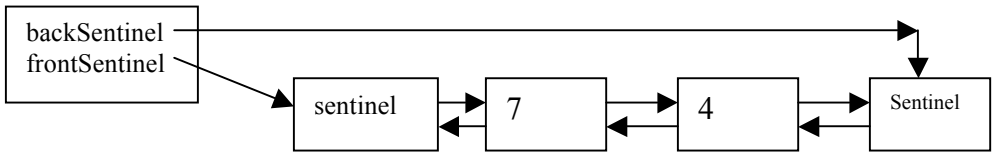
Worksheet 24:Solution: LinkedList Iterator

In preparation: Read Chapter 8 to learn more about the concept of Iterators. If you have not done it already, you should complete Worksheet 17 and 23 to learn more about the basic features of the linked list, as well as the dynamic array iterator.

In this worksheet you will extend the design of the LinkedList abstraction you created in Worksheet 17, creating for it an iterator that uses the same interface as the dynamic array iterator you developed in Worksheet 23. Recall that for the LinkedList that double links are being used, and that there is both a starting and ending sentinel.



An iterator, you will remember, is defined by four functions. The first initializes the iterator, associating it with the container it will iterate over. The function hasNext returns true if there are more elements, false otherwise. The function next returns the current element. The function remove can be used to remove the element last returned by next.

```
/* conceptual interface */
 Boolean  hasNext ( );
 TYPE next ( );
 void remove ( );
```

The dynamic array iterator maintained an integer index into the container. The linked list iterator will do something similar. It will maintain a pointer to one of the links in the container. The functions hasNext and Next must move this pointer forward, so as to eventually reference every method.

Be careful with the remove method. You want to make sure that when the next iteration of the loop is performed the next element in sequence will be produced. However, the actual removal can be made easier using the function you wrote in Lesson 19.

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

You may find it useful to draw a picture of the linked list, adding both the front and the back sentinels, to help you better understand how the fields in the linked list iterator should change as each element is returned.

```
struct linkedListIter {
	struct DLink *cur;
	struct linkedList *lst;
};


void  initlinkedListIter (struct linkedList *lst, struct linkedListIter
*itr) {

	itr->lst = lst;
	itr->cur = lst->frontSentinel->next;
}


struct linkedListIter *createlinkedListIter(struct linkedList *lst){
	struct linkedListIter *newItr = malloc (sizeof(struct
linkedListIter));
	initlinkedListIter(lst, newItr);
	return newItr;
}

int hasNextlinkedListIter (struct linkedListIter *itr) {

	if(itr->cur != itr->lst->backSentinel) {

		itr->cur = itr->cur->next;
		return (1);
	}
	else return(0);

}

TYPE nextlinkedListIter (struct linkedListIter *itr) {

	TYPE val = itr->cur->prev->value;
	return val;
}

 /* removes the last value returned by 'next'
  * Be careful with this one!
  * Notice that we use a tmp to ensure that the following
  * calls to hasNext and next are correct after removal of the
  * current link.*/
void removelinkedListIter (struct linkedListIter *itr) {

	struct DLink *tmp = itr->cur;
	itr->cur = itr->cur->prev;
	_removeLink(itr->lst, tmp);
}
```