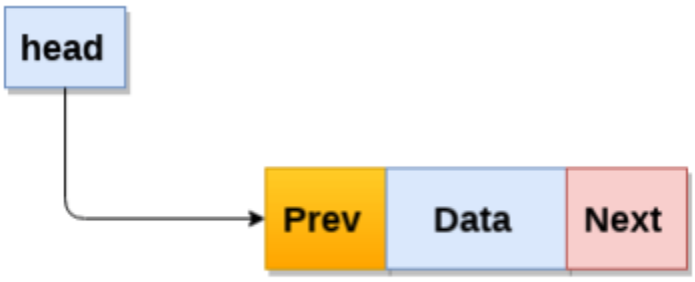
DS-LAB EXPERIMENT Extra

NAME: YASH SARANG D6AD/47

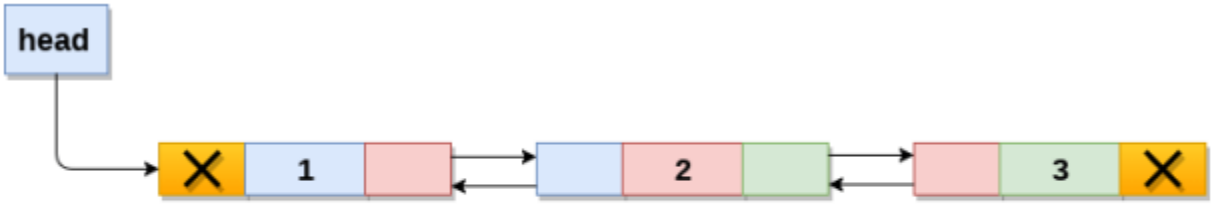
Aim: To implement Doubly linked list ADT.

Theory: Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer) , pointer to the previous node (previous pointer). A sample node in a doubly linked list is shown in the figure.



Node

A doubly linked list containing three nodes having numbers from 1 to 3 in their data part, is shown in the following image.



Doubly Linked List

The prev part of the first node and the next part of the last node will always contain null indicating end in each direction. In a singly linked list, we could traverse only in one direction, because each node contains address of the next node and it doesn't have any record of its previous nodes. However, doubly linked list overcome this

limitation of a singly linked list. Due to the fact that each node of the list contains the address of its previous node, we can find all the details about the previous node as well by using the previous address stored inside the previous part of each node.

Advantages :

1. It is better as, unlike singly linked list, in a doubly-linked list we can traverse in both directions. Thus, if in case any pointer is lost we can still traverse.
2. Thus, in Doubly Linked List we can traverse from Head to Tail as well as Tail to Head. 3. Delete operation is quicker if the pointer to the node to be deleted is given to us already. 4. Insertion is quicker in doubly-linked lists.

Disadvantages :

1. Extra space is required for the previous pointer for doubly-linked lists(DLL)
2. All operations require an additional modification of the previous pointer as well along with the next pointer.

Algorithm:

➢ **Inserting At Beginning of the list**

- Step 1: IF ptr = NULL

Write OVERFLOW
Go to Step 9 [END OF IF]

- Step 2: SET NEW_NODE = ptr
- Step 3: SET ptr = ptr -> NEXT
- Step 4: SET NEW_NODE -> DATA = VAL
- Step 5: SET NEW_NODE -> PREV = NULL
- Step 6: SET NEW_NODE -> NEXT = START
- Step 7: SET head -> PREV = NEW_NODE
- Step 8: SET head = NEW_NODE
- Step 9: EXIT

➢ **Inserting At End of the list**

- Step 1: IF PTR = NULL

Write OVERFLOW
Go to Step 11 [END OF IF]

- Step 2: SET NEW_NODE = PTR
- Step 3: SET PTR = PTR -> NEXT
- Step 4: SET NEW_NODE -> DATA = VAL
- Step 5: SET NEW_NODE -> NEXT = NULL
- Step 6: SET TEMP = START
- Step 7: Repeat Step 8 while TEMP -> NEXT != NULL
- Step 8: SET TEMP = TEMP -> NEXT [END OF LOOP]
- Step 9: SET TEMP -> NEXT = NEW_NODE
- Step 10C: SET NEW_NODE -> PREV = TEMP
- Step 11: EXIT

➢ **Inserting at specific location in the list**

- Step 1: IF PTR = NULL

Write OVERFLOW
Go to Step 15 [END OF IF]

