|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test no | Test description | Input | Expected output | Actual output | Pass/Fail |
| 1 | To see if the skeleton code provided populated the root node as required. | The WriteLine method was added to the skeleton code under where the root node was set. This allowed for the item in the root node to be the correct number. | The data held in the root node was printed out. | The data held in the root node is to be printed. | Pass |
| 2 | See if skeleton code works and populates the binary tree as stated to then carry out relevant tasks. | WriteLine methods were added to where each node was assigned a position in the nested conditional statements in the InsertNode() method. | The binary tree is to be printed out in order with 12 nodes. | 30 nodes printed out. | Fail |
| 3 | Ensure the loop which populates the respective nodes with conditional statements, breaks where needed so values are not repeated which made test two fail. | The break statement was added in each conditional statement within the loop, this was to break the loop, meaning the loop only needed to run once for each node. | The binary tree, printed out in order with 12 nodes. | The 12 nodes printed out. | Pass |
| 4 | Ensure the tree is printed in order, pre-order and post order. | Three methods were added to print the tree in the requested orders. The pseudocode from the lecture slides showed, for each traversal, how to use recursion to print out the required nodes. | All data in nodes are printed out in three specific orders, pre, in and post order. | Traversal of the first three nodes only. | Fail |
| 5 | Ensure the tree is printed in order, pre-order and post order. | The break statements added to the else statements in test 3 were removed from the insert node method, realised that the insert node method is supposed to loop over into the else statement to populate the tree properly. | All data in nodes are printed out in three specific orders, pre, in and post order. | All three traversals are printed in the desired order. | Pass |
| 6 | Allow the user to use the assignment tree or choose to input a random tree. | userChoice method was added, to allow the user to input a custom tree or to choose the coursework tree. Asks the user to enter the word “tree” for the assignment tree or to press any key to enter a custom tree. If the user chose the coursework tree, then there is a conditional statement with the course work tree with the 12 numbers stored and ready to be returned to the main. Otherwise, if the custom tree was chosen then it first asks the user how many nodes they would like to enter, this was read with readLine(), converted to an int and assigned to a variable. This variable would then be used to count the loops up to the users desired size tree. A loop was added and went up to the user’s tree size choice by incrementing the counter variable by one each loop up to that number. Each loop prompts the user for a number converting and storing the numbers. All inputted items for nodes were stored and returned to the main, the size of the array storing the nodes for the custom tree was decided at runtime by the user. | To first ask the user which tree they would like to use, the user to choose the coursework tree and to print out the assignment in the correct order with the correct traversals. | Asks user which tree they would like to use, coursework tree chosen, everything printed out as expected. | Pass |
| 7 | Allow the user to use the assignment tree or choose to input a random tree. | The user chooses a custom tree, inputs the same 12 nodes for the assignment tree. | Ask the user for coursework tree or custom, user to choose custom, the program asks the user how many nodes and the user to enter number 12, prompt user for 12 numbers and then print out same traversals as test no. 6. | Output was as expected. | Pass |
| 8 | Find item 20 traversing the tree with a depth-first traversal algorithm, non-recursively, by using a stack. | Added the preorder DFS algorithm, to find item 20 using a stack, using the pseudocode from the slides as a base. Next, the stack and list were created, and the stack had the root pushed in, ready to enter the loop. In the loop the first thing was to print the root node which was the first visit as this needed to be done to print the path, it was also added to the list to print the reverse-path at the end. Then the add right child and left child, in that order, was added to the stack, so the left child would be at the top to be printed and if not (due to it being a leaf) the right child would be printed. With the item at the top of the stack added to the list for later. If item 20 was found an if statement would break the loop with the break keyword, this was added before the right and left child conditional statements. Then the list was reversed with the .Reverse() method and printed to show the path back from the leaf node, this was written outside of the loop so it would print once the loop was broken. | A preorder traversal printed out of the tree up to item 20, then the path back to the root node from item 20 of the same traversal. | Loops over item 13 which is a leaf node. | Fail |
| 9 | Find item 20 traversing the tree with a depth-first traversal algorithm, non-recursively, by using a stack. | Added pop() to pop what is on the top of the stack so the right node which is under leaf node 13, with item 22 is printed out. This was done after adding the top element in the stack to the list to show the backwards path. | A preorder traversal is to be printed out of the tree up to item 20, then the path back to the root node from item 20 of the same traversal. | Output is as expected. | Pass |
| 10 | Find item 30 traversing the tree with a breadth-first traversal algorithm, non-recursively, by using a queue. | Added the breadth-first search (BFS) algorithm, to find item 30 using a queue, using the pseudocode from the slides as a base. Next, the queue and list were created, and the queue had the root pushed in, ready to enter the loop. In the loop the first thing was to print the root node which was the first visit as this needed to be done to print the path, it was also added to the list to print the reverse-path at the end. Then the add the left child and right child, in that order, was added to the stack, so the left child would be at the front so it could be printed and if not (due to it being a leaf) the right child would be printed. With the item at the front of the queue added to the list for later. If item 20 was found an if statement would break the loop with the break keyword, this was added before the left and right child conditional statements. Then the list was reversed with the .Reverse() method and printed to show the path back from the leaf node, this was written outside of the loop so it would print once the loop was broken. Added dequeue() to remove what is at the front of the queue so the right node which is under the leaf node was printed out. This was done after adding the front element in the queue to the list to show the backwards path. | Use a breadth-first search to print out all the items in nodes until item 30 was found. When item 30 was found print the bath back to the root node, |  | Pass |
| 11 | Find items 20 and 30 traversing the tree, using recursion. | A method was created in the MyBinaryTree class. Within the method, a conditional statement was used which had; a base case which was written with the if statement; an else if statement to check if item 20 was found, if it was found this had the recursive call which recursively called the method with the left child then printed item 20 and finally ran over the method recursively with the right child; the next else if statement checked if the node passed as a parameter recursively to the method was equal to item 30, if it met this condition the recursive call within called the function passing the left child and then printed item 30, this did not have a recursive call for the right child as this was the last item to be found so it would not print anything else; the final else statement was used as long as item 20 and 30 had yet to be found, it recursively called the left child nodes then printed out the item when the last was met and then printing the right child node. | Use a recursive traversal to find both items 20 and 30, while printing out every visited node along the way. |  | Pass |
| 12 | Allow the user to accidentally input a letter instead of a number and use the try-catch statements to recursively call the method for the user to try again. | Here, the code within the else statement to create the custom tree was all put into a try statement, if there was a problem with the code in here such as the user inputting a decimal, word or letter when a whole number was required, then the code in the catch statement needed to be executed, this told the user to only enter whole numbers and recursively called the method asking the user to choose the assignment tree or custom. | Use the try-catch method to allow the user to start over if they input a letter or decimal number and not break the program. |  | Pass |