Data structures I

BINARY SEARCH TREES (ASSIGNMENT 4)

Marwan ElSafty | 4690

Zeyad Ossama | 4555

Rewan Kadry | 4955

Problem Statement:

In this assignment you are required to implement a binary search tree based spell checking system. Initially you will be given a dictionary of words to build your binary search tree, the comparison between strings will be based on strcmp function available in c function. During building your BST, you are required to ensure that your tree is balanced. After finishing the BST you are required to print the height of the generated tree. Then you will be provided with a sentence to spell check. Then you will be required to determine if the word is in your tree or not:

1. If it is in your tree then you will print that the word is correct
2. If it isn't then you will print three suggestions for the correct word, The word in the leaf node you reached, the word in the predecessor node and the node in the successor node.

Binary Search Tree Implementation:

The implementation of the binary tree is found in the header file “bstTree.h” and the source file “bstTree.c” and its main functions used are:

1. arrayToBST:

   this function takes a pointer of the first element in an array of strings sorted alphabetically as a pointer and it makes a new balanced binary search tree from the sorted array using a divide and conquer algorithm using recursion and it returns a pointer to the root node of the tree.

```
TreeNode * arrayToBST(char * arr, int start, int end) {
    if (start > end) return NULL;
    int mid = (start + end) / 2;
    TreeNode * root = newNode(arr + 30 * mid);
    root - > left = arrayToBST(arr, start, mid - 1);
    root - > right = arrayToBST(arr, mid + 1, end);
    return root;
}
```

2. findWord:

   this function search for a given word in the dictionary tree using recursion. The base conditions are to return 0 if the root is NULL or to return 1 if the word in root is equal to the give word.

   But if the given word is smaller than the word in the root we recall the function sending the left subtree as the root else if the given word is bigger than the word in the root we recall the function sending the right subtree as the root.

```
int findWord(TreeNode * root, char wordToFind[]) {
    if (root == NULL) return 0;
    if (strcmp(wordToFind, root - > word) == 0)
      return 1;
    if (strcmp(wordToFind, root - > word) < 0)
      return findWord(root - > left, wordToFind;
    if (strcmp(wordToFind, root - > word) > 0)
      return findWord(root - > right, wordToFind);
}
```

3. findLeaf:

   this function is called whenever a word is not found in the dictionary tree, it returns the word found in the node where the given word was supposed to be found.
   It uses the same algorithm as the function “findWord” but with an extra parameter which is the node variable.

```c
void findLeaf(TreeNode * root, char wordToFind[], TreeNode * leafNode) {
    if (root == NULL)
        return;
    strcpy(leafNode - > word, root - > word);
    if (strcmp(wordToFind, root - > word) < 0)
        return findLeaf(root - > left, wordToFind, leafNode);
    else if (strcmp(wordToFind, root - > word) > 0)
        return findLeaf(root - > right, wordToFind, leafNode);
}
```

4. findSuccessor:

   this function is called whenever the given word is not found in the dictionary tree, it traverses the tree looking for the successor of a node sent to the function as a parameter, if the word in the node is smaller than the word in the root then successor is equal to the root and we go to the left subtree, if the word in the node is bigger than the word in the root we go to the right subtree without changing the successor value and if the word in the node is equal to the word in root we go to the right subtree (the right children are always bigger as we are looking for the successor).
   It stops when the root is NULL.

```c
TreeNode * findSuccessor(TreeNode * root, TreeNode * leafNode) {
    TreeNode * succ = (TreeNode * ) malloc(sizeof(TreeNode));
    while (root != NULL) {
        if (strcmp(leafNode - > word, root - > word) == 0) root = root - > right;
        else if (strcmp(leafNode - > word, root - > word) < 0 || root - > right == NULL)
{
            succ = root;
            root = root - > left;
        }
        else if (strcmp(leafNode - > word, root - > word) > 0 || root - > left == NULL)
            root = root - > right;
              else break;
          }
          return succ;  }
```

5. findPredecessor:

   This function is called whenever the given word is not found in the dictionary tree, it traverses the tree looking for the predecessor of a node sent to the function as a parameter, if the word in the node is bigger than the word in the root then predecessor is equal to the root and we go to the right subtree, if the word in the node is smaller than the word in the root we go to the left subtree without changing the predecessor value.

