```
/* Implementation of a simple circular queue using a static array */
#include <stdio.h>
#include <stdlib.h>
#include "jobstuff.h"
#include <string.h>
#define TRUE 0
#define FALSE 1

// expansion of queue.c

// Makes a job to add it to the queue
job init_job(int id, char *command) {
        job newJob;

   newJob.jobid = id;
   newJob.job_comm = command;
   newJob.status = "Waiting...";
        newJob.outFile = malloc(10);
        newJob.errFile = malloc(10);
   sprintf(newJob.outFile, "%d.out", id);
   sprintf(newJob.errFile, "%d.out", id);

   return newJob;
}

/* create the queue data structure and initialize it */
queue *queue_init(int n) {
        queue *q = (queue *)malloc(sizeof(queue));
        q->size = n;
        q->item = malloc(sizeof(job *)*n);
        q->start = 0;
        q->end = 0;
        q->count = 0;

        return q;
}

void show_jobs(int arrlen, job *jobList) {
   int i;

   // Call you a noob if the array is empty and you try to show jobs
   if (jobList == NULL && arrlen == 0) {
      printf("There are no jobs running or waiting.\n");
   }
```

```
    else {
        // Formatting
        printf("<Job ID>    <Command>          <Status>\n"); // 3 tabs btwn Command and Status
        // Loop through array and print the jobs
        for (i = 0; i < arrlen; i++) {
            if (strcmp(jobList[i].status, "Complete.") != TRUE) {
                printf("%d         %s          %s\n", jobList[i].jobid, jobList[i].job_comm, jobList[i].status);
            }
        }

    }

}

/* insert an item into the queue, update the pointers and count, and
   return the no. of items in the queue (-1 if queue is null or full) */
int queue_insert(queue *q, job *job) {
        if ((q == NULL) || (q->count == q->size))
          return -1;

        q->item[q->end % q->size] = job;
        q->end = (q->end + 1) % q->size;
        q->count++;

        return q->count;
}

/* delete an item from the queue, update the pointers and count, and
   return the item deleted (-1 if queue is null or empty) */
   // Edited to instead grab the job and pop it off the queue
job *queue_delete(queue *q) {
        if ((q == NULL) || (q->count == 0))
          return (job *)-1;

        job *pop = q->item[q->start];
        q->start = (q->start + 1) % q->size;
        q->count--;

        return pop;
}

/* delete the queue data structure */
void queue_destroy(queue *q) {
        free(q->item);
```

```
        free(q);
}
```