Worksheet 18: Linked List Queue, pointer to Tail

On Your Own (No need to submit)

1. Draw a picture showing the values of the various data fields in an instance of ListQueue when it is first created.

2. Draw a picture showing what it looks like after one element has been inserted.

3. Based on the previous two drawings, can you determine what feature you can use to tell if a list is empty?

4. Draw a picture showing what it looks like after two elements have been inserted.

5. What is the algorithmic complexity of each of the queue operations?

6. How difficult would it be to write the method addFront(newValue) that inserts a new element into the front of the collection? A container that supports adding values at either and, but removal from only one side, is sometimes termed a *scroll*.

7. Explain why removing the value from the back would be difficult for this container. What would be the algorithmic complexity of the removeLast operation?

```
struct link {
   TYPE value;
   struct link * next;
};

struct listQueue {
   struct link *firstLink;
   struct link *lastLink;
};

void listQueueInit (struct listQueue *q) {
   struct link *lnk = (struct link *) malloc(sizeof(struct link));
   assert(lnk != 0); /* lnk is the sentinel */
   lnk->next = 0;
   q->firstLink = q->lastLink = lnk;
}

void listQueueAddBack (struct listQueue *q, TYPE e) {


struct link *lnk = (struct link *) malloc(sizeof(struct link));
assert (lnk != 0);
lnk->value = e;
lnk->next = q->lastLink->next;
q->lastLink->next=lnk;
q->lastLink = lnk;



}

TYPE listQueueFront (struct listQueue *q) {


Assert( !listQueueIsEmpty(q) ;
Return q->firstLink->next->value ;


}


void listQueueRemoveFront (struct listQueue *q) {



assert(!listQueueIsEmpty(q)) ;
struct link *garage = q->firstLink->next ;
q->firstLink->next = q->firstLink->next->next ;
free(garbage);

}

int listQueueIsEmpty (struct listQueue *q) {

return q->firstLink->next == 0;


}
```