15-122: Principles of Imperative Computation Section G

Recitation 9 — Josh Zimmerman

Practice!

Credit for this section goes to CMU alumna, Caroline Buckey. It was updated by current 122 TA Alex Cappiello to account for differences in exactly how we teach certain material.

Suppose you have queues implemented using linked lists as shown in lecture, with integer data. Specifically, you have the following structs:

```
struct list_node {
    int data;
    struct list_node *next;
};
typedef struct list_node list;

struct queue_header {
        list *front;
        list *back;
};
typedef struct queue_header* queue;
```

Recall from lecture that in both stacks and queues, we always keep one dummy node. In queues, `back` points to this dummy node. Its fields are uninitialized, and it simply ensures that we never need to worry about `front` or `back` being null.

Problem 1

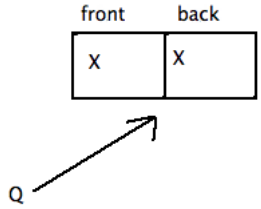
Recall the code for `queue_new()`:

```
queue queue_new()
//@ensures is_queue(\result);
//@ensures queue_empty(\result);
{
  1: queue Q = alloc(struct queue_header);
  2: list* p = alloc(struct list_node);
  3: Q->front = p;
  4: Q->back = p;
  5: return Q;
}
```

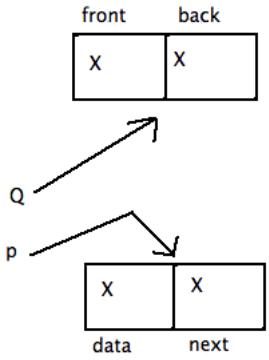
Draw the state of the queue after each given line of code. Use `X` for uninitialized fields.

Solution:

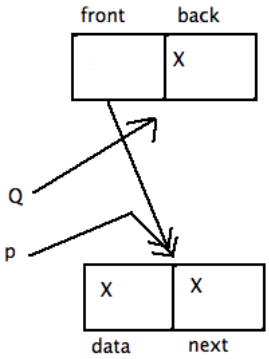
1.



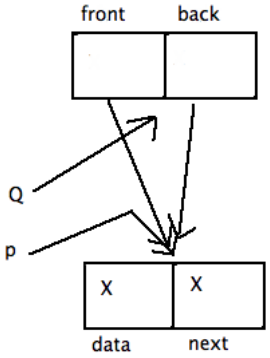
2.



3.



4.



Problem 2

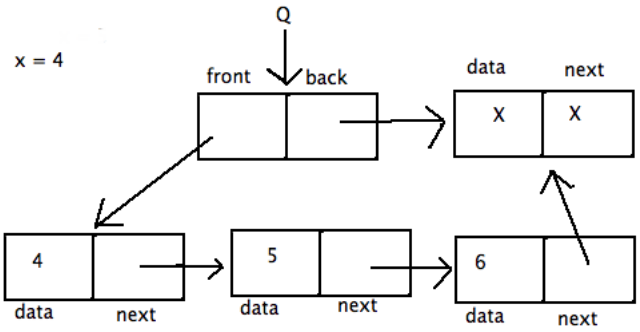
Recall the code for `deq`:

```
int deq(queue Q)
//@requires is_queue(Q);
//@ensures is_queue(Q);
{
    1: int x = Q->front->data;
    2: Q->front = Q->front->next;
    3: return x;
}
```

Suppose `deq(Q)` is called on a queue `Q` that contains before the call, from front to back, `(4, 5, 6)`. Draw the state of the queue after each given line of code. Include an indication of what data the variable `x` holds.

1.

Solution:



2.

Solution: