Iterator ADT

Dynamic Array and Linked List
Goals

• Why do we need iterators?
• Iterator ADT
• Linked List and Dynamic Array Iterators
  – The lecture will focus on link list iterators
void initBag (struct Bag *da, int cap);

void freeBag (struct Bag *da);

void addBag (struct Bag *da, TYPE val);

int containsBag (struct Bag *da, TYPE val);

void removeBag (struct Bag *da, TYPE val);

int sizeBag (struct Bag *da);

What if the user wants to loop through the bag to print out of filter contents?
Iterator Concept

• Problem: How do you provide a user of a container access to the elements, without exposing the inner structure?

• Think of two developers: one writing the container (that’s you!!!), the other using the container (that’s someone using your library to build an application where they need, for example, a stack implementation)
Traversing a Container – as Developer

For example, within the Linked List container you (the developer) wrote a loop such as the following:

```c
struct LinkedList *list;
struct Link *l;
...
/* Initialize list. */
l = list->frontSentinel->next;
while(l!=list->backSentinel){
    ...do something...
    l=l->next;
}
```

This is fine within the container library code itself, but we don’t want users of the container library to have to know about links...or worse yet, be able to manipulate links!
Encapsulation

• Chapter 5: Hide the implementation details behind a simple and easy to remember interface (ie. abstraction mechanism)
• Users should *not* know about links, arrays, size, capacity, etc.
• Users should know and use the public abstractions: push, pop, contains, remove, etc.
How do we abstract away loops...

```c
for(i = 0; i < sizeDynArr; i++)
    val = getDynArr(i);

while( cur != backSent)
    do something
    cur = cur -> next

do {
    something
    cur = cur->next;
}while (cur != backSent);
```
Solution: define an interface that provides methods for writing loops

```c
int hasNextIter(struct Iter *itr);
TYPE nextIter(struct Iter *itr);
void removeIter(struct Iter *itr);
void changeIter(struct Iter *itr, TYPE val);
void addIter(struct Iter *itr, TYPE val);
```
TYPE cur; /* current collection val */
struct linkedList *list;
linkedListIterator *itr;
list = createLinkedListList(...)
itr = createLinkedListListIter(list)

while (hasNextListIter(itr)) {
    cur = nextListIter(itr);
    if (cur ...)
        removeListIter(itr);
}
struct linkedListIterator {
    struct linkedList *lst;
    struct dlink *currentLink;
}

struct linkedListIterator *createLinkedListIterator (struct linkedList *lst)
{
    struct linkedListIterator *itr;
    itr = malloc(sizeof(struct linkedListIterator));
    itr->lst = lst;
    itr->currentLink = lst->frontSentinel;
    return itr;
}
After Initialization
Strategy

- **HasNext**
  - Returns T (F) to the user if there is (is not) another value remaining to be enumerated

- **Next**
  - Return the next value to be enumerated and updates current to point to that value (assumes HasNext was called and returned T)

- **Remove**
  - Removes the value (and link) that was *last returned* by call to Next()

NOTE: Some designs will use HasNext to update the current and Next just returns current.
After Initialization

while (hasNextListIter(itr)) {
    curVal = nextListIter(itr);
    ...
}

List

<table>
<thead>
<tr>
<th>frontSent</th>
</tr>
</thead>
<tbody>
<tr>
<td>prev</td>
</tr>
<tr>
<td>front</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>backSent</th>
</tr>
</thead>
<tbody>
<tr>
<td>prev</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>current</th>
</tr>
</thead>
<tbody>
<tr>
<td>prev</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ltr</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>nextListIter(itr)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
After One Iteration: hasNext, next

```java
while (hasNextListIter(itr)) {
    curVal = nextListIter(itr);
    ...
}
```
While (hasNextListIter(itr)) {
    curVal = nextListIter(itr);
    ...
}

After Two Iterations
After Three Iterations

We’ve enumerated all elements
Subsequent calls to HasNext will evaluate to false

while (hasNextListIter(itr)) {
    curVal = nextListIter(itr);
    ...
}
After Two Iterations – then Remove

Remove the last value enumerated (10)
Where should current be after the removal?

```java
while (hasNextListIter(itr)) {
    curVal = nextListIter(itr);
    if (curVal == 10) {
        removeListIter(itr);
    }
}
```
After Two Iterations – then Remove

Remove the last value enumerated (10)
Where should current be after the removal?
Your Turn

Worksheet#24 Linked List Iterator
// Return T (F) if there is (is not) a next element

int hasNextListIter (struct linkedListIterator *itr)
{
    return itr->currentLink->next !=
            itr->lst->backSentinel;
}

struct linkedListIterator {
    struct linkedList *lst;
    struct DLink *currentLink;
}
/* Returns next element and updates that element to be the current element */

TYPE nextListIter (struct LinkedListIterator *itr)
{
    itr->currentLink = itr->currentLink->next;
    return itr->currentLink->value;
}

struct linkedListIterator {
    struct linkdList *lst;
    struct DLink *currentLink;
}
Assume you have following function:

```c
// removes lnk from the list it is part of
void _removeLink (struct DLink *lnk)
```

```c
void removeIter (struct listIter *itr)
{
    itr->currentLink = itr->currentLink->prev;
    _removeLink(itr->currentLink->next);
}
```

```c
struct linkedListIterator {
    struct linkdList *lst;
    struct DLink *currentLink;
}
```
void _removeLink (struct dlink *lnk)
{
}

frontSentinel

link

Link

db

backSentinel

Link
void _removeLink (struct DLink *lst)
{
    lnk->prev->next = lnk->next;
}

frontSentinel

Link

prev

next

lnk

prev

next

Link

prev

next

backSentinel

db
void _removeLink (struct DLink *lst) {

    lnk->prev->next = lnk->next;
    lnk->next->prev = lnk->prev;

}
void _removeLink (struct linkedList *lst)
{
    lnk->prev->next = lnk->next;
    lnk->next->prev = lnk->prev;
    free(lnk);
}