# Chapter 5:

### Question 37:

Implement an integer function in the client program that returns the number of items in a queue. The queue is unchanged.

**Answer:**

Here is my function. The complete code I used, including a test driver, is attached in a zip.

int GetLength(QueType& queue) {

QueType queue2;

int count;

ItemType item;

while(!queue.IsEmpty()) {

queue.Dequeue(item);

queue2.Enqueue(item);

count++;

}

while(!queue2.IsEmpty()) {

queue2.Dequeue(item);

queue.Enqueue(item);

}

return count;

}

# Chapter 6:

### Question 12:

Of the three variations sy linked lists (circular, with header and trailer nodes, and doubly linked), which would be most appropriate for each of the following applications?

a. You want to search a list for a key and return the keys of the elements that come before it and the keys of the two elements that come after it.

b. A text file contains integer elements, one per line, sorted from smallest to largest. You must read the values from the file and create a sorted linked list containing the values.

c. A list is short and frequently becomes empty. You want a list that is optimal for inserting an element into the empty list and deleting the last element from the list.

**Answer:**

a. Because you want to first find an item, and then return items both before and after the one that was searched for, using a doubly linked list will give you the simplest implementation.

b. In this case, the data is already sorted, so you only need to insert it into the list, and not re-sort it. In this situation, all three list types would work fine, but a circular linked list will have the smallest memory usage.

c. In this case you definitely want to use a list with trailer and header nodes, as it will give you the best performance on emptying and adding the first item into a list. This is because it doesn't need to check for empty list conditions, but can use the same code on an empty and partially filled list.

### Question 14:

Use the linked lists contained in the array pictured in Figure 6.19 to answer the following questions:

a. What elements are in the list pointed to by list1?

b. What elements are in the list pointed to by list2?

c. What array positions (indexes) are part of the free space list?

d. What would the array look like after the deletion of Nell from the first list?

e. What would the array look like after the insertion of Anne into the second list? Assume that before the insertion the array is as pictured in Figure 6.19.

**Answer:**

a. John, Nell, Susan, Susanne.

b. Mark, Naomi, Robert.

c. 7, 2, 3.

d.

Free: 4 list1: 0 list2: 1

|  |  |  |
| --- | --- | --- |
| Nodes | .info | .next |
| 0 | John | 8 |
| 1 | Mark | 5 |
| 2 |  | 3 |
| 3 |  | NUL |
| 4 | \*Nell | 7 |
| 5 | Naomi | 6 |
| 6 | Robert | NUL |
| 7 |  | 2 |
| 8 | Susan | 9 |
| 9 | Susanne | NUL |

\*I left Nell in because the data wouldn't actually be removed, although that address is now part of the free list.

e.

Free: 2 list1: 0 list2: 7

|  |  |  |
| --- | --- | --- |
| Nodes | .info | .next |
| 0 | John | 8 |
| 1 | Mark | 5 |
| 2 |  | 3 |
| 3 |  | NUL |
| 4 | Nell | 8 |
| 5 | Naomi | 6 |
| 6 | Robert | NUL |
| 7 | Anne | 1 |
| 8 | Susan | 9 |
| 9 | Susanne | NUL |