STACK PROGRAM:

AIM:

To create stack using the linked list and doing the stack operation in it.

ALGORITHM:

· Initialize the stack by setting the top pointer to NULL and size to zero.

· Display the menu for push, pop, empty check, full check, and exit operations.

· Read the user's choice from the keyboard.

· If the push option is selected, check whether the stack size has reached the maximum limit.

· Display stack overflow if the stack is full.

· Insert the element at the top of the stack and increment the size if the stack is not full.

· If the pop option is selected, check whether the stack is empty.

· Display stack underflow if the stack is empty.

· Remove the top element from the stack, display value, and decrement the size if the stack is not empty.

· Check whether the stack is empty and display the appropriate message when the empty option is selected.

· Check whether the stack is full and display the appropriate message when the full option is selected.

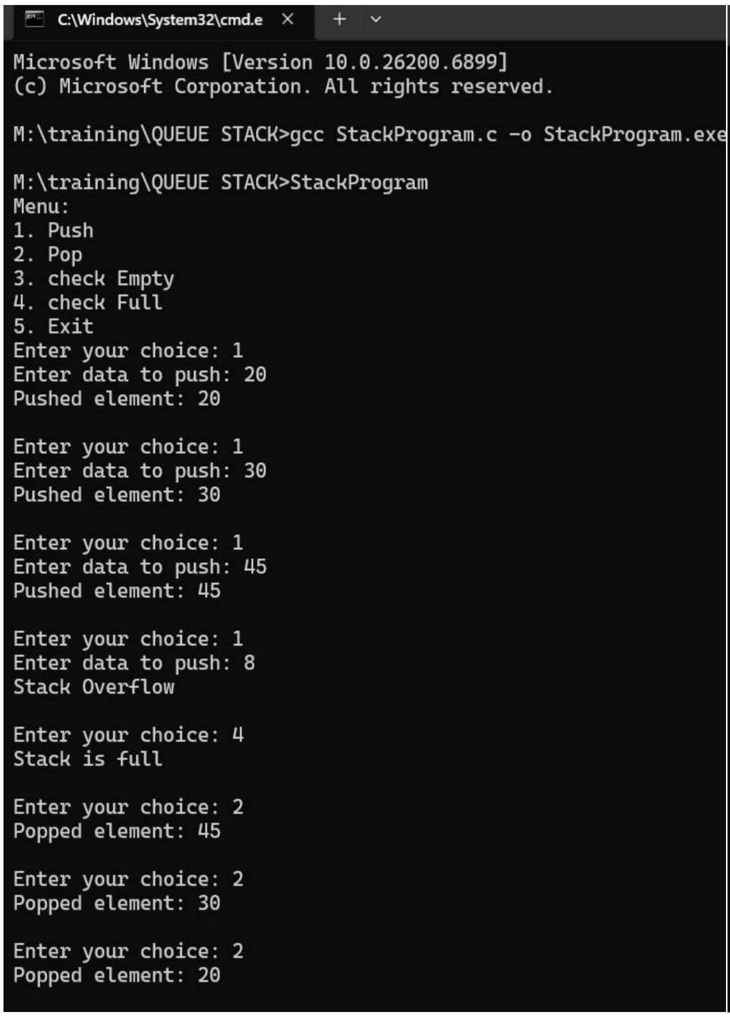
· Repeat the above steps until the exit option is selected.

