Assignment 2 Part 2A

David Gray s2947315

Contents

1. Problem Statement

The goal of Part 2a of the assignment was to create a dictionary generator that would read in a text file and split the text inside into individual words that are placed in a binary search tree. The binary search tree is traversed in order to output the words stored in the tree alphabetically, to a text file used in the spell checking program (stored as dict.txt).

The program also goes through the spell checking file (Eisenhower Spell.txt) and splits its elements into individual words and outputs the words into a new file (newEisenhowerSpell.txt). This new text file will be the file that is spell checked in the spell checking program.

2. User Requirements

There are no user requirements for this program apart from running the executable.

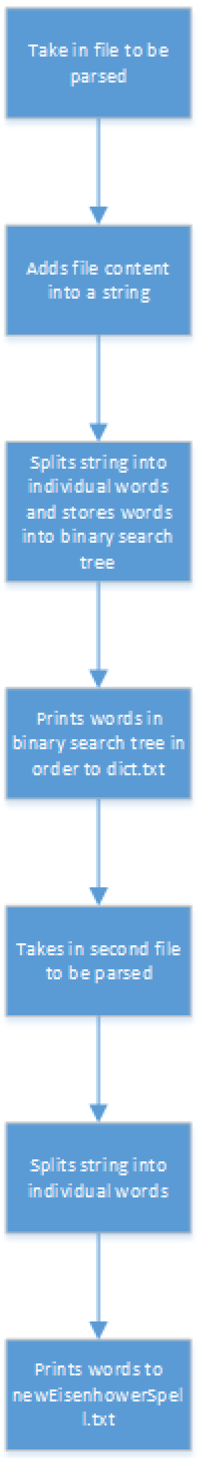
3. Software Requirements

The following outlines the software requirements for the program:

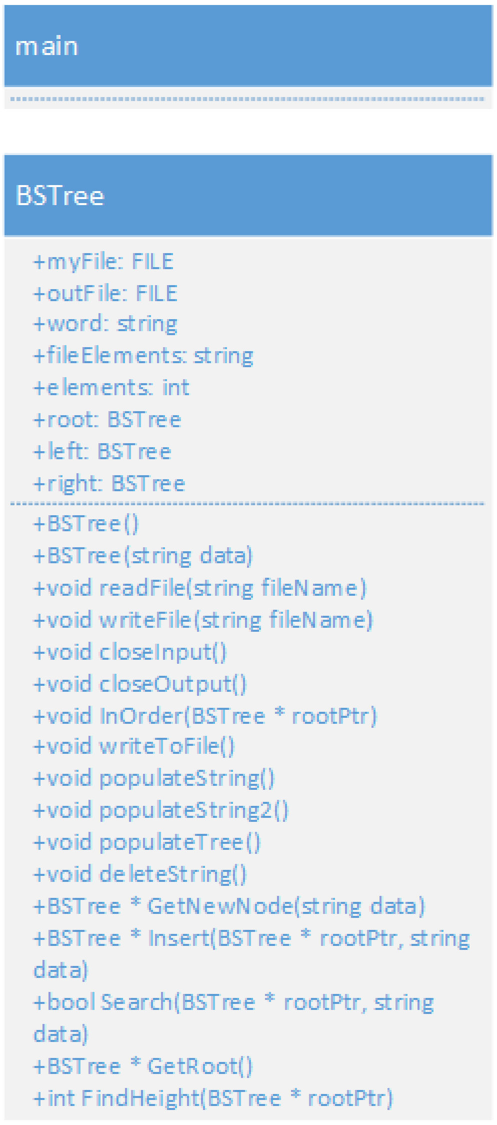
- The program shall convert all words to lower-case
- The program shall check to see if a word is currently stored in the binary search tree and if it is not, it will be added to the binary search tree
- The program shall remove words that contain extra punctuation that is not a single hyphen in the middle of a word, or an apostrophe at the end of a word
- The program shall read in a specific file to parse
- The program shall output a dictionary file separating each word with a newline character
- The program shall output the vital statistics of the binary search tree (for example, the maximum height and the total number of nodes)
- The program shall use C file functions for all file IO and C++ string functions for string handling

