Assignment overview as provided

This assignment requires you to write several functions for a binary tree and a binary search tree (BST). You are also required to write a menu-based application to test these functions. The application reads words from a file and stores them in a BST.

You are supplied with a Dev C++ project, Assignment2.dev, containing various .h and .cpp files. You do not have to modify the project. The code for this assignment must be written in the BinaryTree.cpp, BinarySearchTree.cpp, and Assignment2.cpp files. There is no need to modify the code in the other files supplied.

Introduction to Assignment

The assignment was executed with all the provided inputs from the mail received from Grey Nodes.

From the email, the followings were gathered:-

1. “NodeTypes.h” containing the deceleration of binary tree structure to be reused in the assignment without modification
2. BinaryTree.cpp file was created, as it was not provided. Following functions were created

|  |  |
| --- | --- |
| **Return Type** | **Prototype of Function and Description** |
| BTNode \* | ccreateBTNode (BTNode \* parent, string data)  Creates in a node in the binary tree. The code is almost similar to what was covered in the labs. |
| int | height(BTNode \* root)  Returns the height of the binary tree. The height of the tree is the longest path from the root node to any leaf node in the tree. |
| bool | isEmptyTree(BTNode \* root)  Returns true if the binary tree is empty and false, otherwise. |
| bool | isEqual(BTNode \* root1, BTNode \* root2)  Returns true if the two binary trees supplied as parameters are identical. Two binary trees are identical if they contain the same values in exactly the same positions in each tree. |
| string\* | preOrder (BTNode \* root)  Outputs the pre-order traversal of the binary tree. The code must be non-recursive. |
| Int | weight(BTNode \* root)  Returns the weight of the binary tree. The weight of a binary tree is the amount of leaves in the tree. |
| int | width(BTNode \* root)  Returns the width of the binary tree. The width of a binary tree is the maximum number of nodes at any level of the tree. The Programming Guidelines section explains how to find the amount of nodes in a given level. |
| Int | getWidth(BTNode\* root, int level)  Returns the width at each level |

1. BinarySearchTree.cpp was created, as it was not provided. Following functions were created

|  |  |
| --- | --- |
| **Return Type** | **Prototype of Function and Description** |
| BTNode\* | ccreateBSTNode(BTNode\*& parent, string data). This function creates a node in the Binary Search Tree |
| BTNode \* | ceiling(BTNode\* root, string key)  Returns the address of the node with the smallest key in the BST that is greater than or equal to key, regardless if key is present in the BST. Returns NULL if there is no such node. |
| BTNode \* | deleteNode(BTNode \* root, string key)  Removes the node containing the specified key (if key is present in the BST). Returns the address of the root of the tree without the deleted node. |
| BTNode \* | floor(BTNode \* root, string key)  Returns the address of the node with the biggest key in the BST that is less than or equal to key, regardless if key is present in the BST. Returns NULL if there is no such node. |
| BTNode \* | inOrderPredecessor (BTNode \* node)  Returns the in-order predecessor of the node supplied as a parameter. Returns NULL if the node is NULL or there is no in-order predecessor of the node. |
| bool | insert(BTNode \* root, string key)  If key is not already present, inserts key into the BST and returns true. If key is present, updates the count of key and returns false. |
| string | maxValue (struct BTNode\* node).  This function return the maximum node in tree rooted at node root |
| BTNode\* | minValueNode(struct BTNode\* node)  This function return the minimum node in tree rooted at node root |
| BTNode\* | search(struct BTNode\* root, string key)  Returns a node with the data passed in the argument as key |
| BTNode\* | findMaximum(BTNode\* root)  This function return the maximum node in tree rooted at node root |
| void | findPreSuc(BTNode\* root, BTNode\*& pre, BTNode\*& suc, string data)  Return the nodes by reference that are before and after the current node. |

