1.main.c

```
#include <stdio.h>
#include "queue.h"
int main() {
   queue q;
   int choice, value;
   qinit(&q);
   do {
      printf("\nQueue Operations Menu:\n");
      printf("1. Enqueue\n");
      printf("2. Dequeue\n");
      printf("3. Peek\n");
      printf("4. Display\n");
      printf("5. Check if Queue is Full\n");
      printf("6. Check if Queue is Empty\n");
      printf("7. 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:
            value = peek(&q);
            if (value != -1) {
                printf("Front value: %d\n", value);
            }
            break;
        case 4:
            display(&q);
            break;
        case 5:
            if (isqfull()) {
                printf("The queue is full.\n");
            } else {
                printf("The queue is not full.\n");
            }
            break;
        case 6:
            if (qempty(&q)) {
                printf("The queue is empty.\n");
            } else {
                printf("The queue is not empty.\n");
            }
            break;
        case 7:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Please enter a number between 1 and 7.\n");
            break;
    }
} while (choice != 7);
```

```
    return 0;
}
```

2.header.h

```
// Define the node structure for the list
typedef struct node {
    int data;
    struct node *next;
} node;

// Define the queue structure
typedef struct {
    node *front;
    node *rear;
} queue;

// Function prototypes
void qinit(queue *q);
int qempty(queue *q);
int isqfull();
void enqueue(queue *q, int value);
int dequeue(queue *q);
int peek(queue *q);
void display(queue *q);
```

3.logic.c

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

// Initialize the queue
void qinit(queue *q) {
   q->front = NULL;
   q->rear = NULL;
}

// Check if the queue is empty
int qempty(queue *q) {
   return (q->front == NULL);
}

int isqfull() {
   node *temp = (node *)malloc(sizeof(node));
   if (temp == NULL) {
      return 1; // Queue is full (memory allocation failed)
   }
   free(temp); // Free the memory if allocation was successful
   return 0;   // Queue is not full
}

// Add an element to the queue
void enqueue(queue *q, int value) {
   if (isqfull()) {
      printf("Memory allocation failed, cannot enqueue data.\n");
      return;
   }
```

```
    node *newNode = (node *)malloc(sizeof(node));
    if (newNode == NULL) {
        printf("Memory allocation failed, cannot enqueue data.\n");
        return;
    }
    newNode->data = value;
    newNode->next = NULL;

    if (q->rear == NULL) { // Queue is empty
        q->front = newNode;
        q->rear = newNode;
    } else { // Add the new node to the end of the queue
        q->rear->next = newNode;
        q->rear = newNode;
    }
}

// Remove and return the front element from the queue
int dequeue(queue *q) {
    if (qempty(q)) {
        printf("Queue is empty, cannot dequeue data.\n");
        return -1; // Indicate an error
    }

    int value = q->front->data;
    node *temp = q->front;
    q->front = q->front->next;

    if (q->front == NULL) { // If the queue is now empty
        q->rear = NULL;
```

```
    }

    free(temp);
    return value;
}

// Peek at the front element without removing it
int peek(queue *q) {
    if (qempty(q)) {
        printf("Queue is empty, nothing to peek.\n");
        return -1; // Indicate an error
    }
    return q->front->data;
}

// Display the elements of the queue
void display(queue *q) {
    if (qempty(q)) {
        printf("Queue is empty.\n");
        return;
    }

    node *current = q->front;
    printf("Queue elements: ");
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}
```

Output:

```
tanis@Tanishq MINGW64 /d/COEP/DSA/Serious/LabWork-StackAndQueue/Queue
$ gcc -Wall testqueue.c queue.c

tanis@Tanishq MINGW64 /d/COEP/DSA/Serious/LabWork-StackAndQueue/Queue
$ ./a

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Queue is Full
6. Check if Queue is Empty
7. Exit
Enter your choice: 6
The queue is empty.

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Queue is Full
6. Check if Queue is Empty
7. Exit
Enter your choice: 1
Enter the value to enqueue: 10
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Queue is Full
6. Check if Queue is Empty
7. Exit
Enter your choice: 1
Enter the value to enqueue: 20

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Queue is Full
6. Check if Queue is Empty
7. Exit
Enter your choice: 1
Enter the value to enqueue: 30
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Queue is Full
6. Check if Queue is Empty
7. Exit
Enter your choice: 1
Enter the value to enqueue: 40

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Queue is Full
6. Check if Queue is Empty
7. Exit
Enter your choice: 1
Enter the value to enqueue: 50

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Queue is Full
6. Check if Queue is Empty
7. Exit
Enter your choice: 4
Queue elements: 10 20 30 40 50
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Queue is Full
6. Check if Queue is Empty
7. Exit
Enter your choice: 3
Front value: 10

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Queue is Full
6. Check if Queue is Empty
7. Exit
Enter your choice: 2
Dequeued value: 10

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Queue is Full
6. Check if Queue is Empty
7. Exit
Enter your choice: 4
Queue elements: 20 30 40 50
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Queue is Full
6. Check if Queue is Empty
7. Exit
Enter your choice: 3
Front value: 20
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Queue is Full
6. Check if Queue is Empty
7. Exit
Enter your choice: 5
The queue is not full.

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Queue is Full
6. Check if Queue is Empty
7. Exit
Enter your choice: 7
Exiting program.
```