Skip List Project

Background: Many applications areas such as computer graphics, geographic information systems, and VLSI design require the ability to store and query a collection of rectangles. In 2D, typical queries include the ability to find all rectangles that cover a query point or query rectangle, and to report all intersections from among the set of rectangles. Adding and removing rectangles from the collection are also fundamental operations.

For this project, you will create a simple spatial database for handling inserting, deleting, and performing queries on a collection of rectangles. The data structure used to store the collection will be the Skip List (see Section 12.1 of the textbook for more information about Skip Lists). The Skip List fills the same role as a Binary Search Tree in applications that need to insert, remove, and search for data objects based on some search key such as a name. The Skip List is roughly as complex as a BST to implement, but it generally gives better performance since its worst case behavior depends purely on chance, not on the order of insertion for the data. Thus, the Skip List provides a good organization for answering non-spatial queries on the collection (in particular, for organizing the objects by name). However, as you will discover, the Skip List performs poorly on spatial queries. In Project 2, you will add a more sophisticated data structure that is capable of processing the spatial operations more efficiently.

Invocation and I/O Files:

The name of your executable must be `p1`. There will be no input parameters to the program. Your program should read the command file from the standard input, and write its output to the standard output.

Your program will read from standard input a series of commands, with one command per line. No command line will require more than 80 characters. The formats for the commands are as follows. The commands are free-format in that any number of spaces may come before, between, or after the command name and its parameters. All coordinates will be signed values small enough to fit in a 32-bit `int` variable.

insert *name x y w h*

Insert a rectangle named *name* with upper left corner (x, y), width w and height h. It is permissible for two or more rectangles to have the same name, and it is permissible for two or more rectangles to have the same spatial dimensions and position. The name must begin with a letter, and may contain letters, digits, and underscore characters. Names are case sensitive.

remove *name*

Remove the rectangle with name *name*. If two or more rectangles have the same name, then any one such rectangle may be removed. If no rectangle exists with this name, it should be so reported.

remove x y w h

Remove the rectangle with the specified dimensions. If two or more rectangles have the same dimensions, then any one such rectangle may be removed. If no rectangle exists with these dimensions, it should be so reported.

regionsearch x y w h

Report all rectangles currently in the database that intersect the query rectangle specified by the **regionsearch** parameters. For each such rectangle, list out its name and coordinates.

intersections

Report all pairs of rectangles within the database that intersect.

search *name*

Return the information about the rectangle(s), if any, that have name *name*.

Implementation and Design Considerations:

The rectangles will be maintained in a Skip List, sorted by the name. Use `strcmp` to determine the relative ordering of two names, and to determine if two names are identical.

The biggest design difficulty you are likely to encounter relates to traversing the Skip List during the **intersections** command. The problem is that you need to make a complete traversal of the Skip List for each rectangle in the Skip List. The leads to the question of how do you remember where you are in the "outer loop" of the operation during the processing of the "inner loop" of the operation. A good design choice is to augment the Skip List with an **iterator** class. An iterator object tracks a current position within the Skip List, and has a method that permits the position of the iterator object within the Skip List to move forward. In this way, one iterator object can be tracking the current rectangle in the "outer loop" of the process, while a second iterator can be used to track the current rectangle for the "inner loop."

For the **regionsearch** and **intersections** commands, you need to determine intersections between two rectangles. Rectangles whose edges abut one another, but which do not overlap, are not considered to intersect. For example, (10, 10, 5, 5) and (15, 10, 5, 5) do NOT overlap, while (10, 10, 5, 5) and (14, 10, 5 5) do overlap. Note that rectangles (10, 10, 5, 5) and (11, 11, 1, 1) also overlap.