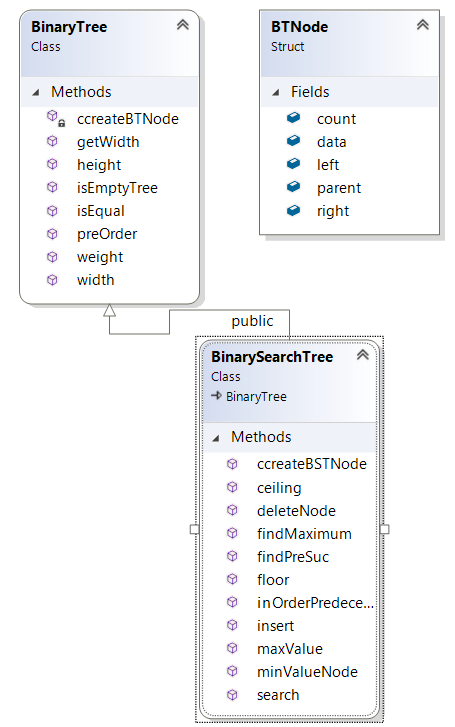
Requirement Specification:-

This assignment came with a specification document that was followed for building the project.

Source Code:

The source code was built as instructed in the assignment document.

Below is a snippet of the class diagram used in this project.



The source files used can been found in the attachment here corresponding to the class diagram above



Application and Implementation:

The project is executed from the Assignment2.cpp file as asked e in the assignment description document.

The application presents a console application where a menu is displayed as defined in the assignment document.

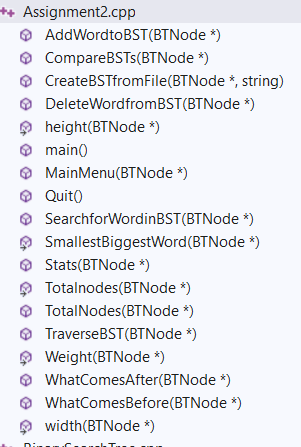
Two classes were created BinaryTree and BinarySearchTree.

BinarySearchTree class is a child class and extends its parent class BinaryTree.

Therefore, all functions at BinaryTree class is also available at BinarySearchTree class.

The source code is documented with the explanation of each functions written in comments just above it.

Below is the snippet of the Assignment2.cpp file



# Brief Overview of the functions in Assignment2.cpp:

## AddWordtoBST

Inserts a new word in the BinarySearchTree. Internally creates a new node. If the word already exists in any node in BST then the count of that node is incremented by one.

## CompareBSTs

Compares two BinarySearchTrees using their root nodes and tells if the two trees are identical or not

## Stats

Provides statically important information of a BinarySearchTree using the below functions within

1. **Height**: Calculate the depth of the tree
2. **Width**: Returns the width of the binary tree. The width of a binary tree is the maximum number of nodes at any level of the tree.
3. **Weight**: The weight of a binary tree is the amount of leaves in the tree
4. **SmallestBiggestWord** : Finds the smallest and the biggest word in the BInarySearchTree but in alphabetical order and not length wise
5. **TotalNodes** : Finds the total number of nodes in the binary search tree

## WhatComesAfter

Prints the word that is stored after the input word in the BinarySearchTree. Internally calls the function **findPreSuc** of the BinarySearchTree class.

## WhatComesBefore

Prints the word that is stored before the input word in the BinarySearchTree. Internally uses the function **search** of BinarySearchTree class to find the node of the input word. Then uses **inOrderPredecessor** function of BinarySearchTree class to find the previous node through which its data is picked.

## TraverseBST

Prints the string data of the entire BST using the **preorder** function of BinaryTree class

## SearchforWordinBST

Searches for a word in the BST using an internal function **search** of the class BinarySearchTree.

## DeleteWordfromBST

Deletes a word in the BinarySearchTree by deleting the node in the tree and updating the parent and child nodes accordingly. This is achieved by using an internal function of class BinarySearchTree, **deletNode.**

## CreateBSTfromFile

This function creates a BinarySearchTree from an input text file.

## Main

This is the main function and the entry point to the application.