4. Software Design

High Level Design – Logical Block Diagram


Take in file to be parsed
Adds file content into a string
Splits string into individual words and stores words into binary search tree
Prints words in binary search tree in order to dict.txt
Takes in second file to be parsed
Splits string into individual words
Prints words to newEisenhowerSpell.txt

Structure Chart



List of main functions in the software.

populateString(): goes through the input file and adds each element to a string called fileString. This method removes words that contain excess characters.

populateTree(): goes through the fileString string and adds individual words from the string into the binary search tree. Before adding a word to the binary search tree, the method checks to see if the word is already in the tree. If it is, the word is not added.

Search(BSTree * rootPtr, string data): searches through the current binary search tree to find if there is already a word in it that is equal to the data being passed through. If there is, the function returns true. If there is not, the function returns false.

InOrder(BSTree * rootPtr): traverses through the binary search tree in order by starting at the left hand side from the root node, printing out all of the contents, printing out the root node, then moving to the right hand side from the root node and printing out all of the contents. This method outputs the words stored in the binary search tree alphabetically to a file specified.

findHeight(BSTree * rootPtr): traverses through the binary search tree to find the longest path.

Insert(BSTree * rootPtr, string data): checks to see if the word to be added is already in the tree. If it is, the process stops. If not, the process continues. If the process continues, the function compares the word to other nodes within the tree to see where it should be placed. When it has found the correct place to insert the new node, it is added to the binary search tree.

List of all data structures in the software. (eg linked lists, trees, arrays etc)

The main data structures this program uses are a binary search tree and linked lists.

The linked lists are used to store the nodes of the binary search tree so that the program can traverse the elements within the tree.

The binary search tree is used to store the words parsed from the input file that will be output as a dictionary file. The binary search tree is balanced on a root node that is 'm'. The binary search tree is traversed to alphabetically output its contents to a dictionary file.

5. Requirement Acceptance Tests

Software Requirement No	**Test**	**Implemented (Full /Partial/ None)**	**Test Results (Pass/ Fail)**	**Comments (for partial implementation or failed test results)**
1	Correctly takes in input file	Full	Pass	
2	Correctly parses file into a string	Full	Pass	
3	Correctly processes string into individual words	Full	Pass	
4	Correctly checks for punctuation in words	Full	Pass	.
5	Correctly adds words to binary search tree	Full	Pass	
6	Correctly prints out vital statistics of binary search tree	Full	Pass	
7	Alphabetically prints out elements of binary search tree to dict.txt text file	Full	Pass	

6. Detailed Software Testing

Test	**Expected Results**	**Actual Results**
Tree Creation		
Tested input registration by entering Ass2 Dictionary.txt	Program should detect that the input value was a valid file	As expected
Tested string creation by parsing through contents of Ass2 Dictionary.txt appending the contents to a large string and removing words with unwanted characters	Program should print out the content from the text file whilst removing words with unwanted characters	As expected
Tested binary search tree element insertion by adding multiple characters such as a, b, c	Program should print out the elements inserted into the binary search tree	As expected
Tested binary search tree inorder traversal by checking to see if input characters are being printed alphabetically	Program should print out contents of the binary search tree alphabetically	As expected
Tested printing of inorder traversal to dict.txt text file	Contents of binary search tree should be printed to dict.txt alphabetically	As expected
Tested vital statistics by printing out the binary search tree’s total number of nodes and maximum depth	Prints out total number of nodes and maximum depth of binary search tree	As expected

7. User Instructions

If running the program using the Visual Studio 2010, open the project and then run the program in debug mode.

If using the executable, double click on the file and let it run.

When prompted, enter the name of the file to create the dictionary from (Ass2 Dictionary.txt).

When prompted, enter the name of the output file for dictionary (dict.txt).

When prompted, enter the name of the file to be split for the spell checking program (Eisenhower Spell.txt).

When prompted, enter the name of the output file for the file to be used in the spell checking program (newEisenhowerSpell.txt).