**CS29003 ALGORITHMS LABORATORY**
**Assignment No: 5**
**Last Date of Submission: 26–August–2015**

The Jovian Institute of Technology (JIT) is building a complex of classrooms for staging several events like classes, labs, quizzes, conferences, and so on. It has a complete list of events before it starts its design process. Each event is specified by an interval $(a, b)$, where $a$ is the start time of the event, and $b$ its finish time. We assume that the events have an hourly schedule, that is, $a$ and $b$ are integers. We also assume that $a < b$ for each event. The intervals are assumed to be open, that is, two intervals like $(a, b)$ and $(b, c)$ are considered non-overlapping. For example, you can schedule the events (1, 3) and (3, 6) in the same classroom. Moreover, multiple events with the same start and finish times may be present. For instance, you may have multiple classes running during the same period (5, 6). The JIT administration plans to design a complex such that all the events can be scheduled with as few classrooms as possible.

**Part 1:** We use a binary search tree in order to store the events. Each node in the tree consists of the following fields: an interval (its two endpoints), a *count* (storing the number of times the interval is duplicated), and a *max* value (storing the maximum finish time of all nodes in the subtree rooted at that node), and two child pointers. Do not use parent pointers.

**Part 2:** Let $I = (a, b)$ and $J = (c, d)$ be two intervals. We say that $I = J$ if both $a = c$ and $b = d$ hold. We say $I < J$ if either (i) $a < c$, or (ii) $a = c$ and $b < d$. Finally, we say $I > J$ if $J < I$. Write a function *intervalcmp*() that, given two intervals $I$ and $J$ as input, returns whether $I < J$ (a negative value), $I = J$ (zero), or $I > J$ (a positive value). Your BST will be ordered with respect to this relation < (this is a total order on the set of integer pairs).

**Part 3:** Write a function *insert*() to insert an interval $I$ to an event tree $T$. Follow the standard BST-insertion procedure with two modifications: (i) if $I$ already exists in $T$, increment its *count*, and (ii) the insertion may cause the *max* fields to change at some nodes on the insertion path—take care of that. The function should return a pointer to the root node of the tree. You do not have to balance the height of the tree.

**Part 4:** Two intervals $I$ and $J$ are said to be overlapping if they share a common real value. Here, we are considering open intervals, that is, bands of real numbers without the endpoints. Two intervals sharing an endpoint may be non-overlapping. For example, the intervals (3, 5) and (5, 6) do not overlap, whereas the interval (3, 5) overlaps with the intervals (2, 4), (2, 5), (3, 4), (3, 5), (4, 5) and (4, 6) (sharing respectively the common real values $e + 1$, $\sqrt{10}$, $\pi$, 4, 4.5, and 14/3, for instance). Write a function *overlapcnt*() that, given an event tree $T$ and a query interval $I$, returns the number of events that overlap with $I$. The function should take into account the event *counts* stored in the nodes of $T$. The running time of your function should be $O(h + t)$, where $h$ is the height of the tree $T$, and $t$ is the number of overlapping events returned by the function.

**Part 5:** Write a function *minclassroomcnt*() to solve the JIT problem, that is, a function to return the minimum possible number of classrooms needed so that all the events can be scheduled. This function should run in $O(n(h + t))$ time (or better), where $n$ is the number of events, and $t$ is the count returned by the function.

**Part 6:** Write a *main*() function to do the following:

- Initialize an event tree $T$ to empty. Read the number $n$ of events from the user. One by one, read $n$ events from the user, and insert in $T$ (Part 3). Print the preorder and inorder listings of the event tree produced. Write [(*a*,*b*),*count*,*max*] for each node.
- Read the number $q$ of overlap queries from the user. One by one, read $q$ queries, and print the overlap counts (using the function of Part 4). Additionally, print the events that overlap with the query interval. The printing format will be [(*a*,*b*),*count*].
- Invoke the function of Part 5 to compute and print the minimum requirement of classrooms.

Submit a single C/C++ file.

**Sample Output**

```
n = 8
+++ Insert   : (14,24) (3,11) (8,12) (7,10) (8,11) (15,20) (8,12) (19,26)

+++ Preorder : [(14,24),1,26] [( 3,11),1,12] [( 8,12),2,12] [( 7,10),1,11] [( 8,11),1,11]
               [(15,20),1,26] [(19,26),1,26]
+++ Inorder  : [( 3,11),1,12] [( 7,10),1,11] [( 8,11),1,11] [( 8,12),2,12] [(14,24),1,26]
               [(15,20),1,26] [(19,26),1,26]
q = 2
+++ Overlap (11,21) : [(14,24),1] [( 8,12),2] [(15,20),1] [(19,26),1]
                      Overlap count = 5
+++ Overlap ( 7,16) : [(14,24),1] [( 3,11),1] [( 8,12),2] [( 7,10),1] [( 8,11),1]
                      [(15,20),1]
                      Overlap count = 7

+++ Minimum class count is 5
```