AIM: LINKED LIST IMPLEMENTATION OF STACK/QUEUE IN REAL LIFE APPLICATION

DESCRIPTION:-

Linked List Implementation of Stack/Queue in Real-Life Applications:

1. **Stack (LIFO)**:
    - **Backtracking**: Used in navigation systems (undo/redo operations, web browsers' back/forward buttons) where the most recent action is reverted first.
    - **Expression Evaluation**: Stack-based algorithms like parsing arithmetic expressions (postfix evaluation) rely on linked list stacks for efficient memory usage.
2. In a linked list stack, each node holds an element and a pointer to the next node, enabling dynamic memory allocation without the size limitation of arrays.
3. **Queue (FIFO)**:
    - **Customer Service Systems**: Queues are used in help desks, customer support centers, or scheduling systems where the first customer request in line is served first.
    - **Task Scheduling**: In operating systems or printer management, tasks are processed in the order they are added, making queues essential.
4. Linked list queues provide flexible memory allocation, allowing the queue to grow dynamically as elements are enqueued and dequeued without the fixed size constraint of arrays.

CODE:-

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

typedef struct StackNode {
    char customer_name[50];
    struct StackNode* next;
} StackNode;

typedef struct Stack {
    StackNode* top;
} Stack;

// Function to create a new stack node
```

```
StackNode* createStackNode(const char* customer_name) {
    StackNode* newNode = (StackNode*)malloc(sizeof(StackNode));
    strcpy(newNode->customer_name, customer_name);
    newNode->next = NULL;
    return newNode;
}

// Initialize the stack
Stack* createStack() {
    Stack* stack = (Stack*)malloc(sizeof(Stack));
    stack->top = NULL;
    return stack;
}

// Push function to add a customer to the stack
void push(Stack* stack, const char* customer_name) {
    StackNode* newNode = createStackNode(customer_name);
    newNode->next = stack->top;  // Link new node to current top
    stack->top = newNode;  // Update top to new node
    printf("Customer added to stack: %s\n", customer_name);
}

// Pop function to serve the last customer
void pop(Stack* stack) {
    if (!stack->top) {
        printf("No customers in stack.\n");
        return;
    }
    StackNode* temp = stack->top;
    printf("Served from stack: %s\n", temp->customer_name);
    stack->top = stack->top->next;  // Move top pointer to next node
    free(temp);  // Free the served customer node
}

// Check if the stack is empty
int isStackEmpty(Stack* stack) {
    return stack->top == NULL;
}

// Peek function to see the last customer
void peek(Stack* stack) {
    if (stack->top) {
        printf("Last customer to be served: %s\n", stack->top->customer_name);
    } else {
```

```
        printf("No customers in stack.\n");
    }
}

// Free the stack
void freeStack(Stack* stack) {
    while (!isStackEmpty(stack)) {
        pop(stack);
    }
    free(stack);
}

// Example usage
int main() {
    Stack* serviceStack = createStack();
    push(serviceStack, "Customer A");
    push(serviceStack, "Customer B");
    push(serviceStack, "Customer C");

    peek(serviceStack);  // Outputs: Customer C
    pop(serviceStack);  // Outputs: Served from stack: Customer C
    peek(serviceStack);  // Outputs: Customer B

    // Clean up
    freeStack(serviceStack);
    return 0;
}
```

OUTPUT:-

```
/tmp/NTM2K2Nh1b.o
Customer added to stack: Customer A
Customer added to stack: Customer B
Customer added to stack: Customer C
Last customer to be served: Customer C
Served from stack: Customer C
Last customer to be served: Customer B
Served from stack: Customer B
Served from stack: Customer A


=== Code Execution Successful ===
```