Assignment 1

Luke Wilde | lw102 | 6459274

A program that reads in a text file, adds each word to an AVL tree then uses a mergesort on the tree.

Since binary trees can be unbalanced when nodes are being inserted into the trees, this makes it much less efficient for a program to search the tree, so by using an AVL tree it will self balance upon frequent insertion and deletion and since they are best used in a situation where we are constantly looking up data. Since binary trees are on average **O(n)** it is a slight edge to use an AVL tree as it is **O(log n)**.
The merge sort algorithm is also the best choice as its worst case it is **O(n log n)** and it gives us a decent amount of space being saved as it's space complexity is only **O(n)**.

Pseudo Code:

```
Open a file
while
Read in each word character by character
        If word
                Process start and end of word;
                Insert word
        Fi
elihw
If word
        Process last word
        Insert word
Fi
visit(root)

mergeSort(index)

Print first and last 10 words
```

Functions

getWord(int);

Takes an int of an index and loops through the start and end of the Word struct, adds the chars to make a string then returns said string

insert(string, int);

Takes a string x and an int index. Checks that the index isn't the root then proceeds.
If x is equal to word at index, increase the count of that word, decrement global index then reduce by the length of the word.
If x < word at index, left node of word at index is = insert string on left node, then proceeds with the balancing of left node.
If x > word at index, right node of word at index is = insert string on right node, then proceeds with the balancing of the right node.
Then make the height = max(left.height, right.height) + 1

getHeight(int);

Gets the height of the node at index
Returns -1 if index is = 0, otherwise returns index.height

max(int, int);

Returns the higher value

rotateRight(int);

Takes int index, makes a temp value = index.left
Proceeds to then make index.left = x.right
X.right = index
Returns x

rotateLeft(int);

Takes int index, makes a temp value = index.right
Proceeds to then make index.right = x.left
X.left = index
Returns x

doubleRight(int);

Makes index.left = rotateLeft on index.left
Returns rotateRight on index

doubleLeft(int);

Makes index.right = rotateLeft on index.left
Returns rotateLeft on index

visit(int);

This is the in order traversal for displaying the tree
Checks if index.left != 0
    Calls visit on index.left
Prints values
Checks if index.right != 0
    Calls visit on index.right

mergeSort(int, int);

Makes an int mid = start and stop / 2
Then checks if top <= start if so returns
Calls mergeSort on start and mid
Calls mergeSort on mid+1 and top
The calls the merge function on start, mid and top

merge(int, int, int);

Takes 3 ints, mid, start and top creates a variable = 1 and creates a new array of struct.
Then while the left is less than mid and right less than top, checks if left count is greater than right count and add left to new array otherwise it adds right
Then checks if left less than mid and adds left then checks for right less than top and adds right
Then overwrites all elements in original with the sorted