PROGRAM:

```c
/*
*  Program to implement a stack using linked list
*  Author: MUTHUGANESH S
*  Date  : 20/1/2026
*  File  : StackProgram.c
*  retval: void
*/

// Include necessary headers
#include <stdio.h>
#include <stdlib.h>

// Define boolean values and stack size
#define TRUE 1
#define FALSE 0
#define STACK_SIZE 5

// Define the Node structure
typedef struct Node {
    int Data;
    struct Node* Next;
} Node;

// Define the Stack structure
typedef struct Stack {
    Node* HeadNode;
    int Size;
} Stack;

// Function to create a new node
Node* CreateNode(int Data){

    Node* NewNode = (Node*)malloc(sizeof(Node));

    if(NewNode == NULL){
        printf("Memory allocation failed\n");
        return NULL;
    }

    NewNode->Data = Data;
    NewNode->Next = NULL;
    return NewNode;
}

// Function to push an element onto the stack
void Push(Stack* MyStack, int Data){

    Node* NewNode = CreateNode(Data);

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

    if(MyStack->Size == STACK_SIZE){
        printf("Stack Overflow\n");
        free(NewNode);
        return;
    }

    MyStack->Size++;
    NewNode->Next = MyStack->HeadNode;
    MyStack->HeadNode = NewNode;
    printf("Pushed element: %d\n", Data);
}

// Function to pop an element from the stack
void Pop(Stack* MyStack){

    if(MyStack->HeadNode == NULL){
        printf("Stack Underflow\n");
        return;
    }

    Node* TempNode = MyStack->HeadNode;
    MyStack->HeadNode = MyStack->HeadNode->Next;
    MyStack->Size--;

    printf("Popped element: %d\n", TempNode->Data);

    free(TempNode);
}

// Function to check if the stack is empty
int IsEmpty(Stack* MyStack){
    return MyStack->HeadNode == NULL;
}

// Function to check if the stack is full
int IsFull(Stack* MyStack){
    return MyStack->Size == STACK_SIZE;
}

int main(void){
    Stack MyStack;
    MyStack.HeadNode = NULL;
    MyStack.Size = 0;

    int Value,Data;

    printf("Menu:\n");
    printf("1. Push\n");
    printf("2. Pop\n");
    printf("3. check Empty\n");
    printf("4. check Full\n");
    printf("5. Exit\n");

    printf("Enter your choice: ");
    scanf("%d", &Value);

    while(Value != 5){
        switch (Value)
    {
    case 1:
        printf("Enter data to push: ");
        scanf("%d", &Data);
        Push(&MyStack, Data);
        break;

    case 2:
        Pop(&MyStack);
        break;

    case 3:
        if(IsEmpty(&MyStack)==TRUE){
            printf("Stack is empty\n");
        } else {
            printf("Stack is not empty\n");
        }
```

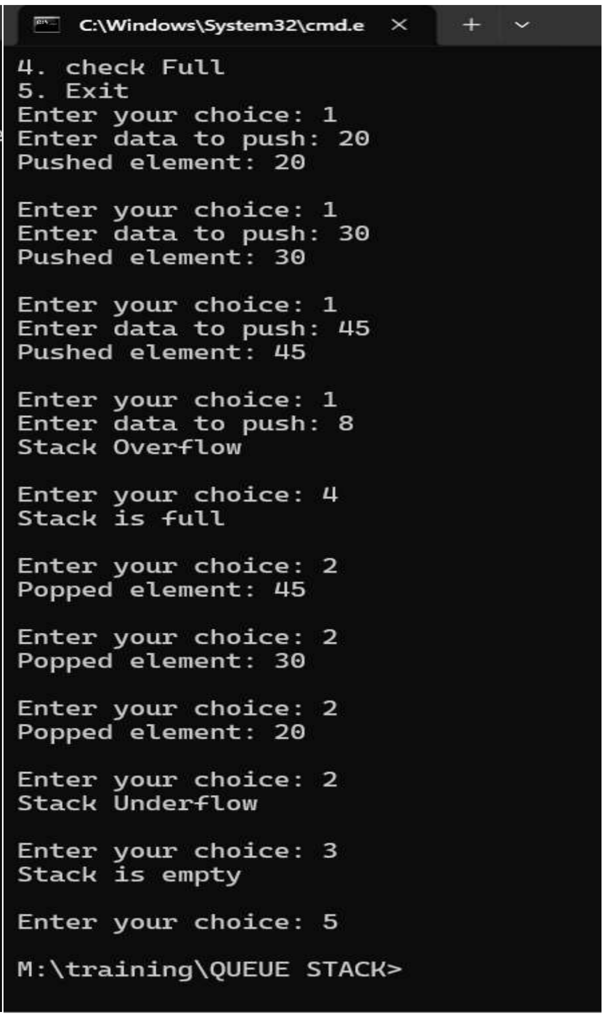
```c
            break;

        case 4:
            if(IsFull(&MyStack)==TRUE){
                printf("Stack is full\n");
            } else {
                printf("Stack is not full\n");
            }
            break;

        case 5:
            exit(0);

        default:
            break;
        }
            printf("\nEnter your choice: ");
            scanf("%d", &Value);
        }

        return 0;
}
```

OUTPUT:



C:\Windows\System32\cmd.e

```
Microsoft Windows [Version 10.0.26200.6899]
(c) Microsoft Corporation. All rights reserved.

M:\training\QUEUE STACK>gcc StackProgram.c -o StackProgram.exe

M:\training\QUEUE STACK>StackProgram
Menu:
1. Push
2. Pop
3. check Empty
4. check Full
5. Exit
Enter your choice: 1
Enter data to push: 20
Pushed element: 20

Enter your choice: 1
Enter data to push: 30
Pushed element: 30

Enter your choice: 1
Enter data to push: 45
Pushed element: 45

Enter your choice: 1
Enter data to push: 8
Stack Overflow

Enter your choice: 4
Stack is full

Enter your choice: 2
Popped element: 45

Enter your choice: 2
Popped element: 30

Enter your choice: 2
Popped element: 20
```



