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

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

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

int isEmpty(struct Node* top) {
    return top == NULL;
}

void push(struct Node** top, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *top;
    *top = newNode;
    printf("%d pushed onto the stack\n", data);
}
```

```c
int pop(struct Node** top) {
    if (isEmpty(*top)) {
        printf("Stack underflow\n");
        return -1;
    }
    struct Node* temp = *top;
    int poppedData = temp->data;
    *top = (*top)->next;
    free(temp);
    return poppedData;
}

int peek(struct Node* top) {
    if (isEmpty(top)) {
        printf("Stack is empty\n");
        return -1;
    }
    return top->data;
}


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

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


int main() {
    struct Node* stackTop = NULL;

    int choice, value;
    do {
        printf("\n--- Stack Operations ---\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Peek\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(&stackTop, value);
                break;
            case 2:
                value = pop(&stackTop);
                if (value != -1)
                    printf("Popped value: %d\n", value);
                break;
            case 3:
                value = peek(stackTop);
```

```
        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(&stackTop, value);
                break;
            case 2:
                value = pop(&stackTop);
                if (value != -1)
                    printf("Popped value: %d\n", value);
                break;
            case 3:
                value = peek(stackTop);
                if (value != -1)
                    printf("Top element: %d\n", value);
                break;
            case 4:
                display(stackTop);
                break;
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 5);

    return 0;
}
```

```
--- Stack Operations ---
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push: 22
22 pushed onto the stack

--- Stack Operations ---
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push: 30
30 pushed onto the stack

--- Stack Operations ---
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 3
Top element: 30

--- Stack Operations ---
1. Push
2. Pop
3. Peek
4. Display
```

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};


struct Queue {
    struct Node *front, *rear;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

struct Queue* createQueue() {
    struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue));
    q->front = q->rear = NULL;
    return q;
}

int isEmpty(struct Queue* q) {
    return q->front == NULL;
```

```
    return q->front == NULL;
}

void enqueue(struct Queue* q, int data) {
    struct Node* newNode = createNode(data);
    if (q->rear == NULL) { // If queue is empty
        q->front = q->rear = newNode;
        printf("%d enqueued into the queue\n", data);
        return;
    }
    q->rear->next = newNode;
    q->rear = newNode;
    printf("%d enqueued into the queue\n", data);
}

int dequeue(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue underflow\n");
        return -1;
    }
    struct Node* temp = q->front;
    int dequeuedData = temp->data;
    q->front = q->front->next;

    if (q->front == NULL)
        q->rear = NULL;

    free(temp);
    return dequeuedData;
}
```

```c
int peek(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    }
    return q->front->data;
}

void display(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return;
    }
    struct Node* temp = q->front;
    printf("Queue elements: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}


int main() {
    struct Queue* queue = createQueue();

    int choice, value;
    do {
        printf("\n--- Queue Operations ---\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
```

```c
        switch (choice) {
            case 1:
                printf("Enter value to enqueue: ");
                scanf("%d", &value);
                enqueue(queue, value);
                break;
            case 2:
                value = dequeue(queue);
                if (value != -1)
                    printf("Dequeued value: %d\n", value);
                break;
            case 3:
                value = peek(queue);
                if (value != -1)
                    printf("Front element: %d\n", value);
                break;
            case 4:
                display(queue);
                break;
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 5);

    return 0;
}
```

```
--- Queue Operations ---
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 12
12 enqueued into the queue

--- Queue Operations ---
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 13
13 enqueued into the queue

--- Queue Operations ---
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 3
Front element: 12

--- Queue Operations ---
1. Enqueue
2. Dequeue
3. Peek
4. Display
```

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

// Define the structure of a node
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a node at the end of the list
struct Node* insertEnd(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    if (head == NULL) {
        return newNode;
    }
    struct Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    return head;
}
```

```c
void printList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// Function to sort the linked list (Bubble Sort)
struct Node* sortList(struct Node* head) {
    if (head == NULL) return head;
    struct Node* current = head;
    struct Node* index = NULL;
    int temp;
    while (current != NULL) {
        index = current->next;
        while (index != NULL) {
            if (current->data > index->data) {
                // Swap data
                temp = current->data;
                current->data = index->data;
                index->data = temp;
            }
            index = index->next;
        }
        current = current->next;
```

```c
struct Node* reverseList(struct Node* head) {
    struct Node* prev = NULL;
    struct Node* current = head;
    struct Node* next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}

// Function to concatenate two linked lists
struct Node* concatenate(struct Node* head1, struct Node* head2) {
    if (head1 == NULL) return head2;
    if (head2 == NULL) return head1;

    struct Node* temp = head1;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = head2;
    return head1;
}

int main() {
    struct Node *list1 = NULL, *list2 = NULL;

    // Creating first linked list
    list1 = insertEnd(list1, 5);
```

```c
    list1 = insertEnd(list1, 9);
    list1 = insertEnd(list1, 3);

    // Creating second linked list
    list2 = insertEnd(list2, 8);
    list2 = insertEnd(list2, 2);
    list2 = insertEnd(list2, 7);

    printf("List 1: ");
    printList(list1);

    printf("List 2: ");
    printList(list2);

    // Sorting list1
    list1 = sortList(list1);
    printf("\nSorted List 1: ");
    printList(list1);

    // Reversing list1
    list1 = reverseList(list1);
    printf("Reversed List 1: ");
    printList(list1);

    // Concatenating list1 and list2
    struct Node* concatenatedList = concatenate(list1, list2);
    printf("\nConcatenated List: ");
    printList(concatenatedList);

    return 0;
```

```
List 1: 5 -> 1 -> 9 -> 3 -> NULL
List 2: 8 -> 2 -> 7 -> NULL

Sorted List 1: 1 -> 3 -> 5 -> 9 -> NULL
Reversed List 1: 9 -> 5 -> 3 -> 1 -> NULL

Concatenated List: 9 -> 5 -> 3 -> 1 -> 8 -> 2 -> 7 -> NULL

Process returned 0 (0x0)   execution time : 0.054 s
Press any key to continue.
```