Tutorial 5: Queues

ELEC 278: Fundamentals of Information Structures

The learning goals for Tutorial 5 are:

- Practice building circular model queues and deques.

Problem 1. Replicated from `https://leetcode.com/problems/design-circular-deque/`

Implement a circular double-ended queue (deque).
Implement the `MyCircularDeque` struct:

- `myCircularDequeCreate(int k)` Initializes the deque with a maximum size of k.
- `bool myCircularDequeInsertFront(MyCircularDeque* obj, int val)` Adds an item at the front of Deque. Returns true if the operation is successful, or false otherwise.
- `bool myCircularDequeInsertLast(MyCircularDeque* obj, int val)` Adds an item at the rear of Deque. Returns true if the operation is successful, or false otherwise.
- `bool myCircularDequeDeleteFront(MyCircularDeque* obj)` Deletes an item from the front of Deque. Returns true if the operation is successful, or false otherwise.
- `bool myCircularDequeDeleteLast(MyCircularDeque* obj)` Deletes an item from the rear of Deque. Returns true if the operation is successful, or false otherwise.
- `int myCircularDequeGetFront(MyCircularDeque* obj)` Returns the front item from the Deque. Returns -1 if the deque is empty.
- `int myCircularDequeGetRear(MyCircularDeque* obj)` Returns the last item from Deque. Returns -1 if the deque is empty.
- `bool myCircularDequeIsEmpty(MyCircularDeque* obj)` Returns true if the deque is empty, or false otherwise.
- `bool myCircularDequeIsFull(MyCircularDeque* obj)` Returns true if the deque is full, or false otherwise.
- `void myCircularDequeFree(MyCircularDeque* obj)` Frees all memory allocated to rge deque.

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

typedef struct {
    // complete
} MyCircularDeque;
```

```c
// Initialize deque with maximum size k
MyCircularDeque* myCircularDequeCreate(int k) {
    // to do
}

// Check if deque is empty
bool myCircularDequeIsEmpty(MyCircularDeque* obj) {
    // to do
}

// Check if deque is full
bool myCircularDequeIsFull(MyCircularDeque* obj) {
    // to do
}

// Add an item at the front
bool myCircularDequeInsertFront(MyCircularDeque* obj, int value) {
    // to do
}

// Add an item at the rear
bool myCircularDequeInsertLast(MyCircularDeque* obj, int value) {
    // to do
}

// Delete an item from the front
bool myCircularDequeDeleteFront(MyCircularDeque* obj) {
    // to do
}

// Delete an item from the rear
bool myCircularDequeDeleteLast(MyCircularDeque* obj) {
    // to do
}

// Get the front item
int myCircularDequeGetFront(MyCircularDeque* obj) {
    // to do
}

// Get the rear item
int myCircularDequeGetRear(MyCircularDeque* obj) {
    // to do
}

// Free the deque
void myCircularDequeFree(MyCircularDeque* obj) {
    // to do
}

int main() {
    MyCircularDeque* deque = myCircularDequeCreate(3); // Size 3

    printf("Insert Last 1: %d\n", myCircularDequeInsertLast(deque, 1)); // returns
     true
    printf("Insert Last 2: %d\n", myCircularDequeInsertLast(deque, 2)); // returns
     true
    printf("Insert Front 3: %d\n", myCircularDequeInsertFront(deque, 3)); //
    returns true
```

