```
5.Delete from last
6.Delete the node after the given data
7.Search
8.Show
9.Exit

Enter your choice?
9

Exited..
```

RESULT:
Thus the program and code for implementation of doubly linked list is successfully written and executed.

LAB 7:
IMPLEMENTATION OF STACK USING ARRAY AND LINKED LIST

AIM:
To execute program based on implementation of Stack using array and linked list.

ALGORITHM:

Stack Operations using Linked List:

Step 1 - Include all the header files which are used in the program. And declare all the user defined functions.
Step 2 - Define a 'Node' structure with two members data and next.
Step 3 - Define a Node pointer 'top' and set it to NULL.
Step 4 - Implement the main method by displaying Menu with list of operations and make suitable function calls in the main method.

push(value) - Inserting an element into the Stack:

Step 1 - Create a new Node with given value.
Step 2 - Check whether stack is Empty (top == NULL)
Step 3 - If it is Empty, then set new Node → next = NULL.
Step 4 - If it is Not Empty, then set new Node → next = top.
Step 5 - Finally, set top = new Node.

Pop() - Deleting an Element from a Stack:

Step 1 - Check whether stack is Empty (top == NULL).
Step 2 - If it is Empty, then display "Stack is Empty!!! Deletion is not possible!!!" and terminate the function
Step 3 - If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'.
Step 4 - Then set 'top = top → next'.
Step 5 - Finally, delete 'temp'. (free(temp)).

Display () - Displaying stack of elements :

Step 1 - Check whether stack is Empty (top == NULL).
Step 2 - If it is Empty, then display 'Stack is Empty!!!' and terminate the function.
Step 3 - If it is Not Empty, then define a Node pointer 'temp' and initialize with top.
Step 4 - Display 'temp → data --->' and move it to the next node. Repeat the same until temp reaches to the first node in the stack. (temp → next != NULL).
Step 5 - Finally! Display 'temp → data ---> NULL'.

Stack Operations using Array:

Step 1 - Include all the header files which are used in the program and define a constant 'SIZE' with specific value.
Step 2 - Declare all the functions used in stack implementation.
Step 3 - Create a one dimensional array with fixed size (int stack[SIZE])
Step 4 - Define a integer variable 'top' and initialize with '-1'. (int top = -1)
Step 5 - In main method, display menu with list of operations and make suitable function calls to perform operation selected by the user on the stack.

push(value) - Inserting value into the stack:

In a stack, push() is a function used to insert an element into the stack. In a stack, the new element is always inserted at top position. Push function takes one integer value as parameter and inserts that value into the stack. We can use the following steps to push an element on to the stack...

Step 1 - Check whether stack is FULL. (top == SIZE-1)
Step 2 - If it is FULL, then display "Stack is FULL!!! Insertion is not possible!!!" and terminate the function.
Step 3 - If it is NOT FULL, then increment top value by one (top++) and set stack[top] to value (stack[top] = value).

Pop() - Delete a value from the Stack:

In a stack, pop() is a function used to delete an element from the stack. In a stack, the element is always deleted from top position. Pop function does not take any value as parameter. We can use the following steps to pop an element from the stack...

Step 1 - Check whether stack is EMPTY. (top == -1)
Step 2 - If it is EMPTY, then display "Stack is EMPTY!!! Deletion is not possible!!!" and terminate the function.
Step 3 - If it is NOT EMPTY, then delete stack[top] and decrement top value by one (top--).
Display() - Displays the elements of a Stack:
Step 1 - Check whether stack is EMPTY. (top == -1)
Step 2 - If it is EMPTY, then display "Stack is EMPTY!!!" and terminate the function.
Step 3 - If it is NOT EMPTY, then define a variable 'i' and initialize with top. Display stack[i] value and decrement i value by one (i--).
Step 3 - Repeat above step until i value becomes '0'.