   It stops when the root is NULL.

```
TreeNode * findPredecessor(TreeNode * root, TreeNode * leafNode) {
    TreeNode * pre = (TreeNode * ) malloc(sizeof(TreeNode));
    while (root != NULL) {
        if (strcmp(leafNode - > word, root - > word) == 0) root = root - > left;
        else if (strcmp(leafNode - > word, root - > word) < 0) {
            root = root - > left;
        } else if (strcmp(leafNode - > word, root - > word) > 0) {
            pre = root;
            root = root - > right;
           } else break;
       }
       return pre;
   }
```

6. getTreeHeight:

   this is a function that returns the height of a given tree.

```
int getTreeHeight(TreeNode * root) {
    if (root == NULL) return 0;
    else {
        int leftHeight = getTreeHeight(root - > left);
        int rightHeight = getTreeHeight(root - > right);
        if (leftHeight > rightHeight) return (leftHeight + 1);
        else return (rightHeight + 1);
    }
}
```

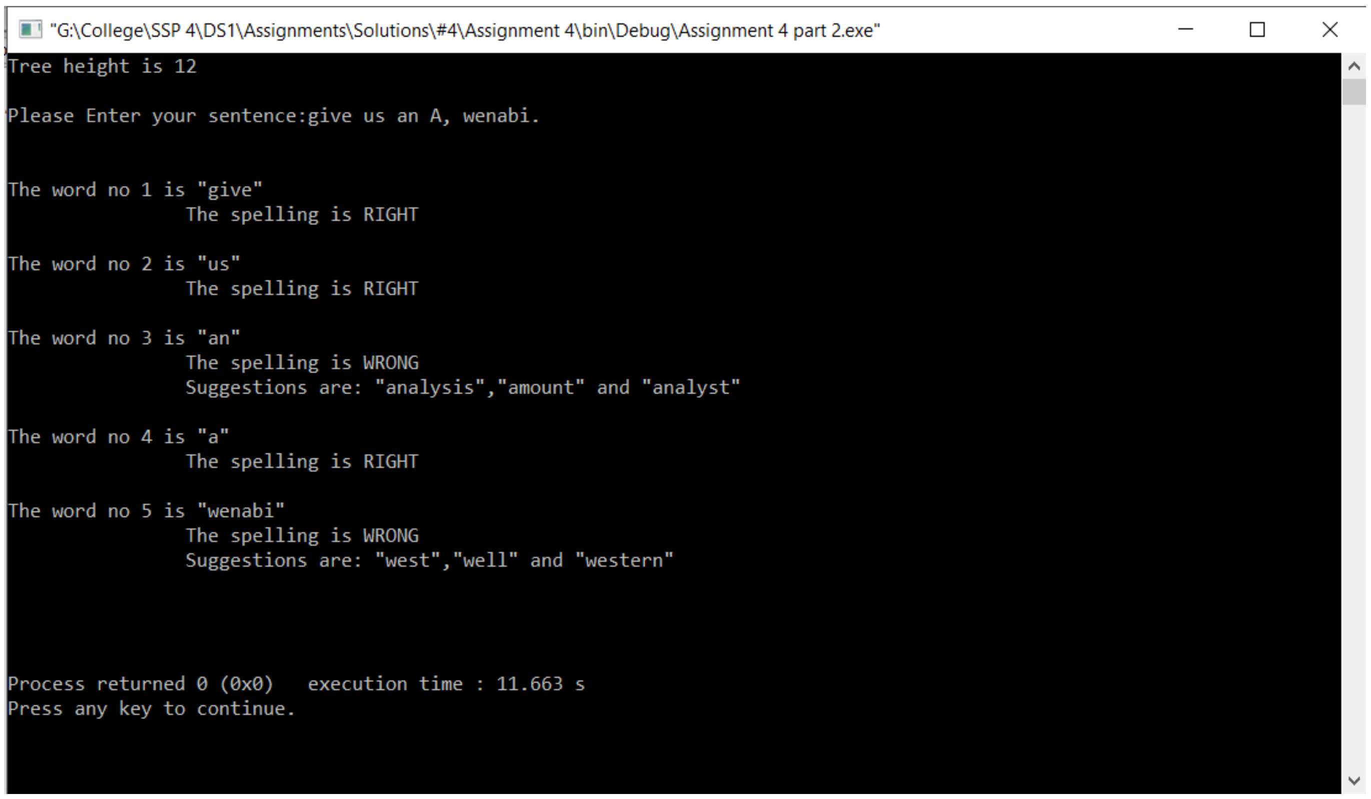
Main file:

It contains the main code, which uses the tree functions and other functions to check the spelling of a given sentence and give suggestions in case of a misspelled word.

“strtok()” function is used to split the sentence given into words when a whitespace or a comma is found.