## MainMenu

This function displays the menu of the application as shown below

Binary Search Tree (BST)

---------------------------------------------

1. Create BST from File

2. Add Word to BST

3. Delete Word from BST

4. Search for Word in BST

5. Traverse BST

6. What Comes Before Word in BST?

7. What Comes After Word in BST?

8. Compare BSTs

9. Statistics

Q. Quit

Please enter an option:

## Quit

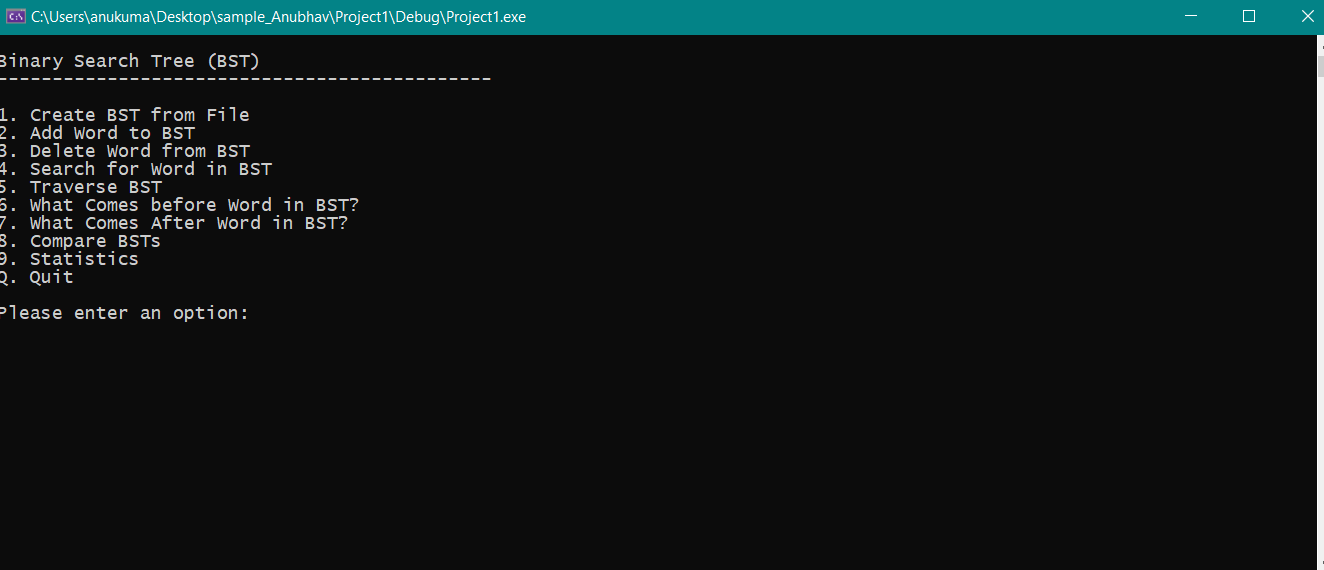
This function exits the console application and the program is terminated.

The implemented Assignmnet2.cpp file has been attached here for reader to analyze.

