Program No: 1

Aim: WAP to find maximum and minimum number in an array .

Algorithm

Read the entered array size and store that value into the variable n.

Read the entered elements using scanf and store the entered array elements into the array using for loop for(i=0;i<n;i++).

Initialise min,max values with the 1st element of the array.

Compare min,max values with a[i],

If min value is greater than a[i] then initialise min=a[i] and if max value is less than a[i] then initialise max=a[i]. Repeat this step for each element of the string using for loop which is having the structure for(i=1;i<n;i++).

Print the minimum of the array and maximum of the array values.

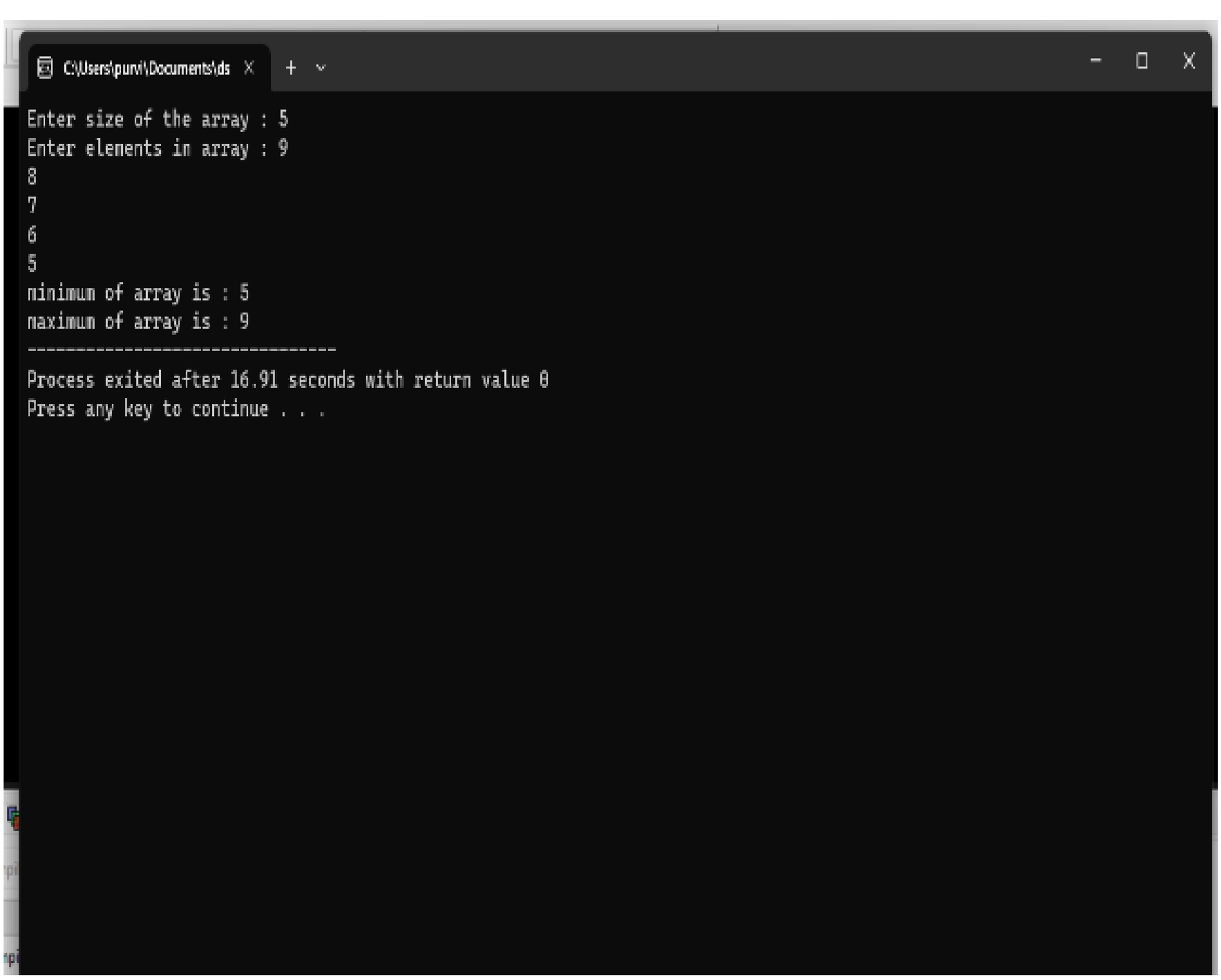
Tool :- Dev-c++

Source code :-

```
#include<stdio.h>
 #include<conio.h>
   int main()
{
    int a[1000],i,n,min,max;

  printf("Enter size of the array: ");
    scanf("%d",&n);

    printf("Enter elements in array: ");
    for(i=0; i<n; i++)
    {
```

```
    scanf("%d",&a[i]);
}

min=max=a[0];
for(i=1;i<n;i++)
{
    if(min>a[i])
                min=a[i];
                 if(max<a[i])
                  max=a[i];
}
 printf("minimum of array is : %d",min);
 printf("\nmaximum of array is : %d",max);


return 0;
}
```

Output :-


C:\Users\purvi\Documents\ds
Enter size of the array : 5
Enter elenents in array : 9
8
7
6
5
ninimun of array is : 5
naximun of array is : 9

Process exited after 16.91 seconds with return value 0
Press any key to continue . . .

Program No:2

AIM:- Perform insertion and deletion operation in 1D array.

Algorithm:-

Insertion Operation Algorithm:

1. Start.
2. Declare an array of size n.
3. Input the size of the array, n.
4. Input the elements of the array.
5. Input the position where you want to insert the element, pos.
6. Input the element to be inserted, newItem.
7. Shift all elements from position pos to n-1 one position to the right.
8. Insert newItem at position pos.
9. Increment the size of the array, n, by 1.
10. Display the updated array.
11. End.

