Programming Assignment 6
Implementation of a Service Queue ADT
DUE: Thursday Nov 20 11:59PM

> **NOTE:** during the Friday, Nov 12 lecture we will walk through an approach that meets the runtime requirements of this assignment. This way, even if you are unable to devise an approach meeting all requirements, you still have an opportunity to complete a correct implementation while still giving everyone an opportunity to figure out a solution on their own. (Please at least do some brainstorming prior to Nov 12!!)

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**. 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.

Your main job: 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: Think about which runtime requirements sq_slow meets and which it does not.

STEP 4: 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 5: Implement, debug and test your implementation.

Tips/Hints in your quest to meet the runtime requirements:

- A linked list implementation using doubly-linked nodes might come in handy
- Dynamically resized arrays might also be handy.

Deliverables

Your submission will include the following:

- All source code:
  - sq.c
  - sq.h (function declarations and typedef must NOT be modified)
  - driver.c
  - any other source files you wrote for your solution. You don't *need* to have additional 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…