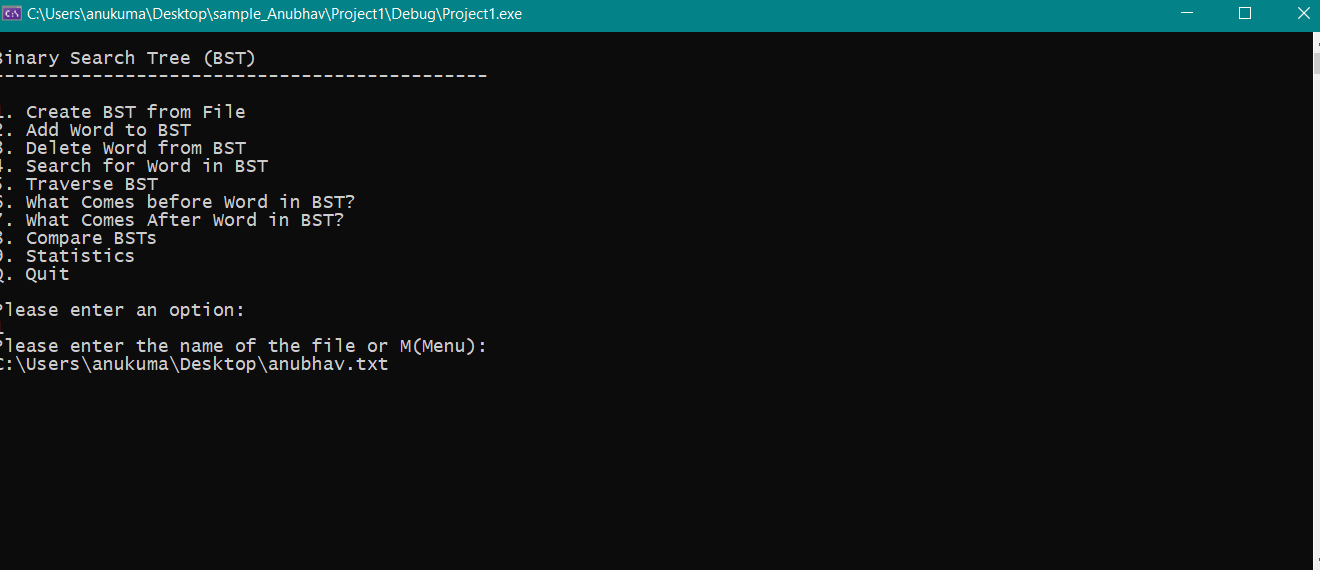


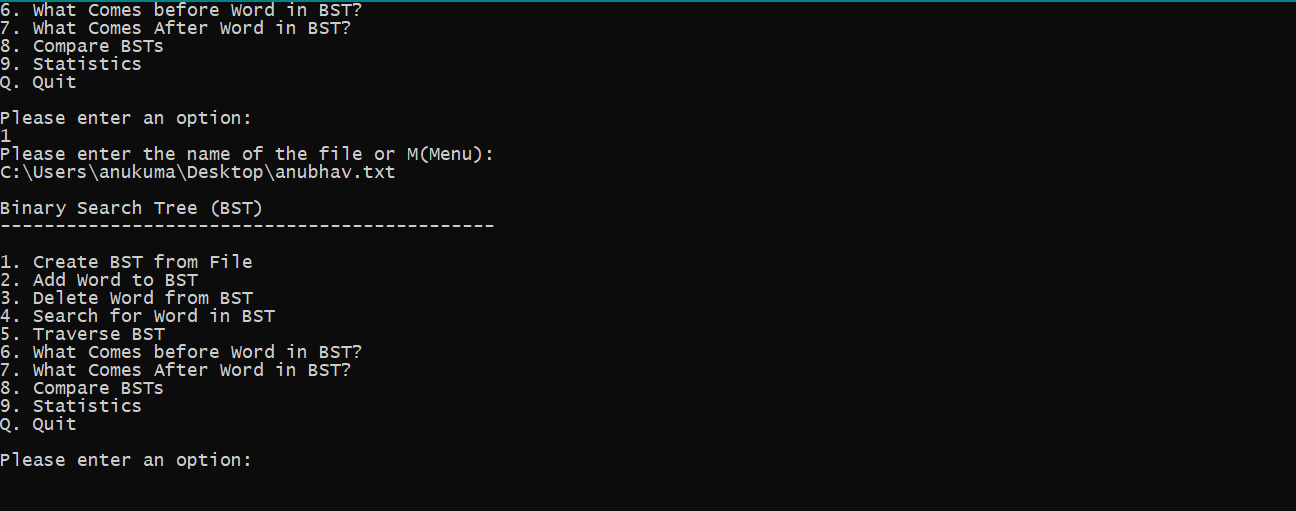
Output Screenshots

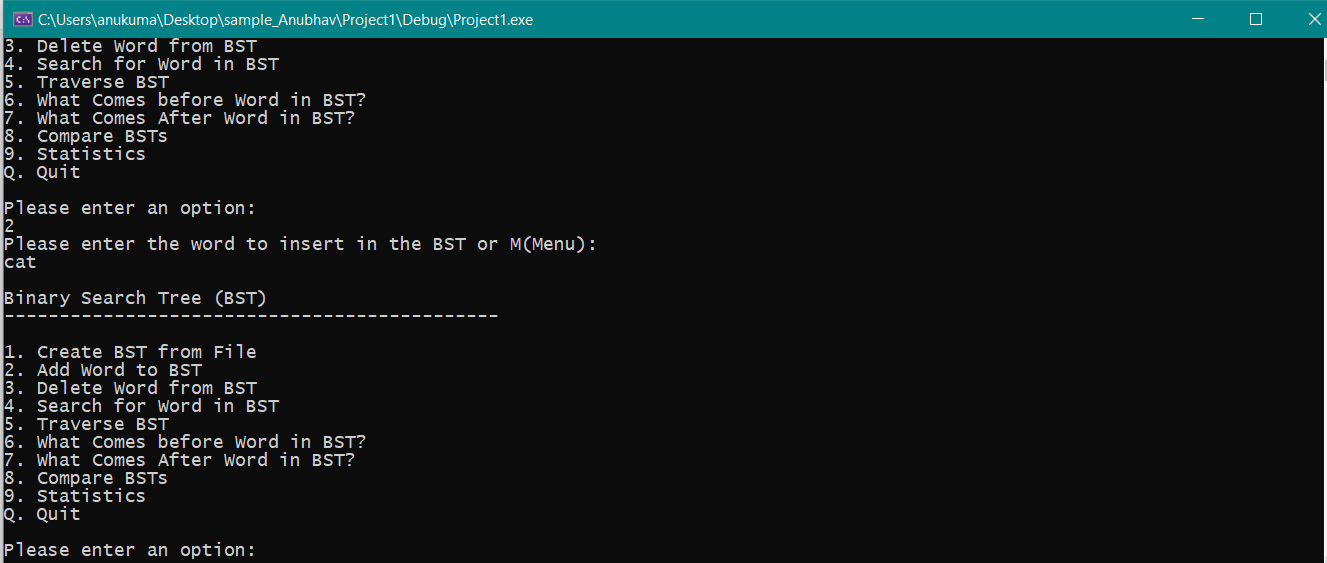