- Step 2: SET NEW_NODE = PTR
- Step 3: SET PTR = PTR -> NEXT
- Step 4: SET NEW_NODE -> DATA = VAL
- Step 5: SET TEMP = START
- Step 6: SET I = 0
- Step 7: REPEAT 8 to 10 until I<="" li="">
- Step 8: SET TEMP = TEMP -> NEXT
- STEP 9: IF TEMP = NULL
- STEP 10: WRITE "LESS THAN DESIRED NO. OF ELEMENTS"

GOTO STEP 15 [END OF IF]
[END OF LOOP]

- Step 11: SET NEW_NODE -> NEXT = TEMP -> NEXT
- Step 12: SET NEW_NODE -> PREV = TEMP
- Step 13 : SET TEMP -> NEXT = NEW_NODE
- Step 14: SET TEMP -> NEXT -> PREV = NEW_NODE
- Step 15: EXIT

➢ **Deleting from Beginning of the list**

- STEP 1: IF HEAD = NULL

WRITE UNDERFLOW
GO TO STEP 6

- STEP 2: SET PTR = HEAD
- STEP 3: SET HEAD = HEAD → NEXT
- STEP 4: SET HEAD → PREV = NULL
- STEP 5: FREE PTR
- STEP 6: EXIT

➢ **Deleting from End of the list**

- Step 1: IF HEAD = NULL

Write UNDERFLOW

Go to Step 7 [END OF IF]

- Step 2: SET TEMP = HEAD
- Step 3: REPEAT STEP 4 WHILE TEMP->NEXT != NULL
- Step 4: SET TEMP = TEMP->NEXT [END OF LOOP]
- Step 5: SET TEMP ->PREV-> NEXT = NULL
- Step 6: FREE TEMP
- Step 7: EXIT

➢ **Deleting a specific Node from the list**

- Step 1: IF HEAD = NULL

Write UNDERFLOW

Go to Step 9 [END OF IF]

- Step 2: SET TEMP = HEAD
- Step 3: Repeat Step 4 while TEMP -> DATA != ITEM
- Step 4: SET TEMP = TEMP -> NEXT [END OF LOOP]
- Step 5: SET PTR = TEMP -> NEXT
- Step 6: SET TEMP -> NEXT = PTR -> NEXT
- Step 7: SET PTR -> NEXT -> PREV = TEMP
- Step 8: FREE PTR
- Step 9: EXIT

➢ **Searching a Node**

- Step 1: IF HEAD == NULL

WRITE "UNDERFLOW"

GOTO STEP 8 [END OF IF]

- Step 2: Set PTR = HEAD
- Step 3: Set i = 0

- Step 4: Repeat step 5 to 7 while PTR != NULL
- Step 5: IF PTR → data = item return i [END OF IF]
- Step 6: i = i + 1
- Step 7: PTR = PTR → next
- Step 8: Exit

➢ **Displaying a Single Linked List**

We can use the following steps to display the elements of a single linked list.

- Step 1: Check whether the list is Empty (head == NULL)
- Step 2: If it is Empty then, display 'List is Empty!!!' and terminate the function.
- Step3: If it is Not Empty then, define a Node pointer 'temp' and initialize with head
- Step4: Keep displaying temp → data with an arrow (--->) until temp reaches the last node.
- Step5: Finally display temp → data with an arrow pointing to NULL (temp → data ---> NULL).
- Step 6: Exit

PROGRAM CODE:

```
#include <stdio.h>
#include <stdlib.h>

struct Node
{
  int data;
  struct Node * prev;
  struct Node * next;
};
struct Node * head = NULL;
//fuctions
void begin_insert();
void last_insert();
void random_insert();
void begin_delete();
void last_delete();
void random_delete();
void search();
void display();
int main()
{
```

```
   int i, choice;
   do {

    printf("\n\n**********Operation
      Menu ** ** ** ** ** \n ");
      printf("\n\t1.insert at beginning \n\t2.insert at end\ n\ t3.insert at position\ n\ t4.delete beginning\ n\
t5.delete end\ n\ t6.delete at position\ n\ t7.search\ n\ t8.display\ n\ t9.exit ");
       printf("\n\tEnter your choice: "); scanf("%d", & choice);
       switch (choice)
       {
       case 1:
         begin_insert();
         break;
       case 2:
         last_insert();
         break;
       case 3:
         random_insert();
         break;
       case 4:
         begin_delete();
         break;
       case 5:
         last_delete();
         break;
       case 6:
         random_delete();
         break;
       case 7:
         search();
         break;
       case 8:
         display();
         break;
       case 9:
         printf("\n\tExiting...");
         break;
       default:
         printf("\n\tEnter the valid choice");
       }
      }
      while (choice != 9);
      return 0;
}
    void begin_insert() {
      int item;
      struct Node * ptr;
      ptr = (struct Node * ) malloc(sizeof(struct Node));
```

