Programming Assignment 2 - Part I

Design and Implementation of a Service Queue ADT

DUE DATES:

(A) Writeup (design document -- corresponding to "Step 3" below.
Friday, Sept 29 by 8:59AM
Submitted Via Gradescope

(B) Implementation: Saturday Oct 7 by 11:59PM (blackboard)

You go to a popular restaurant that doesn't take reservations. When you get there, you just have to wait in line for a table. These days, most restaurants will give you some kind of buzzer and will signal you when your table is ready.

The restaurant essentially has to manage a queue of buzzers. Implementation of an ADT supporting this management is what you will be doing in this assignment.

The ADT interface is specified in the provided file **sq.h** (which you are NOT allowed to modify!). The first thing it does is (partially) specify a type **SQ** for a service queue. A client program uses service queue pointers (type **SQ ***).

Below is the **SQ typedef** and function prototypes as specified in **sq.h**

```
// incomplete type:  struct service fields specified in
//    implementation file
typedef struct service_queue SQ;

extern SQ * sq_create();

void sq_free(SQ *q);

extern void sq_display(SQ *q);

extern int  sq_length(SQ *q);               // O(1)

extern int  sq_give_buzzer(SQ *q);          // O(1) amortized

extern int sq_seat(SQ *q);                  // O(1)

extern int sq_kick_out(SQ *q, int buzzer);  // O(1)

extern int sq_take_bribe(SQ *q, int buzzer); // O(1)
```

Each function has a banner comment above it specifying the required behavior ***AND*** a required runtime. **Read sq.h!**

Your main job: design and write an implementation file **sq.c** meeting ***all*** of the requirements specified in **sq.h**

STEP 1: read the banner comments in sq.h and make sure you understand the required behavior of all of the functions.

STEP 2: For reference, you have been given a complete implementation file **sq_slow.c**. The catch is: it doesn't meet all of the runtime requirements.

Read through this implementation.

Reading through this pretty simple implementation should help you understand the expected behavior (but not how to meet the runtime requirements). This implementation uses the previously existing ADT for lists.

You will also notice an application program **driver.c**. It initializes a service queue and starts a simple interactive interface that lets the user perform the various operations (basically a one-to-one correspondence with the functions plus a quit command).

STEP 3: Now the fun part! Start designing *your* implementation of the service queue which meets the runtime requirements. Advice:

- Take a "blank slate" approach: do NOT try to modify sq_slow.c to meet the runtime requirements.
- Start with pencil and paper.
- Remember, you get to specify what goes in the **service_queue** structure.
- Take advantage of the fact that the buzzer-IDs are not just any old integers -- they start from zero and increase from there.
- It will probably ***NOT*** be useful to use the given **list** ADT like **sq_slow.c** does. This does not mean that ***some kind of*** linked list implementation won't be useful. Take a blank slate approach to this aspect as well.
- Note that almost all of the runtime requirements are O(1). So if you find yourself writing a loop, think twice. (In the case of **sq_give_buzzer**, you probably will end up with a loop, but read the runtime requirements carefully).

When you think you have a correct "design" (i.e., an organization of the data which enables the specified run times), feel free to post a Piazza note to instructors only for a sanity check. Such posts are expected to be very clear if you expect a meaningful response!!!

STEP 4: Implement, debug and test your implementation.

Deliverable A: Writeup

Submitted via gradescope. Due Friday Sept 29 by 8:59AM.

You will submit a writeup which includes two things:

1. Your "design"
2. An illustration of your approach through a specific example (details below).

<u>(1) Design:</u>

- Your "design" is a explanation of how you plan to organize the data to make the runtime requirements achievable.
- Some Key Points:
    - This is not "code" per-se. It is a concise description of your strategy **almost certainly including diagram**s
    - Focus on the data structures -- how things are organized. You can express this organization in terms of C structures (but probably not code actually operating on them).
    - Give step-by-step descriptions of how the individual operations will work and meet runtime requirements.

What makes a good design document?

<u>Think of it this way</u>:

You are part of a team and have been tasked with coming up with a strategy to meet the requirements stated above.

You have a team meeting at which you are to present what you have figured out and what recommendations you have (and maybe what things you haven't figured out).

At this meeting, you will give every team member a handout.

The handout should be your design document!

<u>Here is a related rule</u>: any competent computer scientist should be able to read this document and:

1. Be able to translate it into an actual implementation
2. Be convinced of its correctness and meeting claimed runtimes.

<u>Candor</u>: if you just aren't able to resolve a requirement, it is not a crime to say so and explain your obstacles. This is better than random stuff you know won't work.

Once you have a draft of your design description, try to re-read as if you've never seen it before. Be honest with yourself if you've met the above criteria. Then improve it! Do your best -- with practice you will get better at it.

Your design document can be handwritten, but please make it clear and neat!!

(2) Examples

The 2nd part of your writeup will illustrate how your approach behaves for a specific scenario. Do this by answering the questions below:

Suppose your service queue is created and the following happens (in this order):

- There are 20 calls to sq_get_buzzer
- There are two calls to sq_seat

QUESTION 1: Give a detailed diagram of the state of your data structure after these operations.

QUESTION 2: Now suppose there is a call to sq_kick_out on buzzer number 9. Explain how your strategy will perform this operation; refer to the diagram from Question-1 in your explanation. Suggestion: You might use multiple colors in your diagram with certain colors corresponding to the operation steps.

QUESTION 3: Now re-draw the state of your data structure after completion of the sq_kick_out operation.

QUESTION 4: Now do the same kind of simulation, but for a sq_bribe operation on buzzer 11.

QUESTION 5: Finally, redraw your data structure after completion of the sq_bribe operation.

Deliverable B: Implementation.

(Submitted via blackboard): You will submit a single zip file containing the following:

- All source code:
  - sq.c
  - sq.h (unmodified)
  - driver.c
  - any other source files you wrote for your solution. You probably won’t have any other files, but if you decide to partition your code, it is acceptable.
- Makefile:

  the makefile must enable the following:

  ```
  > make sq.o
  > make fdriver
  ```

  The target fdriver is just the given driver program which uses your (fast) implementation of the ADT.

  Just modify the given makefile…

Hints:

- You may find doubly-linked list nodes handy!
- It's cliche, but try to think outside the box. Think about your toolbox -- you have structs, arrays, pointers, etc.. Be creative about how to use them!

Submission

Your zip file will be uploaded to Blackboard as usual. Note that there will be separate submissions for Part-I and Part-II.

Checklist

Item	Description	Due Date	Comments
(A) Writeup	**This document has two parts:** **(1) 'Design':** How you plan to organize your data structures and how this organization meets the requirements (mainly runtime). Click here for details and suggestions. **(2) Examples:** You will draw and explain the state of your data structure after a specified operations. Details here.	Friday, Sept 29 by 8:59AM **NO EXCEPTIONS!**	**Submission will be through gradescope.** **Handwritten submissions are fine but are expected to be neat and easy to read!**
On Friday, Sept 29 we will go over a viable design strategy in lecture. Thus, even if your initial design did not meet all requirements, you still have an opportunity to write an implementation which does!			
(B) Implementation	You will submit your completed implementation in C. The key file will be sq.c, but you will also submit a makefile, etc. Details here.	Saturday, Oct 7 by 11:59PM **NO EXCEPTIONS**	**Submission via Blackboard.**