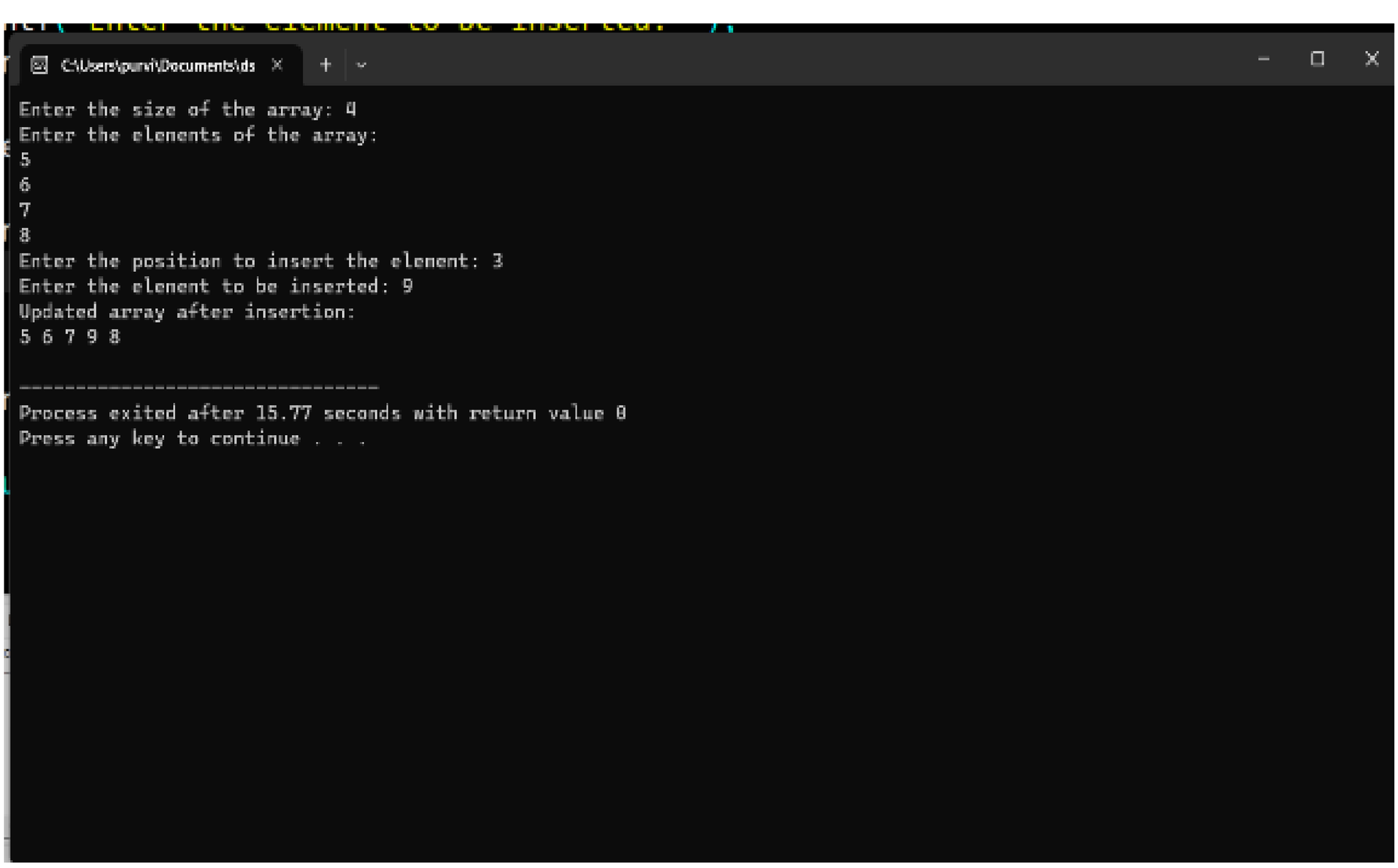
Tool :- Dev-c++

Source code:-

```
#include<stdio.h>

void insertElement(int array[], int *n, int pos, int newItem) {
    (*n)++; // Increase the size of the array by 1
    int i;
    for (i = (*n) - 1; i >= pos; i--) {
        array[i] = array[i - 1];
    }

    array[pos] = newItem;
}
```

```
int main() {
    int array[100],i, n, pos, newItem;

    printf("Enter the size of the array: ");
    scanf("%d", &n);

    printf("Enter the elements of the array:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &array[i]);
    }

    printf("Enter the position to insert the element: ");
    scanf("%d", &pos);

    printf("Enter the element to be inserted: ");
    scanf("%d", &newItem);

    insertElement(array, &n, pos, newItem);

    printf("Updated array after insertion:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");

    return 0;
}
```

Output :-


C:\Users\purvi\Documents\ds
Enter the size of the array: 4
Enter the elements of the array:
5
6
7
8
Enter the position to insert the element: 3
Enter the element to be inserted: 9
Updated array after insertion:
5 6 7 9 8

Process exited after 15.77 seconds with return value 0
Press any key to continue . . .

Algorithm :-

Deletion Operation Algorithm:

1. Start.
2. Declare an array of size n.
3. Input the size of the array, n.
4. Input the elements of the array.
5. Input the position of the element to be deleted, pos.
6. Shift all elements from position pos+1 to n-1 one position to the left.
7. Decrement the size of the array, n, by 1.
8. Display the updated array.
9. End.

