Ve281 Data Structures and Algorithms
Homework Two

This homework is announced on September 29th, 2012. It is due by 11:40 am on October 9th, 2012. The homework consists of three problems. Although this homework asks you to write code, it is a written assignment, not a programming assignment.

1. Remove a given node in a linked list.

   (a) Consider the class of the single-ended, singly linked list `LinkedList`, which we have talked in our lecture.

   ```
   class LinkedList {
       node *first;
   public:
       // Public operational methods and maintenance methods
       // (omitted here)
   };

   struct node {
       node *next;
       int value;
   };
   ```

   We have mentioned the following *member* function `removeNode` in our lecture. Please write the code to implement this method. Your implementation **cannot** call any other functions.

   ```
   void removeNode(node *n);
   // REQUIRES: "this" list is not empty and "n" is in the list
   // EFFECTS: remove the node n
   ```

(b) Implement the same `removeNode` function as above for a double-ended, doubly linked list `DList`. Your implementation **cannot** call any other functions. We require that the time complexity of your implementation is $\Theta(1)$. The definition of `Dlist` is:

```
class DList {
    node *first;
    node *last;
public:
    // Public operational methods and maintenance methods
    // (omitted here)
};

struct node {
    node *next;
    node *prev;
    int value;
};
```

(c) A *circular* linked list is one in which the next field for the last node of the list points to the first node of the list. This can be useful when you wish to have a relative positioning for elements. Suppose that we have a class of single-ended, singly linked circular linked list `CList` as follows:

```
class CList {
    node *first;
public:
    // Public operational methods and maintenance methods
    // (omitted here)
};

struct node {
    node *next;
    int value;
};
```

Implement the same member function `removeNode` as defined above for `CList`. Your implementation **cannot** call any other functions. We require that the time complexity of your implementation is $\Theta(1)$. Hint: for a linked list of objects, we only care about the order of the values stored in the linked list and the "first" pointer of the linked list.

2. Remove elements based on a given value.

   (a) Write the code to implement the following function with the minimal time complexity. Your implementation **cannot** call any other functions.

```
int removeKey(int a[], unsigned int size, int key);
// REQUIRES: "a" is an integer array of size as "size"
// MODIFIES: "a"
// EFFECTS: remove all the elements whose values are "key".
//          For example, suppose that a = {4, 2, -5, 2, 9}.
//          Calling removeKey function with key = 2 will change
//          the array a to {4, -5, 9}.
//          The function returns the size of the new array.
```

   (b) Suppose we have defined a similar function `removeKey` as a member function for the class of single-ended, singly linked list `LinkedList` defined in Problem 1(a). The specification of this function is

```
void removeKey(int key);
// MODIFIES: this
// EFFECTS: remove all the nodes in the list whose values are "key"
```

   Write the code to implement this member function with the minimal time complexity. Your implementation **cannot** call any other functions.

3. Overload the output operator `<<` for the single-ended, singly linked list `LinkedList` defined in Problem 1(a). However, we require this operator to print the elements in the linked list in the *reverse* order, separated by a single space. For example, if the list is

```
1 -> 2 -> 3,
```

   then the output is

```
3 2 1
```

   Write the code to implement this output operator. We require that the time complexity of your implementation is $\Theta(n)$, where n is the size of the linked list. If your implementation needs to call any functions, you should write these functions. If there are any changes required to the class definition of `LinkedList`, mention them.