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

// Returns true if the queue is full or queue is NULL
bool isfull(queue *q){
	return q == NULL || q->count == MAXQUEUE;
}

// Creates a queue and returns it to the user
queue *createq(void) {
	queue *newq;
	// Atemps to allocate the memory for queue
	if ((newq = (queue *) malloc(sizeof(queue))) == NULL) {
		// If the allocation fails
		printf("createq: Error allocating memory\n");
	}else {
		// If the allocation worked
		// Sets the head and tail to the first array spot
		newq->head = newq->tail = newq->count = 0;
	}
	return newq;
}

// Adds the a new person to the queue
bool enqueue(queue * q, person p) {
	bool success = false;

	// Check the status of queue
	if (q == NULL){
		// If the queue doesn't exisits
		printf("enqueue: Error - queue is NULL\n");
	} else if (isfull(q)) {
		// If the queue is full
		printf("enqueue: Error - queue is full\n");
	} else {
		// If the queue can be added
		// Inserts the new user at the tail
		q->theq[q->tail] = p;
		// Increments the count by
		q->count++;
		// Sets the tail pointer to the next spot in the array
		q->tail = (q->tail +1) % MAXQUEUE;
		success = true;
	}
	return success;
}

// Returns the person at the head of the queue
person dequeue(queue *q) {
	// Creates a person struct
	person retval = {"", 0};

	// Checks the status of queue
	if(q == NULL) { // non-existent queue, return empty person
		printf("dequeue: Error - queue is NULL\n");
	} else if (!isemptyqueue(q)) {
		// If the queue is not empty
		// Gets the head of the queue
```

```c
            retval = q->theq[q->head];
            // Decrement the account
            q->count--;
            // Updates head pointer to next person
            q->head = (q->head + 1) % MAXQUEUE;
      } else {
            // If queue is empty
            printf("dequeue: Error - empty queue\n");
      }
      // Returns retval updated or not
      return retval;
}

// Return true if the queue is empty or doesn't exists
bool isemptyqueue(queue *q) {
      return q == NULL || q->count == 0;
}


bool emptyq(queue * q){
      bool success = (q != NULL);

      // success is true
      if (success) {
            // sets the head and tail to the first spot in the queue
            q->head = q->tail = q->count = 0;
      }

      return success;
}

// Deallocates the memory used for the queue
bool freeq(queue *q){
      bool success = isemptyqueue(q);

      if (success && q != NULL){
            free(q);
      }
      return success;
}
```