

IP Manager

For this project you will implement a binary search tree (without balancing). The BST will be used here to store records, but it should be implemented as a template for reusability (by you on a future project).

The BST must support all the usual functionality, as discussed in class. For this project we will focus primarily on building the initial tree from a list of values, inserting and deleting values, searching for specific values, and providing support for one data editing operation. In order to determine if the tree structure is correct, you will instrument the search function to return the level in which the sought item is found.

The BST will store records that represent network addresses. For our purposes, each of these records will consist of an IP address, the network name assigned to that address, and the physical location of the corresponding machine. For example:

```
298.120.40.101    daffodil.cs.vt.edu    McBryde 118
```

The BST will organize the records using the IP address as the primary key. Each IP address must be unique; i.e., the tree must never store two records containing the same IP address. Your implementation must check for that and reject any attempt to create a duplicate entry. Logically, it would also be an error for two records to contain the same network name, but your implementation may ignore that issue.

Data Structures:

The primary data structure element of this project is a binary search tree (BST). Your implementation is under the following specific requirements:

- The BST must be encapsulated as a C++ template.
- The underlying structure must be linked, and you must use a C++ template for the tree nodes.
- The behavior of the BST must conform to the description given in class. Note that in this project, we do not allow entries with duplicate key values.
- For testing, your BST should have the ability to display itself to a specified output stream, as described in the notes. If you expect to receive help, be sure that your display function conforms to the formatting described in the course notes.

Your design must make appropriate use of classes. The specification may imply the existence of additional classes besides those involved in the implementation of the BST. Aside from binary nodes used only within an encapsulating class, data members of classes must be private.

If an error occurs during the parsing of the input file, there's an error in your code. However, your program should still attempt to recover, by "flushing" the current input line and proceeding to the next input line.

Command file description:

Your program **must** read commands from a file named `IPCommands.txt` — use of another input file name will almost certainly result in a score of 0. As before, lines beginning with a semicolon (`;`) character are comments; your program will ignore comments. An arbitrary number of comment lines may occur in the input file.

Each non-comment line of the commands file will specify one of the commands described below. Each command line consists of a sequence of "tokens" which will be separated by single tab characters. A newline character will immediately follow the final "token" on each line. Commands are case-sensitive.

The mandatory supported commands are:

```
new      <IP address>      <name>      <location>
```

If the given IP address does not already occur in the BST, a new record is inserted at the proper location in the tree and the level of the new node is printed. If the given IP address already occurs, an error message is printed.

remove <IP address>

If the BST contains a record with the given IP address, that record is deleted from the BST, and its level is printed. If no such record exists in the tree, an error message is printed.

find <IP address>

This causes the BST to be searched for a record containing the given IP address. After the search, output is generated to report the results of the search. If a matching record is found, the level, IP address, network name, and location are printed. If no match is found, an error message is printed.

rename <IP address> <name>

If the BST contains a record with the given IP address, the given name is substituted for the existing name, and a confirming the renaming is printed. If no matching record is found, an error message is printed.

display

This causes the BST to display itself to the log file. The display must show the IP addresses (only), and must be formatted as shown in the course notes, so that the parent-child and level relationships are clear.

exit

This causes the IPManager application to terminate. The command file is guaranteed to end with an exit command.

Legend: in the commands above:

<IP address>	a string in the usual form of an IP address
<name>	a string naming the machine
<location>	a string specifying the location of the machine

Since this assignment will be automatically graded, you must use the same formatting and labeling shown in the sample output file at the end of this specification.

You may assume that both input files will conform to the given syntax, so syntactic error checking is not required. A sample command file is given at the end of this specification.

Output File Description:

Your program **must** write its output to a file named `IPResults.txt` — use of another output file name will undoubtedly result in a score of 0. Since your output for this assignment will be graded automatically, you must adhere to the specified format exactly. A sample output file is given at the end of this specification.

The first two lines should contain your name, course and project title, followed by separator line, as shown below. The next section reports some general statistics on the initial BST.

You must echo the commands (but not the comments) to the output file, labeled as shown. The output, if any, from processing each command must be written to the output file immediately after the command echo, as shown. Each command and its accompanying output must be delimited in some way (e.g., the lines of hyphens I used).

The output file at the end of this specification was produced by my solution, executed on the given sample command file. The separator lines may use any symbol and be of any length you like.

Submitting Your Program:

You will submit this assignment to the Curator System (read the *Student Guide*), and it will be graded automatically. Instructions for submitting, and a description of how the grading is done, are contained in the *Student Guide*.

Note that the automated grading system requires that your program be submitted as a single source file. For this project you should develop your implementation using separate compilation, and then manually concatenate the code into a single .cpp file, being careful to get the order right and remove obsolete #include directives.

You will be allowed up to five submissions for this assignment. Use them wisely. Test your program thoroughly before submitting it. If you do not get a perfect score, analyze the problem carefully and test your fix before submitting again. The highest score you achieve will be counted.

