Department of Computer Science and Engineering
Assignment 10
Subject : Programming and Data Structure (CS19003)

Instructions:

- Give the name of the programs files as <Your roll>_<assignment number>_<question number>.c. For example, 21XX12345_A1_Q1.c for question 1 of assignment 1 of a student with roll 21XX12345. **Follow the file naming convention strictly**.
- Apart from the main .c file for each program, you should also upload one additional temporary .c file for each program (such as when you have finished half of the code). The naming for the temporary file should be in the format <Your roll>_<assignment number>_<question number>_temp.c. For example, 21XX12345_A1_Q1_temp.c **Make sure that your main code do not deviate much from its temporary code for each program**.
- You should upload the main .c file and the temporary .c file individually to the Moodle course web page once you finish answering each question. No need to zip the files.
- The **deadline** to upload the programs is 12:00PM. Beyond that, submission will be closed (No extensions will be granted).
- If you do not follow the instructions, your marks may be deducted.

[Total Marks = 100 (30 + 30 + 40)]

Answer all questions

1. In a doubly linked list, each node is given two pointers, the first pointing to the next element in the list and the second to the previous element in the list. The next pointer of the last node and the previous pointer of the first node should be NULL. Describe a suitable C data type to store a node in a doubly linked list. Suppose that each node in the list stores an integer key value. Now, consider a **sorted doubly linked list of integers in ascending order**.
   - Write a C function to insert an integer 'x' in 'L' in the sorted order. If x is already present in L, then the insertion produces no change. If x is absent from L, it is inserted in the appropriate position so that after the insertion, L continues to remain sorted.
   - Write a C function to return the **minimum search distance** for a particular element within L. The minimum search distance for an element 'x' within 'L', if 'x' is present in 'L', refers to the minimum of the distances from the front and rear end of the list to the particular element 'x'. For example, if the list contains (2,4,6,8,9,10), then the minimum search distance of element '9' is 2, as distance from front end to '9' = 5 and distance from rear end to '9' = 2, and min(2,5) = 2. If 'x' is not present in 'L', then search distance returns -1. If there are duplicate elements 'x' within 'L', then the first occurrence of 'x' from both the sides should be considered as the respective distance and the minimum of that is returned. If the distance from both the left end and right end is the same (i.e the element is in the middle of an odd length list), then just that distance should be returned. Note that min(a,a) = a only. The search element should be taken as input from the keyboard and passed to the function as one of its arguments. You **HAVE** to use the concept of doubly linked in your logic for this function.

- Write a function to print the sorted linked list starting from the head node. If the list is empty, then print ‘List empty’.

  For each of the above mentioned functions, write the function name, function definitions and function arguments as per your choice, but make sure that the functions perform the tasks as specified.and one of the arguments must be the head pointer of the linked list. In main(), write a menu-driven code that provides the user with four choices, insertion, finding minimum search distance, printing the list and exiting the program. Based on the user choice, the appropriate function is called. The menu is run continuously until the user wishes to exit. For insertion and search distance, the respective element is also taken as input. The linked list initially starts empty, with 0 number of nodes. **[30]**

2. Define a linked list where the head node sits in the middle and you can go either to the left or right of the node depending on the user input. The head node contains two pointer variables left and right to traverse either to the left or right of the node. All other nodes contain either the left pointer or the right pointer. Write a program to create such a linked list by inserting new nodes either to the left or to the right of the head node. Logically, the list looks like as shown below,

$$\ldots \leftarrow A \leftarrow B \leftarrow HEAD \rightarrow C \rightarrow D \rightarrow \ldots$$

   Write a C program to perform the following tasks.

   - Let the user choose continuously from the keyboard (until ‘-1’ is pressed) the new element to be added and which direction the new insertion should happen.
   - Depending on the choice, create a new node containing either the left or right pointer.
   - Start from the head and traverse in the appropriate direction till the end of the respective left/right sub chain to insert a new node there.
   - At the end display both the left sub-chain and the right sub-chain.

   You should write two functions ***insertLeft(node* head, int data)*** and ***insertRight(node* head, int data)*** that inserts a new node either to the left or right of the head respectively. You can assume that the nodes contain only integer data. You can further assume that ’1’ represents left and ’0’ represents right as choice during user input.