Program :-Stack Implementation Using Linked List

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

struct node
{
   int info;
   struct node *ptr;
}*top,*top1,*temp;

int topelement();
void push(int data);
void pop();
void empty();
void display();
void destroy();
void stack_count();
void create();

int count = 0;

void main()
{
   int no, ch, e;

   create();

   while (1)
   {
      printf("\n 1 - Push");
      printf("\n 2 - Pop");
      printf("\n 3 - Top");
      printf("\n 4 - Empty");
      printf("\n 5 - Exit");
      printf("\n 6 - Display");
      printf("\n 7 - Stack Count");
      printf("\n 8 - Destroy stack");
```

```
        printf("\n Enter choice : ");
        scanf("%d", &ch);

        switch (ch)
        {
        case 1:
            printf("Enter data : ");
            scanf("%d", &no);
            push(no);
            break;
        case 2:
            pop();
            break;
        case 3:
            if (top == NULL)
                printf("No elements in stack");
            else
            {
                e = topelement();
                printf("\n Top element : %d", e);
            }
            break;
        case 4:
            empty();
            break;
        case 5:
            exit(0);
        case 6:
            display();
            break;
        case 7:
            stack_count();
            break;
        case 8:
            destroy();
            break;
        default :
            printf(" Wrong choice, Please enter correct choice  ");
            break;
        }
    }
}

/* Create empty stack */
void create()
{
    top = NULL;
}

/* Count stack elements */
```

```
void stack_count()
{
   printf("\n No. of elements in stack : %d", count);
}

/* Push data into stack */
void push(int data)
{
   if (top == NULL)
   {
      top =(struct node *)malloc(1*sizeof(struct node));
      top->ptr = NULL;
      top->info = data;
   }
   else
   {
      temp =(struct node *)malloc(1*sizeof(struct node));
      temp->ptr = top;
      temp->info = data;
      top = temp;
   }
   count++;
}

/* Display stack elements */
void display()
{
   top1 = top;

   if (top1 == NULL)
   {
      printf("Stack is empty");
      return;
   }

   while (top1 != NULL)
   {
      printf("%d ", top1->info);
      top1 = top1->ptr;
   }
 }

/* Pop Operation on stack */
void pop()
{
   top1 = top;

   if (top1 == NULL)
   {
      printf("\n Error : Trying to pop from empty stack");
```

```
        return;
    }
    else
        top1 = top1->ptr;
    printf("\n Popped value : %d", top->info);
    free(top);
    top = top1;
    count--;
}

/* Return top element */
int topelement()
{
    return(top->info);
}

/* Check if stack is empty or not */
void empty()
{
    if (top == NULL)
        printf("\n Stack is empty");
    else
        printf("\n Stack is not empty with %d elements", count);
}

/* Destroy entire stack */
void destroy()
{
    top1 = top;

    while (top1 != NULL)
    {
        top1 = top->ptr;
        free(top);
        top = top1;
        top1 = top1->ptr;
    }
    free(top1);
    top = NULL;

    printf("\n All stack elements destroyed");
    count = 0;
}
```

Output :-

```
 1 - Push
 2 - Pop
 3 - Top
 4 - Empty
 5 - Exit
 6 - Display
 7 - Stack Count
 8 - Destroy stack
 Enter choice : 1
Enter data : 10

 1 - Push
 2 - Pop
 3 - Top
 4 - Empty
 5 - Exit
 6 - Display
 7 - Stack Count
 8 - Destroy stack
 Enter choice : 1
Enter data : 20

 1 - Push
 2 - Pop
 3 - Top
 4 - Empty
 5 - Exit
 6 - Display
 7 - Stack Count
 8 - Destroy stack
 Enter choice : 6
20 10
 1 - Push
 2 - Pop
 3 - Top
 4 - Empty
 5 - Exit
 6 - Display
 7 - Stack Count
 8 - Destroy stack
 Enter choice : 2

 Popped value : 20
 1 - Push
 2 - Pop
 3 - Top
 4 - Empty
 5 - Exit
 6 - Display
 7 - Stack Count
 8 - Destroy stack
 Enter choice : 6
10
```

