

Code:

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

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

```
typedef struct {  
    int* data;  
    int front;  
    int rear;  
    int size;  
} Queue;
```

```
typedef struct {  
    Queue* q1;  
    Queue* q2;  
} MyStack;
```

```
Queue* createQueue(int size) {  
    Queue* queue = (Queue*)malloc(sizeof(Queue));  
    queue->data = (int*)malloc(size * sizeof(int));  
    queue->front = queue->rear = -1;  
    queue->size = size;  
    return queue;  
}
```

```
void enqueue(Queue* queue, int value) {  
    if (queue->rear == -1) {  
        queue->front = queue->rear = 0;  
    } else {  
        queue->rear = (queue->rear + 1) % queue->size;  
    }  
    queue->data[queue->rear] = value;  
}
```

```
int dequeue(Queue* queue) {  
    int value = queue->data[queue->front];  
    if (queue->front == queue->rear) {  
        queue->front = queue->rear = -1;  
    } else {  
        queue->front = (queue->front + 1) % queue->size;  
    }  
}
```

```

    return value;
}

bool isEmpty(Queue* queue) {
    return queue->front == -1;
}

MyStack* myStackCreate() {
    MyStack* stack = (MyStack*)malloc(sizeof(MyStack));
    stack->q1 = createQueue(1000); // Adjust the size as needed
    stack->q2 = createQueue(1000);
    return stack;
}

void myStackPush(MyStack* obj, int x) {
    enqueue(obj->q1, x);
}

int myStackPop(MyStack* obj) {
    if (isEmpty(obj->q1)) {
        return -1; // Stack is empty
    }

    while (obj->q1->front != obj->q1->rear) {
        enqueue(obj->q2, dequeue(obj->q1));
    }

    int poppedValue = dequeue(obj->q1);

    // Swap q1 and q2
    Queue* temp = obj->q1;
    obj->q1 = obj->q2;
    obj->q2 = temp;

    return poppedValue;
}

int myStackTop(MyStack* obj) {
    if (isEmpty(obj->q1)) {
        return -1; // Stack is empty
    }

```

```

    }

    while (obj->q1->front != obj->q1->rear) {
        enqueue(obj->q2, dequeue(obj->q1));
    }

    int topValue = dequeue(obj->q1);
    enqueue(obj->q2, topValue);

    // Swap q1 and q2
    Queue* temp = obj->q1;
    obj->q1 = obj->q2;
    obj->q2 = temp;

    return topValue;
}

bool myStackEmpty(MyStack* obj) {
    return isEmpty(obj->q1);
}

void myStackFree(MyStack* obj) {
    free(obj->q1->data);
    free(obj->q1);
    free(obj->q2->data);
    free(obj->q2);
    free(obj);
}

```

Output:

The screenshot displays a LeetCode submission interface. On the left, the 'Submissions' tab is active, showing a green 'Accepted' status. Below this, a 'Runtime' of 3 ms and 'Memory' of 6.58 MB are listed. A 'Testcase' section shows 'Case 1' as 'Accepted' with a runtime of 3 ms. The input sequence is ["MyStack", "push", "push", "top", "pop", "empty"] and the stack state is [[], [1], [2], [], [], []]. The output is [null, null, null, 2, 2, false] and the expected output is [null, null, null, 2, 2, false]. On the right, the C++ code is shown, implementing a stack using two queues (q1 and q2). The code includes functions for enqueue, dequeue, top, pop, and empty operations.

```
81
82 while (obj->q1->front != obj->q1->rear) {
83     enqueue(obj->q2, dequeue(obj->q1));
84 }
85
86 int topValue = dequeue(obj->q1);
87 enqueue(obj->q2, topValue);
88
89 // Swap q1 and q2
90 Queue* temp = obj->q1;
91 obj->q1 = obj->q2;
92 obj->q2 = temp;
93
94 return topValue;
95 }
96
97 bool myStackEmpty(MyStack* obj) {
98     return isEmpty(obj->q1);
99 }
100
101 void myStackFree(MyStack* obj) {
102     free(obj->q1->data);
103     free(obj->q1);
104     free(obj->q2->data);
105     free(obj->q2);
106     free(obj);
107 }
108
```