**The screenshots are attached in sequential order of execution**



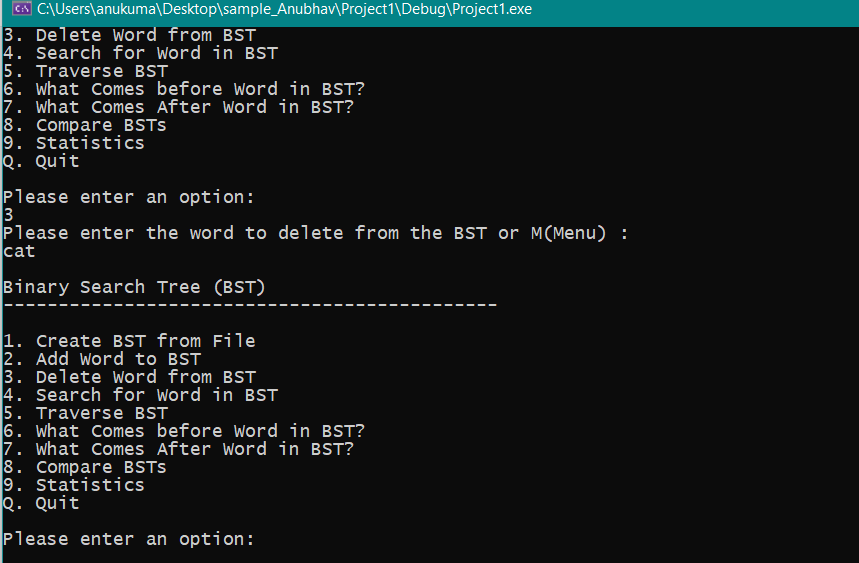
**Create a Binary Search Tree using an input text file**



