

CS2604 (Fall 2002)

PROGRAMMING ASSIGNMENT #2

Due Tuesday, Oct 8 @ 11:00 PM for 100 points

Early bonus date: Monday, October 7 @ 11:00 PM for a 10 point bonus

Late date: Wednesday, October 9 at 11:00 PM for a 20 point penalty

Background:

This project continues the theme of a Geographic Information System that was begun in Project 1. This time, the focus is organizing the city records into a database for fast search. For each city you will store the name and its location (X and Y coordinates). Consider what happens if you store the city records using a linked list. The cost of key operations (insert, remove, search) using this representation would be $\Theta(n)$ where n is the number of cities. For a few records this is acceptable, but for a large number of records this becomes too slow. Some other representation is needed that makes these operations much faster. In this project, you will implement one such representation, known as a bintree.

A binary search tree gives $O(\log n)$ performance for insert, remove, and search operations (if you ignore the possibility that it is unbalanced). This would allow you to insert and remove cities, and locate them by name. However, the BST does not help when doing a search by coordinate. In particular, the primary search operation that we are concerned with in this project is a form of range query called “regionsearch.” In a regionsearch, we want to find all cities that are within a certain distance of a query point.

You could combine the (x, y) coordinates into a single key and store cities using this key in a second BST. That would allow search by coordinate, but does nothing to help regionsearch. The problem is that the BST only works well for one-dimensional keys, while a coordinate is a two-dimensional key.

To solve these problems, you will implement a bintree. The bintree (see Section 13.3.3 of the textbook) is one of many **hierarchical data structures** commonly used to store data such as city coordinates. It allows for efficient insertion, removal, and regionsearch queries. You should read Section 13.3 of the textbook completely, since to understand the description of the bintree, you first need to understand the structure of the k-d tree and the PR quadtree. You should pay particular attention to the discussion regarding parameter passing in recursive functions, and to the discussion regarding flyweights, both in Section 13.3.2. Both of these discussions are relevant to your implementation of the bintree.

Invocation and I/O Files:

Your program must be named **binprog**. Your program will read from standard input, and write to standard output. The program should terminate when it reads an end of file mark from standard input.

The input for this project will consist of a series of commands (some with associated parameters, separated by spaces), one command for each line. Commands are free format in that an arbitrary number of additional spaces may be interspersed between parameters. The input file may also contain blank lines, which your program should ignore. You do not need to check for syntax errors in the command lines (although you **do** need to check for logical errors such as duplicate insertions or removals of non-existent cities).

Each input command should result in meaningful feedback in terms of an output message. Each input command should be echo’ed to the output. In addition, some indication of success or error should be reported. Some of the command specifications below indicate particular additional information that is to be output.

Commands and their syntax are as follows. Note that a *name* is defined to be a string containing only upper and lower case letters and the underscore character.

insert *x y name*

A city at coordinate (x, y) with name *name* is entered into the database. Spatially, the database should be viewed as a square whose origin is in the upper left (NorthWest) corner at position $(0, 0)$. The world is 2^{14} by 2^{14} units wide, and so x and y are integers in the range 0 to $2^{14} - 1$. The x -coordinate increases to the right, the y -coordinate increases going down. Note that it is an error to insert two cities with identical coordinates, but **not** an error to insert two cities with identical names.

remove *x y*

The city with coordinate (x, y) is removed from the database (if it exists). Be sure to print the name of the city that is removed.

search *x y radius*

All cities within *radius* distance from location (x, y) are listed. A city that is exactly *radius* distance from the query point should be listed. You should also output a count of the number of bintree nodes looked at during the search process. x and y are (signed) integers with absolute value less than 2^{14} . *radius* is a non-negative integer.

debug

Print a listing of the bintree nodes in preorder. In each dimension, the node with the lower-valued coordinate should appear first. The tree should split first in the x dimension (i.e., with a vertical line), followed by the y dimension. The node listing should appear as a single string (without internal newline characters or spaces) as follows:

- For an internal node, print “G”.
- For an empty leaf node, print “E”.
- For a leaf node containing a data point, print X,Y,NAME where X is the x-coordinate, Y is the y-coordinate, and NAME is the city name. Making up plausible coordinates in a 32 by 32 world, Figure 13.5a of the textbook would be printed as:

GG10,10,AG5,17,B11,20,EGGG18,2,C18,10DE20,20,F

makenull

Initialize the database to be empty.

