Data Structure Lab Assignment (CS 2172)
Assignment 5: Singly Linked List

Time: 2 weeks

1. Use of linked list to maintain list of integers

Let integers to be stored in a linked list(s) whose nodes are of the following type.

```
struct intNode {
    int element;              /* stores the integer value (positive/negative/zero) */
    struct intNode *next;     /* points to the next node in the liked-list */
};
```

It is understood that in a single program multiple such linked lists can be maintained by having variables like **s1, s2**, etc, defined as "**struct intNode *s1, *s2;**" where **s1** and **s2** point to the first nodes of two linked lists, respectively.

You have to write functions as stated below for maintaining linked lists whose nodes are of type "**struct intNode**" and **nodes are in ascending orde**r of the field "**element**". That is, the following functions work on such sorted linked and also leaves the lists in sorted state.

Write a function "***struct intNode *addIntNodeSorted(struct intNode *start, int elem)***" that allocates (***malloc***s) a new node, copies the contents of the parameter "elem" in the newly allocated node and adds the node to the linked list referred to by "***start***" (that is, whose 1st node is pointed to by the parameter ***start***). **The function returns the address of the 1st node of the resulting linked list.** Please note that in this case the pointer to the 1st node of a linked list changes only when the new node being added becomes the 1st node. That is, either the new node is being added to an empty list or the new node comes first in the prescribed order.

Write a function "***int isPresentIntNodeSorted(struct intNode *start, int elem)***" that searches for presence of a node whose "**element**" matches with "**elem**" (the 2nd parameter of the function). If such a node is found, the function returns its position in the linked list, otherwise (such a node is not there) the function returns 0.

Write a function "***struct student *deleteIntNodeSorted(struct intNode *start, int elem)***" that delete the node of the linked list referred to by "***start***" whose "**element**" field matches with the parameter "**elem**". The deleted node must be freed too. The function returns the address of the 1st node of the resulting linked list. Please note that in this case the pointer to the 1st node of a linked list changes only when the 1st node of the linked list gets deleted.

Write another function "***void printLL(struct intNode *start)***" that prints the integer information in the following format.

```
Start -> 2 -> 5 -> 11 -> 12 -> NULL
```

Write a main() function to demonstrate that the above-specified functions are working as desired.

2. **Management of the integer linked list (Once problems 1 is completed, then attempt this problem)**

   - Write a function "***struct intNode * freeIntNodeSorted(struct intNode *start)***" that frees all the nodes of a linked list.

   - Write a function "***struct intNode * reverseIntNodeSorted(struct intNode *start)***" that reverse all the nodes of a linked list. At this point list is sorted in descending order. Note that you must not use any temporary storage (like array or other variables) to do this. Do it without swapping data.

   - Write a function "***void updateIntNode(struct intNode *start)***" that randomly adds a value (between 0 to 9) to each element of the list. At this point of time the list is not sorted anymore.

   - Write a function ***struct intNode * sortIntNode (struct intNode *start)*** that sorts back all the nodes of a linked list in ascending order of elements. Note that you must not use any temporary storage (like array or other variables) to do this. Do it without swapping data.

   - Then write a function "***struct intNode * mergeSNodeSorted(struct intNode *start1, struct intNode *start2)***" that merges both the lists and returns the newly created list. The resulting list also should be in the sorted order with respect to the "element" and finally one list should exists. Note that you must not allocate any new node through malloc.

     Note: In the function "***mergeIntNodeSorted***", try to propose a mechanism that sets both ***start1*** and ***start2*** to NULL, this is because after the completion of merging, both the input lists will become empty.