C:\Windows\System32\cmd.e

```
4. check Full
5. Exit
Enter your choice: 1
Enter data to push: 20
Pushed element: 20

Enter your choice: 1
Enter data to push: 30
Pushed element: 30

Enter your choice: 1
Enter data to push: 45
Pushed element: 45

Enter your choice: 1
Enter data to push: 8
Stack Overflow

Enter your choice: 4
Stack is full

Enter your choice: 2
Popped element: 45

Enter your choice: 2
Popped element: 30

Enter your choice: 2
Popped element: 20

Enter your choice: 2
Stack Underflow

Enter your choice: 3
Stack is empty

Enter your choice: 5

M:\training\QUEUE STACK>
```

QUEUE PROGRAM

AIM:

To create a Queue using linked list and perform Queue functionalities.

ALGORITHM:

· Initialize the queue by setting the front and rear pointers to NULL and the size to zero.

· Display the menu for enqueue, dequeue, and exit operations.

· Read the user's choice from the keyboard.

· If the enqueue option is selected, check whether the queue size has reached the maximum limit.

· Display queue overflow if the queue is full.

· Create a new node and insert the element at rear of the queue and increment the size if the queue is not full.

· If the dequeue option is selected, check whether the queue is empty.

· Display queue underflow if the queue is empty.

· Remove the elementfrom the front of queue,display value, and decrement size if the queue is not empty.

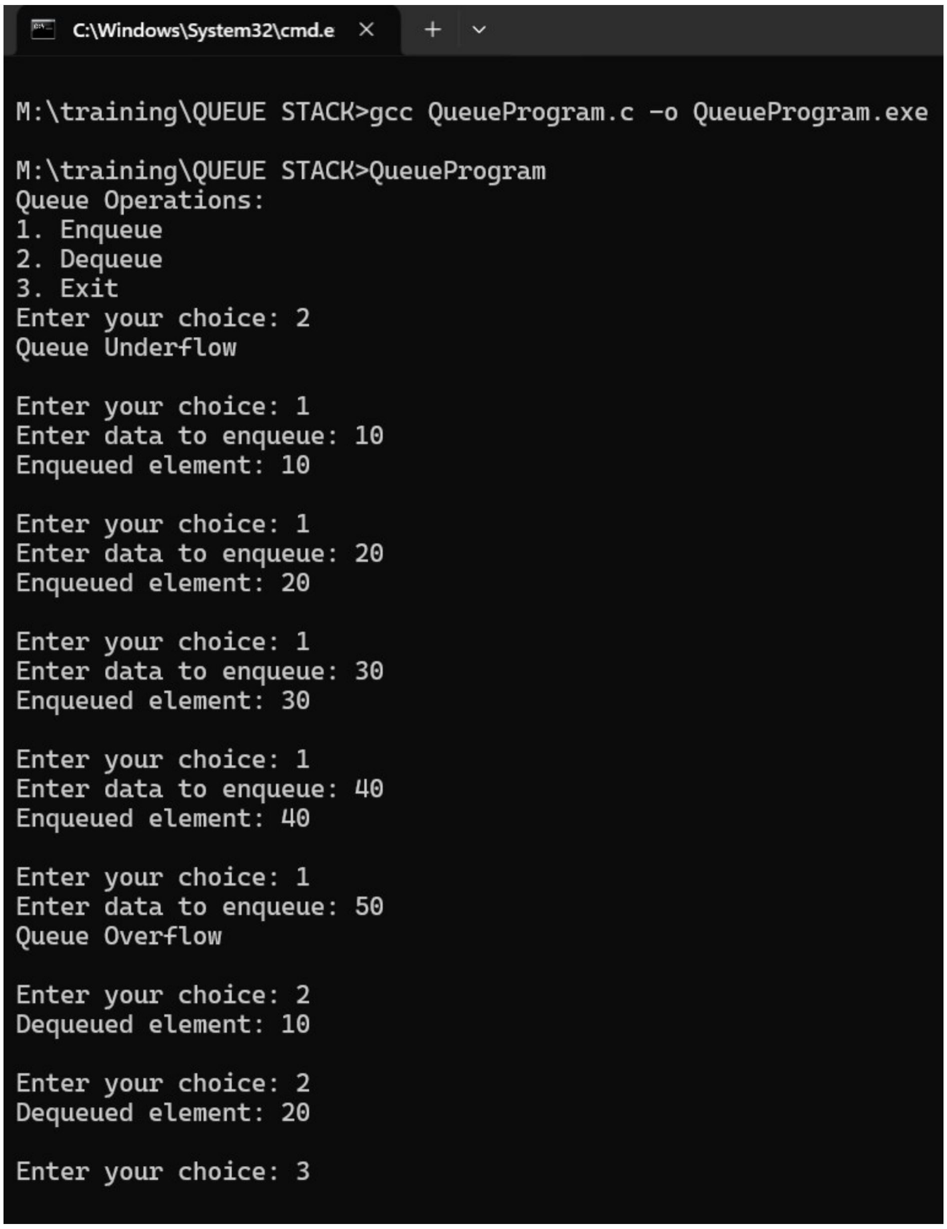
· Repeat the above steps based on the user's choice until the exit option is selected.