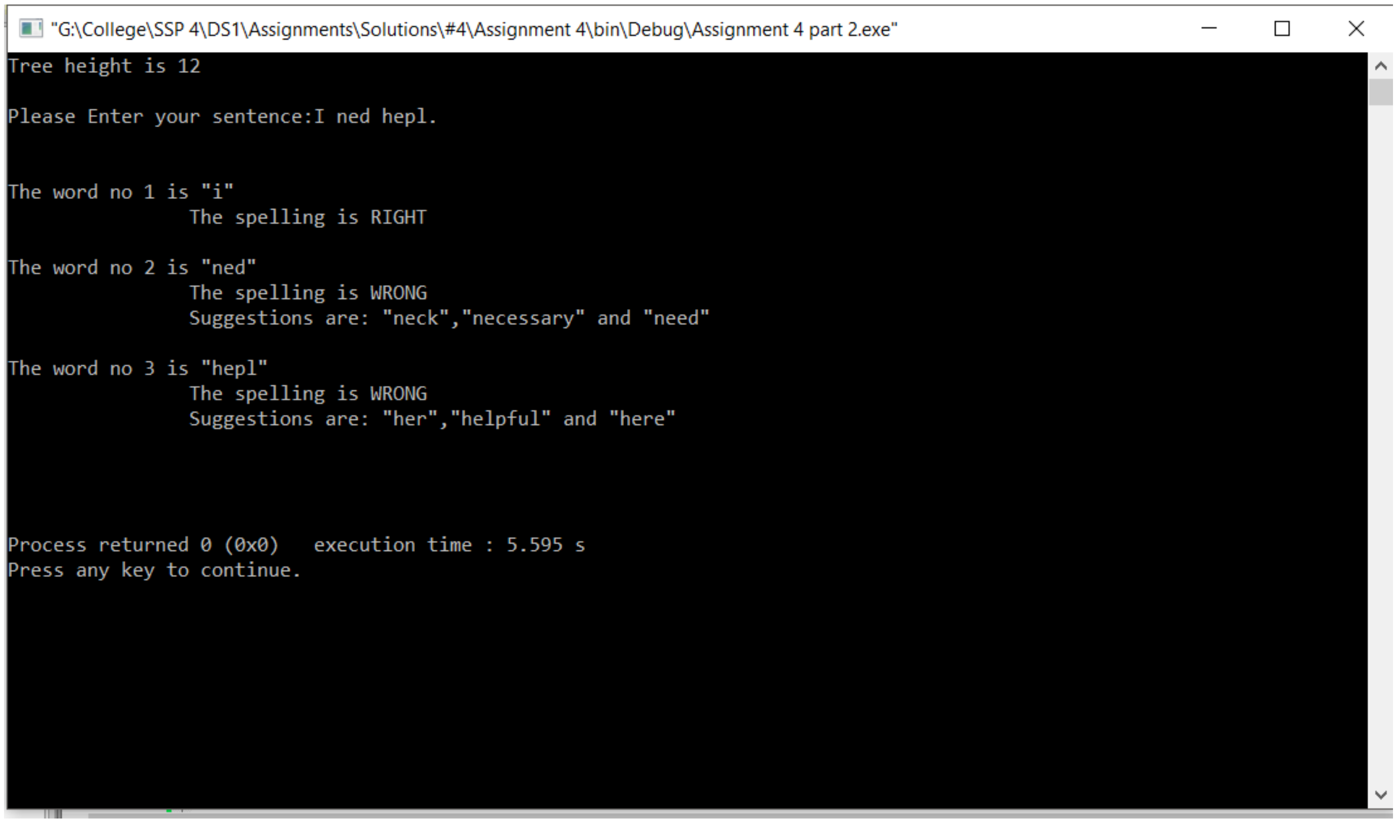
“generateDictionary()” is function created to read the sorted words from the text file and put them in a global array to facilitate its use.

Screenshots:

