Linear Queue Operations using a stack

Code:

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

#define MAX 100 // Maximum size of the stack

// Stack structure
typedef struct {
    int data[MAX];
    int top;
} Stack;

// Queue using two stacks
typedef struct {
    Stack s1;
    Stack s2;
} Queue;

// Function prototypes
void initializeStack(Stack *s);
int isStackEmpty(Stack *s);
int isStackFull(Stack *s);
void push(Stack *s, int value);
int pop(Stack *s);

void initializeQueue(Queue *q);
void enqueue(Queue *q, int value);
int dequeue(Queue *q);
void displayQueue(Queue *q);

// Main function
int main() {
    Queue q;
    initializeQueue(&q);

    int choice, value;

    while (1) {
        printf("\nQueue Operations using Stacks:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display Queue\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to enqueue: ");
                scanf("%d", &value);
                enqueue(&q, value);
                break;
            case 2:
                value = dequeue(&q);
                if (value != -1) {
                    printf("Dequeued value: %d\n", value);
                }
                break;
            case 3:
                displayQueue(&q);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
```

```
    return 0;
}

// Initialize stack
void initializeStack(Stack *s) {
    s->top = -1;
}

// Check if the stack is empty
int isStackEmpty(Stack *s) {
    return s->top == -1;
}

// Check if the stack is full
int isStackFull(Stack *s) {
    return s->top == MAX - 1;
}

// Push an element onto the stack
void push(Stack *s, int value) {
    if (isStackFull(s)) {
        printf("Stack overflow! Cannot push.\n");
        return;
    }
    s->data[++(s->top)] = value;
}

// Pop an element from the stack
int pop(Stack *s) {
    if (isStackEmpty(s)) {
        printf("Stack underflow! Cannot pop.\n");
        return -1;
    }
    return s->data[(s->top)--];
}

// Initialize queue
void initializeQueue(Queue *q) {
    initializeStack(&(q->s1));
    initializeStack(&(q->s2));
}

// Enqueue operation
void enqueue(Queue *q, int value) {
    push(&(q->s1), value);
    printf("Enqueued %d into the queue.\n", value);
}

// Dequeue operation
int dequeue(Queue *q) {
    if (isStackEmpty(&(q->s1)) && isStackEmpty(&(q->s2))) {
        printf("Queue is empty! Cannot dequeue.\n");
        return -1;
    }
    // Transfer elements from s1 to s2 if s2 is empty
    if (isStackEmpty(&(q->s2))) {
        while (!isStackEmpty(&(q->s1))) {
            push(&(q->s2), pop(&(q->s1)));
        }
    }
    return pop(&(q->s2));
}

// Display queue elements
void displayQueue(Queue *q) {
    if (isStackEmpty(&(q->s1)) && isStackEmpty(&(q->s2))) {
        printf("Queue is empty!\n");
        return;
    }

    Stack temp;
    initializeStack(&temp);
```

```
    // Transfer elements from s2 to temp to maintain order
    while (!isStackEmpty(&(q->s2))) {
        push(&temp, pop(&(q->s2)));
    }

    // Display elements from temp
    printf("Queue elements: ");
    while (!isStackEmpty(&temp)) {
        int value = pop(&temp);
        printf("%d ", value);
        push(&(q->s2), value);
    }

    // Transfer elements from s1 to temp to maintain order
    while (!isStackEmpty(&(q->s1))) {
        push(&temp, pop(&(q->s1)));
    }

    // Display elements from temp and restore them to s1
    while (!isStackEmpty(&temp)) {
        int value = pop(&temp);
        printf("%d ", value);
        push(&(q->s1), value);
    }

    printf("\n");
}}
```

OUTPUT:

```
Queue Operations using Stacks:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter the value to enqueue: 1
Enqueued 1 into the queue.

Queue Operations using Stacks:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter the value to enqueue: 2
Enqueued 2 into the queue.

Queue Operations using Stacks:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 3
Queue elements: 1 2

Queue Operations using Stacks:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 2
Dequeued value: 1

Queue Operations using Stacks:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice:
```