```
 1 - Push
 2 - Pop
 3 - Top
 4 - Empty
 5 - Exit
 6 - Display
 7 - Stack Count
 8 - Destroy stack
 Enter choice : 5
```

RESULT:

Thus the program and code for implementation of Stack using array and linked list is successfully written and executed.

LAB 8:
IMPLEMENTATION OF QUEUE USING ARRAY AND LINKED LIST

AIM:

To execute program based on implementation of Queue using array and linked list.

ALGORITHM:

Queue representing array:
Step 1 - Include all the header files which are used in the program and define a constant 'SIZE' with specific value.
Step 2 - Declare all the user defined functions which are used in queue implementation.
Step 3 - Create a one dimensional array with above defined SIZE (int queue[SIZE])
Step 4 - Define two integer variables 'front' and 'rear' and initialize both with '-1'. (int front = -1, rear = -1)
Step 5 - Then implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on queue.
Inserting value into the Queue:
Step 1 - Check whether queue is FULL. (rear == SIZE-1)
Step 2 - If it is FULL, then display "Queue is FULL!!! Insertion is not possible!!!" and terminate the function.
Step 3 - If it is NOT FULL, then increment rear value by one (rear++) and set queue[rear] = value.

Deleting value from the Queue:
Step 1 - Check whether queue is EMPTY. (front == rear)
Step 2 - If it is EMPTY, then display "Queue is EMPTY!!! Deletion is not possible!!!" and terminate the function.
Step 3 - If it is NOT EMPTY, then increment the front value by one (front ++). Then display queue[front] as deleted element. Then check whether both front and rear are equal (front == rear), if it TRUE, then set both front and rear to '-1' (front = rear = -1).
Queue representing linked list:
Algorithm:
Step 1 - Include all the header files which are used in the program. And declare all the user defined functions.
Step 2 - Define a 'Node' structure with two members data and next.
Step 3 - Define two Node pointers 'front' and 'rear' and set both to NULL.
Step 4 - Implement the main method by displaying Menu of list of operations and make suitable function calls in the main method to perform user selected operation.
Inserting element into Queue:
Step 1 - Create a new Node with given value and set 'new Node → next' to NULL.
Step 2 - Check whether queue is Empty (rear == NULL)
Step 3 - If it is Empty then, set front = new Node and rear = new Node.
Step 4 - If it is Not Empty then, set rear → next = new Node and rear = new Node.
Deleting element from Queue:
Step 1 - Check whether queue is Empty (front == NULL).
Step 2 - If it is Empty, then display "Queue is Empty!!! Deletion is not possible!!!" and terminate from the function
Step 3 - If it is Not Empty then, define a Node pointer 'temp' and set it to 'front'.
Step 4 - Then set 'front = front → next' and delete 'temp' (free(temp)).

Program :-

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

```
#define MAX 50

void insert();
void delete();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
    int choice;
    while (1)
    {
        printf("1.Insert\n");
        printf("2.Delete\n");
        printf("3.Display\n");
        printf("4.Quit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
            insert();
            break;
            case 2:
            delete();
            break;
            case 3:
            display();
            break;
            case 4:
            exit(1);
            default:
            printf("Wrong choice \n");
        }
    }
}

void insert()
{
    int add_item;
    if (rear == MAX - 1)
    printf("Queue Overflow \n");
    else
    {
        if (front == - 1)
        /*If queue is initially empty */
        front = 0;
        printf("Inset the element in queue : ");
        scanf("%d", &add_item);
        rear = rear + 1;
        queue_array[rear] = add_item;
    }
}

void delete()
{
    if (front == - 1 || front > rear)
    {
        printf("Queue Underflow \n");
        return ;
    }
    else
    {
        printf("Element deleted from queue is : %d\n", queue_array[front]);
        front = front + 1;
    }
}

void display()
{
    int i;
    if (front == - 1)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is : \n");
        for (i = front; i <= rear; i++)
            printf("%d ", queue_array[i]);
        printf("\n");
    }
}
```

Output :-

```
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 1
Inset the element in queue : 30
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 1
Inset the element in queue : 40
1.Insert
2.Delete
```