PROGRAM:

```c
/*
* Program to implement a Queue using linked list
* Author: MUTHUGANESH S
* Date  : 20/1/2026
* File  : QueueProgram.c
* retval: void
*/

// Include necessary headers
#include <stdio.h>
#include <stdlib.h>

// Define queue size
#define MAX_QUEUE_SIZE 4

// Define the Node structure
typedef struct Node {
    int Data;
    struct Node* Next;
} Node;

// Define the Queue structure
```

```c
typedef struct Queue {
    Node* Front;
    Node* Rear;
    int Size;
} Queue;

// Function to create a new node
Node* CreateNode(int Data){
    Node* NewNode = (Node*)malloc(sizeof(Node));
    if(NewNode == NULL){
        printf("Memory allocation failed\n");
        return NULL;
    }
    NewNode->Data = Data;
    NewNode->Next = NULL;
    return NewNode;
}

// Function to enqueue an element into the queue
void Enqueue(Queue* MyQueue, int Data){
    if(MyQueue->Size == MAX_QUEUE_SIZE){
        printf("Queue Overflow\n");
        return;
    }

    Node* NewNode = CreateNode(Data);
    if(NewNode == NULL){
        printf("Memory allocation failed\n");
        return;
    }

    if(MyQueue->Rear == NULL){
        MyQueue->Front = MyQueue->Rear = NewNode;
    }
    else {
        MyQueue->Rear->Next = NewNode;
        MyQueue->Rear = NewNode;
    }

    MyQueue->Size++;
    printf("Enqueued element: %d\n", Data);
}

// Function to dequeue an element from the queue
void Dequeue(Queue* MyQueue){

    if(MyQueue->Size == 0){
        printf("Queue Underflow\n");
        return;
    }

    Node* TempNode = MyQueue->Front;
    MyQueue->Front = MyQueue->Front->Next;
    if(MyQueue->Front == NULL){
        MyQueue->Rear = NULL;
    }

    MyQueue->Size--;
    printf("Dequeued element: %d\n", TempNode->Data);
    free(TempNode);
}

int main(void){

    // Initialize the queue
    Queue MyQueue;
    MyQueue.Front = NULL;
    MyQueue.Rear = NULL;
    MyQueue.Size = 0;

    printf("Queue Operations:\n");
    printf("1. Enqueue\n");
    printf("2. Dequeue\n");
    printf("3. Exit\n");

    int Value, Data;
    printf("Enter your choice: ");
    scanf("%d", &Value);

    while(Value != 3){
        switch(Value){

            case 1:

                printf("Enter data to enqueue: ");
                scanf("%d", &Data);
                Enqueue(&MyQueue, Data);
                break;

            case 2:

                Dequeue(&MyQueue);
                break;

            default:

                printf("Invalid choice\n");
                break;

        }

        printf("\nEnter your choice: ");
        scanf("%d", &Value);
    }

    return 0;
}
```

OUTPUT:



```
M:\training\QUEUE STACK>gcc QueueProgram.c -o QueueProgram.exe

M:\training\QUEUE STACK>QueueProgram
Queue Operations:
1. Enqueue
2. Dequeue
3. Exit
Enter your choice: 2
Queue Underflow

Enter your choice: 1
Enter data to enqueue: 10
Enqueued element: 10

Enter your choice: 1
Enter data to enqueue: 20
Enqueued element: 20

Enter your choice: 1
Enter data to enqueue: 30
Enqueued element: 30

Enter your choice: 1
Enter data to enqueue: 40
Enqueued element: 40

Enter your choice: 1
Enter data to enqueue: 50
Queue Overflow

Enter your choice: 2
Dequeued element: 10

Enter your choice: 2
Dequeued element: 20

Enter your choice: 3
```