Tool :- Dev c++

Source code :-

```
#include<stdio.h>
void deleteElement(int array[], int *n, int pos) {
   int i;
    for (i = pos; i < (*n) - 1; i++) {
        array[i] = array[i + 1];
    }
 (*n)--; // Decrease the size of the array by 1
}
int main() {
    int array[100],i, n, pos;
    printf("Enter the size of the array: ");
```

```
    scanf("%d", &n);
    printf("Enter the elements of the array:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &array[i]);
    }
    printf("Enter the position of the element to delete: ");
    scanf("%d", &pos);
    deleteElement(array, &n, pos);
    printf("Updated array after deletion:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");
    return 0;
}
```

Output :-


C:\Users\purvi\Documents\ds
Enter the size of the array: 5
Enter the elements of the array:
4
5
6
7
8
Enter the position of the element to delete: 3
Updated array after deletion:
4 5 6 8

Process exited after 21.19 seconds with return value 0
Press any key to continue . . .

Program No:3

AIM:- WAP to perform various operation in the single linked list.

a. Create First Node
b. Insert At Beginning
c. Insert At Last
d. Delete From Beginning
e. Delete From Last
f. Display List

Algorithm:-

1. Start.
2. Declare a structure for a node with two fields: data and next.
3. Declare a pointer variable, head, and set it to NULL.
4. Declare a variable, choice.
5. Repeat the following steps:
    Display a menu of operations: (a) Create First Node, (b) Insert At Beginning, (c) Insert At Last, (d) Delete From Beginning, (e) Delete From Last, (f) Display List.
    Input the choice.
    Switch on the choice.
    For each case:
        Perform the respective operation using functions and the head pointer.
        Display appropriate messages.
    If choice is not valid, display an error message.
6. Until the choice is to exit.
7. End.

Tool:- Dev c++

Source code:-

```
#include<stdio.h>

#include<stdlib.h>

struct Node{
```

```
    int data;
    struct Node* next;
};
void createFirstNode(struct Node** head) {
    int newData;
    printf("Enter the data for the first node: ");
    scanf("%d", &newData);
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = newData;
    temp->next = NULL;
    *head = temp;
    printf("First node created successfully!\n");
}
void insertAtBeginning(struct Node** head) {
    int newData;
    printf("Enter the data for the new node: ");
    scanf("%d", &newData);
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = newData;
    temp->next = *head;
    *head = temp;
    printf("Node inserted at the beginning successfully!\n");
```

```
}
void insertAtLast(struct Node** head) {
    int newData;
    printf("Enter the data for the new node: ");
    scanf("%d", &newData);
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = newData;
    temp->next = NULL;
    if (*head == NULL) {
        *head = temp;
    } else {
        struct Node* current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = temp;
    }
    printf("Node inserted at the end successfully!\n");
}
void deleteFromBeginning(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty. Unable to delete!\n");
```

```
    return;
}

    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);

    printf("Node deleted from the beginning successfully!\n");
}

void deleteFromLast(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty. Unable to delete!\n");
        return;
    }
    struct Node* current = *head;
    struct Node* prev = NULL;
    while (current->next != NULL) {
        prev = current;
        current = current->next;
    }
    if (prev == NULL) {
```

```
    // Only one node in the list
    *head = NULL;
}else{
    prev->next = NULL;
    }
    free(current);
    printf("Node deleted from the end successfully!\n");
}
void displayList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    printf("Linked List: ");
    while (head != NULL) {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}
int main() {
    struct Node* head = NULL;
```

```
int choice;
while (1) {
    printf("\n— MENU —\n");
    printf("1. Create First Node\n");
    printf("2. Insert At Beginning\n");
    printf("3. Insert At Last\n");
    printf("4. Delete From Beginning\n");
    printf("5. Delete From Last\n");
    printf("6. Display List\n");
    printf("7. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            createFirstNode(&head);
            break;
        case 2:
            insertAtBeginning(&head);
            break;
        case 3:
            insertAtLast(&head);
```

```
            break;
        case 4:
            deleteFromBeginning(&head);
            break;
        case 5:
            deleteFromLast(&head);
            break;
        case 6:
            displayList(head);
            break;
            case 7:
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}
```

Output :-

```
--- MENU ---
1. Create First Node
2. Insert At Beginning
3. Insert At Last
4. Delete From Beginning
5. Delete From Last
6. Display List
7. Exit
Enter your choice: 1
Enter the data for the first node: 9
First node created successfully!

--- MENU ---
1. Create First Node
2. Insert At Beginning
3. Insert At Last
4. Delete From Beginning
5. Delete From Last
6. Display List
7. Exit
Enter your choice: 2
Enter the data for the new node: 8
Node inserted at the beginning successfully!

--- MENU ---
1. Create First Node
2. Insert At Beginning
3. Insert At Last
4. Delete From Beginning
```

```
5. Delete From Last
6. Display List
7. Exit
Enter your choice: 3
Enter the data for the new node: 7
Node inserted at the end successfully!

--- MENU ---
1. Create First Node
2. Insert At Beginning
3. Insert At Last
4. Delete From Beginning
5. Delete From Last
6. Display List
7. Exit
Enter your choice: 4
Node deleted from the beginning successfully!

--- MENU ---
1. Create First Node
2. Insert At Beginning
3. Insert At Last
4. Delete From Beginning
5. Delete From Last
6. Display List
7. Exit
Enter your choice: 5
Node deleted from the end successfully!

--- MENU ---
```

```
5. Delete From Last
6. Display List
7. Exit
Enter your choice: 5
Node deleted from the end successfully!

--- MENU ---
1. Create First Node
2. Insert At Beginning
3. Insert At Last
4. Delete From Beginning
5. Delete From Last
6. Display List
7. Exit
Enter your choice: 6
Linked List: 9

--- MENU ---
1. Create First Node
2. Insert At Beginning
3. Insert At Last
4. Delete From Beginning
5. Delete From Last
6. Display List
7. Exit
Enter your choice: 7

--------------------------------
Process exited after 31.03 seconds with return value 0
Press any key to continue . . .
```

Program No:4

AIM:- WAP to perform insertion and deletion operation in circular linked list.

Algorithm

1. Start
2. Declare a structure named **Node** with an integer **data** member and a pointer to the next node named **next**.
3. Declare a pointer variable **head** and set it to NULL to indicate an empty circular linked list.
4. Repeat the following steps until the user chooses to exit:
    Display the menu options: insert at beginning, insert at end, delete from beginning, delete from end, display, and exit.
    Read the user's choice.
    Perform the corresponding operation based on the choice using a switch statement:
        If the choice is 1 (insert at beginning):
            Prompt the user to enter a value to insert.
            Create a new node dynamically using **malloc**.
            Assign the entered value to the **data** member of the new node.
            If **head** is NULL, set both the **next** pointer of the new node and **head** to the new node itself.
                Otherwise, set the **next** pointer of the new node to **head->next** and update the **next** pointer of **head** to the new node.
            Display "Element <value> inserted at the beginning successfully."
        If the choice is 2 (insert at end):
            Prompt the user to enter a value to insert.
            Create a new node dynamically using **malloc**.
            Assign the entered value to the **data** member of the new node.
            If **head** is NULL, set both the **next** pointer of the new node and **head** to the new node itself.
            Otherwise, set the **next** pointer of the new node to **head->next** and update the **next** pointer of **head** to the new node.

- Update **head** to point to the new node.
  Display "Element <value> inserted at the end successfully."

If the choice is 3 (delete from beginning):

- Check if the circular linked list is empty (head == NULL).
  - If it is, display "Circular Linked List is empty, cannot delete."
    Otherwise, create a temporary pointer **temp** and set it to **head->next**.
    - If **temp** points to **head**, it means there is only one node in the circular linked list.
      - Set both **temp** and **head** to NULL to indicate an empty list.
    - Otherwise, set the **next** pointer of **head** to **temp->next**.
      Free the memory occupied by **temp**.
  - Display "Element deleted from the beginning."

If the choice is 4 (delete from end):

- Check if the circular linked list is empty (head == NULL).
  - If it is, display "Circular Linked List is empty, cannot delete."
    Otherwise, create two temporary pointers **prev** and **temp** and set them to **head**.
    - Traverse the circular linked list until **temp->next** points to **head**.
      Set the **next** pointer of **prev** to **head** and free the memory occupied by **temp**.
  - Display "Element deleted from the end."

If the choice is 5 (display):

- Check if the circular linked list is empty (head == NULL).
  - If it is, display "Circular Linked List is empty."
    Otherwise, create a temporary pointer **temp** and set it to **head**.
    - Traverse the circular linked list starting from **temp** until **temp->next** points to **head**.
      Display the value of **temp->data** and update **temp** to **temp->next**.

If the choice is 6 (exit), display "Exiting the program."

	If the choice is invalid, display "Invalid choice. Please try again."
5. End	

Tool: Dev c++.

Source Code:

```
#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node* next;

};

struct Node* head = NULL;

void insertAtBeginning(int value) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

    newNode->data = value;

    if (head == NULL) {

        newNode->next = newNode;

        head = newNode;

    } else {
```

```
struct Node* temp = head->next;

newNode->next = temp;

        head->next = newNode;

    }

    printf("Element %d inserted at the beginning successfully.\n", value);

}

void insertAtEnd(int value) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

newNode->data = value;

if (head == NULL) {

    newNode->next = newNode;

    head = newNode;

} else {

    struct Node* temp = head->next;

newNode->next = temp;

head->next = newNode;

head = newNode;
```

```
}

printf("Element %d inserted at the end successfully.\n", value);

}

void deleteFromBeginning() {

    if (head == NULL) {

        printf("Circular Linked List is empty, cannot delete.\n");

    } else {

        struct Node* temp = head->next;

    if (temp == head) {

        head = NULL;

    } else {

        head->next = temp->next;

    }

    free(temp);

    }

    printf("Element deleted from the beginning.\n");

}
```

```
void deleteFromEnd() {

    if (head == NULL) {

        printf("Circular Linked List is empty, cannot delete.\n");

    } else {

        struct Node* prev = head;

        struct Node* temp = head->next;

        while (temp->next != head) {

            prev = temp;

            temp = temp->next;

        }

        prev->next = head;

        free(temp);

    }

    printf("Element deleted from the end.\n");

}

void display() {

    if (head == NULL) {
```

```
    printf("Circular Linked List is empty.\n");

} else {

    struct Node* temp = head->next;

    printf("Circular Linked List elements: ");

    while (temp != head) {

        printf("%d ", temp->data);

        temp = temp->next;

    }

        printf("\n");

    }

}

int main() {

    int choice, value;

    do {

        printf("\n—- Circular Linked List Menu -—\n");

        printf("1. Insert at Beginning\n");

        printf("2. Insert at End\n");
```

```
printf("3. Delete from Beginning\n");

printf("4. Delete from End\n");

        printf("5. Display\n");

        printf("6. Exit\n");

        printf("-----------------\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

    switch (choice) {

        case 1:

            printf("Enter the value to insert at the beginning: ");

            scanf("%d", &value);

            insertAtBeginning(value);

            break;

        case 2:

            printf("Enter the value to insert at the end: ");

            scanf("%d", &value);

            insertAtEnd(value);
```

```
            break;

        case 3:
            deleteFromBeginning();
            break;
        case 4:
            deleteFromEnd();
            break;
        case 5:
            display();
            break;
        case 6:
            printf("Exiting the program.\n");
            break;
        default:
                printf("Invalid choice. Please try again.\n")
        }
    } while (choice != 6);
```

```
 return 0;

}
```

