Data Structures Lab Using C/C++

Subject Code: MCAL11

A Practical Journal Submitted in Fulfilment of Degree of

Master in Computer Application

Year 2023-2024

By

Mr. Sagar Govind Kamtekar

(Application ID: 81690)

Semester – 1

Institute of Distance and Open Learning Vidya Nagari, Kalina,
Santacruz East – 400098,

University of Mumbai

PCP Center

[Vidyavardhini's College of Engineering & Technology, Vasai Road]

Sagar Govind Kamtekar (81690)

Institute of Distance and Open Learning Vidya Nagari, Kalina, Santacruz East – 400098,

University of Mumbai

CERTIFICATE

This is to certify that**, Mr. Sagar Govind Kamtekar** appearing **Master's in Computer Application (Semester 1) Application ID: 81690**, has satisfactorily completed the prescribed practical of **MCA11 – Data Structures Lab Using C/C++** as laid down by the University of Mumbai for the academic year 2023-24.

Teacher in charge Examiners Coordinator

IDOL, MCA

University of Mumbai

Date: Place:

INDEX

Practical 1

AIM:

Implementation of Bubble Sort in C.

OBJECTIVE:

The objective of implementing Bubble Sort in the C language is to create a straightforward and intuitive sorting algorithm that rearranges elements in an array, either in ascending or descending order, through the repetitive swapping of adjacent elements.

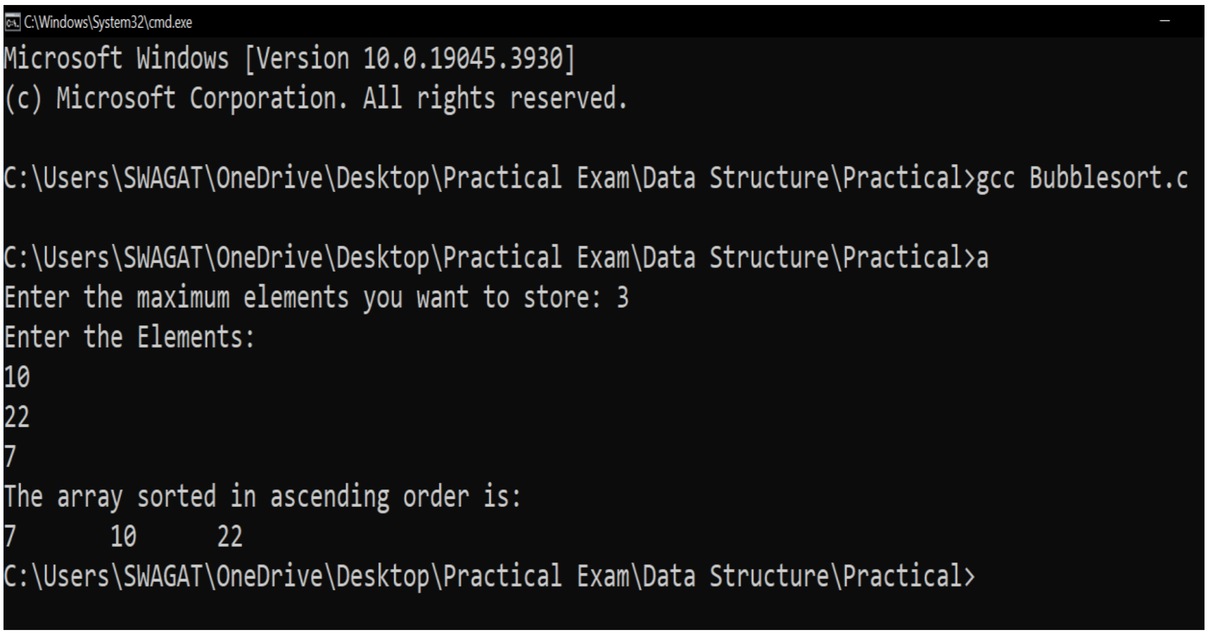
THEORY:

Bubble Sort is a simple sorting algorithm that works by repeatedly stepping through the array, comparing adjacent elements, and swapping them if they are in the wrong order. The process is repeated for the entire array until no more swaps are needed, indicating that the array is sorted.

CODE:

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

int main()
{
   int i, n, temp, j, arr[10];

   printf("Enter the maximum elements you want to store: ");
   scanf("%d", &n);

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

   for(i=0; i<n; i++)
   {
      for(j=0; j<n-1; j++)
      {
         if(arr[j] > arr[j+1])
         {
                temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }

    printf("The array sorted in ascending order is: \n");
    for(i=0; i<n; i++)
        printf("%d\t", arr[i]);

    return 0;
}
```

OUTPUT:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.3930]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical>gcc Bubblesort.c

C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical>a
Enter the maximum elements you want to store: 3
Enter the Elements:
10
22
7
The array sorted in ascending order is:
7       10      22
C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical>
```

Practical 2

AIM:

Implementation of Insertion Sort in C.

OBJECTIVE:

The objective of implementing Insertion Sort in the C language is to arrange elements in an array in ascending order by iteratively inserting elements from the unsorted part into their correct positions within the sorted part of the array.

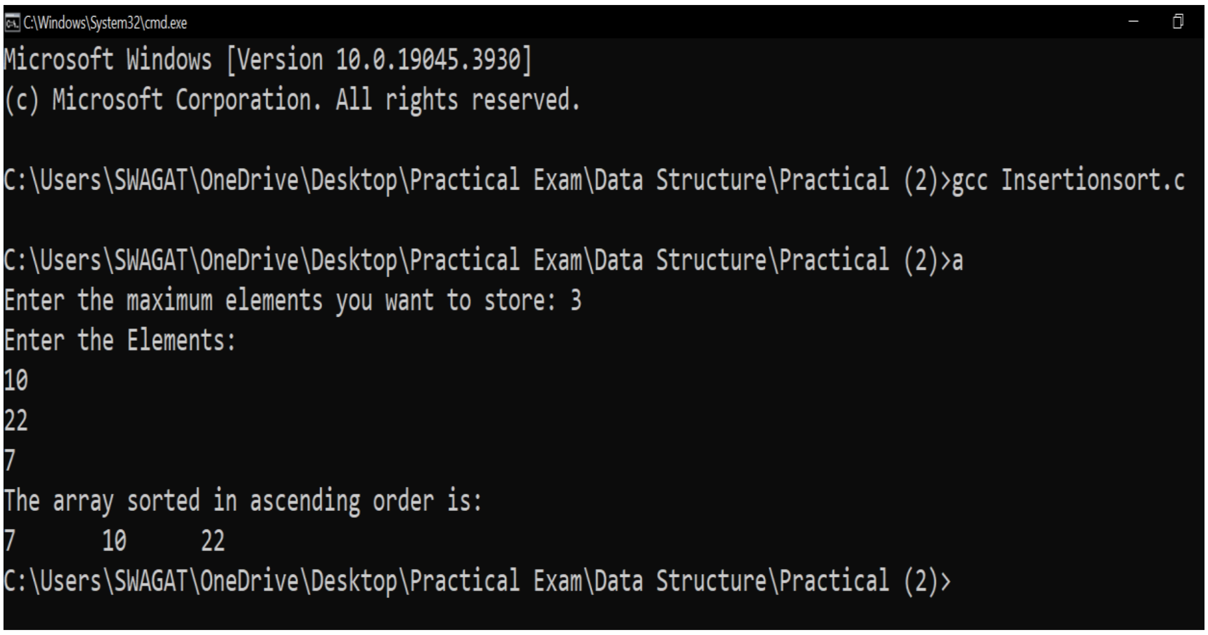
THEORY:

