Task 1 – Design and implement your Binary Search Tree (BST) data structure

1. Practice modular programming. That is, compartmentalize the data structures and operations by storing the implementation codes in separate source code files.
2. Design your BST. At the minimum, you must have function definitions for the following BST operations:
   - create() – produces an empty BST
   - search() – determines if a search key exists in the BST
   - insert() – adds a new node in the BST
   - inorder() – performs inorder traversal of the BST
   - destroy() – This function should be called as a clean-up operation before the program actually terminates. If dynamic memory allocation was used, then this function will also free up the memory space. The BST will become empty after calling destroy().
3. Implement the BST yourselves

Task 2 - Implement the algorithms for the BST Application

1. Practice good programming. Compartmentalize your solution/functions in separate source code file(s).
   NOTE: You should **NOT CALL** any of the usual sorting algorithm for the project. The BST should handle the alphabetical ordering of words.
2. Accept a named text file as input. Assume that the input text file contains a collection of words, phrases, sentences, and paragraphs in English.
   - We can assume that the input doesn't contain words with special characters or accents like in ["Café", "Naïve"]
   - The table below shows sample contents of a input text file named **INPUT.txt**

INPUT.txt
Hello, how are you today? I am fine, thank you! How about you? Today is the present. Tomorrow is a gift.

3. Design and implement an algorithm that will automatically detect and count all the English words in the input text file. The case of the words is ignored (e.g. Word = word, Hello = hello) and only words with a length of at least
   3 must be included.
4. Output a text file named **WORDS.txt**. It should contain a list of the English words found in the input text file with their corresponding frequency count. The words must be in lowercase and are arranged in alphabetical order.
   - The table below shows sample contents of the output text file named **WORDS.txt** based on the contents of INPUT.txt in step 2.

WORDS.txt	
about	1
are	1
fine	1
gift	1
hello	1
how	2
present	1
thank	1
the	1
today	2
tomorrow	1
you	3

   - Something to observe with WORDS.txt is that the words do not contain symbols like ["?", ".", "!"] that are present in INPUT.txt
5. Implement error checking. For example, if the input text file does not exist, the program outputs “<FILENAME.TXT> not found.” error message.

Task 3 - Integration and Testing

1. Create the main module which will include the other modules.
2. Call the appropriate functions to achieve the tasks.
3. Create test cases and perform exhaustive testing.