```
3.Display
4.Quit
Enter your choice : 3
Queue is :
30 40
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 2
Element deleted from queue is : 30
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 3
Queue is :
40
```

<u>**Queue Implementation Using Linked List**</u>

<u>**Program :-**</u>

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

struct node
{
   int info;
   struct node *ptr;
}*front,*rear,*temp,*front1;

int frontelement();
void enq(int data);
void deq();
void empty();
void display();
void create();
void queuesize();

int count = 0;

void main()
{
   int no, ch, e;

   create();
   while (1)
   {
        printf("\n 1 - Insert");
        printf("\n 2 - Delete");
        printf("\n 3 - Front element");
        printf("\n 4 - Empty");
        printf("\n 5 - Exit");
        printf("\n 6 - Display");
        printf("\n 7 - Queue size");

        printf("\n Enter choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("Enter data : ");
            scanf("%d", &no);
            enq(no);
            break;
        case 2:
            deq();
            break;
        case 3:
            e = frontelement();
            if (e != 0)
                printf("Front element : %d", e);
            else
                printf("\n No front element in Queue as queue is empty");
            break;
        case 4:
            empty();
            break;
        case 5:
            exit(0);
        case 6:
            display();
            break;
        case 7:
            queuesize();
            break;
        default:
            printf("Wrong choice, Please enter correct choice  ");
            break;
        }
    }
}

/* Create an empty queue */
void create()
{
    front = rear = NULL;
}
```

```
/* Returns queue size */
void queuesize()
{
   printf("\n Queue size : %d", count);
}

/* Enqueing the queue */
void enq(int data)
{
   if (rear == NULL)
   {
      rear = (struct node *)malloc(1*sizeof(struct node));
      rear->ptr = NULL;
      rear->info = data;
      front = rear;
   }
   else
   {
      temp=(struct node *)malloc(1*sizeof(struct node));
      rear->ptr = temp;
      temp->info = data;
      temp->ptr = NULL;

      rear = temp;
   }
   count++;
}

/* Displaying the queue elements */
void display()
{
   front1 = front;

   if ((front1 == NULL) && (rear == NULL))
   {
      printf("Queue is empty");
      return;
   }
   while (front1 != rear)
   {
      printf("%d ", front1->info);
      front1 = front1->ptr;
   }
   if (front1 == rear)
      printf("%d", front1->info);
}

/* Dequeing the queue */
void deq()
```

```
{
   front1 = front;

   if (front1 == NULL)
   {
      printf("\n Error: Trying to display elements from empty queue");
      return;
   }
   else
      if (front1->ptr != NULL)
      {
         front1 = front1->ptr;
         printf("\n Dequed value : %d", front->info);
         free(front);
         front = front1;
      }
      else
      {
         printf("\n Dequed value : %d", front->info);
         free(front);
         front = NULL;
         rear = NULL;
      }
      count--;
}

/* Returns the front element of queue */
int frontelement()
{
   if ((front != NULL) && (rear != NULL))
      return(front->info);
   else
      return 0;
}

/* Display if queue is empty or not */
void empty()
{
    if ((front == NULL) && (rear == NULL))
      printf("\n Queue empty");
   else
     printf("Queue not empty");
}
```

Output :-

```
 1 - Insert
 2 - Delete
 3 - Front element
 4 - Empty
 5 - Exit
 6 - Display
 7 - Queue size
 Enter choice : 1
Enter data : 10

 1 - Insert
 2 - Delete
 3 - Front element
 4 - Empty
 5 - Exit
 6 - Display
 7 - Queue size
 Enter choice : 1
Enter data : 20

 1 - Insert
 2 - Delete
 3 - Front element
 4 - Empty
 5 - Exit
 6 - Display
 7 - Queue size
 Enter choice : 3
Front element : 10
 1 - Insert
 2 - Delete
 3 - Front element
 4 - Empty
 5 - Exit
 6 - Display
 7 - Queue size
 Enter choice : 2

 Dequed value : 10
 1 - Insert
 2 - Delete
 3 - Front element
 4 - Empty
 5 - Exit
 6 - Display
 7 - Queue size
 Enter choice : 7

 Queue size : 1
 1 - Insert
 2 - Delete
 3 - Front element
 4 - Empty
 5 - Exit
 6 - Display
 7 - Queue size
 Enter choice : 6
20
```

RESULT:

Thus the program and code for implementation of Queue using array and linked list is successfully written and executed.

LAB 9:
APPLICATIONS OF STACK, QUEUE

AIM:

To execute program based on applications of stack and queue

ALGORITHM: