Week 8 Section Handout

This week’s section handout has practice with Linked Lists and uses the following structs:

- **ListNode**
  ```c
  struct ListNode{
      int data;
      ListNode* next;
  };
  ```

- **ListNodeString**
  ```c
  struct ListNodeString{
      string data;
      ListNode* next;
  };
  ```

- **ListNodeDouble**
  ```c
  struct ListNodeDouble{
      double data;
      ListNode* next;
  };
  ```

1. **Merge**
   Write a recursive function `merge` that accepts two sorted linked lists and returns a pointer to a merged, sorted list. For example, suppose your function is given the following lists:
   ```c
   { 1, 4, 5, 10, 11 }   { 2, 4, 6, 7, 8 }
   ```
   A call to your function should return a pointer to the following list:
   ```c
   { 1, 2, 4, 4, 5, 6, 7, 8, 10, 11 }
   ```
   **Constraints:** Do not swap data values or create any new nodes to solve this problem; you must create the merged list by rearranging the links of the lists passed to your function. Do not use auxiliary structures like arrays, vectors, stacks, queues, etc., to solve this problem.

   ```c
   ListNode* merge(ListNode* a, ListNode* b) { ...
   ```

2. **Split**
   Write a function `split` that rearranges the elements of a list of integers so that all negative values appear before all of the non-negatives, with each group in the same relative order. For example, suppose a list stores the following values:
   ```c
   {8, 7, -4, 19, 0, 43, -8, -7, 2}
   ```
   After a call to your function, the list's contents would be:
   ```c
   {-4, -8, -7, 18, 7, 19, 0, 43, 2}
   ```
   Do not swap data values or create any new nodes to solve this problem; you must rearrange the list by rearranging the links of the list. Do not use auxiliary structures like arrays, vectors, stacks, queues, etc, to solve this problem.

   ```c
   void split(ListNode*& front) { ...
   ```

3. **RemoveAllThreshold**
   Write a function `removeAllThreshold` that removes all occurrences of a given double value +/- a threshold value from the list. For example, if a list contains the following values:
   ```c
   {3.0, 9.0, 4.2, 2.1, 3.3, 2.3, 3.4, 4.0, 2.9, 2.7, 3.1, 18.2}
   ```
   The call of `removeAllThreshold(front, 3.0, .3)` where `front` denotes a pointer to the front of list, would remove all occurrences of the value 3.0 +/- .3 from the list, yielding the following values:
   ```c
   {9.0, 4.2, 2.1, 2.3, 3.4, 4.0, 18.2}
   ```
   If the list is empty or the value doesn't appear in the list at all, then the list should not be changed by your function. You must preserve the original order of the elements of the list. **Note that we will be using the ListNodeDouble structure.**

   ```c
   void removeAllThreshold(ListNodeDouble*& front, double value, double threshold) { ...
   ```

Thanks to CS106B and X instructors and TAs for contributing problems on this handout.
4. Insert
Write a function named insert that accepts a reference to a ListNodeString pointer representing the front of a linked list, along with an index and a string value. Your function should insert the given value into a new node at the specified position of the list. For example, suppose the list passed to your function contains the following sequence of values:

   { "Katherine", "Julie", "Kate" } 

The call of insert(front, 2, "Mehran") should change the list to store the following:

   { "Katherine", "Julie", "Mehran", "Kate" } 

The other values in the list should retain the same order as in the original list. You may assume that the index passed is between 0 and the existing size of the list, inclusive.

Constraints: Do not modify the data field of existing nodes; change the list by changing pointers only. Do not use any auxiliary data structures to solve this problem (no array, Vector, Stack, Queue, string, etc).

```cpp
void insert(ListNode*& front, int index, string value) {...
```

5a. Reverse
Write a function reverse that reverses the order of the elements in a list of integers. For example, if a list initially stores the sequence of integers below at left, it should store the sequence at right after your function is called:

   {1, 8, 19, 4, 17}  ->  {17, 4, 19, 8, 1} 

```cpp
void reverse(ListNode*& front) {...
```

Bonus: This one also has an interesting recursive solution.

5b. Braid
Now, write a function braid that takes a linked list, and weaves the reverse of that list into the original. (In this case, you will need to create new nodes.) Here are a few examples:

   {1, 4, 2}  ->  {1, 2, 4, 4, 2, 1}

   {3}  ->  {3, 3} 

   {1, 3, 6, 10, 15}  ->  {1, 15, 3, 10, 6, 6, 10, 3, 15, 1} 

Bonus: This one also has an interesting recursive solution.

```cpp
void braid(ListNode*& front) {...
```

Thanks to CS106B and X instructors and TAs for contributing problems on this handout.