nsertion Sort is a simple sorting algorithm that builds the final sorted array one element at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort, but it has several advantages in certain scenarios.

CODE:

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

int main()
{
   int i, j, key, n, arr[10];

   printf("Enter the maximum elements you want to store: ");
   scanf("%d", &n);

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

   for(i=1; i<n; i++)
   {
      key = arr[i];
      j = i - 1;

      while(j >= 0 && arr[j] > key)
      {
            arr[j + 1] = arr[j];
            j = j - 1;
         }

         arr[j + 1] = key;
      }

   printf("The array sorted in ascending order is: \n");
   for(i=0; i<n; i++)
      printf("%d\t", arr[i]);

   return 0;
}
```

OUTPUT:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.3930]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (2)>gcc Insertionsort.c

C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (2)>a
Enter the maximum elements you want to store: 3
Enter the Elements:
10
22
7
The array sorted in ascending order is:
7       10      22
C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (2)>
```

Practical 3

AIM:

Implementation of Selection Sort in C.

OBJECTIVE:

The objective of implementing Selection Sort in the C language is to arrange elements in an array in ascending order by iteratively selecting the minimum element from the unsorted part of the array and swapping it with the first unsorted element.

THEORY:

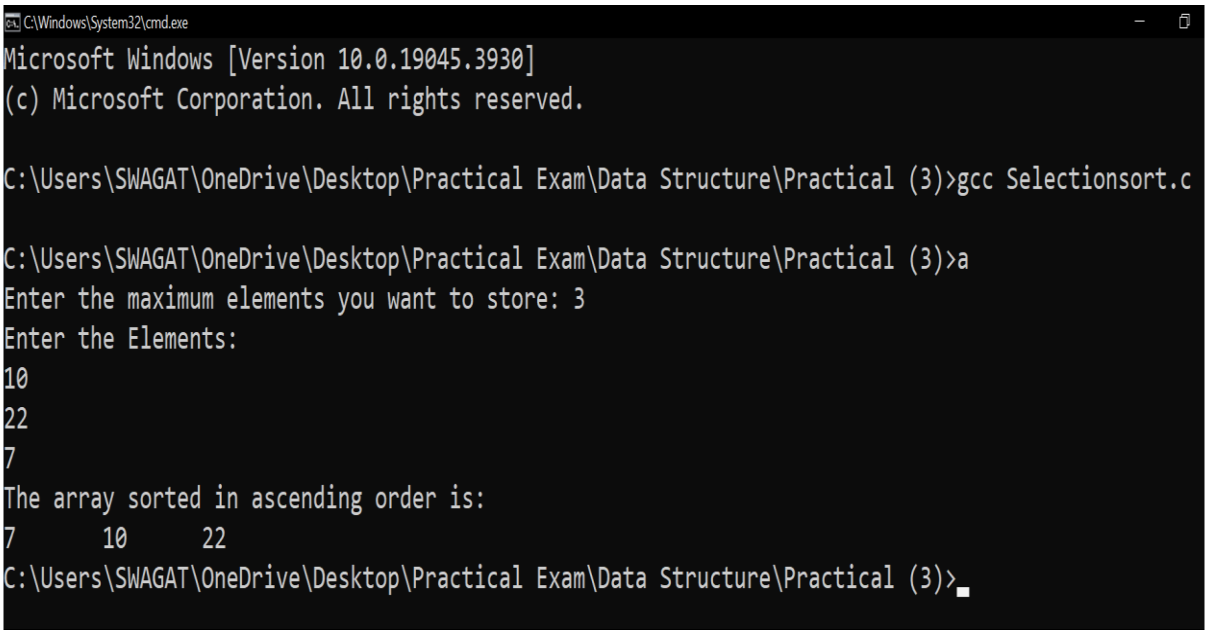
Selection Sort is a simple sorting algorithm that sorts an array by repeatedly finding the minimum element from the unsorted part of the array and putting it at the beginning. The algorithm maintains two subarrays within the given array: the sorted subarray, which is built up from left to right, and the unsorted subarray, containing the remaining elements.

CODE:

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

int main()
{
   int i, n, temp, j, arr[10];

   printf("Enter the maximum elements you want to store: ");
   scanf("%d", &n);

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

   for(i=0; i<n-1; i++)
   {
      int min_index = i;

      for(j=i+1; j<n; j++)
```

```
        {
            if(arr[j] < arr[min_index])
            {
                min_index = j;
            }
        }

        temp = arr[i];
        arr[i] = arr[min_index];
        arr[min_index] = temp;
    }

    printf("The array sorted in ascending order is: \n");
    for(i=0; i<n; i++)
        printf("%d\t", arr[i]);

    return 0;
}
```

OUTPUT:



```
Microsoft Windows [Version 10.0.19045.3930]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (3)>gcc Selectionsort.c

