Homework 1: Linked Lists

Objective

In this programming homework you are being asked to implement the following data structures:

1. A *singly linked list*
2. A *doubly linked list*

and to carry out simple computational experiments. These exercises will help you to polish up your skills in *manipulating pointers* in the language `C/C++`.

Introduction

In this homework, singly linked lists and doubly linked lists are *internal linked lists* where each node contains a data item (type `int`) and pointer(s) to the other nodes of the same type. To build these linked lists you will need to implement the following classes:

SingleNode

An object of this class represents the node we use for building an *internal* singly linked list. You may like to include the usual member functions (e.g. constructors, destructors, accessors, mutators etc.) that you find appropriate.

SingleList

An object of this class represents a singly linked list. You may include the usual member functions (e.g. constructors, destructors, accessors, mutators etc.) and others that you find appropriate.

DoubleNode

An object of this class is the node we use to build an *internal* doubly linked list. Again, you may like to include the usual member functions (e.g. constructors, destructors, accessors, mutators etc.) and others that you find appropriate.

DoubleList

An object of this class represent a doubly linked list. You may include the usual member functions (e.g. constructors, destructors, accessors, mutators etc.) that you find appropriate. If you use the pseudocode given Cormen's text (CLRS) as a guide, please acknowledge it in your `readme.txt` file.

Note In this homework, the required operations must be implemented by modifying pointers. For example, to reverse a list, you must re-arrange the nodes in reverse order. Simply by swapping and/or modifying the data field of each node will not be accepted.

Experiments To practice pointer manipulations in the current setting, you are asked to implement the following operations:

1. Team of Four:

Given a linked list of numbers, it will rearrange the list so that, for each block in the list of size 4, the node will be sorted in ascending order, according to the data stored in these nodes. For example, in the singly linked list case, if the list is:

```
Head ->[ 2 ]->[ 3 ]->[ 1 ]->[ 6 ]->[ 8 ]-[ 5 ]->[ 7 ]->[ 13 ]->[ 0 ]->nil
```

then, the resulting list, after performing the operation, is:

```
Head ->[ 1 ]->[ 2 ]->[ 3 ]->[ 6 ]->[ 5 ]->[ 7 ]->[ 8 ]->[ 13 ]->[ 0 ]->nil
```

To present your results, you may print the list in the following format to the screen:

```
***** Before the operation *****
2, 3, 1, 6,
8, 5, 7,13,
0

***** After the team of four operation *****
1, 2, 3, 6
5, 7, 8,13
0
```

2. Reverse:

Given a linked list of numbers, it will rearrange the list so that the numbers will be in reverse order. For example, in the singly linked list case, if the list is:

```
Head ->[ 2 ]->[ 3 ]->[ 1 ]->[ 6 ]->[ 8 ]-[ 5 ]->[ 7 ]->[ 13 ]->[ 0 ]->nil
```

then the resulting list, after performing the operation, is:

```
Head ->[ 0 ]->[ 13 ]->[ 7 ]->[ 5 ]->[ 8 ]-[ 6 ]->[ 1 ]->[ 3 ]->[ 2 ]->nil
```

To present your results, you may print the list in the following format to the screen:

```
***** Before the operation *****
2, 3, 1, 6,
8, 5, 7, 13,
0

***** After the team of four operation *****
0, 13, 7, 5,
8, 6, 1, 3,
2
```

3. Shuffle:

Given a linked list of numbers, it will shuffle the first half of the list with the second half of the list so that the nodes will be listed from each half alternatively. For example, in the singly linked list case, if the list is:

```
Head ->[ 2 ]->[ 3 ]->[ 1 ]->[ 6 ]->[ 8 ]->[ 5 ]->[ 7 ]->[ 13 ]->[ 0 ]->nil
```

then the resulting list, after the operation, is:

```
Head ->[ 2 ]->[ 5 ]->[ 3 ]->[ 7 ]->[ 1 ]->[ 13 ]->[ 6 ]->[ 0 ]->[ 8 ]->nil
```

Note that when the number of elements in the list is odd, say $n = 2k+1$, the first half consists the first $k+1$ elements and the second consists of the remaining k elements.

To present your results, you may print the list in the following form to the screen:

```
***** Before the operation *****
 ** first half **
2, 3, 1, 6,
8
 ** second half **
5, 7, 13, 0


***** After the shuffle operation *****
 ** the list **
2, 5, 3, 7,
1, 13, 6, 0,
8
```

Results Your test program should test each of the above operations for both singly linked lists and doubly linked lists and print the results to the screen. In particular, you need to complete the following 12 test cases:

Test Case no.	Data Structures	Operation Being Tested	initial data sequence
1	Singly linked list	Team of Four	S_1
2	Doubly linked list	Team of Four	S_1
3	Singly linked list	Team of Four	S_2
4	Doubly linked list	Team of Four	S_2
5	Singly linked list	Reverse	S_1
6	Doubly linked list	Reverse	S_1
7	Singly linked list	Reverse	S_2
8	Doubly linked list	Reverse	S_2
9	Singly linked list	Shuffle	S_1
10	Doubly linked list	Shuffle	S_1
11	Singly linked list	Shuffle	S_2
12	Doubly linked list	Shuffle	S_2

The sequence S_1 and S_2 are defined as follows:

$S_1 : 1, 2, \ldots, 100$; S_2 : a pseudo-random sequence of *distinct* numbers of length 100.

Due Date

Upload your submission on or before 10:00PM (EST Time), 9/30/2016. 24 **hour extension rules apply** (See Syllabus for the details). For those who met this criteria, and if the final submission's received after 10:00PM (EST Time), 10/1/2016, the penalty is 25 percent off per day late.

Submission

Your submission will contain a collection of `C++` header files (`.h` files), implementation files (`.cpp` files) and the main program (named as `main.cpp`). They are expected to have adequate comments.

In addition, you will need to submit two text files:

i. `readme.txt`:

it should contain a concise description of how the two data structures (i.e. singly and doubly linked list) work. Include a description regarding your implementation of the three operations: Team of Three, Reverse and Shuffle and evidence that your code implement the required operationsby modifying the pointers. Acknowledge the reference(s) you use, if any, in this file.

ii. `output.txt`: it should contain a sample execution result in the `ubuntu/g++` environment. This file can be obtained by

```
ubuntu>g++ *.cpp -o hw1
ubuntu>./hw1 > output.txt
```

It should be in a single zip file. It should be named via the following convention:

```
<SU-EMAIL>-<FIRST-Name>-HW1.zip
```

That is, if my SU email address is `abc111@syr.edu` and my first name is Andrew, then I should name my submission as

```
abc111-andrew-HW1.zip
```

Additional Requirement Relevant information regarding the submission process will be posted within our blackboard site or via our Piazza forum.