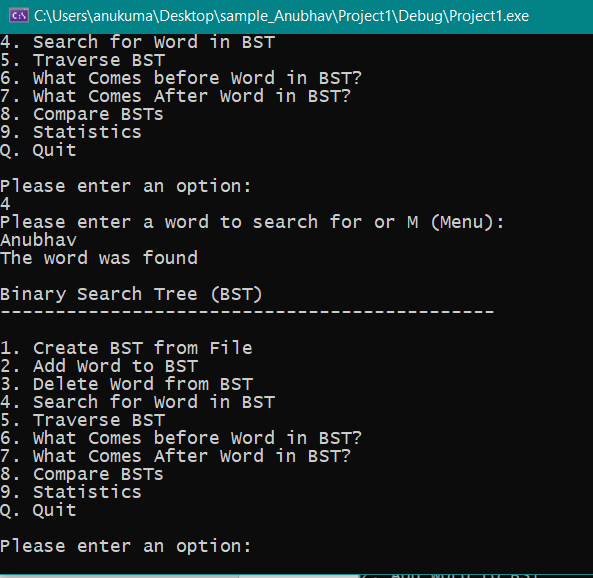


**Add a word: cat**

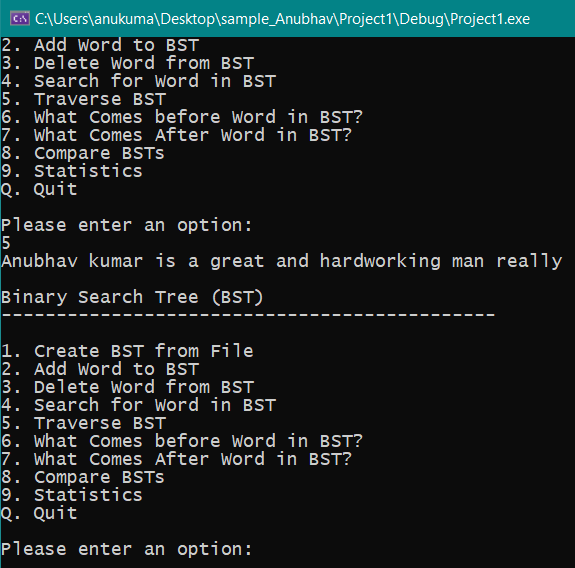
**Delete a word: cat**



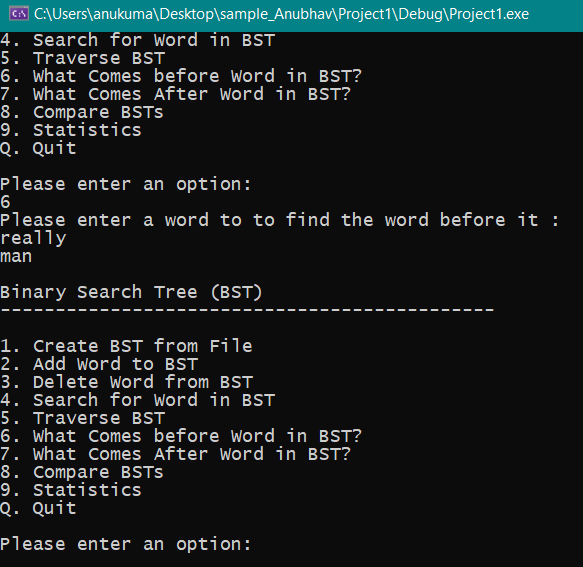
**Search for a word: Anubhav**



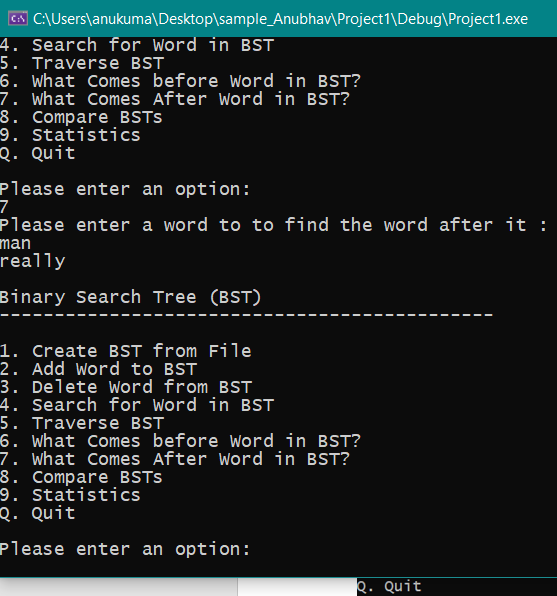
