BITS Pilani, Pilani Campus
2nd Semester 2017-18
CS F211 Data Structures & Algorithms
Lab Test 1 - (18th Feb 2018)

Maximum Marks: 30 + 30 = 60 **Duration: 160 + 10 = 170 min**

Instructions

- In this lab test you have to solve two problems given in two different files. Total time for solving the problems is 160 minutes with at most 10 additional minutes for uploading the solution on the portal. Portal closing timer is shown at top-right corner of your page.
- You may make multiple submissions on the online portal, but only the latest submission shall be considered for evaluation.
- For each of the given two problems, upload all relevant source code (*.c) and header (*.h) files as a single submission. Do not upload any other file.
- If your program is having a "*compilation error*" or "*segmentation fault*" or other similar errors, we shall **evaluate your code for 50% of the marks only**. So, try to keep your code as clean as possible. Also note that **commented code** will **NOT** be considered for evaluation.
- Each problem is associated with a zip file which contains a "**main.c**" file and a few additional files:
  - DO **NOT** make any changes in the file - "**main.c**" in either of the problems.
  - Write your code in the provided additional files only. Do NOT rename any file, or create a new file.
  - Please note that each function may be tested separately for various test cases. So, don't make changes to the signatures and return types of the pre-defined functions.

Problem 1 of 2

[Expected Time: 75 minutes.]

An array of student records `studArray`, where each record is a tuple – {<`name`>, <`marks`>} is defined in the file named *main.c*, where <`name`> is a string of size 10 and <`marks`> is a *float* value. You can use the files in **q1.zip** and build your code over them. **DO NOT** make any changes to the **structure definitions** and **signatures** and **return types** of the predefined functions in these files. However, you may add more functions as required.

a. Implement the function `createList`(), which takes an array of student records along with its size, creates a linked list of records contained in the array and returns the list. This function **must** execute in O(n) time, where n is the size of the array. **[5M]**

b. Implement the function `insertInOrder`(), which takes a sorted linked list and a list node containing a new student record and inserts it into the list at its appropriate place such that the list remains sorted in *ascending order* by <`marks`>. **[8M]**

c. Implement the function `insertionSort`(), which takes an unsorted list and sorts it using the `insertInOrder`() function and returns the sorted list. **[8M]**

d. Implement a function `measureSortingTime`(), which takes an unsorted list and sorts it by calling insertionSort() and returns the time taken for the for the execution of `insertionSort`() in *milliseconds*. **[4M]**

e. Create your custom `myalloc`() and `myfree`() functions to replace calls to native malloc() and free() functions to profile the heap memory. The total memory allocated at any given time should be stored in a global variable known as "`globalCnt`". **[5M]**