C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (3)>a
Enter the maximum elements you want to store: 3
Enter the Elements:
10
22
7
The array sorted in ascending order is:
7       10      22
C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (3)>
```

Practical 4

AIM:

Implementation of Linear Search in C.

OBJECTIVE:

The objective of implementing Linear Search in the C language is to find the position of a specific element within an array.

THEORY:

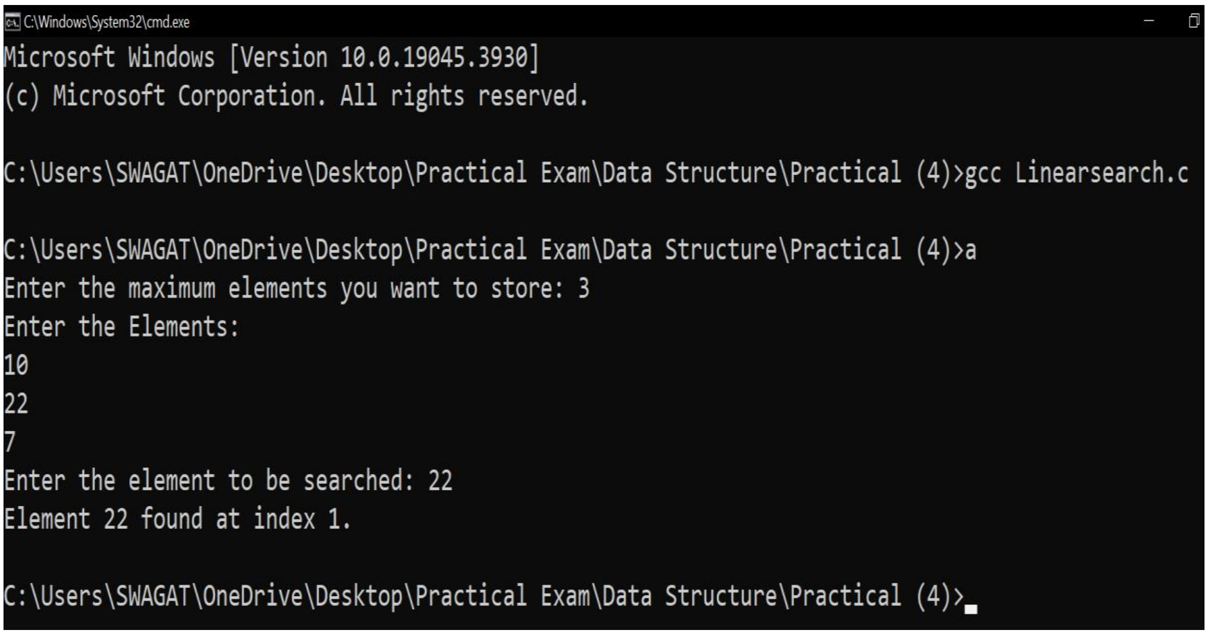
Linear Search, also known as Sequential Search, is a simple algorithm used to find the position of a target element within an array. It involves iterating through each element of the array and comparing it with the target value.

CODE:

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

int main()
{
   int i, n, key, arr[10];

   printf("Enter the maximum elements you want to store: ");
   scanf("%d", &n);

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

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

   int found = 0;
   for(i = 0; i < n; i++)
   {
      if(arr[i] == key)
      {
         found = 1;
```

```
            printf("Element %d found at index %d.\n", key, i);
            break;
        }
    }

    if(!found)
    {
        printf("Element %d not found in the array.\n", key);
    }

    return 0;
}
```

OUTPUT:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.3930]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (4)>gcc Linearsearch.c

C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (4)>a
Enter the maximum elements you want to store: 3
Enter the Elements:
10
22
7
Enter the element to be searched: 22
Element 22 found at index 1.

C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (4)>
```

Practical 5

AIM:

Implementation of Binary Search in C.

OBJECTIVE:

The objective of implementing Binary Search in the C language is to efficiently locate a specific element within a sorted array.

THEORY:

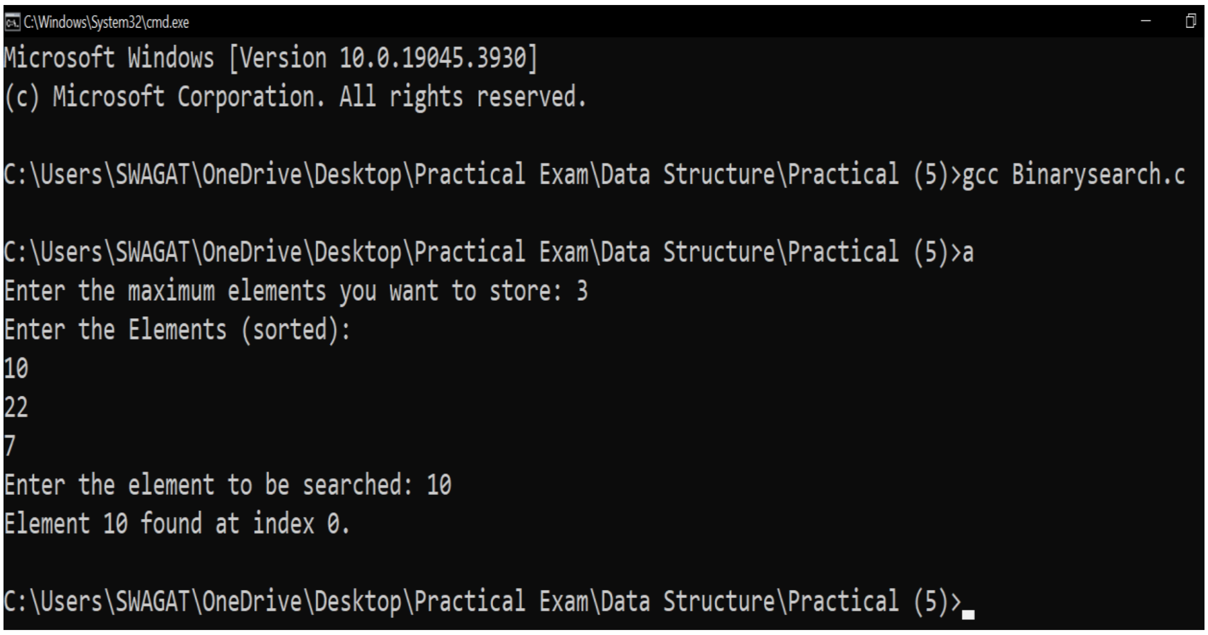
Binary Search is a divide-and-conquer algorithm used to find the position of a target value within a sorted array. The basic idea is to repeatedly divide the search space in half until the target element is found or the search space is empty.

CODE:

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

int binarySearch(int arr[], int low, int high, int key);

int main()
{
   int i, n, key, arr[10];

   printf("Enter the maximum elements you want to store: ");
   scanf("%d", &n);

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

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

   int index = binarySearch(arr, 0, n - 1, key);

   if(index != -1)
```

```
    {
        printf("Element %d found at index %d.\n", key, index);
    }
    else
    {
        printf("Element %d not found in the array.\n", key);
    }
    return 0;
}

int binarySearch(int arr[], int low, int high, int key)
{
    while (low <= high)
    {
        int mid = low + (high - low) / 2;

        if (arr[mid] == key)
            return mid;

        if (arr[mid] < key)
            low = mid + 1;

        else
            high = mid - 1;
    }
    return -1;
}
```

OUTPUT:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.3930]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (5)>gcc Binarysearch.c

C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (5)>a
Enter the maximum elements you want to store: 3
Enter the Elements (sorted):
10
22
7
Enter the element to be searched: 10
Element 10 found at index 0.

