CSE 373 Homework 3 Project Description

Assigned:	Wednesday, April 17, 2002
Due:	Wednesday, April 24, 2002
	At the start of class

Introduction

For this homework project, you will develop a code module that implements an AVL tree package and answer some questions. All the source files that we are providing for this assignment are in a zip file on the web site. You should download and unzip the file. The questions were handed out in class; you can also get a copy from the web site.

Tree. The mainTree program implements a simple Symbol reader and tree display formatter. For the first time, we are using an additional program called "dot" which takes text input files describing a graph of some sort, and produces postscript output files with nicely drawn plots. The drawings in this case are drawings of the trees that your procedures have created.

The main program for the homework is supplied in ADT/Tree/mainTree.c, and you are to write the individual tree management functions as defined in ADT/include/tree.h. The functions are based very closely on the discussion in the textbook, however, they are not exactly the same.

Grading

The 7 homework assignments of the quarter will count for a total of 50% of your class grade, and each individual homework assignment will count for about 7% of the total class grade. The grading for this project is as follows.

Questions:	10 points
Implementation:	10 points
Total	20 points

NOTE: You need to turn in several things:

1. the paper copy of your answer sheet
2. a printout of a wintree.dot (or tree.dot) showing the output from your program
3. a printout of the two plots generated by dot from the file in item 2
4. a copy of the receipt you got when you did the web turn in
5. and do a web turnin of your implementation of tree.c

Staple the answer sheet, the dot file, and the two pages of plots together and turn them in on Wednesday.

Directory Structure

All of the homework projects are implementations of one of the Abstract Data Types that we discuss in class. The directory structure of the files you receive is as follows:

ADT/	top level directory
ADT/include	header files
ADT/symbols	example symbol table files
ADT/Tree	directory for the Tree project
ADT/Lecture	examples from the lectures, if any
ADT/lib	precompiled binaries, if any

Program: Tree

The purpose of this program is to read a symbol table file and create two AVL binary search trees from it, then create a drawing showing the structure of the trees.

Using one set of symbols, the program creates one tree sorted by symbol name, another by symbol value. The program prints information about the resulting trees in the format defined by the dot program. Input is taken from stdin or a named symbol table file, output is to stdout (which can be redirected to a disk file). The dot program reads this file and produces a postscript drawing.

This program requires you to implement a set of AVL tree management functions. The main program and the header files are supplied; you add code to the implementation file "tree.c" to supply the actual tree management functions.

Remember that most of the code you need to implement is outlined in the book or on the web site associated with the book. There is a link to the code from our syllabus page. You need to modify it for our data types and comparison strategies and integrate it into the application.

Program: dot

This program is already written (by ATT). It is installed on the lab computers, and it is available from the class web site for installing on your own system. You don't need to know anything about how the input to dot is formatted in order to do this assignment, since the format routines are already written and supplied to you in the homework zip file. If you are running on your own system, you will have to do the installation of dot (which is just extracting it from the graphviz download and making it available in your path).

Note: If you want to add printf statements to your code, you should write out your information surrounded by /* and */, because then dot can still read the output file and it will ignore these lines (it considers them to be comments).

To do:

1. Create the Tree project just as you did in Homework 1 for the Sum and List projects. For the Program arguments entry, use "..\symbols\tvsymbols.txt".
2. Note that although there is a file "tree.c" provided to you it is very incomplete, and so the project will not link correctly. All the routines that you will implement are missing.
3. Change the name that is included in the function getTreeAuthor. As delivered to you, it says "Anonymous Author." You should change that to be your own name.
4. Write the routines that are needed by mainTree as described below, rebuild the project, and run it. Debug until done.
5. If you run the project from Visual C, the output gets displayed in a console window. This may not be very understandable. I have supplied a batch file called winrunit.bat that runs your program (Debug\tree.exe) from a command line and stores the output in wintree.dot. It then runs dot, and stores the result in wintree.ps. To use this batch file, get a command prompt (Start->Programs->Accessories->Command Prompt) and use the cd command to move to the ADT\Tree directory. Then type "winrunit".

   The wintree.dot file contains whatever information your program wrote out. If you have been careful to surround any output you have added with /* and */ then dot should run okay.

   The wintree.ps file is a postscript file that contains an actual drawing of the two trees that were created. You can use Ghostview (available from our web site) or any other postscript viewer to display and print the drawings.

6. Review the code as needed in order to answer the questions in the homework.

The definition of struct ListNode is in include/privatetree.h. It is not in tree.c because the plot functions need to be able to see the structure definition also. Similarly, the definition of struct Symbol is in include/symbol.h because the symbol functions in symbol.c need to be able to see the structure of a Symbol.

The function headers for the routines you need to write are in ADT/include/tree.h. Note that the functions are very similar, but NOT IDENTICAL, to the ones defined in the textbook. The functions are as follows.

char *getTreeAuthor(void)

Returns a text string to the caller naming the person who wrote the program. Change "Anonymous Author" to your name.

SearchTree CreateTree(void)

Create a new AVL tree data structure. Since there is no data structure needed (the root node entry is created when the first item is inserted in the tree) this function does not do anything other than return NULL.

SearchTree DestroyTree(SearchTree T)

Release all memory allocated for this tree. This requires traversing the tree and releasing the TreeNodes. This function is called MakeEmpty in the textbook. Notice that this function does not release memory associated with any items that might be pointed to by entries in the tree. It is the responsibility of the calling program to manage the memory associated with objects other than the tree itself. This function returns a NULL.

SearchTree InsertTreeNode(SearchTree T, Comparator C, ElementType X)

Add a new node to the tree. The Comparator C is a function pointer. The comparator function is used in place of the "<" and ">" signs in the examples in the textbook. Comparators work exactly the same way that int strcmp(char *a,char *b) works. The function returns –1 if the first argument is less than the second argument, returns 0 if the arguments are equal, and returns +1 if the first argument is greater then the second argument. You call a comparator that has been passed to you as a function pointer with a statement like:

```
if ((*C)(X,T->Element) < 0) {
    … do things for X less than T->Element …
}
```

The reason for doing this is that you can use one set of tree management code and let the caller decide how to order the tree by providing a special comparator function. In this homework, I have supplied two comparators: one by value, one by name.

Position FindTreeNode(SearchTree T, Comparator C, ElementType X)

Find a node in a tree. Note that the Comparator supplied by the caller will always be the same Comparator as was supplied when the nodes were inserted in the tree. If you build a tree with one comparator, then try to find nodes using another, it will not work. This function returns a pointer to a TreeNode or NULL if not found.

Position FindTreeNodeMin(SearchTree T)

Find the minimum node in the tree. No comparator is needed because the minimum node is the leftmost child in the tree, no matter what the comparator is. Returns a pointer to a TreeNode or NULL if empty tree.

Position FindTreeNodeMax(SearchTree T)

Find the maximum node in the tree. No comparator is needed because the maximum node is the rightmost child in the tree, no matter what the comparator is. Returns a pointer to a TreeNode or NULL if empty tree.

ElementType RetrieveTreeNodeElement(Position P)

Retrieves the element pointer from a TreeNode for use by the caller.

In addition to the functions listed above, there are several utility functions that are used in tree.c, but are not needed anywhere else.

static int GetHeight(SearchTree T);
static Position SingleRotateWithLeft(Position K2);
static Position SingleRotateWithRight(Position K2);
static Position DoubleRotateWithLeft(Position K3);
static Position DoubleRotateWithRight(Position K3);

These functions are defined in the textbook.