Folder SEM 3\Exp6

2 printable files

(file list disabled)

SEM 3\Exp6\Queue_ARR.c

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

#define MAX 100 // Maximum size of the queue

// Queue structure using arrays
struct QueueArray
{
    int front, rear;
    int arr[MAX];
};

// Function to create a queue
struct QueueArray *createQueue()
{
    struct QueueArray *queue = (struct QueueArray *)malloc(sizeof(struct
QueueArray));
    queue->front = -1;
    queue->rear = -1;
    return queue;
}

// Check if the queue is full
int isFull(struct QueueArray *queue)
{
    return queue->rear == MAX - 1;
}

// Check if the queue is empty
int isEmpty(struct QueueArray *queue)
{
    return queue->front == -1 || queue->front > queue->rear;
}

// Enqueue an element into the queue
void enqueue(struct QueueArray *queue, int value)
{
    if (isFull(queue))
    {
        printf("Queue overflow!\n");
        return;
    }
    if (isEmpty(queue))
    {
        queue->front = 0; // Initialize front if queue was empty
    }
    queue->arr[++queue->rear] = value;
    printf("%d enqueued to queue\n", value);
}

// Dequeue an element from the queue
int dequeue(struct QueueArray *queue)
{
    if (isEmpty(queue))
    {
        printf("Queue underflow!\n");
        return -1; // Return -1 for underflow
    }
    return queue->arr[queue->front++];
}

// Peek at the front element of the queue
int peek(struct QueueArray *queue)
{
    if (isEmpty(queue))
    {
        printf("Queue is empty!\n");
        return -1; // Return -1 if empty
    }
    return queue->arr[queue->front];
}

// Display the queue
void display(struct QueueArray *queue)
{
    if (isEmpty(queue))
    {
        printf("Queue is empty!\n");
        return;
    }
    printf("Queue elements: ");
    for (int i = queue->front; i <= queue->rear; i++)
    {
        printf("%d ", queue->arr[i]);
    }
    printf("\n");
}

int main()
{
    struct QueueArray *queue = createQueue();
    int choice, value;

    while (1)
    {
        printf("\nQueue Operations (Array Implementation):\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\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 the 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 value: %d\n", value);
            break;
        case 4:
            display(queue);
            break;
        case 5:
            free(queue);
            exit(0);
        default:
            printf("Invalid choice!\n");
        }
    }

    return 0;
}

```

SEM 3\Exp6\Queue_LL.c

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

// Node structure for the linked list
struct Node
{
    int data;
    struct Node *next;
};

// Queue structure using linked lists
struct QueueLinkedList
{
    struct Node *front;
    struct Node *rear;
```

```c
};

// Function to create a queue
struct QueueLinkedList *createQueue()
{
    struct QueueLinkedList *queue = (struct QueueLinkedList *)malloc(sizeof(struct QueueLinkedList));
    queue->front = queue->rear = NULL; // Initialize front and rear
    return queue;
}

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

// Enqueue an element into the queue
void enqueue(struct QueueLinkedList *queue, int value)
{
    struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
    new_node->data = value;
    new_node->next = NULL;

    if (isEmpty(queue))
    {
        queue->front = queue->rear = new_node; // First node
        printf("%d enqueued to queue\n", value);
        return;
    }

    queue->rear->next = new_node; // Add new node at the end
    queue->rear = new_node;       // Update the rear pointer
    printf("%d enqueued to queue\n", value);
}

// Dequeue an element from the queue
int dequeue(struct QueueLinkedList *queue)
{
    if (isEmpty(queue))
    {
        printf("Queue underflow!\n");
        return -1; // Return -1 for underflow
    }
    struct Node *temp = queue->front;
    int dequeued_value = temp->data;
    queue->front = queue->front->next;

    // If the front becomes NULL, set rear to NULL as well
    if (queue->front == NULL)
    {
        queue->rear = NULL;
    }

    free(temp);
    return dequeued_value;
}

// Peek at the front element of the queue
int peek(struct QueueLinkedList *queue)
{
    if (isEmpty(queue))
    {
        printf("Queue is empty!\n");
        return -1; // Return -1 if empty
    }
    return queue->front->data;
}

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

int main()
{
    struct QueueLinkedList *queue = createQueue();
    int choice, value;

    while (1)
    {
        printf("\nQueue Operations (Linked List Implementation):\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\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 the 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 value: %d\n", value);
            break;
        case 4:
            display(queue);
            break;
        case 5:
            // Free linked list nodes (cleanup)
            while (!isEmpty(queue))
            {
                dequeue(queue);
            }
            free(queue);
            exit(0);
        default:
            printf("Invalid choice!\n");
        }
    }

    return 0;
}

```