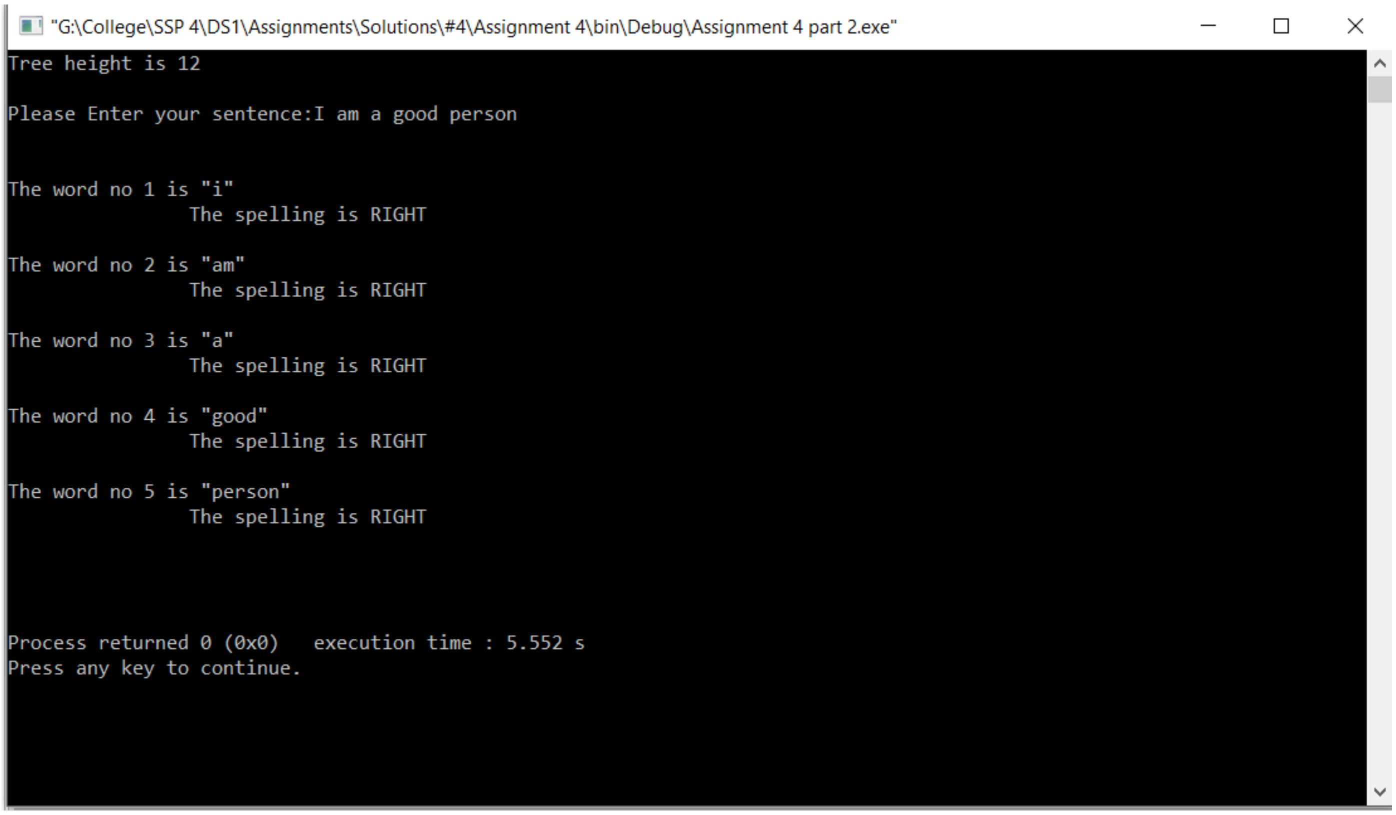


"G:\College\SSP 4\DS1\Assignments\Solutions\#4\Assignment 4\bin\Debug\Assignment 4 part 2.exe"

```
Tree height is 12

Please Enter your sentence:give us an A, wenabi.


The word no 1 is "give"
                The spelling is RIGHT

The word no 2 is "us"
                The spelling is RIGHT

The word no 3 is "an"
                The spelling is WRONG
                Suggestions are: "analysis","amount" and "analyst"

The word no 4 is "a"
                The spelling is RIGHT

The word no 5 is "wenabi"
                The spelling is WRONG
                Suggestions are: "west","well" and "western"




Process returned 0 (0x0)   execution time : 11.663 s
Press any key to continue.
```


"G:\College\SSP 4\DS1\Assignments\Solutions\#4\Assignment 4\bin\Debug\Assignment 4 part 2.exe"

```
Tree height is 12

Please Enter your sentence:I ned hepl.


The word no 1 is "i"
                The spelling is RIGHT

The word no 2 is "ned"
                The spelling is WRONG
                Suggestions are: "neck","necessary" and "need"

The word no 3 is "hepl"
                The spelling is WRONG
                Suggestions are: "her","helpful" and "here"




Process returned 0 (0x0)   execution time : 5.595 s
Press any key to continue.
```


"G:\College\SSP 4\DS1\Assignments\Solutions\#4\Assignment 4\bin\Debug\Assignment 4 part 2.exe"

```
Tree height is 12

Please Enter your sentence:I am a good person


The word no 1 is "i"
                The spelling is RIGHT

The word no 2 is "am"
                The spelling is RIGHT

The word no 3 is "a"
                The spelling is RIGHT

The word no 4 is "good"
                The spelling is RIGHT

The word no 5 is "person"
                The spelling is RIGHT




Process returned 0 (0x0)   execution time : 5.552 s
Press any key to continue.
```