Project 10

**CONCORDANCE WITH A BINARY SEARCH TREE**

Full Name: Prashul Shrestha

Section #: 2

Project #: 10

Due Date: April 27, 2017

**Design** **Document**

**Introduction**

A **binary search tree** (BST) is a binary tree in which the value at each node is greater than the values in the node's left subtree and less than the values in the node's right subtree. A binary search tree can implement an ordered list.

**Data** **Structures**

The program uses a class called Class **Binary\_Search()**, a char **array**[] named **Word**, and a binary tree to store the word into the node inputted by the user. First, the file is read from the .txt file using **ifstream file** and **get(ch).** The character is checked for alphabet or not using **isalpha()** function and then the filtered word is saved to a **char array Word file[]** and is send to public **insert()** function to recursive private **help\_insert**() function. There we use recursion and **strcmp()** to check for same words already present in the list and to alphabetically arrange the tree.

**Functions**

The program uses **Six** functions to implement the Concordance through Singly-linked List. The functions are called from main() and some are member function to return the result within the function which called it. The list of the functions are given below:

* **Binary**\_**Search**(): A constructor to initialize the root to be NULL.
* ~ **Binary**\_**Search**(): A destructor to call the Destroy function.
* void Destroy(node\* ptr); Deletes all the nodes in the tree.
* **Void insert**(**WORD& String**) – This will help private insert function to send one word as char array in private insert() function.
* **void help\_insert(node \*&ptr, Word& entry)**; to compare the string and insert into the tree in-order.
* **Bool is\_empty():** checks if the binary tree is empty or not.
* **int get\_count(Word& target):** To count the repetitive of words present in the binary tree structure.
* **int bst\_length():** A recursive public function to call the private function.
* **int r\_length(node\* ptr):** A private recursive function to return the length of the tree.
* **bool is\_Present(Word Entry);** To check if the word is present in the binary tree.
* **void Count\_of\_word();** To print the count of words present in the tree.
* **void print();** A public function to call the private function to print the content of the tree.
* **void bst\_print(node\* ptr);**  to print the content in recursive way.

The program uses **string**.**size**() from **string**.h library to get the length of the string. It also uses **strcmp()** and **strcpy()** to compare the char array and the string, and copy one char array into another.

**Menu**() is a the main function from where we first create an object or an instance of class **Binary\_Search** called **BST** and which helps invoke the functions such as **BST.insert**(Word& String); and **BST.get\_count(entry)** to get the count of the word in the list.

**The Main Program**

**Menu**() is a the main function from where we first create an object or an instance of class **Binary\_Search** called **BST** and which helps invoke the functions such as **BST.insert**(Word& String); and **BST.get\_count(entry)** to get the count of the word in the list.

The program is quite simple first, the strings are inserted into the node while checking through the list using **strcmp()** and inserting if it’s not present on the left side or right side of the root depending upon the word, while incrementing the count if its present.

User Document

A **binary search tree** (BST) is a binary tree in which the value at each node is greater than the values in the node's left subtree and less than the values in the node's right subtree. A binary search tree can implement an ordered list.

The main program named **BSTmain**.**cpp** can be compiled and run, using the code:

**g++ BSTmain**.**cpp**

**a.out**

**g++** function will compile the function and make it ready to be run using **a.out**. The function will prompt the following output:

**Ouptut:**

**Word Count**

**-----------------------------**

**a 6**

**all 1**

**belowme 1**

**binary 2**

**bst 1**

**data    1**

**equal 1**

**follow 1**

**greater 1**

**has 2**

**in 1**

**ioned 1**

**is 1**

**its 2**

**key 3**

**left 1**

**less 1**

**node 2**

**nodes 3**

**of 2**

**or 1**

**parent 2**

**propert 1**

**right 1**

**s 1**

**search 2**

**structu 1**

**subtree 2**

**than 2**

**the 4**

**to 2**

**tree 3**

**which 1**

**-------------------------------**

**The File contains 34 words.**

**Enter the word: a**

**Count of the word a = 6**

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

**Thank you for using the Program.**

**Coded By: Prashul Shrestha.**

**Summary**

Completing this project, I learnt the use of Binary tree list in the real-world experience. Since in this project we implemented insert through Binary Tress, I became aware of the implementation part if the Binary Search tree, but could not use remove implementation which I am going to implement in Expression trees using Binary Tree.

Other thing, that I learnt while doing this project is that, while we pass a char array[], a pointer cannot point to the array as whole, so we need to use **strcnpy()** to copy the elements of the array into another and ignore the NULL characters in that copying array.

Working with pointers and their semantics is helping me broaden my knowledge and analyze me the use of pointer in making complex program into simpler versions.

**TESTING**

**CODING**