   Example:

```
Enter data: 10
Enter direction: 1

Enter data: 20
Enter direction: 0

Enter data: 30
Enter direction: 0
```

Enter data: 40

Enter direction: 1

Enter data: 50

Enter direction: 0

Enter data: -1 [NOTE: '-1' is the exiting condition for taking further inputs]

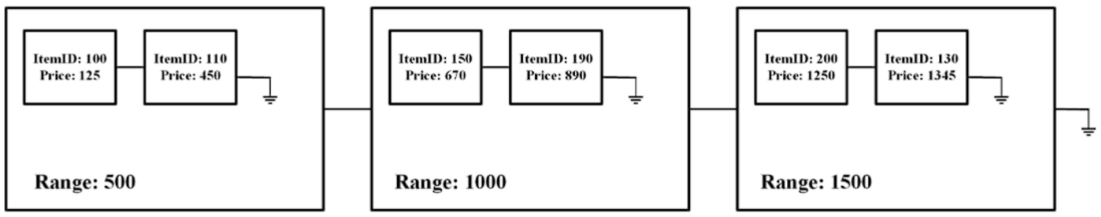
Output:

Left linked list: 10 → 40 → NULL

Right linked list: 20 → 30 → 50 → NULL

[You have to print the arrow heads and the word 'NULL' to clearly represent the list] **[30]**

3. You must have done online shopping where you can filter different items based on a number of parameters, one such being sorted as per the range of price of the item, i.e whether the price lies in the range, say, (0-500), then (501-1000), (1001-1500), (1501-2000) and so on. We can think of each range to be a bucket and within each bucket all the items whose price falls in the respective range are kept. In order to build such a filtering system, we can take the help of nested linked lists. Consider the following diagram below.



As per the diagram shown, there is one outer linked list, and within each node of the outer list there is an inner linked list. Each node of the outer linked list denotes a bucket as per the price range in increasing order, starting from 0. It holds an integer variable 'Range' which denotes the upper limit of the price range for that bucket, the lower limit starting from after the previous bucket's range. Overall the price starts from 0 (lower limit for the first bucket) and each successive bucket has a range of 500 units of currency. That is, if the value of 'Range' is 1000 for a bucket, this means that bucket holds all the items whose prices fall in the range 501 to 1000. The inner linked list denotes the list of all the individual items whose price falls within the range for that particular bucket. Each node of the inner list consists of two integer variables, 'ItemID' denoting the numerical ID of the item and 'Price', denoting the price of that particular item. Within each bucket, the inner list of items is also sorted in increasing order of their prices.

Write a C program to build such a filtering system.

- Define the suitable structure variable that can form such a nested linked list. In particular, the structure variable itself should be a nested structure to incorporate the outer and inner lists.
- Initially, the entire list is empty, i.e no elements are present. Write a function ***void insert(Bucket *head ,int itemID, int Price)***, to insert a new item into the linked list as per the filtering order, where 'Bucket *head' points to the head of the outer linked list, 'itemID' refers to the ID of the particular item and 'Price' refers to the individual price of that item. You must first locate the appropriate bucket node of the outer list where the

item has to be kept, then within that bucket you have to insert that item in the correct position of the inner list. If the appropriate bucket node is not present within the current list, then add a new bucket node that can hold that item, along with any intermediate bucket nodes that may be required. This is required because the buckets have a range of 500 only. The intermediate bucket nodes can hold an empty inner list, if no items for that bucket are present. Keep in mind that both the outer and inner lists are sorted in ascending order, and must remain sorted after each insertion.

- From main(), take input the itemID and price of 'n' such items and insert them one by one within the filtered list by calling insert(). You may assume that for all items, minimum price is 0 and maximum price is 5000.
- Finally after taking input is complete, print the entire list, as per the format below.

Example sample output (refer to the diagram):

```
Bucket 1:
Lower limit: 0
Upper limit: 500
Items:
(100,125) [NOTE: The item tuple is of the form (itemID,Price)]
(110,450)

Bucket 2:
Lower limit: 501
Upper limit: 1000
Items:
(150,670)
(190,890)

Bucket 3:
Lower limit: 1001
Upper limit: 1500
Items:
(200,1250)
(130,1345)
```

[40]