Output:-

```
C:\Users\purvi\Documents\ds

----- Circular Linked List Menu -----
1. Insert at Beginning
2. Insert at End
3. Delete from Beginning
4. Delete from End
5. Display
6. Exit
--------------------------------------
Enter your choice: 1
Enter the value to insert at the beginning: 5
Element 5 inserted at the beginning successfully.

----- Circular Linked List Menu -----
1. Insert at Beginning
2. Insert at End
3. Delete from Beginning
4. Delete from End
5. Display
6. Exit
--------------------------------------
Enter your choice: 2
Enter the value to insert at the end: 9
Element 9 inserted at the end successfully.

----- Circular Linked List Menu -----
1. Insert at Beginning
2. Insert at End
3. Delete from Beginning
4. Delete from End
```

```
C:\Users\purvi\Documents\ds

3. Delete from Beginning
4. Delete from End
5. Display
6. Exit
--------------------------------------
Enter your choice: 3
Element deleted from the beginning.

----- Circular Linked List Menu -----
1. Insert at Beginning
2. Insert at End
3. Delete from Beginning
4. Delete from End
5. Display
6. Exit
--------------------------------------
Enter your choice: 4
Element deleted from the end.

----- Circular Linked List Menu -----
1. Insert at Beginning
2. Insert at End
3. Delete from Beginning
4. Delete from End
5. Display
6. Exit
--------------------------------------
Enter your choice: 5
Circular Linked List elements:
```

```
C:\Users\purvi\Documents\ds

6. Exit
--------------------------------------
Enter your choice: 4
Element deleted from the end.

----- Circular Linked List Menu -----
1. Insert at Beginning
2. Insert at End
3. Delete from Beginning
4. Delete from End
5. Display
6. Exit
--------------------------------------
Enter your choice: 5
Circular Linked List elements:

----- Circular Linked List Menu -----
1. Insert at Beginning
2. Insert at End
3. Delete from Beginning
4. Delete from End
5. Display
6. Exit
--------------------------------------
Enter your choice: 6
Exiting the program.

--------------------------------
Process exited after 40.57 seconds with return value 0
Press any key to continue . . .
```

Program No: 5

Aim :- Write a menu driven program to implement various operations as push, pop, display, isFull and isEmpty in a stack with the help of static memory allocation.

Algorithm

1. Start
2. Define a constant **MAX_SIZE** to represent the maximum size of the stack.
3. Declare an integer array **stack** of size **MAX_SIZE** to store the stack elements.
4. Declare an integer variable **top** and initialize it to -1 to indicate an empty stack.
5. Repeat the following steps until the user chooses to exit:
    - Display the menu options: push, pop, display, isFull, isEmpty, and exit.
    - Read the user's choice.
    - Perform the corresponding operation based on the choice using a switch statement:
        - If the choice is 1 (push):
            - Check if the stack is full (top == MAX_SIZE - 1).
                - If it is, display "Stack Overflow: Cannot push element, stack is full."
                - Otherwise, prompt the user to enter a value to push.
                    - Increment **top** by 1.
                    - Assign the entered value to **stack[top]**.
                    - Display "Element <value> pushed successfully."
        - If the choice is 2 (pop):
            - Check if the stack is empty (top == -1).
                - If it is, display "Stack Underflow: Cannot pop element, stack is empty."
                - Otherwise, display "Element <value> popped." where **<value>** is **stack[top]**.
                    - Decrement **top** by 1.
        - If the choice is 3 (display):
            - Check if the stack is empty (top == -1).

If it is, display "Stack is empty."
Otherwise, display "Stack elements: " followed by the elements in the stack from **stack[top]** to **stack[0]**.

If the choice is 4 (isFull):
Check if the stack is full (top == MAX_SIZE - 1).
If it is, display "Stack is full."
Otherwise, display "Stack is not full."

If the choice is 5 (isEmpty):
Check if the stack is empty (top == -1).
If it is, display "Stack is empty."
Otherwise, display "Stack is not empty."

If the choice is 6 (exit), display "Exiting the program."
If the choice is invalid, display "Invalid choice. Please try again."

6. End

Tool :- Dev-c++

Sourse code :-

```
#include<stdio.h>
#define MAX_SIZE 10

int stack[MAX_SIZE];
int top = -1;

void push(int value){
    if(top == MAX_SIZE-1){
        printf("Stack Overflow: Cannot push element, stack is full.\n");
    }else{
```

```
    stack[++top] = value;
    printf("Element %d pushed successfully.\n", value);
}
}
void pop(){
    if(top == -1){
        printf("Stack Underflow: Cannot pop element, stack is empty.\n");
    }else{
        int value = stack[top--];
        printf("Element %d popped.\n", value);
    }
}
void display(){
    if(top == -1){
        printf("Stack is empty.\n");
    }else{
        printf("Stack elements: ");
        int i;
        for(i = top; i >= 0; i--){
            printf("%d ", stack[i]);
        }
        printf("\n");
```

```
    }
}

int isFull(){
    return top == MAX_SIZE-1;
}
int isEmpty(){
    return top == -1;
}
int main(){
    int choice, value;
    do{
        printf("\n---Stack Menu---\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. isFull\n");
        printf("5. isEmpty\n");
        printf("6. Exit\n");
        printf("--------------\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
switch(choice){
    case 1:
        printf("Enter the value to push: ");
        scanf("%d", &value);
        push(value);
        break;
    case 2:
        pop();
        break;
    case 3:
        display();
        break;
    case 4:
        if (isFull()){
            printf("Stack is full.\n");
        }else{
            printf("Stack is not full.\n");
        }
        break;
    case 5:
        if (isEmpty()){
```

```
            printf("Stack is empty.\n");
        }else{
            printf("Stack is not empty.\n");
        }
        break;
    case 6:
        printf("Exiting the program.\n");
        break;
    default:
        printf("Invalid choice. Please try again.\n");
        }
    }while(choice!= 6);

    return 0;
}
```