C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (5)>
```

Practical 6

AIM:

Implementation of Stack using Array in C.

OBJECTIVE:

The objective of implementing a Stack using an array in the C language is to create a data structure that follows the Last In, First Out (LIFO) principle.

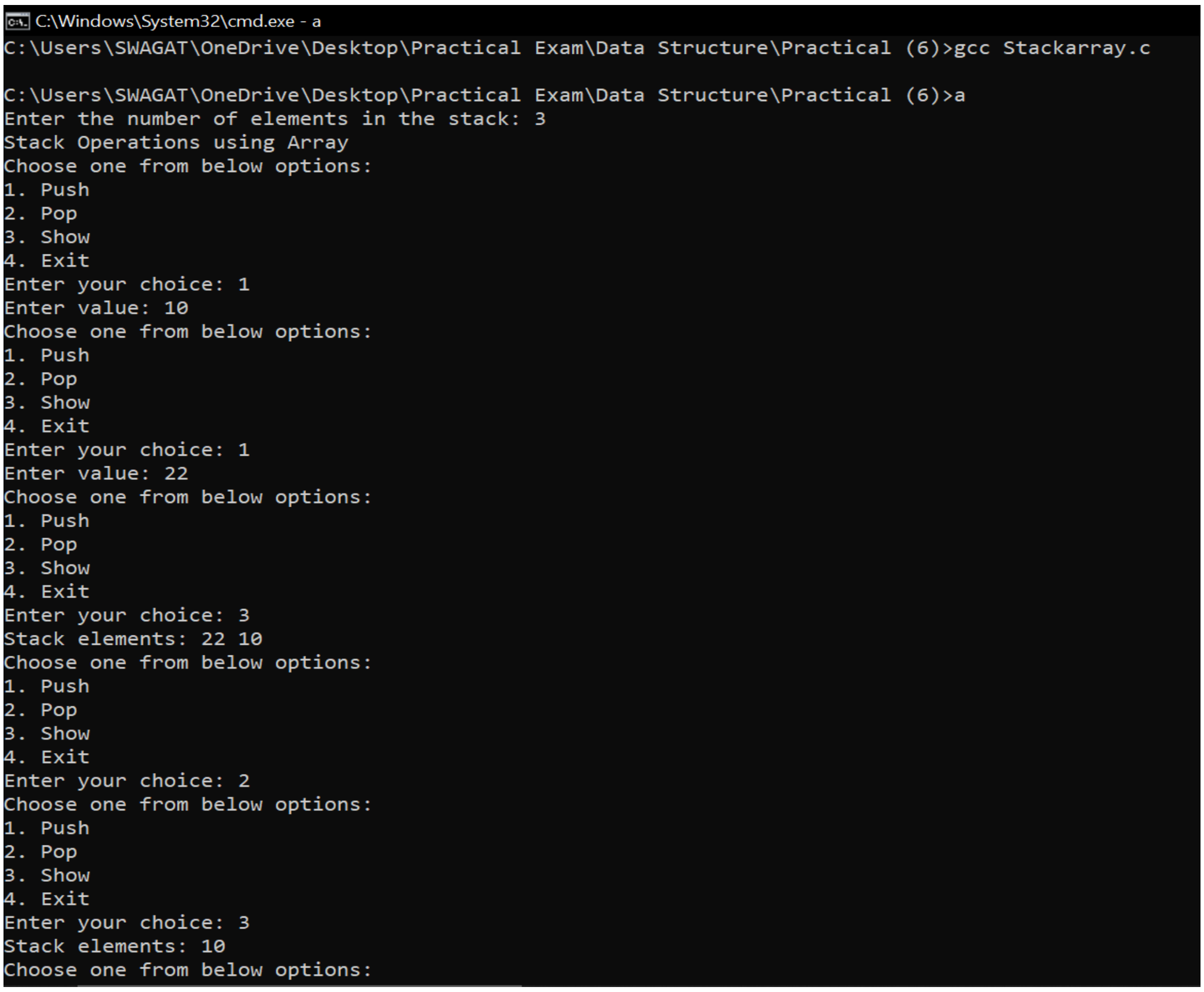
THEORY:

A Stack is a linear data structure that follows the Last In, First Out (LIFO) principle. Implementing a Stack using an array provides a simple and efficient way to manage elements in a sequential manner.

CODE:

```
#include<stdio.h>

int stack[100], i, j, choice = 0, n, top = -1;

void push();
void pop();
void show();

void main() {
    printf("Enter the number of elements in the stack: ");
    scanf("%d", &n);
    printf("Stack Operations using Array\n");

    while (choice != 4) {
        printf("Choose one from below options:\n");
        printf("1. Push\n2. Pop\n3. Show\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                push();
                break;

            case 2:
                pop();
                break;

            case 3:
                show();
                break;

            case 4:
                printf("Exiting...\n");
                break;

            default:
                printf("Please enter a valid choice.\n");
        }
    }
}

void push() {
    int val;
    if (top == n - 1)
        printf("Overflow: Stack is full.\n");
    else {
        printf("Enter value: ");
        scanf("%d", &val);
        top = top + 1;
        stack[top] = val;
    }
}

void pop() {
    if (top == -1)
        printf("Underflow: Stack is empty.\n");
    else
        top = top - 1;
}

void show() {
    if (top == -1) {
        printf("Stack is Empty.\n");
    } else {
        printf("Stack elements: ");
        for (i = top; i >= 0; i--) {
            printf("%d ", stack[i]);
        }
        printf("\n");
    }
}
```

OUTPUT:



```
C:\Windows\System32\cmd.exe - a
C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (6)>gcc Stackarray.c

