Worksheet 19: Linked List Deque

1. Draw a picture of an initially empty LinkedList, including the two sentinels.

2. Draw a picture of the LinkedList after the insertion of one value.

3. Based on the previous two pictures, can you describe what property characterizes an empty collection?

4. Draw a picture of a LinkedList with three or more values (in addition to the sentinels).

5. Draw a picture after a value has been inserted into the front of the collection. Notice that this is between the front sentinel and the following element. Draw a picture showing an insertion into the back. Notice that this is again between the last element and the ending sentinel. Abstracting from these pictures, what would the function addBefore need to do, where the argument is the link that will follow the location in which the value is inserted.

6. Draw a picture of a linkedList with three or more values, then examine what is needed to remove both the first and the last element. Can you see how both of these operations can be implemented as a call on a common operation, called _removeLink?

7. What is the algorithmic complexity of each of the deque operations?

```
struct dlink {
   TYPE value;
   struct dlink * next;
   struct dlink * prev;
};

struct linkedList {
   int size;
   struct dlink * frontSentinel;
   struct dlink * backSentinel;
};

            /* these functions are written for you */
void LinkedListInit (struct linkedList *q) {
   q->frontSentinel = malloc(sizeof(struct dlink));
   assert(q->frontSentinel != 0);
   q->backSentinel = malloc(sizeof(struct dlink));
   assert(q->backSentinel);
   q->frontSentinel->next = q->backSentinel;
   q->backSentinel->prev = q->frontSentinell;
   q->size = 0;
}

void linkedListFree (struct linkedList *q) {
   while (q->size > 0)
      linkedListRemoveFront(q);
   free (q->frontSentinel);
   free (q->backSentinel);
   q->frontSentinel = q->backSentinel = null;
}

void LinkedListAddFront (struct linkedList *q, TYPE e)
   { _addBefore(q, q->frontSentinel_>next, e); }

void LinkedListAddback (struct linkedList *q, TYPE e)
   { _addBefore(q, q->backSentinel, e); }

void linkedListRemoveFront (struct linkedList *q) {
   assert(! linkedListIsEmpty(q));
   _removeLink (q, q->frontSentinel->next);
}

void LinkedListRemoveBack (struct linkedList *q) {
   assert(! linkedListIsEmpty(q));
   _removeLink (q, q->backSentinel->prev);
}

int LinkedListIsEmpty (struct linkedList *q) {
   return q->size == 0;
}
```

```
/* write addLink and removeLink. Make sure they update the size field correctly */

/* _addBefore places a new link BEFORE the provide link, lnk */
void _addBefore (struct linkedList *q, struct dlink *lnk, TYPE e) {


struct dlink *newLink = (struct dlink *) malloc(sizeof(struct dlink);
assert (newLink !=0);
newLionk -> value = e;
newLink->next = lnk;
newLink->prev = lnk->prev;
lnk->prev->next = newLink;
q->size++;




}

void _removeLink (struct linkedList *q, struct dlink *lnk) {


struct dlink *garbage = lnk;
lnk->prev->next = lnk->next;
lnk->next->prev = lnk->prev;
free(garbage);
q->size--;




}

TYPE LinkedListFront (struct linkedList *q) {

Assert(!LinkedListIsEmpty(q));
Return q->frontSentinel->next->value;


}

TYPE LinkedListBack (struct linkedList *q) {

assert (!LinkedListIsEmpty(q));
return q->backSentinel->prev->value;



}
```