```
    printf("Insert Front 4: %d\n", myCircularDequeInsertFront(deque, 4)); //
    returns false, queue is full

    printf("Get Rear: %d\n", myCircularDequeGetRear(deque)); // returns 2
    printf("Is Full: %d\n", myCircularDequeIsFull(deque)); // returns true

    printf("Delete Last: %d\n", myCircularDequeDeleteLast(deque)); // returns true
    printf("Insert Front 4: %d\n", myCircularDequeInsertFront(deque, 4)); //
    returns true
    printf("Get Front: %d\n", myCircularDequeGetFront(deque)); // returns 4

    myCircularDequeFree(deque); // Free the deque
    return 0;
}
```

Solution.

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

typedef struct {
    int *data;
    int front; // index of first element
    int rear; // index of last element
    int size; // how much we have used
    int capacity; // how much capacity we have available from the start
} MyCircularDeque;

// Initialize deque with maximum size k
MyCircularDeque* myCircularDequeCreate(int k) {
    MyCircularDeque *deque = (MyCircularDeque*)malloc(sizeof(MyCircularDeque));
    deque->data = (int*)malloc(k * sizeof(int));
    deque->front = 0; // can you start-off with different numbers for front and
    rear?
    deque->rear = -1;
    deque->size = 0;
    deque->capacity = k;
    return deque;
}

// Check if deque is empty
bool myCircularDequeIsEmpty(MyCircularDeque* obj) {
    return obj->size == 0;
}

// Check if deque is full
bool myCircularDequeIsFull(MyCircularDeque* obj) {
    return obj->size == obj->capacity;
}

// Add an item at the front
bool myCircularDequeInsertFront(MyCircularDeque* obj, int value) {
    if (myCircularDequeIsFull(obj)) {
        return false; // Deque is full
    }
    obj->front = (obj->front - 1 + obj->capacity) % obj->capacity;
    obj->data[obj->front] = value;
    obj->size++;
    return true;
```

```c
}

// Add an item at the rear
bool myCircularDequeInsertLast(MyCircularDeque* obj, int value) {
    if (myCircularDequeIsFull(obj)) {
        return false; // Deque is full
    }
    obj->rear = (obj->rear + 1) % obj->capacity;
    obj->data[obj->rear] = value;
    obj->size++;
    return true;
}

// Delete an item from the front
bool myCircularDequeDeleteFront(MyCircularDeque* obj) {
    if (obj->size == 0) {
        return false; // Deque is empty
    }
    obj->front = (obj->front + 1) % obj->capacity;
    obj->size--;
    return true;
}

// Delete an item from the rear
bool myCircularDequeDeleteLast(MyCircularDeque* obj) {
    if (obj->size == 0) {
        return false; // Deque is empty
    }
    obj->rear = (obj->rear - 1 + obj->capacity) % obj->capacity;
    obj->size--;
    return true;
}

// Get the front item
int myCircularDequeGetFront(MyCircularDeque* obj) {
    if (obj->size == 0) {
        return -1; // Deque is empty
    }
    return obj->data[obj->front];
}

// Get the rear item
int myCircularDequeGetRear(MyCircularDeque* obj) {
    if (obj->size == 0) {
        return -1; // Deque is empty
    }
    return obj->data[obj->rear];
}

// Free the deque
void myCircularDequeFree(MyCircularDeque* obj) {
    free(obj->data);
    free(obj);
}

int main() {
    MyCircularDeque* deque = myCircularDequeCreate(3); // Size 3

    printf("Insert Last 1: %d\n", myCircularDequeInsertLast(deque, 1)); // returns
```

```
    true
    printf("Insert Last 2: %d\n", myCircularDequeInsertLast(deque, 2)); // returns
    true
    printf("Insert Front 3: %d\n", myCircularDequeInsertFront(deque, 3)); //
   returns true
    printf("Insert Front 4: %d\n", myCircularDequeInsertFront(deque, 4)); //
   returns false, queue is full

    printf("Get Rear: %d\n", myCircularDequeGetRear(deque)); // returns 2
    printf("Is Full: %d\n", myCircularDequeIsFull(deque)); // returns true

    printf("Delete Last: %d\n", myCircularDequeDeleteLast(deque)); // returns true
    printf("Insert Front 4: %d\n", myCircularDequeInsertFront(deque, 4)); //
   returns true
    printf("Get Front: %d\n", myCircularDequeGetFront(deque)); // returns 4

    myCircularDequeFree(deque); // Free the deque
    return 0;
}
```