C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (6)>a
Enter the number of elements in the stack: 3
Stack Operations using Array
Choose one from below options:
1. Push
2. Pop
3. Show
4. Exit
Enter your choice: 1
Enter value: 10
Choose one from below options:
1. Push
2. Pop
3. Show
4. Exit
Enter your choice: 1
Enter value: 22
Choose one from below options:
1. Push
2. Pop
3. Show
4. Exit
Enter your choice: 3
Stack elements: 22 10
Choose one from below options:
1. Push
2. Pop
3. Show
4. Exit
Enter your choice: 2
Choose one from below options:
1. Push
2. Pop
3. Show
4. Exit
Enter your choice: 3
Stack elements: 10
Choose one from below options:
```

Practical 7

AIM:

Implementation of Stack using Linkedlist in C.

OBJECTIVE:

The objective of implementing a Stack using a linked list in the C language is to create a data structure that follows the Last In, First Out (LIFO) principle.

THEORY:

A Stack is a linear data structure that follows the Last In, First Out (LIFO) principle. Implementing a Stack using a linked list provides a dynamic structure that can efficiently manage elements without the need for a fixed-size array.

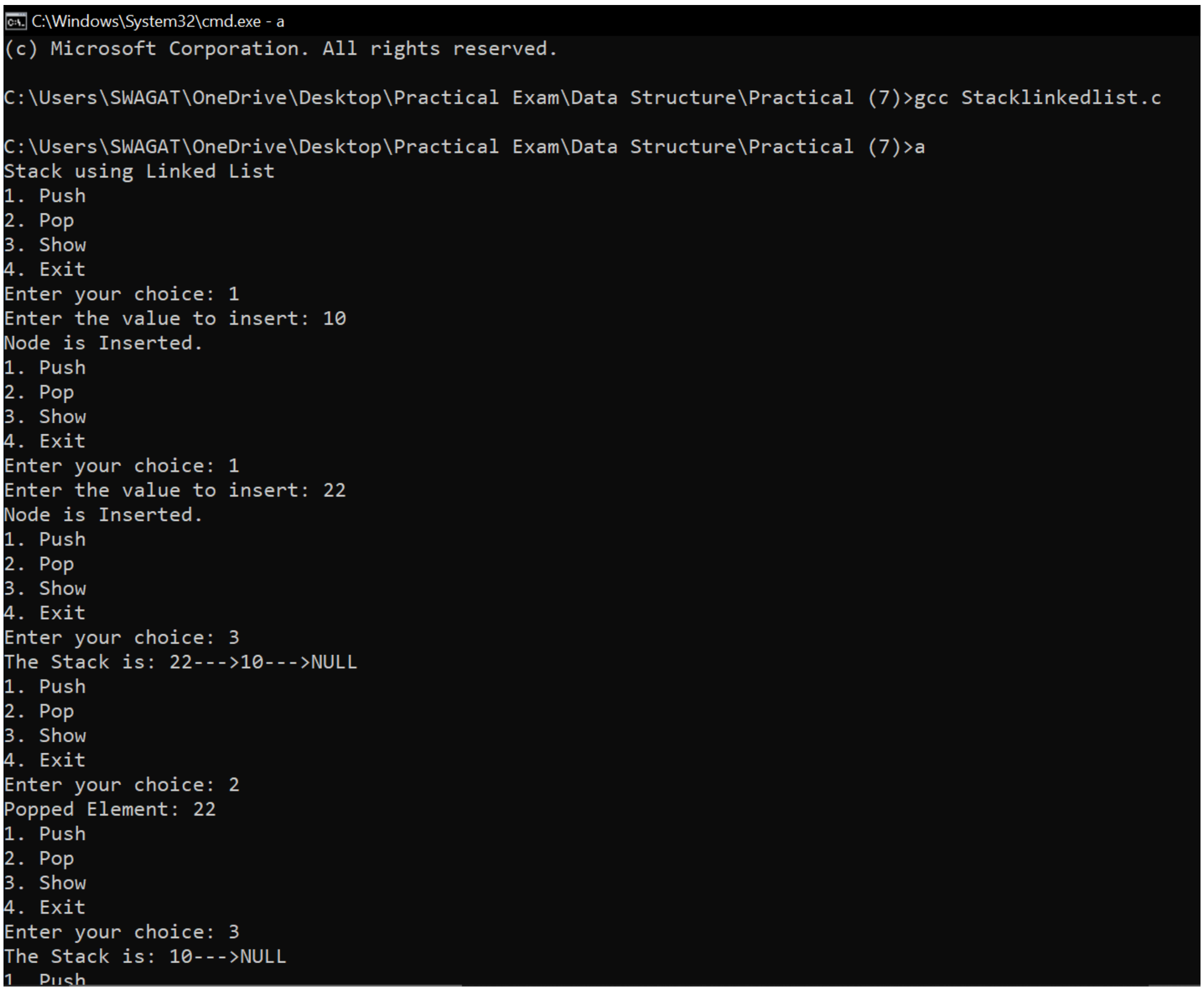
CODE:

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

struct Node {
   int data;
   struct Node *next;
} *top = NULL;

void push(int);
void pop();
void display();

int main() {
   int choice, value;
   printf("Stack using Linked List\n");

   while (1) {
      printf("1. Push \n2. Pop \n3. Show \n4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            printf("Enter the value to insert: ");
            scanf("%d", &value);
```

```
                push(value);
                break;

            case 2:
                pop();
                break;

            case 3:
                display();
                break;

            case 4:
                exit(0);
                break;

            default:
                printf("Invalid Choice\n");
        }
    }
}

void push(int value) {
    struct Node *newNode;
    newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;

    if (top == NULL)
        newNode->next = NULL;
    else
        newNode->next = top;

    top = newNode;
    printf("Node is Inserted.\n");
}

void pop() {
    if (top == NULL)
        printf("Empty Stack\n");
    else {
        struct Node *temp = top;
        printf("Popped Element: %d\n", temp->data);
        top = temp->next;
        free(temp);
    }
```

```
}

void display() {
   if (top == NULL)
      printf("Empty Stack\n");
   else {
      printf("The Stack is: ");
      struct Node *temp = top;
      while (temp->next != NULL) {
         printf("%d--->", temp->data);
         temp = temp->next;
      }
      printf("%d--->NULL\n", temp->data);
   }
}
```

OUTPUT:



```
C:\Windows\System32\cmd.exe - a
(c) Microsoft Corporation. All rights reserved.

C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (7)>gcc Stacklinkedlist.c

C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (7)>a
Stack using Linked List
1. Push
2. Pop
3. Show
4. Exit
Enter your choice: 1
Enter the value to insert: 10
Node is Inserted.
1. Push
2. Pop
3. Show
4. Exit
Enter your choice: 1
Enter the value to insert: 22
Node is Inserted.
1. Push
2. Pop
3. Show
4. Exit
Enter your choice: 3
The Stack is: 22--->10--->NULL
1. Push
2. Pop
3. Show
4. Exit
Enter your choice: 2
Popped Element: 22
1. Push
2. Pop
3. Show
4. Exit
Enter your choice: 3
The Stack is: 10--->NULL
1. Push
```

Practical 8

AIM:

Implementation of Queue using Array in C.

OBJECTIVE:

The objective of implementing a Queue using an array in the C language is to create a data structure that follows the First In, First Out (FIFO) principle.