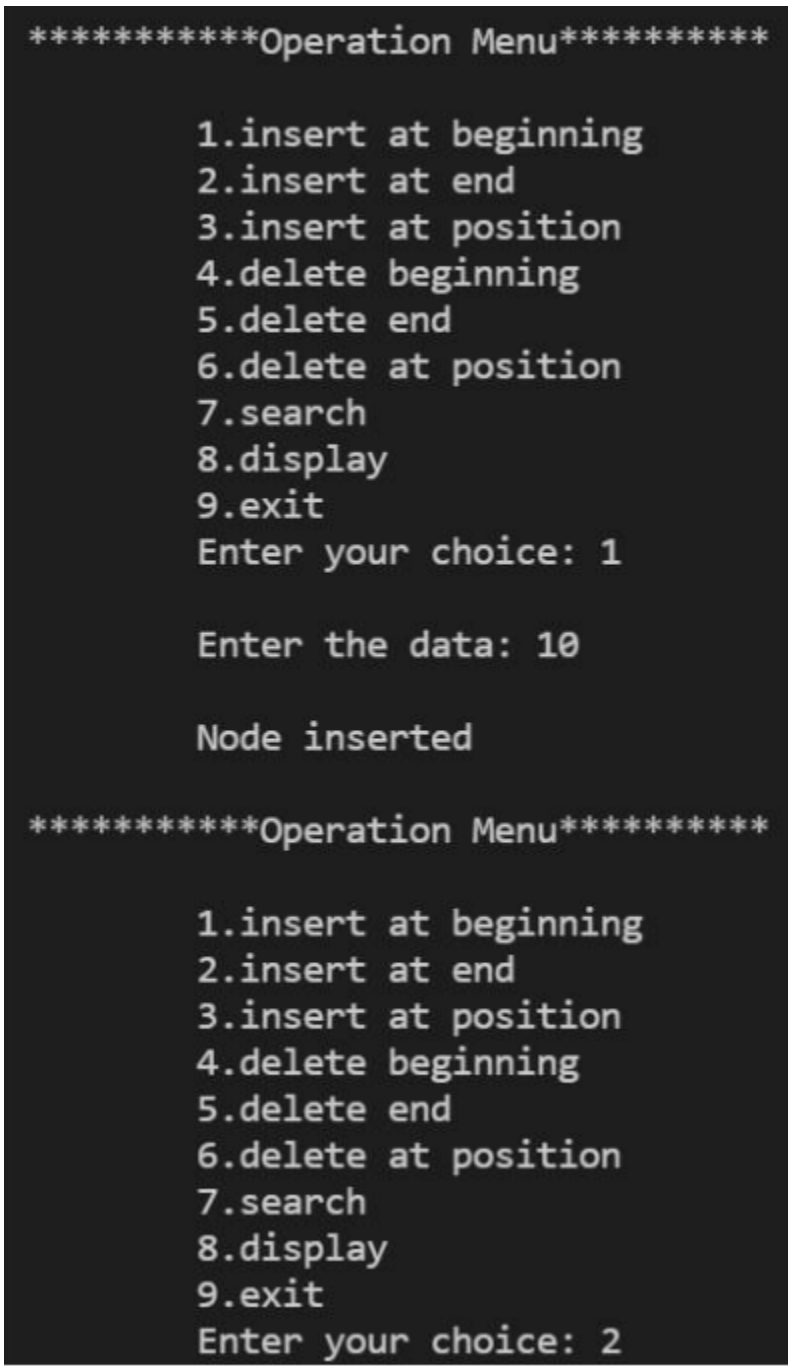
```
  if (ptr == NULL) {
    printf("\n\tOVERFLOW");

  } else {
    printf("\n\tEnter the data: ");
    scanf("%d", & item);
    if (head == NULL) {
      ptr -> prev = NULL;
      ptr -> next = NULL;
      ptr -> data = item;
      head = ptr;
    } else {
      head -> prev = ptr;
      ptr -> next = head;
      ptr -> prev = NULL;
      ptr -> data = item;
      head = ptr;
    }
    printf("\n\tNode inserted");
  }
}
void last_insert() {
  int item;
  struct Node * ptr,
    * temp;
  ptr = (struct Node * ) malloc(sizeof(struct Node));
  temp = head;
  if (ptr == NULL) {
    printf("\n\tOVERFLOW");
  } else {
    printf("\n\tEnter the data: ");
    scanf("%d", & item);
    if (head == NULL) {
      ptr -> prev = NULL;
      ptr -> next = NULL;
      ptr -> data = item;
      head = ptr;
    } else if (head -> next == NULL) {
      ptr -> data = item;
      ptr -> prev = temp;
      ptr -> next = NULL;
      temp -> next = ptr;
    } else {
      do {
        temp = temp -> next;
      } while (temp -> next != NULL);
      ptr -> data = item;
      ptr -> prev = temp;
```

