Cairo University
Faculty of Engineering
Data Structures & Algorithms

Data Structures and Algorithms Linked List Assignment

Spring 2021

Regulations & Hints

1. In this assignment you are required to submit **one cpp file**, "**code.cpp**", that includes all the requirements of this assignment.
2. You are provided a partially implemented "code.cpp" file that includes the node class and the partially-implemented linked list class and main function. You should complete the implementation of this file to be exactly the same as specified in the requirements section.
3. **Code is corrected automatically, so you should pay attention to print the output in the same required format, in order not to lose marks. All printed outputs are case-sensitive.**
4. **Compilation errors result in zero grade.**
5. **Complete the main function exactly the same as described below because implementing a function without writing its part in the main function will make you lose this function's grade.**
6. **[Important]** Don't use any cin or cout in the code (except in **["choice" equals 1]** part in the main function and must be exactly the same as required). Whenever you want to read or print linked list elements, just use:

- L.PrintList ( ) when you want to print the elements of linked list L to the user.
- L.ReadList ( ) when you want to read the elements of linked list L from the user.
- **Note**: you must NOT change the implementation of these functions.

7. **[Important]** Don't add system("pause"), cin.get() or anything that halt the execution at the end of your code.
8. For numeric values: integers only are allowed. float and double types aren't allowed.
9. Random submissions may be chosen for discussions or re-evaluation.
10. Take care of corner cases because the judging tool will contain many hidden test cases (don't check on only the test cases mentioned in the document).
11. You can "**pre-check"** it multiple times using the automatic judge platform. Pre-checking means it will be executed against examples and will give you feedback whether your solution is correct or not for a small subset of test cases.
12. **"Check" button means submission. The first 2 submissions are considered penalty free. Third submission will be penalized 50%. Fourth submissions and beyond will be zero graded.**
13. Suggested steps:
    a. Start working locally on your machine until you are sure that the flow of your program is correct. You can test on the given examples.
    b. Pre-check your code to make sure it works correctly on the platform.
    c. After finishing your work and being sure nothing else can be added, submit it.
14. **Cheating will cause you to take ZERO grade in the assignment and deducting more grades from other coursework grades for ALL involved students.**

Requirements

Notes:

- All the following required Linked List functions are **member functions**.
- You should keep the sequence of elements of the input linked lists unless specified otherwise (Don't deal with the linked list as a Bag in this assignment).
- Assume that you cannot add an element of value (-1) in your linked list because L.ReadList() takes the value (**-1**) to indicate the **end of linked list**.
- Each "Sample Input" in the functions below contain an **integer** at the beginning of it indicting the number/id of the function to check (this integer is the "**choice**" integer mentioned in the main function – read the main function)

1- Write a function called "**countAnomalies**" that prints the count of anomalous elements in the linked list. An element in the linked list is said to be anomalous if it is greater than the **sum** of the two elements before it. If the element has fewer than two elements before it, it is NOT considered an anomaly.

Sample Input	Sample Output	Notes
1 10 12 10 **25 90** 4 3 **20** -1	3	25 > (12+10) 90 > (10+25) 20 > (4+3)
1 1 2 **5 9** 14 4 **20** 9 **91** -1	4	5 > (1+2) 9 > (2+5) 20 > (14+4) 91 > (20+9) Note that 14 is not greater than 5+9.
1 10 50 -1	0	Note that the first two element are not considered anomalies .

2- Write a function called **insertBeforePositive** that inserts before every positive element in the linked list its negative value. You should modify the input list itself (don't return the result in a new linked list).

Sample Input	Sample Output	Notes
2 -10 **2** -5 **6 7** -1	-10 **-2** 2 -5 **-6** 6 **-7** 7	Here the added elements are: -2, -6 and -7
2 **10** -2 **3** -1	**-10** 10 -2 **-3** 3	Here the head changed to point the added (-10) at the beginning.
2 **10 2 3** -1	**-10** 10 **-2** 2 **-3** 3	All positive values, so insert a node before each element.
2 -10 -2 -1	-10 -2	No positive values, so the output is the same as original.

