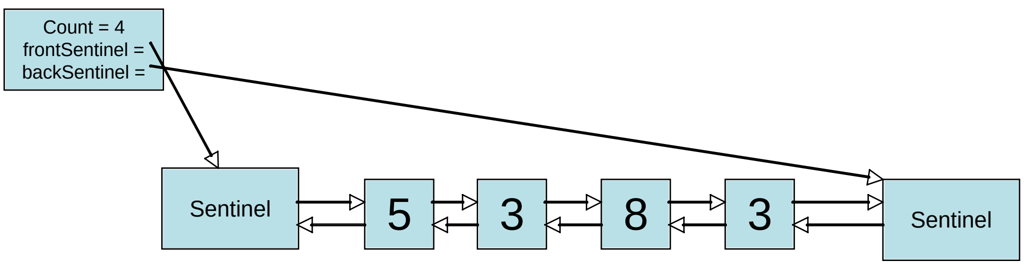
Worksheet 22: Constructing a Bag using a Linked List
Group 11:
Tatyana Vlaskin
Danny Mejia
Katherine Jensen
Michael Sigel – did not participate at all

In Preparation: Read Chapter 8 to learn more about the Bag data abstraction. If you have not done so already, complete Worksheets 17 and 18 to learn about the basic features of the linked list.

In this lesson we continue developing the LinkedList data structure started in Worksheet 19. In the earlier worksheet you implemented operations to add and remove values from either the front or the back of the container. Recall that this implementation used a sentinel at both ends and double links. Because we want to quickly determine the number of elements in the collection, the implementation also maintained an integer data field named count, similar to the count field in the dynamic array bag.



Also recall that adding or removing elements is a problem that you have already solved. Adding a new value at either end was implemented using a more general internal function, termed addLink:

void _addLink (struct LinkedList * lst, struct dlink * lnk, TYPE e);

Similarly removing a value, from either the front or the back, used the following function:

void _removeLink (struct linkedList * lst, struct dlink * lnk);

To create a bag we need three operations: add, contains, and remove. The add operation can simply add the new value to the front, and so is easy to write. The method contains must use a loop to cycle over the chain of links. Each element is tested against the argument, using the EQ macro. If any are equal, then the Boolean value true is returned. Otherwise, if the loop terminates without finding any matching element, the value False is returned.

The remove method uses a similar loop. However, this time, if a matching value is found, then the method removeLink is invoked. The method then terminates, without examining the rest of the collection.

Complete the implementation of the ListBag based on these ideas:

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

struct linkedList {
  struct dlink * frontSentinel;
  struct dlink * backSentinel;
  int size;
};
        /* the following functions were written in earlier lessons */
void linkedlistInit (struct linkedList *lst);
void linkedListFree (struct linkedList *lst);
void _addLink (struct linkedList * lst, struct dlink * lnk, TYPE e);
void _removeLink (struct linkedList * lst, struct dlink * lnk);

void linkedListAdd (struct linkedList * lst, TYPE e)
  { _addLink(lst, lst->frontSentinel->next, e); }

/* you must write the following */
```

The method contains must use a loop to cycle over the chain of links. Each element is tested against the argument, using the EQ macro. If any are equal, then the Boolean value true is returned. Otherwise, if the loop terminates without finding any matching element, the value False is returned.

```
int linkedListContains (struct linkedList *lst, TYPE e) {
   struct DLink *current; //declare a structure
   assert(!isEmptyList(lst)); //check and make sure that there are elements in the list
   current = lst->frontSentinel->next; // Initialize current
   while(current!= lst->backSentinel){//continue until we reach the end
      if(EQ(current->value,e)) //if current matches the value
         return 1;
      current=current->next;
   }
   return 0;
}
```

The remove method uses a similar loop. However, this time, if a matching value is found, then the method removeLink is invoked. The method then terminates, without examining the rest of the collection.

```
void LinkedListRemove (struct linkedList *lst, TYPE e) {
   struct DLink *current;
   assert(!isEmptyList(lst));// make sure that the list is not empty
   current = lst->frontSentinel->next; // Initialize current
   while(current!= lst->backSentinel) {// loop until the end is reached
      if(EQ(current->value,e)) { // if there passed paremeter matches one of the elements
in the list
         _removeLink(lst,current);// remove the element, this function was writen in
earlier lessons
         break; // Only one copy of element will be removed
      }
      current=current->next;// set current to the next element
   }
}
```

1. What were the algorithmic complexities of the methods addLink and removeLink that you wrote back in linked list for Deque?
   - □ addLink **is O(1)**, and can be done at the head or tail
   - □ removeLink **is O(1)**, and can be done at the head or tail

2. Given your answer to the previous question, what are the algorithmic complexities of the three principle Bag operations?

void linkedListAdd : the element is added to the front, thus algorithmic complexity if O(1)

int linkedListContains: must iterate through the linked list in order to find the element to return its value. Worst case is n operations, so the contains function is O(n).

void LinkedListRemove : first we need to iterate trough the list to find the element to remove, so algorithmic complexity if O(n).