Vincent Crabtree

Programming Assignment 2

Cross-Reference Index

Report

Overview

The purpose of this assignment was to create a cross-reference index for a given text file by utilizing a binary search tree to store the words that appear in the file. Each word is then displayed on the index in alphabetical order, alongside a list of numbers of the lines that word appears on. This list of numbers is stored as a linked list, and every node of the search tree that corresponds to a word has one. My program begins by asking the user for the name of the file they wish to cross-reference, and then begins to analyze that file line by line and char by char. Every time a complete word is found, it is truncated to ten characters if necessary, then either it is added to the tree as a new node, or the line number it was found on is added to the list of line numbers for that word if it already exists in the tree. Finally, the cross-reference index is displayed to the user and written to an output file when the file has been completely scanned.

I implemented the majority of this assignment by making three main classes. The first of these classes is a tree node class, which acts as word on the binary search tree. The second is a line number class, which is essentially a linked list node, and each tree node has a pointer to the head of a linked list used for storing the numbers of lines that the corresponding word appears on. The third class I made was a top level binary search tree classes, which has a pointer to the root of the tree, and various methods for the tree. This class encapsulates the other two, and as a whole they act as the binary search tree.

File Descriptions

Main.cpp:

The main function acts as the driver for the program, and handles most of the file streams and iterating through them. It begins with declarations and initializations, and most notably instantiates a binary search tree object. Then, it prompts the user to enter the name of the file to be cross referenced, and opens a stream to that file, or exits with a warning if it doesn't exist. Next, a while loop is used with a getline function as its condition, so it will iterate for every line of the file until the end. Inside this loop is a for loop, which iterates character by character through the line using a string iterator. Each character is checked, and if it is an alphanumerical, it is concatenated to a temporary string that exists to hold words as they appear in the file. Once a character is found that is not an alphanumerical, the temporary string is cut down to a ten character substring, and then converted into a c-style string to store in a tree node object. The temporary string is then cleared to start a new word, and the binary search tree insert method is called with the new c-string as the key, and the line number as well, provided it starts with a letter. After the for loop has iterated through the entire line, the temp string is checked for the last word, and it is added to the search tree if appropriate. Finally, the display tree method of the binary search tree is called, which displays the contents of the tree as a cross-reference index, and outputs the index to an output file as well. To wrap everything up, a free tree method is

called to free all of the memory allocated to the tree, the tree is deleted, and the file stream to the text file is closed.

treeNode.cpp/h:

This is the tree node class which houses all of the necessary attributes and methods for operating on a tree node. The attributes include left and right subtree pointers, and a pointer to the head of a linked list of corresponding line numbers. It also includes a c-style string for the word, and getters and setters for the private attributes. Each tree node has a method for inserting a number into the list, and freeing the memory used by the list of line numbers. The insert number method takes the number to be inserted as an argument, and then iterates to the end of the list and inserts a new linked list node there with that number. This will put the numbers in increasing order inherently, since the main function iterates through the lines in increasing order.

lineNum.cpp/h:

This class is simply used for the linked list node for the list of line numbers at each tree node. It contains a line number integer attribute to use as the key, and a pointer to the next number in the list. It also has getters and setters for each attribute.

Bst.cpp/h:

The “bst” class stands for binary search tree, and it acts as the top level class for the binary search tree data structure in this program. It contains one element: a tree node pointer that points to the root of the tree. The main use of this class lies with the methods, though, as it includes a method for inserting a new node into the tree, a method for displaying the contents for the tree as a cross-reference index and a method for freeing all of the memory used by the tree on the heap.

The insert method either inserts a new node into the tree, or adds a new lone number to a node that already exists, based on the word that is passed to the method as the key. It begins with an if statement that checks to see if the tree is empty. If it is, then a new tree node object is instantiated as the root of the tree, with the passed word and line number. If the tree is not empty, meaning the root is not NULL, then a string compare is performed on the root’s string and the passed string, and the result is stored in a variable. If the strings match, then the line number passed is added to the list of line numbers at that node. If the string to be inserted is lower in value, then the left subtree is checked. If the left subtree is NULL, then a new tree node is instantiated with the passed string as the key, and set as the left subtree. If the subtree is not NULL, then the method is recursively called on the left subtree. If the passed string is greater in value than the string at the root, the same process occurs, only with the right subtree instead.

The display tree method uses a recursive, in-order traversal of the tree to display the contents of each tree node in alphabetical order. If the passed node is NULL, it returns. Otherwise, the method recursively calls itself on the left subtree of the passed node, then prints the word at the current node, as well as the list of line numbers that correspond. It also writes the same output to an output file. After this, the method recursively calls itself with the right subtree of the passed node.

The free tree method frees the memory allocated on the heap by the binary search tree by using post-order traversal of the tree. If the tree node passed to the method is NULL, it returns. Otherwise, it recursively calls itself on the left subtree of the passed node, followed by a second recursive call on the right subtree. Then it calls the free list method of the current node and finally deletes the current node.