# CS19001: Programming and Data Structures Laboratory

**Lab Test – 1 (EVEN-PC)**

**Date: 06-September-2019**

### Problem Statement A: [ *Cutting-Canals* ] Marks: 35

Let a set of intervals is defined as, $\mathcal{I} = \{[a_1, b_1]; [a_2, b_2]; \ldots; [a_n, b_n]\}$, where $a_i \le b_i$, when $1 \le i \le n$ and $b_i < a_{i+1}$, when $1 \le i < n$ (both $a_i$ and $b_i$ are integers within range $[-10^9, 10^9]$ and $n \le 10^5$). This indicates that each of the intervals are non-overlapping with the others and enjoys a monotonically increasing nature.

You are given with a set of $n$ existing intervals, $\mathcal{E}$, and another set of $m$ new intervals, $\mathcal{I}$, that you wish to delete in the existing interval set. You have to write a C-program that generates the final interval set, $\mathcal{F}$, by deleting $\mathcal{I}$ from $\mathcal{E}$. While deleting $\mathcal{I}$ into $\mathcal{E}$, you must consider the overlapping (as well as subsuming) nature of the intervals and produce the refined final set of intervals (following the above definition).

Your C-program needs to carry out the following steps:

- Take from user as input the number of existing intervals, i.e. $n$ (an integer).
- Take from user each of the start ($a_i$) and end ($b_i$) values of these $n$ existing intervals. Keep these into two arrays (one for keeping the start values and the other for the keeping the end values).
- Take from user as input the number of new intervals, i.e. $m$ (an integer).
- Take from user each of the start ($a_i$) and end ($b_i$) values of these $m$ new intervals to be deleted. Keep these into two arrays (one for keeping the start values and the other for the keeping the end values).
- Print the two user input interval sets in proper format (the existing and the new) by iterating over the arrays where you kept the data.
- Delete all the $m$ new intervals from the existing set of intervals and Print the final set of intervals after the deletion.

### Example Inputs/Outputs A:

*Sample-1:*

```
Enter Number of Existing Intervals: 8
Enter 8 Existing Intervals:
    Enter Interval-1: 4 7
    Enter Interval-2: 9 10
    Enter Interval-3: 13 14
    Enter Interval-4: 20 21
    Enter Interval-5: 24 28
    Enter Interval-6: 31 33
    Enter Interval-7: 34 36
    Enter Interval-8: 39 41

Enter Number of New Intervals: 9
Enter 9 New Intervals to Delete:
    Delete Interval-1: 1 2
    Delete Interval-2: 5 6
    Delete Interval-3: 12 15
    Delete Interval-4: 17 18
    Delete Interval-5: 23 25
    Delete Interval-6: 27 29
    Delete Interval-7: 32 35
    Delete Interval-8: 38 40
    Delete Interval-9: 42 45

++ Existing Intervals: [ 4, 7 ]; [ 9, 10 ]; [ 13, 14 ]; [ 20, 21 ];
                       [ 24, 28 ]; [ 31, 33 ]; [ 34, 36 ]; [ 39, 41 ];
++ Deleting Intervals: [ 1, 2 ]; [ 5, 6 ]; [ 12, 15 ]; [ 17, 18 ]; [ 23, 25 ];
                       [ 27, 29 ]; [ 32, 35 ]; [ 38, 40 ]; [ 42, 45 ];
++ Final Intervals: [ 4, 5 ]; [ 6, 7 ]; [ 9, 10 ]; [ 20, 21 ]; [ 25, 27 ];
                    [ 31, 32 ]; [ 35, 36 ]; [ 40, 41 ];
```

*Sample-2:*

```
Enter Number of Existing Intervals: 2
Enter 2 Existing Intervals:
    Enter Interval-1: -10000 -100
    Enter Interval-2: 10 1000000

Enter Number of New Intervals: 1
Enter 1 New Intervals to Delete:
    Delete Interval-1: -100000000 1000000000

++ Existing Intervals: [ -10000, -100 ]; [ 10, 1000000 ];
++ Deleting Intervals: [ -100000000, 1000000000 ];
++ Final Intervals: NIL
```

*Sample-3:*

```
Enter Number of Existing Intervals: 2
Enter 2 Existing Intervals:
    Enter Interval-1: 1 3
    Enter Interval-2: 6 9

Enter Number of New Intervals: 1
Enter 1 New Intervals to Delete:
    Delete Interval-1: 3 6

++ Existing Intervals: [ 1, 3 ]; [ 6, 9 ];
++ Deleting Intervals: [ 3, 6 ];
++ Final Intervals: [ 1, 3 ]; [ 6, 9 ];
```

### Problem Statement B: [ *Snake-Rotation-AntiClockwise* ] Marks: 15

Write a C-program that does the following:

- It takes as input $n$ non-negative integer array elements ($1 \leq n \leq 10^5$).
- It also takes as input a replication-number, $m$ ($1 \leq m \leq 10^5$).
- It prints the replicated array entries (for same array) line-by-line $m$ times. *Printing in such a style looks the output like an* ($m \times n$) *matrix.*
- Then, it prints all the elements of this ($m \times n$) output *spirally in Anti-Clockwise direction*, starting from the first element of top printed array and without repeating for any position during traversal.

**Note:** *You are NOT allowed to use any extra array except the ONLY One-Dimensional input array required.*

### Example Inputs/Outputs B:

*Sample-1:*

```
Enter No. of Elements: 10
Enter 10 Array Elements: 0 1 2 3 4 5 6 7 8 9
Enter No. of Such Repeated Arrays: 5
++ The Replicated Set of Array Elements:
 0 1 2 3 4 5 6 7 8 9
 0 1 2 3 4 5 6 7 8 9
 0 1 2 3 4 5 6 7 8 9
 0 1 2 3 4 5 6 7 8 9
 0 1 2 3 4 5 6 7 8 9
++ The Arrays Elements Together (AntiClock-wise Printing):
0 0 0 0 0 1 2 3 4 5 6 7 8 9 9 9 9 9 8 7 6 5 4 3 2 1 1 1 1 2 3 4 5 6 7 8 8 8 7 6 5 4 3 2 2 3 4 5 6 7
```

*Sample-2:*

```
Enter No. of Elements: 2
Enter 2 Array Elements: 1 2
Enter No. of Such Repeated Arrays: 2
++ The Replicated Set of Array Elements:
 1 2
 1 2
++ The Arrays Elements Together (AntiClock-wise Printing):
1 1 2 2
```

**Submit a single C source file for each problem. Do not use global/static variables.**