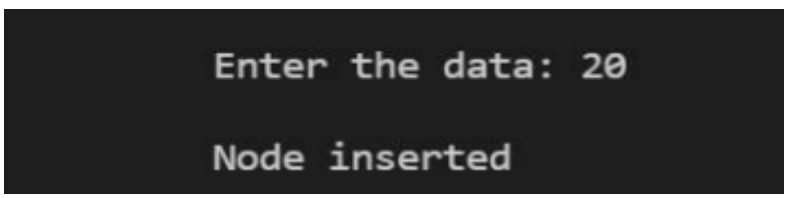
```
      ptr -> next = NULL;
      temp -> next = ptr;

    }
    printf("\n\tNode inserted");
   }
}
void random_insert() {
    int item, loc;
    struct Node * ptr,
      * temp;
    ptr = (struct Node * ) malloc(sizeof(struct Node));
    temp = head;
    if (ptr == NULL) {
      printf("\n\tOVERFLOW");
    } else {
      printf("\n\tAt what position you want to enter
        Node ? ");
      scanf("%d", & loc);
      if (loc == 1 || loc == 0) {
        begin_insert();
      } else {
        for (int i = 1; i < loc - 1; i++) {
          temp = temp -> next;
          if (temp == NULL) {
            printf("\n\tCannot insert, as list end
              is reached ");
            }
          }
          printf("\n\tEnter the data: ");
          scanf("%d", & item);
          if (temp -> next == NULL) {
            ptr -> data = item;
            ptr -> prev = temp;
            ptr -> next = NULL;
            temp -> next = ptr;
          } else {
            ptr -> data = item;
            temp -> next -> prev = ptr;
            ptr -> next = temp -> next;
            ptr -> prev = temp;
            temp -> next = ptr;
          }
        }
      }
    }
    void begin_delete() {
      struct Node * temp = head;
```

```
  if (head == NULL) {
    printf("\n\tCannot delete, as list is empty");

  } else if (head -> next == NULL) {
    free(head);
    printf("\n\tDeleted the only element of list");

  } else {
    head = temp -> next; //shifting head to next Node
    temp -> next = NULL;
    temp -> prev = NULL;
    free(temp);
  }
  printf("\n\tFirst Node deleted");
}
void last_delete() {
  struct Node * temp = head;
  if (head == NULL) {
    printf("\n\tCannot delete, as list is empty");
  } else if (head -> next == NULL) {
    free(head);
    printf("\n\tDeleted the only element of list");
  } else {
    do {
      temp = temp -> next;
    } while (temp -> next != NULL);
    temp -> prev -> next = NULL;
    free(temp);
  }
  printf("\n\tLast Node deleted");
}
void random_delete() {
    struct Node * temp = head;
    int loc;
    printf("\n\tAt what position you want to delete Node?
        ");
        scanf("%d", & loc);
        if (loc == 1 || loc == 0) {
          begin_delete();
        } else {
          for (int i = 1; i < loc; i++) {
            temp = temp -> next;
            if (temp == NULL) {
              printf("\n\tCannot delete, as list end is
                reached ");
              }
            }
            if (temp -> next == NULL) {
```

```
      last_delete();
    } else {

      temp -> prev -> next = temp -> next;
      temp -> next -> prev = temp -> prev;
      temp -> next = NULL;
      temp -> prev = NULL;
      free(temp);
    }
  }
  printf("\n\tDeleted the Node at position %d", loc);
}
void search() {
  struct Node * temp = head;
  int i, item, count = 0, flag = 1;
  printf("\n\tEnter the element to be searched: ");
  scanf("%d", & item);
  while (temp != NULL) {
    count++;
    if (temp -> data == item) {
      printf("\n\tElement %d found at %d position",
        item, count);
      flag = 0;
    }
    temp = temp -> next;
  }
  if (flag == 1) {
    printf("\n\tElement not found");
  }
}
void display() {
  struct Node * temp = head;
  if (temp == NULL) {
    printf("\n\tLinked list not found");
  } else {
    printf("\n\tLinked list is as follows:\n\t");
    while (temp != NULL) {
      printf("%d %c ", temp -> data, 26);
      temp = temp -> next;
    }
  }
}
```