3- Write a function called "**deleteSecondLargest**" that deletes the second largest **distinct** element in the Linked List. If there is no largest second element, you shouldn't do anything. You should modify the input list itself (don't return the result in a new linked list).

(Hint: One case where there is no second largest element is when the Linked List has only one distinct element)

Sample Input	Sample Output	Notes
3 19 23 3 **56** 78 -1	19 23 3 78	78 is the largest element and 56 is the second largest element, so we deleted 56.
3 1 2 5 20 20 **12** 20 **12** -1	1 2 5 20 20 20	20 is the largest element and 12 is the second largest distinct element, so we deleted all occurrences of 12. Note how we handled duplicates here.
3 7 -1	7	No second largest element found, so we did nothing.
3 4 4 4 4 4 -1	4 4 4 4 4	No second largest element found, so we did nothing.

4- Write a function called "**reverseHalf**" that reverses the first half of the linked list. If the linked list has an odd number of elements, consider the middle element to be part of the first half. You should modify the input list itself (don't return the result in a new linked list).

Sample Input	Sample Output	Notes
4 1 2 3 4 5 6 -1	3 2 1 4 5 6	We reversed the first 3 elements and left the other 3 in the same order.
4 1 2 3 4 5 -1	3 2 1 4 5	Note how we handled the middle element when the list had odd number of elements.
4 4 5 -1	4 5	The first half of the list is just one element. The reverse of a list with one element is itself.
4 12 -1	12	The reverse of a list with one element is itself.

5- Write a function called **sortedOddsAndEvens** that takes two input lists *sorted* ascendingly (the calling linked list and one linked list passed as function parameter) then the function modifies the first linked list to contain all odd value elements of the two input linked lists and modifies the second linked list to contain all even value elements of the two input linked lists. The two linked lists after modifications must be **sorted** ascendingly too. (Try not to use a sort function to sort the linked list as a whole)

Note: Assume the two input linked lists are already entered sorted (no need to check).

Sample Input	Sample Output	Notes
5 1 4 6 7 9 -1 2 3 5 7 11 12 -1	1 3 5 7 7 9 11 2 4 6 12	Note that duplicate elements are not removed and the output lists are sorted.

5 4 5 7 7 8 10 -1 3 4 5 7 9 11 -1	3 5 5 7 7 7 9 11 4 4 8 10	Note that the head of the two linked lists pointed to a different node than original.
5 2 4 6 -1 1 3 5 -1	1 3 5 2 4 6	Here the linked lists are totally swapped.
5 -1 5 -1	5	Here the first linked list is empty and the second linked list contains one odd value, so the first linked list becomes pointing to this odd value and the second linked list becomes empty

☐ Complete the partially-implemented "main" function provided in "code.cpp" file to work as follows:

- Reads an integer "**choice**" from the user indicating the function number he wants to test.
- **If "choice" equals 1, then make the following (in order):**
  a) Create an object of Linked list of integers called L1
  b) Call L1.ReadList()
  c) Call the function **countAnomalies** on L1 and save the count (return value) in a variable called ca;
  d) Assuming you saved the count in a variable called ca, print the count using "std::cout<<ca;" (Use the exact format – Don't add a new line or space) → **this is the only cout allowed for you to use in your code.**
- **If "choice" equals 2, then make the following (in order):**
  a) Create an object of Linked list of integers called L1
  b) Call L1.ReadList()
  c) Call the function **insertBeforePositive** on L1
  d) Call L1.PrintList()
- **If "choice" equals 3, then make the following (in order):**
  a) Create an object of Linked list of integers called L1
  b) Call L1.ReadList()
  c) Call the function **deleteSecondLargest** on L1
  d) Call L1.PrintList()
- **If "choice" equals 4, then make the following (in order):**
  a) Create an object of Linked list of integers called L1
  b) Call L1.ReadList()
  c) Call the function **reverseHalf** on L1
  d) Call L1.PrintList()
- **If "choice" equals 5, then make the following (in order):**
  a) Create two objects of Linked list of integers called L1 and L2
  b) Call L1.ReadList()
  c) Call L2.ReadList()
  d) Call the function **sortedOddsAndEvens** using L1 as the calling object and pass L2 to it as parameter
  e) Call L1.PrintList()
  f) Call L2.PrintList()