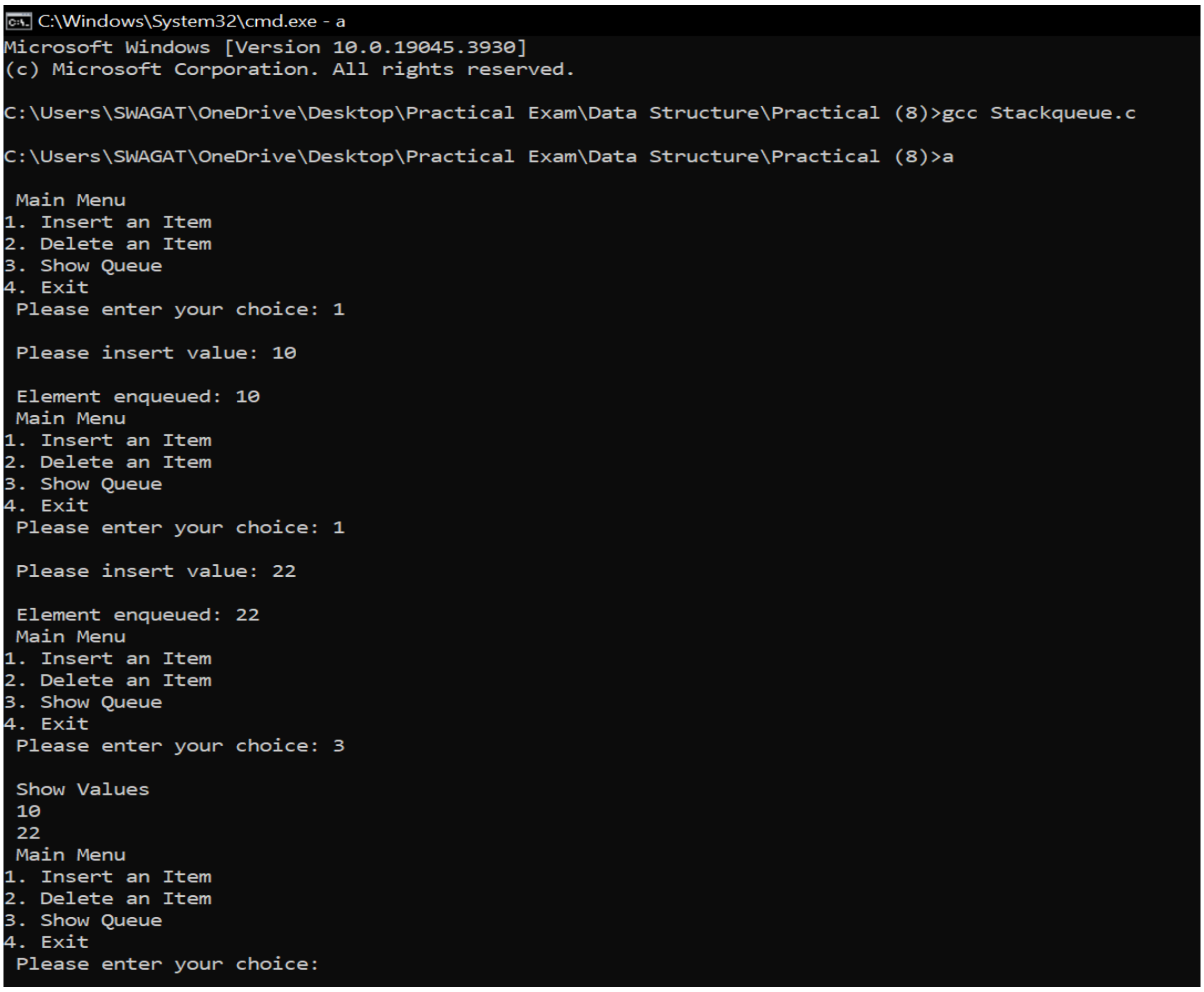
THEORY:

A Queue is a linear data structure that follows the principle of First In, First Out (FIFO). Implementing a Queue using an array provides a simple and efficient way to manage elements in a sequential manner.

CODE:

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

#define MAX_SIZE 100

struct Queue
{
   int items[MAX_SIZE];
   int front, rear;
};

void enqueue(struct Queue *q, int element);
int dequeue(struct Queue *q);
void display(struct Queue *q);

int main()
{
   struct Queue q;
   q.front = -1;
   q.rear = -1;

   int choice, element;

   while (1)
   {
        printf("\n Main Menu");
        printf("\n1. Insert an Item \n2. Delete an Item \n3. Show Queue \n4.
Exit");
        printf("\n Please enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            printf("\n Please insert value: ");
            scanf("%d", &element);
            enqueue(&q, element);
            break;

        case 2:
            element = dequeue(&q);
            if (element != -1)
                printf("\n Removed element: %d", element);
            break;

        case 3:
            display(&q);
            break;

        case 4:
            exit(0);
            break;

        default:
            printf("\n Please enter a valid choice: ");
        }
    }

    return 0;
}

void enqueue(struct Queue *q, int element)
{
    if (q->rear == MAX_SIZE - 1)
    {
        printf("\n Overflow");
        return;
    }
    if (q->front == -1)
        q->front = 0;

    q->rear++;
    q->items[q->rear] = element;

    printf("\n Element enqueued: %d", element);
}

int dequeue(struct Queue *q)
{
    int element;

    if (q->front == -1)
    {
        printf("\n Underflow");
        return -1;
    }

    element = q->items[q->front];
    q->front++;

    if (q->front > q->rear)
    {
        q->front = -1;
        q->rear = -1;
    }

    return element;
}

void display(struct Queue *q)
{
    if (q->front == -1)
    {
        printf("\n Empty Queue");
    }
    else
    {
        printf("\n Show Values");
        for (int i = q->front; i <= q->rear; i++)
        {
            printf("\n %d", q->items[i]);
        }
    }
}
```

OUTPUT:



```
C:\Windows\System32\cmd.exe - a
Microsoft Windows [Version 10.0.19045.3930]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (8)>gcc Stackqueue.c

C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (8)>a

 Main Menu
1. Insert an Item
2. Delete an Item
3. Show Queue
4. Exit
 Please enter your choice: 1

 Please insert value: 10

 Element enqueued: 10
 Main Menu
1. Insert an Item
2. Delete an Item
3. Show Queue
4. Exit
 Please enter your choice: 1

 Please insert value: 22

 Element enqueued: 22
 Main Menu
1. Insert an Item
2. Delete an Item
3. Show Queue
4. Exit
 Please enter your choice: 3