Example:

Note: in this example, statements enclosed in `{ }` are comments to help you under the example; comments do NOT appear in the data file!

```
insert 900 500 Blacksburg
      insert 500 140 Roanoke
insert 910 510 New_York
debug                                     { print coords, name for 3 cities }
      remove 500 140                       { its there to remove }
search 901 501 5                           { print info for one city }
makenull                                    { reinitialize }
```

Implementation:

Note that in Project 4 you will be re-implementing this project to store the bintree on disk. To avoid having to completely rewrite your bintree class code, you should be sure to access bintree nodes **ONLY** through **set** and **get** methods in the bintree node class. Make the child pointers private data members of the node class to insure that this happens.

All operations that traverse or descend the bintree structure MUST be implemented recursively.

You must use inheritance to implement bintree nodes. You should have an abstract base class with separate subclasses for the internal nodes and leaf nodes. A bintree internal node should store pointers to its children. Leaf nodes should store a (pointer to a) city record object. Empty leaf nodes should be implemented by a flyweight.

Programming Standards:

You must conform to good programming/documentation standards, as described in the Elements of Programming Style. Some specifics:

- You must include a header comment, preceding `main()`, specifying the compiler and operating system used and the date completed.
- Your header comment must describe what your program does; don't just plagiarize language from this spec.
- You must include a comment explaining the purpose of every variable or named constant you use in your program.
- You must use meaningful identifier names that suggest the meaning or purpose of the constant, variable, function, etc.
- Always use named constants or enumerated types instead of literal constants in the code.
- Precede every major block of your code with a comment explaining its purpose. You don't have to describe how it works unless you do something so sneaky it deserves special recognition.
- You must use indentation and blank lines to make control structures more readable.
- Precede each function and/or class method with a header comment describing what the function does, the logical significance of each parameter (if any), and pre- and post-conditions.
- Decompose your design logically, identifying which components should be objects and what operations should be encapsulated for each.

Neither the GTAs nor the instructors will help any student debug an implementation, unless it is properly documented and exhibits good programming style. Be sure to begin your internal documentation right from the start.

You may only use code you have written, either specifically for this project or for earlier programs, or code taken from the textbook. Note that the textbook code is not designed for the specific purpose of this assignment, and is therefore likely to require modification. It may, however, provide a useful starting point. You may not use code from STL, MFC, or a similar library in your program.

Testing:

A sample data file will be posted to the website to help you test your program. This is not the data file that will be used in grading your program. While the test data provided should be useful, you should also do testing on your own test data to ensure that your program works correctly.

Deliverables:

You will submit a tarred and gzipped file containing all the source code for the project, a makefile and an optional readme file. The file should end in a tar.gz extension. The makefile will allow the TA to simply type make at the command line and create your executable. The optional readme file will explain any pertinent information needed by the TA to aid them in the grading of your submission.

Once you have assembled the archive for submission, for your own protection, please move it to a location other than your development directory, unzip the contents, build an executable, and test that executable on at least one input. Failure to do this may result in delayed evaluation of your program and a loss of points.

You will submit your project to the automated Curator server. The instructions and necessary software are available at: <http://ei.cs.vt.edu/~eags/CuratorGuides.html>. If you make multiple submissions, only your last submission will be evaluated.

Pledge:

Your project submission must include a statement, pledging your conformance to the Honor Code requirements for this course. Specifically, you must include the following pledge statement in the header comment preceding the function main() in your program. The text of the pledge will also be posted online.

```
// On my honor:
//
// - I have not used C++ language code obtained from another student,
//   or any other unauthorized source, either modified or unmodified.
//
// - All C++ language code and documentation used in my program
//   is either my original work, or was derived, by me, from the source
//   code published in the textbook for this course.
//
// - I have not discussed coding details about this project with anyone
//   other than my instructor, ACM/UPE tutors or the GTAs assigned to this
//   course. I understand that I may discuss the concepts of this program
//   with other students, and that another student may help me debug my
//   program so long as neither of us writes anything during the discussion
//   or modifies any computer file during the discussion. I have violated
//   neither the spirit nor letter of this restriction.
//
```

Programs that do not contain this pledge will not be graded.