CSc 111 Assignment 7

Learning Outcomes:

When you have completed this assignment, you should understand:

- How to define functions that take pointers to a list structure

Getting started

1. Download the provided file called `assignment7.c` to your computer. Drag the file from the downloaded location and drop/upload into Jupyter Hub.
2. Design the functions according to the specifications.
3. Test to make sure your code produces the expected output.
4. Ensure that the filename is `assignment7.c` (files with an incorrect name will not be marked).
5. Download your assignment7.c file from Jupyter Hub (File→Download) and submit this single file to the Assignment 7 dropbox (under Assignments) on the CSC111 BrightSpace Page.
6. Download your file from BrightSpace to confirm that name and file contents are as expected.

Reminder: Your code is to be designed and written by only you and not to be shared with anyone else. See the Course Outline for details explaining the policies on Academic Integrity. Submissions that violate the Academic Integrity policy will be forwarded directly to the Computer Science Academic Integrity Committee.

Grading:

1. Late submissions will not be accepted
2. You must submit a single file with the name assignment7.c or you will be given a zero grade.
3. Your function names must match the specification exactly or you will be given a zero grade.
4. Any code written in the main function may not be marked.
5. We will do spot-check grading for code quality. All assignments are graded BUT only a subset of your code may be graded for code quality. You will not know which portions of the code will be graded, so all of your code must be complete and adhere to specifications to receive marks.
6. Your code must compile and run, as submitted, without errors in the Computer Science department's Jupyter Hub environment using the compilation command provided in this document.
7. We reserve the right to replace your main function with our own, allowing it to better interact with our testing software.
8. It is the responsibility of the student to submit any and all correct files. Only submitted files will be marked. Submitting an incorrect file or forgetting a file is not grounds for a regrade.

Marks will be given for…

- your code producing the correct output
- your code following good coding conventions
    - Proper indentation
    - Documentation
    - Proper use of variables to store intermediate computation results
    - Proper use of helper functions

Compiling and Running

Use the command: `gcc -Wall -Werror -std=c18 -o assignment7 assignment7.c` and the command: `./assignment7` to run your code.

Function Specifications

IMPORTANT: Be careful with your choice of types! All floating-point return types and floating point arguments types **MUST** be of type `double`. When the return type or argument type is clearly not a fractional number, use you must use the type `int`.

Your function MUST accept the parameters in the order they are described within the function specification!

1. Design a function called `add_front` that takes a pointer to a list of integers and adds another element to the front of the list.
2. Design a function called `get_last` that takes a pointer to a list of integers and returns the value of the last element in the list.
    - If the list is empty, return -999.
3. Design a function called `remove_first` that takes a pointer to a list of integers and removes the first element of the list. Remember to free any previously allocated memory.
4. Design a function called `sum_squares` that takes the following arguments in this order: a pointer to a list of integers and returns the sum of the squares of all the values in the list.
    - **For example**: If the list contains the values: 2, 4 and 3 the function should return the sum of 4, 16 and 9 which is 29.
    - If the list is empty, return -999.
5. Design a function called `add_to_all` that takes the following arguments in this order: a pointer to a list of integers and an integer value as an additional argument. The function should update the list by adding the additional argument value to every element in the list.
    - **For example:** If the list contains the values: 2, 4, 3 and the additional argument is 5 when the function returns, the list should contain the values 7, 9, 8.
6. Design a function called `get_all_below` that takes the following arguments in this order: a pointer to an input list of integers, a pointer to an empty result list of integers and a threshold value as an integer. The function should add to the destination list new elements using only the values from the input list that are below the given threshold in the order they appear in the input list. The function should return the number of new elements that were added to the result list. The input list should remain unchanged.
    - **For example:** If the input list contains the following values: 2, 3, 5, 6, 1 in this order and the threshold is 4, after the function executes the result list will contain the values: 2, 3, 1 in this order and the function will return the value 3.
7. Design a function called `are_all_above` that takes the following arguments in this order: a pointer to a list of integers and an integer value as an additional argument. The function should return 1 if all elements in the list are bigger than the additional argument value, and 0 otherwise.
8. Design a function called `does_contain_multiples_of` that takes the following arguments in this order: a pointer to a list of integers and an integer value as an additional argument. The function should return 1 if any elements in the list are multiples of the additional argument value, and 0 otherwise.
    - Multiple is defined here: https://www.mathsisfun.com/numbers/factors-multiples.html#:~:text=Multiples%3A,is%20a%20multiple%20of%206
9. Design a function called `count_if_contains_multiples` that takes the following arguments in this order: a pointer to a list of integers, a pointer to a second list of integers. The function should return a count of the number of elements in the second list for which the first list contains multiples.
    - **For example**: If the first list contains the values {9, 7, 18, 12, 21} and the second list contains the values {12, 5, 14, 3} the function should return 2 since looking at each element in the second list:
        i. a multiple of 12 is in the first list and a multiple of 3 is in the first list
        ii. the first list does not contain any multiples of 5 and 14

Note: The lists are not guaranteed to be the same length, either the first list or the second list could be longer and either of them could be empty!

You should be making use of at least one of the previous functions you wrote as a helper function!