The Student Guide can be found at: <http://ei.cs.vt.edu/~eags/Curator.html>

The submission client can be found at: <http://spasm.cs.vt.edu:8080/curator/>

Evaluation:

Your submitted program will be assigned a score based upon the runtime testing performed by the Curator System. After that, your program will be given a brief evaluation by one of the GTAs, who will consider:

- whether your implementation makes appropriate use of data structures (in particular, the use of a well-designed BST interface), and
- whether your design shows a good object-oriented decomposition of the given problem, and
- whether the internal documentation of your code is acceptable.

This evaluation will produce a deduction (possibly zero, of course) that will be applied to your runtime testing score to produce your final score for the project.

Programming Standards:

While we won't be carefully evaluating your source code on this assignment for programming style, you should still observe good practice. See the Programming Standards page on the course website for specific requirements that should be observed in this course.

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment for your program:

```
// On my honor:  
//  
// - I have not discussed the C++ language code in my program with  
//   anyone other than my instructor or the teaching assistants  
//   assigned to this course.  
//  
// - I have not used C++ language code obtained from another student,  
//   or any other unauthorized source, either modified or unmodified.  
//  
// - If any C++ language code or documentation used in my program  
//   was obtained from another source, such as a text book or course  
//   notes, that has been clearly noted with a proper citation in  
//   the comments of my program.  
//  
// - I have not designed this program in such a way as to defeat or  
//   interfere with the normal operation of the Curator System.  
//  
// <Student Name>
```

Failure to include this pledge in a submission is a violation of the Honor Code.

Sample Command File:

```
; Sample commands file for IP address project:
;
; Build initial tree:
new 100.101.40.3      daffodil.cs.vt.edu      McB 118
new 100.101.40.2      hyacinth.cs.vt.edu      McB 116
new 100.101.40.1      peony.cs.vt.edu         McB 116
new 100.101.40.4      tigerlily.cs.vt.edu     McB 118
new 100.101.40.5      rose.cs.vt.edu          McB 118
new 100.101.40.8      daisy.cs.vt.edu         McB 104
new 100.101.40.9      nightshade.cs.vt.edu   McB 124
new 100.101.40.7      belladonna.cs.vt.edu   McB 124
;
; Take a look at it:
display
;
; Try a few searches:
find 100.101.40.3
find 100.101.40.4
find 100.101.40.7
find 999.999.99.9
;
; Delete a leaf node:
remove          100.101.40.9
;
display
;
; Delete the root node:
remove          100.101.40.1
;
display
;
; Try inserting a duplicate IP address:
new 100.101.40.7      fern.cs.vt.edu          McB 116
;display
;
; Try renaming:
rename          100.101.40.7      pansy.cs.vt.edu
;
; ...and check the results:
find 100.101.40.7
;
exit
```

Sample Results File:

Bill McQuain
CS 2604 IP Manager

```
-----  
Command: new          100.101.40.3      daffodil.cs.vt.edu  
          McB 118  
0  
-----  
Command: new          100.101.40.2      hyacinth.cs.vt.edu  
          McB 116  
1  
-----  
Command: new          100.101.40.1      peony.cs.vt.edu    McB  
116  
2  
-----  
Command: new          100.101.40.4      tigerlily.cs.vt.edu  
          McB 118  
1  
-----  
Command: new          100.101.40.5      rose.cs.vt.edu     McB  
118  
2  
-----  
Command: new          100.101.40.8      daisy.cs.vt.edu    McB  
104  
3  
-----  
Command: new          100.101.40.9      nightshade.cs.vt.edu  
          McB 124  
4  
-----  
Command: new          100.101.40.7      belladonna.cs.vt.edu  
          McB 124  
4  
-----  
Command: display  
0 1 2 3 4  
    100.101.40.1  
    100.101.40.2  
100.101.40.3  
    100.101.40.4  
    100.101.40.5  
        100.101.40.7  
        100.101.40.8  
        100.101.40.9  
-----  
Command: find          100.101.40.3  
Level:    0  
Address:  100.101.40.3  
Name:     daffodil.cs.vt.edu  
Location: McB 118  
-----  
Command: find          100.101.40.4  
Level:    1  
Address:  100.101.40.4  
Name:     tigerlily.cs.vt.edu  
Location: McB 118  
-----  
Command: find          100.101.40.7  
Level:    4
```

```
Address: 100.101.40.7
Name:    belladonna.cs.vt.edu
Location: McB 124
```

```
-----
Command: find          999.999.99.9
999.999.99.9not found
```

```
-----
Command: remove       100.101.40.9
4
```

```
-----
Command: display
0 1 2 3 4
    100.101.40.1
    100.101.40.2
100.101.40.3
    100.101.40.4
    100.101.40.5
    100.101.40.7
    100.101.40.8
```

```
-----
Command: remove       100.101.40.1
2
```

```
-----
Command: display
0 1 2 3 4
    100.101.40.2
100.101.40.3
    100.101.40.4
    100.101.40.5
    100.101.40.7
    100.101.40.8
```

```
-----
Command: new          100.101.40.7          fern.cs.vt.edu    McB
116
100.101.40.7 already exists
```

```
-----
Command: rename       100.101.40.7          pansy.cs.vt.edu
100.101.40.7 renamed from belladonna.cs.vt.edu
```

```
-----
Command: find          100.101.40.7
Level:    4
Address:  100.101.40.7
Name:     pansy.cs.vt.edu
Location: McB 124
```

```
-----
Command: exit
-----
```