22_HEMANT_15.c

```c
/*
Roll no : 22
Batch: A
Author name: Hemant Gupta
Date: 23/08/2024
Description: Linked list implementation of stack
*/

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

// Node structure for linked list
struct Node {
    int data;             // Data stored in the node
    struct Node* next;    // Pointer to the next node
};

// Stack structure
struct Stack {
    struct Node* top;     // Pointer to the top node of the stack
};

// Function to create a new stack
struct Stack* createStack() {
    struct Stack* stack = malloc(sizeof(struct Stack)); // Allocate memory for the stack
    stack->top = NULL; // Initialize top to NULL (stack is empty)
    return stack;      // Return the newly created stack
}

// Function to push data onto the stack
void push(struct Stack* stack, int data) {
    struct Node* newNode = malloc(sizeof(struct Node)); // Allocate memory for a new node
    newNode->data = data;    // Set the node's data
    newNode->next = stack->top; // Link the new node to the previous top
    stack->top = newNode;    // Update top to the new node
}

// Function to pop data from the stack
int pop(struct Stack* stack) {
    if (stack->top == NULL) { // Check if the stack is empty
        printf("Stack underflow!\n");
        exit(EXIT_FAILURE); // Exit if the stack is empty
    }
    struct Node* temp = stack->top; // Temporary node to hold the top
    int poppedValue = temp->data;    // Get the data from the top node
    stack->top = stack->top->next;   // Move top to the next node
    free(temp); // Free the memory of the popped node
    return poppedValue; // Return the popped value
}

// Function to display the elements of the stack
```

```c
void display(struct Stack* stack) {
    struct Node* current = stack->top; // Start from the top of the stack
    if (current == NULL) {
        printf("Stack is empty!\n");
        return;
    }
    printf("Stack elements: ");
    while (current != NULL) {
        printf("%d ", current->data); // Print the data of each node
        current = current->next; // Move to the next node
    }
    printf("\n");
}

// Function to free the stack memory
void freeStack(struct Stack* stack) {
    while (stack->top != NULL) {
        pop(stack); // Pop all elements
    }
    free(stack); // Free the stack structure itself
}

// Main function to demonstrate stack operations
int main() {
    struct Stack* stack = createStack(); // Create a new stack
    int choice, value;

    do {
        printf("\nMenu:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: // Push operation
                printf("Enter a value to push: ");
                scanf("%d", &value);
                push(stack, value); // Push the user input onto the stack
                break;
            case 2: // Pop operation
                printf("Popped element: %d\n", pop(stack)); // Pop and display the top
element
                break;
            case 3: // Display operation
                display(stack); // Display the current elements in the stack
                break;
            case 4: // Exit
                freeStack(stack); // Clean up memory
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}

```