Output :-

```
Enter the value to push: 4
Element 4 pushed successfully.

----- Stack Menu -----
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
----------------------
Enter your choice: 2
Element 4 popped.

----- Stack Menu -----
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
----------------------
Enter your choice: 3
Stack elements: 3

----- Stack Menu -----
1. Push
2. Pop
3. Display
4. isFull
```

```
----- Stack Menu -----
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
----------------------
Enter your choice: 4
Stack is not full.

----- Stack Menu -----
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
----------------------
Enter your choice: 5
Stack is not empty.

----- Stack Menu -----
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
```

```
6. Exit
----------------------
Enter your choice: 4
Stack is not full.

----- Stack Menu -----
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
----------------------
Enter your choice: 5
Stack is not empty.

----- Stack Menu -----
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
----------------------
Enter your choice: 6
Exiting the program.

--------------------------------
Process exited after 30.81 seconds with return value 0
Press any key to continue . . .
```

Program No: 6

AIM:- Write a menu driven program to implement various operations as push, pop, display, isFull and isEmpty in a stack with the help of dynamic memory allocation.

Algorithm

1. Start
2. Declare a structure named `Node` with an integer `data` member and a pointer to the next node named `next`.
3. Declare a pointer variable `top` and set it to NULL to indicate an empty stack.
4. Repeat the following steps until the user chooses to exit:
    - Display the menu options: push, pop, display, isFull, isEmpty, and exit.
    - Read the user's choice.
    - Perform the corresponding operation based on the choice using a switch statement:
        - If the choice is 1 (push):
            - Prompt the user to enter a value to push.
            - Create a new node dynamically using `malloc`.
            - Assign the entered value to the `data` member of the new node.
            - Set the `next` pointer of the new node to the current top node.
            - Update the `top` pointer to point to the new node.
            - Display "Element <value> pushed successfully."
        - If the choice is 2 (pop):
            - Check if the stack is empty (top == NULL).
                - If it is, display "Stack Underflow: Cannot pop element, stack is empty."
                - Otherwise, create a temporary pointer `temp` and set it to the current top node.
                - Retrieve the value from the `data` member of `temp`.
                - Update the `top` pointer to point to the next node.
                - Free the memory occupied by `temp`.
                - Display "Element <value> popped."
        - If the choice is 3 (display):
            - Check if the stack is empty (top == NULL).

If it is, display "Stack is empty."
Otherwise, create a temporary pointer **temp** and set it to the current top node.
Display "Stack elements: " followed by the elements in the stack by traversing the nodes using the **next** pointer of **temp**.
If the choice is 4 (isFull):
Dynamic memory allocation does not have a concept of a full stack, so display "Stack is not full."
If the choice is 5 (isEmpty):
Check if the stack is empty (top == NULL).
If it is, display "Stack is empty."
Otherwise, display "Stack is not empty."
If the choice is 6 (exit), display "Exiting the program."
If the choice is invalid, display "Invalid choice. Please try again."

5. End

Tool :- Dev-c++

Sourse code :-

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

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

```
struct Node* top = NULL;


void push(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = top;
    top = newNode;
    printf("Element %d pushed successfully.\n", value);
}


void pop() {
    if (top == NULL) {
        printf("Stack Underflow: Cannot pop element, stack is empty.\n");
    } else {
        struct Node* temp = top;
        int value = temp->data;
        top = top->next;
        free(temp);
        printf("Element %d popped.\n", value);
    }
}
```

```
void display() {
    if (top == NULL) {
        printf("Stack is empty.\n");
    } else {
        struct Node* temp = top;
        printf("Stack elements: ");
        while (temp != NULL) {
            printf("%d ", temp->data);
            temp = temp->next;
        }
        printf("\n");
    }
}
int isEmpty() {
    return top == NULL;
}
int main() {
    int choice, value;
    do {
        printf("\n—- Stack Menu —–\n");
        printf("1. Push\n");
        printf("2. Pop\n");
```

```
printf("3. Display\n");
printf("4. isFull\n");
printf("5. isEmpty\n");
printf("6. Exit\n");
printf("————————\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter the value to push: ");
        scanf("%d", &value);
        push(value);
        break;
    case 2:
        pop();
        break;
    case 3:
        display();
        break;
    case 4:
        printf("Stack is not full.\n");
```

```
            break;
        case 5:
            if (isEmpty()) {
                printf("Stack is empty.\n");
            } else {
                printf("Stack is not empty.\n");
            }
            break;
        case 6:
            printf("Exiting the program.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 6);

    return 0;
}
```

Output :-

```
-----  Stack Menu  -----
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
-------------------------
Enter your choice: 1
Enter the value to push: 2
Element 2 pushed successfully.

-----  Stack Menu  -----
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
-------------------------
Enter your choice: 1
Enter the value to push: 3
Element 3 pushed successfully.

-----  Stack Menu  -----
1. Push
2. Pop
3. Display
4. isFull
```

```
4. isFull
5. isEmpty
6. Exit
-------------------------
Enter your choice: 2
Element 3 popped.

-----  Stack Menu  -----
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
-------------------------
Enter your choice: 3
Stack elements: 2

-----  Stack Menu  -----
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
-------------------------
Enter your choice: 4
Stack is not full.

-----  Stack Menu  -----
```

```
6. Exit
-------------------------
Enter your choice: 4
Stack is not full.

-----  Stack Menu  -----
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
-------------------------
Enter your choice: 5
Stack is not empty.

-----  Stack Menu  -----
1. Push
2. Pop
3. Display
4. isFull
5. isEmpty
6. Exit
-------------------------
Enter your choice: 6
Exiting the program.

--------------------------------
Process exited after 27.9 seconds with return value 0
Press any key to continue . . .
```

Program No: 7

AIM :- Illustrate queue implementation using array with following operation as enQueue, deQueue, isEmpty, displayQueue.