 Show Values
 10
 22
 Main Menu
1. Insert an Item
2. Delete an Item
3. Show Queue
4. Exit
 Please enter your choice:
```

Practical 9

AIM:

Implementation of Queue using Circular Queue in C.

OBJECTIVE:

The objective of implementing a Circular Queue in the C language is to create a data structure that follows the First In, First Out (FIFO) principle with a circular arrangement of elements.

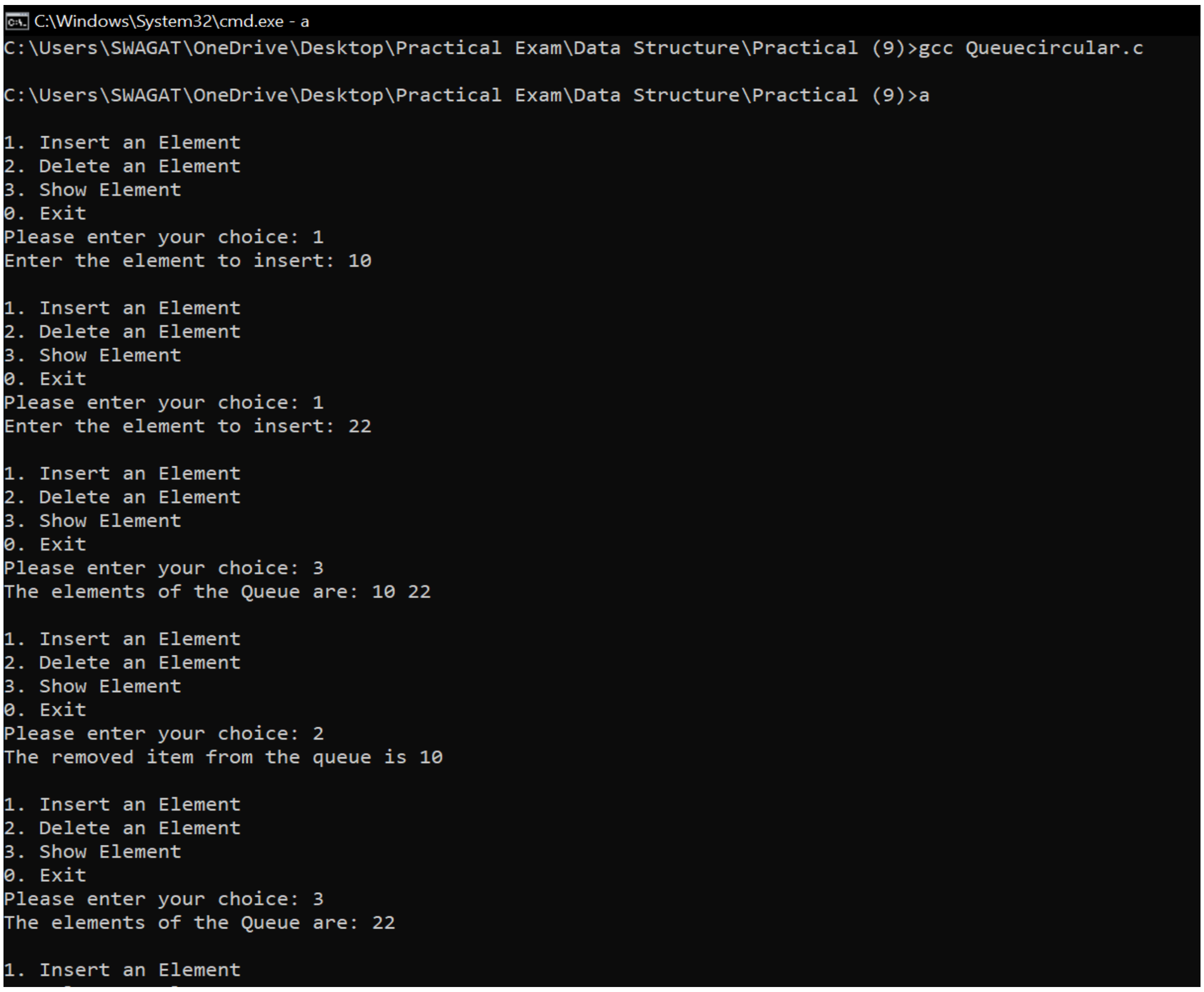
THEORY:

A Circular Queue is a variation of a regular queue where the last element is connected to the first element, forming a circle. This implementation allows for better space utilization and avoids the waste of memory space.

CODE:

```
#include<stdio.h>
#define MAX_SIZE 6

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

void enqueue(int element) {
    if ((rear + 1) % MAX_SIZE == front) {
        printf("The queue is Overflowing.\n");
    } else if (front == -1 && rear == -1) {
        front = 0;
        rear = 0;
        queue[rear] = element;
    } else {
        rear = (rear + 1) % MAX_SIZE;
        queue[rear] = element;
    }
}

void dequeue() {
    if (front == -1 && rear == -1) {
        printf("Queue is Underflow.\n");
    } else if (front == rear) {
        printf("The removed item from the queue is %d\n", queue[front]);
        front = -1;
        rear = -1;
    } else {
        printf("The removed item from the queue is %d\n", queue[front]);
        front = (front + 1) % MAX_SIZE;
    }
}

void display() {
    int i = front;
    if (front == -1 && rear == -1) {
        printf("The Queue is Empty.\n");
    } else {
        printf("The elements of the Queue are: ");
        while (i != rear) {
            printf("%d ", queue[i]);
            i = (i + 1) % MAX_SIZE;
        }
        printf("%d\n", queue[rear]);
    }
}

int main() {
    int choice = 1, x;
    while (choice < 4 && choice != 0) {
        printf("\n1. Insert an Element\n");
        printf("2. Delete an Element\n");
        printf("3. Show Element\n");
        printf("0. Exit\n");
        printf("Please enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the element to insert: ");
                scanf("%d", &x);
                enqueue(x);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 0:
                printf("Exiting the program.\n");
                break;
            default:
                printf("Please enter a valid choice.\n");
        }
    }
    return 0;
}
```

OUTPUT:



```
C:\Windows\System32\cmd.exe - a
C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (9)>gcc Queuecircular.c

C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (9)>a

1. Insert an Element
2. Delete an Element
3. Show Element
0. Exit
Please enter your choice: 1
Enter the element to insert: 10

1. Insert an Element
2. Delete an Element
3. Show Element
0. Exit
Please enter your choice: 1
Enter the element to insert: 22

1. Insert an Element
2. Delete an Element
3. Show Element
0. Exit
Please enter your choice: 3
The elements of the Queue are: 10 22

1. Insert an Element
2. Delete an Element
3. Show Element
0. Exit
Please enter your choice: 2
The removed item from the queue is 10

1. Insert an Element
2. Delete an Element
3. Show Element
0. Exit
Please enter your choice: 3
The elements of the Queue are: 22

