

CS2604 (Fall 2001)  
PROGRAMMING ASSIGNMENT #3: Splay Trees

(Last Revised 3:15pm, October 15)

Due Thursday, November 1 @ 11:00 PM for 125 points

Early bonus date: Wednesday, October 31 @ 11:00 PM for 13 point bonus

Late date: Friday, November 2 at 11:00 PM with a 40 point penalty

**Assignment:**

For this project, you will implement a simple database system with multiple key indices. Specifically, the records in the database will contain three pieces of information about cities: Their population, their economic “net worth” and their name. The records will be organized for search by a collection of three search tree structures. The trees you will implement are a variant of the Binary Search Tree called the splay tree. The splay tree is essentially a modification to the standard BST insert/delete/search routines that guarantee good performance over a series of operations. Read Section 13.2 of the textbook regarding balanced trees for more information (read the section on AVL trees as well as the one on splay trees).

**Implementation:**

Records in the database each contain three fields. The *population* and *economy* fields are integer values. The *name* field is a variable length string beginning with a letter and containing any number of letters (upper and lower case) and digits. Names are case sensitive. You may (and should) compare names for equality using the standard `strcmp` function or something equivalent.

Your database will contain three separate splay trees. Each splay tree node will contain a pointer to the appropriate data record, and the various splay trees are distinguished by their comparator functions. It is recommended, but not required, that you derive your splay tree implementation from the Binary Search Tree and binary tree node classes provided as part of the class textbook source code. It is **required** that your program include the source code file `dictionary.h` without any modifications, and that your splay tree class inherit from the Dictionary class.

All splay tree operations must be implemented recursively. That is, any function that traverses or otherwise moves through the tree must be recursive. There will be a minor point deduction if your node class uses parent pointers. Note that on an unsuccessful search, the last node encountered will be the object of the splay operation. For example, in Figure 13.10, the same splay operations would take place if the value 89.5 were searched for in the tree of part (a).

**Input and Output:**

The input to this program should be **read from standard input** and the output should be **directed to standard output**. The name of the program should be “splay”.

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. You do not need to check for syntax errors in the command lines, however you do need to check for logical errors (such as deleting a value that is not in the tree).

Each input command should result in meaningful feedback in terms of an output message. Each command’s parameters should be printed, and in addition, some indication of success or failure of the operation should be reported. Some of the command specifications below indicate particular additional information that is to be output. Some of the commands refer to a specific data field, by number. Field “1” is population, field “2” is net worth, and field “3” is name.

Commands and their syntax are as follows.

**insert** *population economy name*

Insert a record into the database. You will create a new record object, and insert it into each of the three splay trees.

No record is permitted to duplicate the value of the same field in any other record. For example, it is permitted for the database to contain the two records “1000 12000 Atown” and “12000 1000 Btown” but it is **not** permitted for the database to contain the two records “1000 12000 Atown” and “1000 15000 Btown”. If an attempt is made to insert a record with a duplicate field value, then the insert attempt should be rejected. Note that this might require you to reverse the insertion already done on some tree(s) for that record.

**find** *field value*

Print some record with value *value* in field *field*.

**remove** *field value*

Remove some record with value *value* in field *field* from the database, if it exists. To do so, you will need to remove the record from all three splay trees.

**list** *field*

List out the records in the database, sorted by *field*. The listing should be processed by an inorder traversal of the appropriate tree, with each record printed on a separate line. If the record appears in a node at depth  $i$  in the tree, it will be printed starting in character column  $4i$  on the page (counting the first column as 0).

**makenull**

Reinitialize the database to be empty.

### 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:

Sample data files will be posted to the website to help you test your program. These are not the data files 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 this project electronically. In particular, you will create a zip'ed archive file containing the following items (and nothing else):

- all source code files necessary to build an executable
- either the project workspace files (.dsw and .dsp) for Visual C++ users, or a makefile for g++ users.

Windows users should be sure to use a modern zip tool which preserves long file names. A suitable freeware command-line zip tool will be posted on the course website. UNIX users should submit a gzip'ed and tar'ed file.

Once you have assembled the archive file 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 file. 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.