Algorithm

1. Start
2. Declare an integer array **queue** to store the queue elements.
3. Declare two integer variables **front** and **rear** and initialize them to -1 to indicate an empty queue.
4. Repeat the following steps until the user chooses to exit:
    - Display the menu options: enqueue, dequeue, isEmpty, displayQueue, and exit.
    - Read the user's choice.
    - Perform the corresponding operation based on the choice using a switch statement:
        - If the choice is 1 (enqueue):
            - Check if the queue is full (rear == MAX_SIZE - 1).
                - If it is, display "Queue Overflow: Cannot enqueue element, queue is full."
                - Otherwise, prompt the user to enter a value to enqueue.
                    - Increment **rear** by 1.
                    - Assign the entered value to **queue[rear]**.
                    - If **front** is -1, set it to 0.
                    - Display "Element <value> enqueued successfully."
        - If the choice is 2 (dequeue):
            - Check if the queue is empty (front == -1 or front > rear).
                - If it is, display "Queue Underflow: Cannot dequeue element, queue is empty."
                - Otherwise, retrieve the element at the front of the queue.
                    - Increment **front** by 1.
                    - Display "Element <value> dequeued."

If **front** becomes greater than **rear**, set both **front** and **rear** to -1 to indicate an empty queue.

If the choice is 3 (isEmpty):

Check if the queue is empty (front == -1 or front > rear).

If it is, display "Queue is empty."

Otherwise, display "Queue is not empty."

If the choice is 4 (displayQueue):

Check if the queue is empty (front == -1 or front > rear).

If it is, display "Queue is empty."

Otherwise, display "Queue elements: " followed by the elements in the queue from **queue[front]** to **queue[rear]**.

If the choice is 5 (exit), display "Exiting the program."

If the choice is invalid, display "Invalid choice. Please try again."

5. End

Tool :- Dev-c++

Sourse code :-

```
#include <stdio.h>
#define MAX_SIZE 10

int queue[MAX_SIZE];
int front = -1, rear = -1;

void enqueue(int value) {
    if (rear == MAX_SIZE - 1) {
        printf("Queue Overflow: Cannot enqueue element, queue is full.\n");
    } else {
```

```
    queue[++rear] = value;
    if (front == -1) {
        front = 0;
    }
    printf("Element %d enqueued successfully.\n", value);
  }
}

void dequeue() {
  if (front == -1 || front > rear) {
    printf("Queue Underflow: Cannot dequeue element, queue is
empty.\n");
  } else {
    int value = queue[front++];
    printf("Element %d dequeued.\n", value);
    if (front > rear) {
      front = rear = -1;
    }
  }
}
int isEmpty() {
  return (front == -1 || front > rear);
```

```
}
void displayQueue() {
    if (front == -1 || front > rear) {
        printf("Queue is empty.\n");
    } else {
        printf("Queue elements: ");
        int i;
                for (i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}
int main() {
    int choice, value;
    do {
        printf("\n—- Queue Menu -—\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. isEmpty\n");
        printf("4. Display Queue\n");
        printf("5. Exit\n");
```

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

switch (choice) {
    case 1:
        printf("Enter the value to enqueue: ");
        scanf("%d", &value);
        enqueue(value);
        break;
    case 2:
        dequeue();
        break;
    case 3:
        if (isEmpty()) {
            printf("Queue is empty.\n");
        } else {
            printf("Queue is not empty.\n");
        }
        break;
    case 4:
        displayQueue();
```

```
            break;
        case 5:
            printf("Exiting the program.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
  } while (choice != 5);
  return 0;
}
```

Output :-

```
C:\Users\purvi\Documents\ds

----- Queue Menu -----
1. Enqueue
2. Dequeue
3. isEmpty
4. Display Queue
5. Exit
----------------------
Enter your choice: 1
Enter the value to enqueue: 8
Element 8 erqueued successfully.

----- Queue Menu -----
1. Enqueue
2. Dequeue
3. isEmpty
4. Display Queue
5. Exit
----------------------
Enter your choice: 1
Enter the value to enqueue: 9
Element 9 erqueued successfully.

----- Queue Menu -----
1. Enqueue
2. Dequeue
3. isEmpty
4. Display Queue
5. Exit
----------------------
```

```
C:\Users\purvi\Documents\ds
4. Display Queue
5. Exit
----------------------
Enter your choice: 2
Element 8 dequeued.

----- Queue Menu -----
1. Enqueue
2. Dequeue
3. isEmpty
4. Display Queue
5. Exit
----------------------
Enter your choice: 3
Queue is not empty.

----- Queue Menu -----
1. Enqueue
2. Dequeue
3. isEmpty
4. Display Queue
5. Exit
----------------------
Enter your choice: 4
Queue elements: 9

----- Queue Menu -----
1. Enqueue
2. Dequeue
3. isEmpty
```

```
C:\Users\purvi\Documents\ds
3. isEmpty
4. Display Queue
5. Exit
----------------------
Enter your choice: 3
Queue is not empty.

----- Queue Menu -----
1. Enqueue
2. Dequeue
3. isEmpty
4. Display Queue
5. Exit
----------------------
Enter your choice: 4
Queue elements: 9

----- Queue Menu -----
1. Enqueue
2. Dequeue
3. isEmpty
4. Display Queue
5. Exit
----------------------
Enter your choice: 5
Exiting the program.

