Washington State University Vancouver

CS 360 Precheck

Waitlist Simulator

Professor:
Ben McCamish

Overall Assignment

Write a program (in C) targeted at the Linux platform which implements a queue structure to simulate a line or waitlist of people.

Program Interface

You will implement at least the following interface. Feel free to create additional functions as needed, but do not modify the header file provided.

```
struct individual {
    char firstName[128];
    char lastName[128];
    char issue[256];
};

struct node {
    struct individual person;
    struct node *next;
};

struct node *queueInit();
void queueAdd(struct node *queue, char *firstName, char *lastName, char *issue);
struct individual queueDelete(struct node* queue);
struct individual queueHead(struct node* queue);
struct individual queueTail(struct node* queue);
void freeQueue(struct node* queue);
```

Further, you will have to create your own testbed for the program in order to test different inputs into your functions and to check whether the functions are working as intended before submitting on Autolab.

The Structures

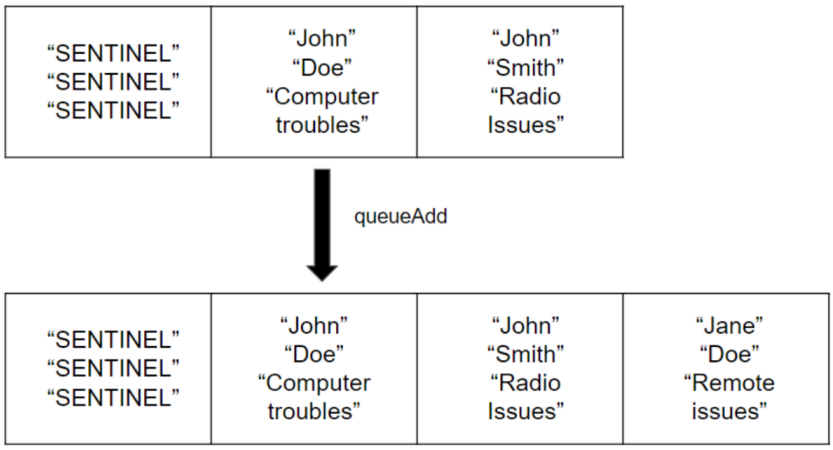
There are two structures here, the person and the node structures, you must implement both. Note that the struct indivudal defined inside of the node structure is not a pointer (and thus will need dot notation to be accessed).

struct node queueInit()

This function will initialize a sentinel node for the queue data structure. The function will simply return a node with a person with firstName, lastName, and issue being set to "SENTINEL", and the pointer to the next node being set to NULL.

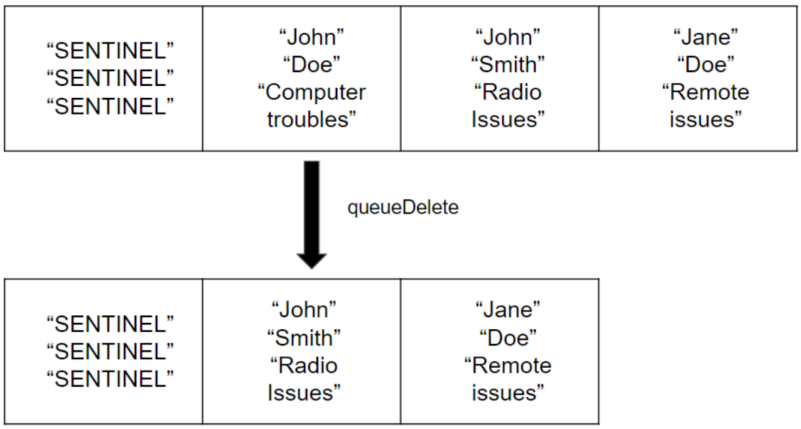
void queueAdd(struct node *queue, char *firstName, char *lastName, char *issue)

This function will take a node of the queue (node given must be the sentinel node) and the first name, last name, and issue of the struct of the person that is to be inserted into the queue and insert into the tail of the queue.



struct individual queueDelete(struct node* queue)

This function will remove the head of the queue (in this case that would be the node pointed to by the sentinel) and return a structure of the person tied to the header node (note that this should not be a pointer to a structure, but rather a normal structure).



If the queue passed is empty, then the function should simply return a struct individual with all empty strings for each of its values (this will be tested on Autolab). Don't forget to free the removed node.

struct individual queueHead(struct node* queue)

This function will simply take the sentinel node of the queue and return a struct individual of the person in the head of the queue (in our case the head of the queue is the node pointed to by the sentinel node). You may assume that there will be no empty queues passed to this function.

struct individual queueTail(struct node* queue)

This function will simply take the sentinel node of the queue and return a regular structure of the tail of the queue, which is the last node in the queue. You may assume that there will be no empty queues passed to this function.

void freeQueue(struct node* queue)

This function will simply free each node in the given queue.

Important Notes

- Make sure to pay close attention to when you are dealing with a pointer to a structure and when when you are dealing with a regular structure, and use the appropriate notation (dot notation for regular structures, arrow notation for

pointers to a structure).

- Remember that you don't need to and shouldn't use malloc to declare a regular structure, only for a pointer to a structure.
- Remember to free all memory which was allocated, you will be tested on this in Autolab. If you are coding on the linux server you can use Valgrind to make sure all heap blocks were freed.

What to turn in (on Autolab):

- A single C file called "waitlist.c"