1. Insert an Element
```

Practical 10

AIM:

Implementation of Queue using Linkedlist in C.

OBJECTIVE:

The objective of implementing a Queue using a linked list in the C language is to create a data structure that follows the First In, First Out (FIFO) principle.

THEORY:

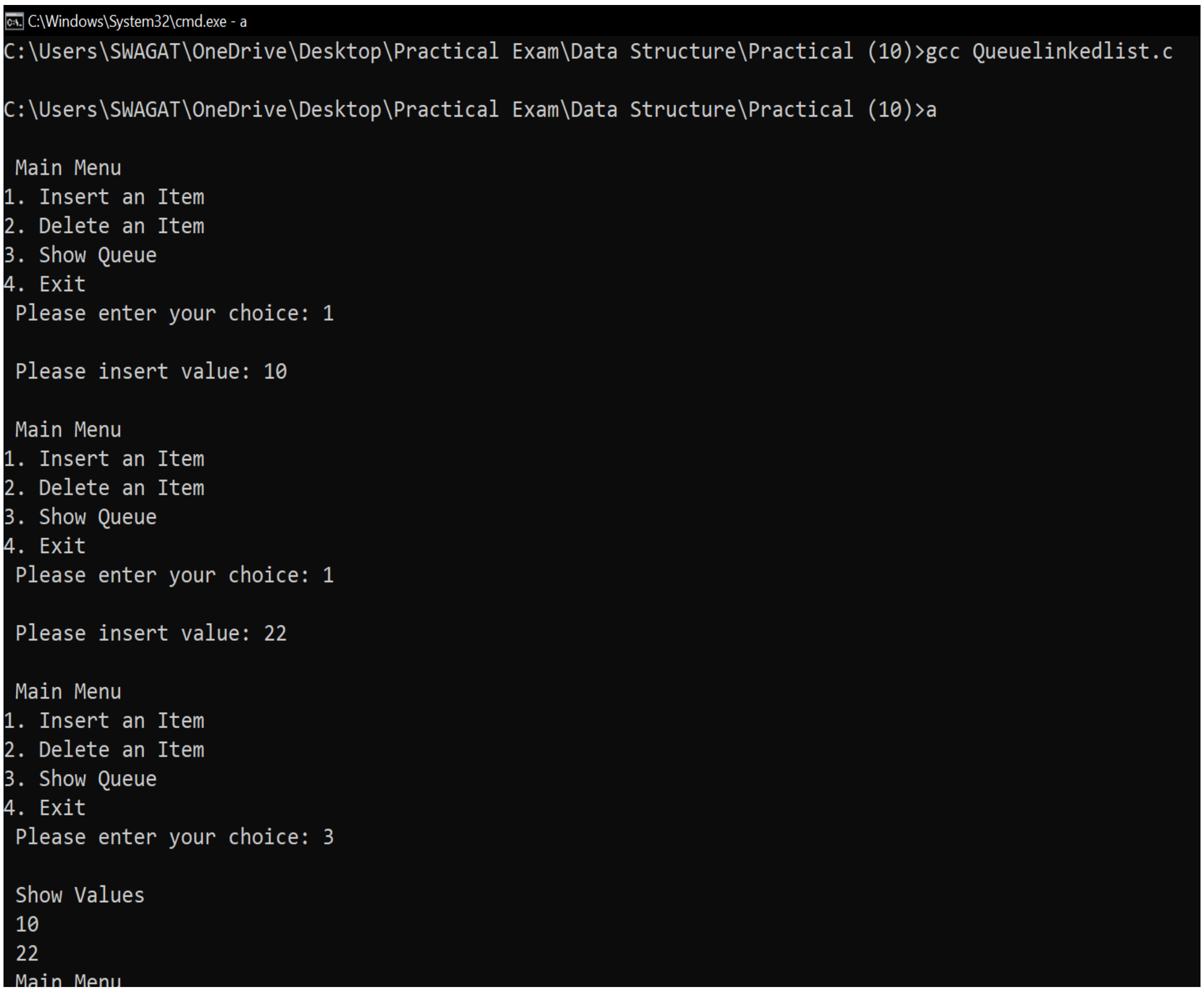
A Queue is a linear data structure that follows the principle of First In, First Out (FIFO). Using a linked list to implement a queue provides a dynamic structure that can efficiently manage elements.

CODE:

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

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

struct node *front = NULL;
struct node *rear = NULL;

void enqueue();
void dequeue();
void display();

int main()
{
   int choice;

   while (choice != 4)
   {
      printf("\n Main Menu");
      printf("\n1. Insert an Item \n2. Delete an Item \n3. Show Queue \n4.
Exit");
```

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

        switch (choice)
        {
            case 1:
                enqueue();
                break;

            case 2:
                dequeue();
                break;

            case 3:
                display();
                break;

            case 4:
                exit(0);
                break;

            default:
                printf("\n Please enter a valid choice: ");
        }
    }

    return 0;
}

void enqueue()
{
    struct node *ptr;
    int element;

    ptr = (struct node *)malloc(sizeof(struct node));

    if (ptr == NULL)
    {
        printf("\n Overflow");
        return;
    }

    printf("\n Please insert value: ");
    scanf("%d", &element);
    ptr->data = element;
    ptr->next = NULL;

    if (front == NULL)
    {
        front = ptr;
        rear = ptr;
    }
    else
    {
        rear->next = ptr;
        rear = ptr;
    }
}

void dequeue()
{
    struct node *ptr;

    if (front == NULL)
    {
        printf("\n Underflow");
        return;
    }

    ptr = front;
    front = front->next;
    free(ptr);
}

void display()
{
    struct node *ptr;
    ptr = front;

    if (front == NULL)
    {
        printf("\n Empty Queue");
    }
    else
    {
        printf("\n Show Values");
        while (ptr != NULL)
        {
            printf("\n %d", ptr->data);
            ptr = ptr->next;
        }
    }
}
```

OUTPUT:



```
C:\Windows\System32\cmd.exe - a
C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (10)>gcc Queuelinkedlist.c

C:\Users\SWAGAT\OneDrive\Desktop\Practical Exam\Data Structure\Practical (10)>a

 Main Menu
1. Insert an Item
2. Delete an Item
3. Show Queue
4. Exit
 Please enter your choice: 1

 Please insert value: 10

 Main Menu
1. Insert an Item
2. Delete an Item
3. Show Queue
4. Exit
 Please enter your choice: 1

 Please insert value: 22

 Main Menu
1. Insert an Item
2. Delete an Item
3. Show Queue
4. Exit
 Please enter your choice: 3

 Show Values
 10
 22
 Main Menu
```