--------------------------------
Process exited after 13.65 seconds with return value 0
Press any key to continue . . .
```

Program No:8

AIM :- Illustrate queue implementation using linked list with following operation as enQueue, deQueue, isEmpty, displayQueue.

Algorithm :-

1. Start
2. Declare a structure named **Node** with an integer **data** member and a pointer to the next node named **next**.
3. Declare two pointer variables **front** and **rear** and set them to NULL to indicate an empty queue.
4. Repeat the following steps until the user chooses to exit:
    - Display the menu options: enqueue, dequeue, isEmpty, displayQueue, and exit.
    - Read the user's choice.
    - Perform the corresponding operation based on the choice using a switch statement:
        - If the choice is 1 (enqueue):
            - Prompt the user to enter a value to enqueue.
            - Create a new node dynamically using **malloc**.
            - Assign the entered value to the **data** member of the new node.
            - Set the **next** pointer of the new node to NULL.
            - If **front** is NULL, set both **front** and **rear** to the new node.
                - Otherwise, set the **next** pointer of **rear** to the new node and update **rear** to the new node.
            - Display "Element <value> enqueued successfully."
        - If the choice is 2 (dequeue):
            - Check if the queue is empty (front == NULL).
                - If it is, display "Queue Underflow: Cannot dequeue element, queue is empty."
                - Otherwise, create a temporary pointer **temp** and set it to the current front node.
                - Retrieve the value from the **data** member of **temp**.
                - Update the **front** pointer to point to the next node.

Free the memory occupied by **temp**.
If **front** becomes NULL, set **rear** to NULL as well to indicate an empty queue.

Display "Element <value> dequeued."

If the choice is 3 (isEmpty):

Check if the queue is empty (front == NULL).

If it is, display "Queue is empty."
Otherwise, display "Queue is not empty."

If the choice is 4 (displayQueue):

Check if the queue is empty (front == NULL).

If it is, display "Queue is empty."
Otherwise, create a temporary pointer **temp** and set it to the current front node.

Display "Queue elements: " followed by the elements in the queue by traversing the nodes using the **next** pointer of **temp**.

If the choice is 5 (exit), display "Exiting the program."
If the choice is invalid, display "Invalid choice. Please try again."

5. End

Tool:- Dev c++.

Source Code:

```
#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node* next;

};

struct Node* front = NULL;

struct Node* rear = NULL;
```

```
void enqueue(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    if (front == NULL) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
    printf("Element %d enqueued successfully.\n", value);
}
void dequeue() {
    if (front == NULL) {
                printf("Queue Underflow: Cannot dequeue element, queue is empty.\n");
    } else {
        struct Node* temp = front;
        int value = temp->data;
        front = front->next;
        free(temp);
        if (front == NULL) {
```

```
    rear = NULL;
  }
  printf("Element %d dequeued.\n", value);
    }
}
int isEmpty() {
    return front == NULL;
}
void displayQueue() {
    if (front == NULL) {
    printf("Queue is empty.\n");
  } else {
    struct Node* temp = front;
    printf("Queue elements: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
        }
        printf("\n");
    }
}
int main() {
```

```
int choice, value;
do {
    printf("\n—- Queue Menu -—\n");
    printf("1. Enqueue\n");
    printf("2. Dequeue\n");
    printf("3. isEmpty\n");
    printf("4. Display Queue\n");
    printf("5. Exit\n");
    printf("————————\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch (choice) {
        case 1:
            printf("Enter the value to enqueue: ");
            scanf("%d", &value);
            enqueue(value);
            break;
        case 2:
            dequeue();
            break;
        case 3:
            if (isEmpty()) {
```

```
            printf("Queue is empty.\n");
        } else {
            printf("Queue is not empty.\n");
        }
        break;
    case 4:
        displayQueue();
        break;
    case 5:
        printf("Exiting the program.\n");
        break;
    default:
        printf("Invalid choice. Please try again.\n");
    }
  } while (choice != 5);
  return 0;
}
```

Output :-

```
C:\Users\purvi\Documents\ds

----- Queue Menu -----
1. Enqueue
2. Dequeue
3. isEmpty
4. Display Queue
5. Exit
----------------------
Enter your choice: 1
Enter the value to enqueue: 8
Element 8 enqueued successfully.

----- Queue Menu -----
1. Enqueue
2. Dequeue
3. isEmpty
4. Display Queue
5. Exit
----------------------
Enter your choice: 1
Enter the value to enqueue: 8
Element 8 enqueued successfully.

----- Queue Menu -----
1. Enqueue
2. Dequeue
3. isEmpty
4. Display Queue
5. Exit
----------------------
```

```
    case 4:
C:\Users\purvi\Documents\ds
----------------------
Enter your choice: 2
Element 8 dequeued.

----- Queue Menu -----
1. Enqueue
2. Dequeue
3. isEmpty
4. Display Queue
5. Exit
----------------------
Enter your choice: 3
Queue is not empty.

----- Queue Menu -----
1. Enqueue
2. Dequeue
3. isEmpty
4. Display Queue
5. Exit
----------------------
Enter your choice: 4
Queue elements: 8

----- Queue Menu -----
1. Enqueue
2. Dequeue
3. isEmpty
4. Display Queue
5. Exit
```

```
    case 4:
C:\Users\purvi\Documents\ds
1. isEmpty
4. Display Queue
5. Exit
----------------------
Enter your choice: 3
Queue is not empty.

----- Queue Menu -----
1. Enqueue
2. Dequeue
3. isEmpty
4. Display Queue
5. Exit
----------------------
Enter your choice: 4
Queue elements: 8

----- Queue Menu -----
1. Enqueue
2. Dequeue
1. isEmpty
4. Display Queue
5. Exit
----------------------
Enter your choice: 5
Exiting the program.

--------------------------------
Process exited after 14.86 seconds with return value 0
Press any key to continue . . .
```