OUTPUT:


**********Operation Menu**********
1.insert at beginning
2.insert at end
3.insert at position
4.delete beginning
5.delete end
6.delete at position
7.search
8.display
9.exit
Enter your choice: 1
Enter the data: 10
Node inserted
**********Operation Menu**********
1.insert at beginning
2.insert at end
3.insert at position
4.delete beginning
5.delete end
6.delete at position
7.search
8.display
9.exit
Enter your choice: 2


Enter the data: 20
Node inserted

```
**********Operation Menu**********

        1.insert at beginning
        2.insert at end
        3.insert at position
        4.delete beginning
        5.delete end
        6.delete at position
        7.search
        8.display
        9.exit
        Enter your choice: 2

        Enter the data: 40

        Node inserted

**********Operation Menu**********

        1.insert at beginning
        2.insert at end
        3.insert at position
        4.delete beginning
        5.delete end
        6.delete at position
        7.search
        8.display
        9.exit
        Enter your choice: 2

        Enter the data: 50

        Node inserted
```

```
**********Operation Menu**********

        1.insert at beginning
        2.insert at end
        3.insert at position
        4.delete beginning
        5.delete end
        6.delete at position
        7.search
        8.display
        9.exit
        Enter your choice: 3

        At what position you want to enter Node? 3

        Enter the data: 30


**********Operation Menu**********

        1.insert at beginning
        2.insert at end
        3.insert at position
        4.delete beginning
        5.delete end
        6.delete at position
        7.search
        8.display
        9.exit
        Enter your choice: 8

        Linked list is as follows:
        10 → 20 → 30 → 40 → 50 →
```

```
**********Operation Menu**********

        1.insert at beginning
        2.insert at end
```

```
        3.insert at position
        4.delete beginning
        5.delete end
        6.delete at position
        7.search
        8.display
        9.exit
        Enter your choice: 4

        First Node deleted

**********Operation Menu**********

        1.insert at beginning
        2.insert at end
        3.insert at position
        4.delete beginning
        5.delete end
        6.delete at position
        7.search
        8.display
        9.exit
        Enter your choice: 5

        Last Node deleted

**********Operation Menu**********

        1.insert at beginning
        2.insert at end
        3.insert at position
        4.delete beginning
        5.delete end
        6.delete at position
        7.search
        8.display
        9.exit
        Enter your choice: 6
```

```
        At what position you want to delete Node? 2

        Deleted the Node at position 2
```

```
**********Operation Menu**********

        1.insert at beginning
        2.insert at end
        3.insert at position
        4.delete beginning
        5.delete end
        6.delete at position
        7.search
        8.display
        9.exit
        Enter your choice: 8

        Linked list is as follows:
        20 → 40 →

**********Operation Menu**********

        1.insert at beginning
        2.insert at end
        3.insert at position
        4.delete beginning
        5.delete end
        6.delete at position
        7.search
        8.display
        9.exit
        Enter your choice: 7

        Enter the element to be searched: 4

        Element not found
```

```
**********Operation Menu**********

        1.insert at beginning
        2.insert at end
        3.insert at position
        4.delete beginning
        5.delete end
        6.delete at position
        7.search
        8.display
        9.exit
        Enter your choice: 7

        Enter the element to be searched: 40

        Element 40 found at 2 position

**********Operation Menu**********

        1.insert at beginning
        2.insert at end
        3.insert at position
        4.delete beginning
        5.delete end
        6.delete at position
        7.search
        8.display
        9.exit
        Enter your choice: 9

        Exiting...
```
