Question 2 – AC52002

This program is designed to create a Binary Search Tree (BST) with words which have been input from an external text file. The tree is populated, and then used to order the words into alphabetical order. The results are then displayed, which involves showing the words present in the text file, accompanied by a word count for each.

Design

To begin, the user is asked to enter in a name for the file they wish to open. This is then appended with ".txt" and fed into the pre-written 'readCharFile' function which was given to the class. A vector of strings is produced which allows for manipulation of the data (saved in a vector named 'words'). This is then fed into the function 'countDuplicates' which returns a paired vector, comprised of a specific word, and a count of its occurrences. Some complicated logic appears in this function, and admittedly this is something I had to research in order to overcome. References to the relevant material which I have re-purposed can be found in comments inside the code. Once this paired vector has been created, and a count of each word paired inside the vector, the next challenge was to remove the duplicate entries of a word. Again, for this some research was undertaken as to how to do this efficiently (references given in the code). The 'removeDuplicate' function as the name would suggest, iterates through the paired vector and removes duplicates.

This now brings us to a point where there is a paired vector of strings with word counts - which no longer has any duplicate values. It is at this point that data can be inserted into the Binary Search Tree (BST). The BST class used is converted from an integer BST template found while researching (references in code). Each node holds a string, a count of the word, and pointers to the Left and Right nodes. The 'Insertion' method is typical, with recursive conditions determining where to place each word. The 'inorder' method traverses the tree in order and prints out the contents of each node. The 'clearData' method is a little unnecessary and works with the destructor to remove the data from the tree, rather than just to re-assign the pointer to the root node. The 'Insert' public function simply allows a call to the private function, passing the data in accordingly. The 'Results' public method provides some display headers for the data, and then calls the private 'inorder' method to display each node in order.

Input testing

Testing using a larger test file named "blah.txt" (will include for reference)	Parsing was successful, though formatting was an issue with words that were larger being miss-aligned.
Testing using a simple test file ("testfile.txt")	Successful
Testing using a £ sign inside the text file	FAIL – Major error at comparison stage. Most special characters handled okay – though pound sign threw a major error.
Testing using a text file containing numbers and some special characters("testfile2")	The program ignored numbers completely and instead produced results for the character data only. Formatting issues also occurred.