$\boxed{\text{Burt Rosenberg}}$

Problem Set 6

OUT: 30 SEPTEMBER, 1993
DUE: 8 OCTOBER, 1993

Goals

Learning about Trees and Recursion.

Reading Assignment

Read Chapters 4 and 5 from *Algorithms,* and Chapter 17 from *Oh! Pascal!*

Programming Assignment

Write a program that takes words from an input file and looks for them in a binary search tree. If found, a count associated with the word is incremented. If not found, the word is added to the tree (maintaining the important search tree order).

Recall that this search tree order is that for any node n and its left and right children l and r, the word in l comes before the word in n in dictionary ordering, and the word in r comes after the word in n in dictionary ordering.

Write out the tree using an *in-order traversal.* It could be a recursive routine or a stack based routine — your choice. The words will be printed in increasing dictionary order. You are also *required* to write a routine which prints the tree out pre-order and with indentation to show its structure. For instance, after the file *romeo, romeo, wherefore art thou, romeo?*, printing the tree would give:

```
1  romeo, 3
2      art, 1
3          -
3          -
2      wherefore, 1
3          thou, 1
3          -
```

(The numbers in the left hand column relate to node depth, and null children are shown to help differentiate an only-left-child from an only-right-child.)

Writing this print function is something you will *want* to do, because it is so much fun watching the tree grow word by word. The most fun is to write a small main program which asks for one word, inserts it into the tree, prints the tree in the above form, then asks for another word. When you type "." the program prints the tree in-order and exits.

Then modify this to form the off-line version which reads words from a text file, and prints the tree in-order on completion.

◇ ◇ ◇

Extra Credit: Implement deletion for you binary tree. Figuring out a good way to delete from a binary tree is the most interesting part of this extra credit.

A "good" deletion algorithm will require only small restructurings of the tree. That is, the run time for a good deletion algorithm will be proportional to the height of the tree. As an example of a bad deletion algorithm, consider rebuilding the tree from scratch. It is a bad algorithm because it will take time proportional to the number of items in the tree, which could be much greater than the tree's height.

To test your program, have it print the tree after each word is inserted or deleted. A word in the input text is to be deleted if it begins with a dash. The dash is stripped, and the word, if found in the tree, is deleted. It need not be an error to ask to delete a word not found it the tree. For instance, *romeo, romeo, wherefore art thou, -dumb -romeo?* might have final tree:

```
1  thou, 1
2      art, 1
3          -
3          -
2      wherefore, 1
3          -
3          -
```

(This is an example, but it is also a hint!)