**Traverse and print the BST**



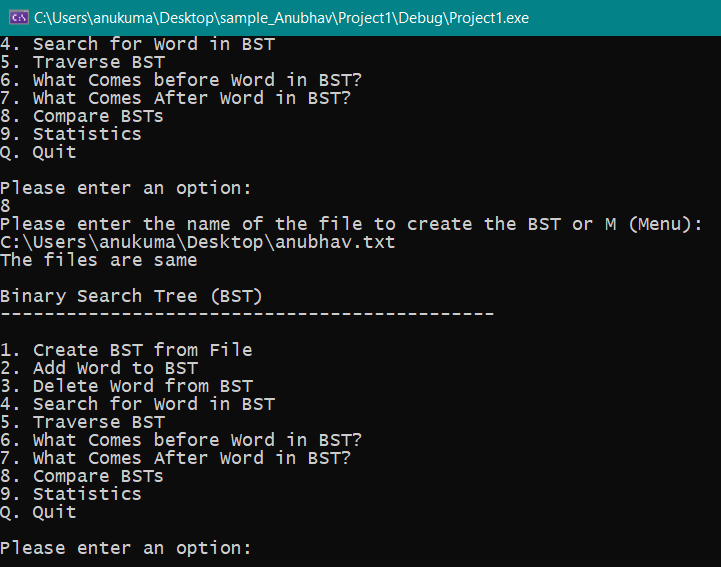
**What comes before: really**



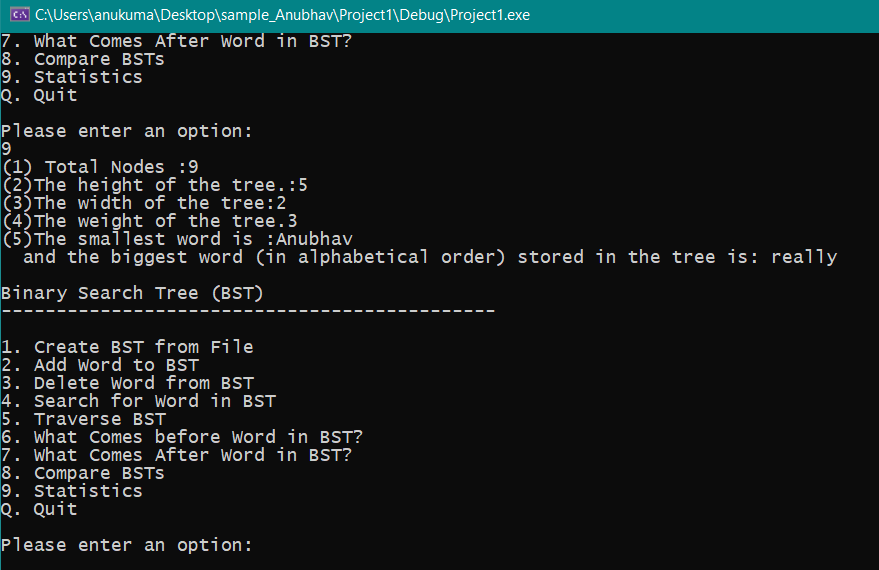
**What comes after: man**



**Comparing the BST: passing the same input file again to check**



**Stats of the BST**



**Quitting the program**

