Faculty of Arts and Sciences
Department of Computer Science
CMPS 212 – Intermediate Programming with Data Structures
Assignment 2
Due Date: Sunday June 26 at midnight

Problem 1 – Singly Linked List

a) Write a method **removeEveryOther**() for the LinkedIntList class that removes every other node in the list. If the list contained [11,4,34,67,5], then a call to *removeEveryOther* would result in [11,34,5]. If the list is empty or there is only a single node, the method does nothing and simply returns. You may NOT create any new nodes or overwrite the data variable of any node. You must solve this problem by manipulating the next variable in the ListNode class. Do NOT use recursion.

b) Write a method **compress()** that could be added to the LinkedIntList class that accepts an integer *n* representing a "compression factor" and replaces every *n* elements with a single element whose data value is the sum of those *n* nodes. Suppose a LinkedIntList variable named *list* stores the following values:

[2, 4, 18, 1, 30, -4, 5, 58, 21, 13, 19, 27]

The call to list.compress(2) gives the following result:

[6, 19, 26, 63, 34, 46]

since (2 + 4 = 6, 18 + 1 = 19, 30 + (-4) = 26, ...)

If the previous call is following by a call to list.compress(3) replaces every three elements with their sum (6 + 19 + 26 = 51, 63 + 34 + 46 = 143) resulting in the following list:

[51, 143]

If the list's size is not an even multiple of *n*, whatever elements are left over at the end are compressed into one node. For example, the original list contained 12 elements. Therefore, a call to list.compress(5) compresses every five elements, (2 + 4 + 18 + 1 + 30 = 55, -4 + 5 + 58 + 21 + 13 = 93), with the last two as leftover elements. The latter will then be compressed alone (19 + 27 = 46), resulting in the following list:

[55, 93, 46]

If *n* is greater than or equal to the list size, the entire list compresses into a single element. If the list is empty, then the result is an empty list. You may assume that the value passed for *n* is >= 1.

For full credit, you may not create any new ListNode objects, though you may have as many ListNode variables as you would like. For full credit, your solution must also run in the order of O(*n*).

Problem 2 - Doubly Linked List

Add the following functions to the doubly linked list class of integers we saw in class:

1. *public void* ***reverse***(){… }; which reverses the order of the nodes in a given doubly linked list (Without allocating any extra space!) For example, after calling the reverse method, the list [3 ⇄ 9 ⇄ 5 ⇄ 1] becomes [1 ⇄ 5 ⇄ 9 ⇄ 3]
2. *public void* ***remove***(){… }; removes the front node of the doubly linked list.

3. $public\ void\ \boldsymbol{clean}()\{\dots\}$; given a non-sorted doubly linked list of integers, finds and deletes all the duplicates from the list. For example, after calling the $clean$ function, the following list [1 ⇄ 5 ⇄ 9 ⇄ 5 ⇄ 1 ⇄ 3 ⇄ 3 ⇄ 1 ⇄ 1] becomes [1 ⇄ 5 ⇄ 9 ⇄ 3]
4. $public\ void\ \boldsymbol{merge}(DNode\ head)\{\dots\}$; merges two sorted doubly linked lists of integers into one, verifying the following:
    a. The merged list must remain sorted
    b. No node allocation or deletion is allowed.
    c. Only one iteration is allowed for each list.
    d. The function must return the head of the resulting merged list.
5. Add a main method and create a list to test the above functions.