**ASSIGNMENT 2**

**BINARY SEARCH TREE**

**DUE DATE: WEEK 14, MONDAY (19/4/2021) BEFORE 5.00PM**

**1.** **OBJECTIVE**

1. To review and strengthen the concept of trees.
2. To practice the recursion techniques.
3. To review different ways of traversing a binary tree.

**2.** **LABORATORY**

You can do this assignment individually or in a group of two persons (from either the same or different practical session).

**3.** **EQUIPMENT**

Hardware: The PCs in the lab for conducting your practical.

Software: The Microsoft Visual Studio 2017. Your programming will only be tested

by the markers using Visual Studio 2017.

**4.** **TASKS:**

The student records are stored in a Binary Search Tree for fast access and searching. This assignment handles:

1. The input of student records from a text file using C++.
2. The storage of student records into a BST.
3. Searching for a student record in a BST.

Use the given source codes in this assignment and add the following functions below:

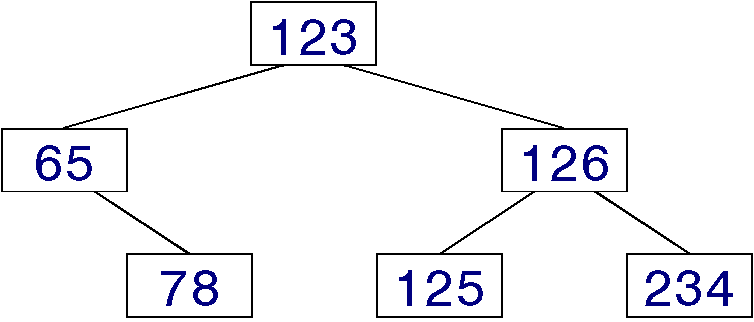
(a) Write a function **bool readFile(char \*filename, BST \*t1)**in main that reads student records from a text file and stores them according to the **student id** in a binary search tree (BST). The function prints to screen the number of student records successfully read. The function return false if file not successfully open and true otherwise. Refer to sample input text file student.txt given. An example of a student record BST is shown in Figure 3.1. Its data in the text file format, **student.txt** is included in the sample textfile folder. In the tree, each node stores a student record. The **key** for the BST is the **id** of each student.

(b) Write a function **bool BST::deepestNodes()**in BST class that finds and prints all the nodes at the deepest level. For example, the deepest nodes of tree1 in Figure 3.2 are 2, 7 and 12 while tree2 has only 21. (**Hint**: You can either use height of tree to look for deepest). Function will return false for empty tree and true otherwise.

1. Write a function **bool BST::display(int order, int source)** in BST class that traverses a BST tree recursively and print the content of every node. The order variable is to specify if user want to print in ascending or descending order according to id. Use order = 1 for ascending and order = 2 for descending. The source variable is used to specify where to print the output. Use source = 1 to print to screen and source = 2 to print to a file name “student-info.txt”. **(Hint: To print in ascending visit left son and then right son. As to print in descending visit right son and then left son.)** Function will return false for empty tree and true otherwise.
2. Provide a function **bool BST::CloneSubtree(BST t1, type item)** that will clone a subtree in t1 where the root of the subtree contain value item. For example, if ***t2.CloneSubtree(t1, 126);*** is called, then the subtree in t1 (t1 is tree in Figure 3.1 and t2 is cloned subtree in Figure 3.1 (a)) where the root contain node 126 will be cloned and store in t2 as shown in Figure 3.1 (a) below. t1 will not change after function call and t2 must be an empty tree before cloning. Display t1 and t2 using preOrderPrint() after function call. Function will return false if t1 is empty and true otherwise.
3. Write a function **bool BST::printAncestor(type item)** to print all the ancestors for item. Table below shows the ancestor for each found in the tree. Print message “There is no ancestor for this item.” if item is the root item.

|  |  |
| --- | --- |
| **Nodes** | **Ancestors** |
| **28** | **No ancestor** |
| **22** | **28** |
| **19** | **22 , 28** |
| **23** | **22, 28** |
| **21** | **19, 22, 28** |

1. Write a function **bool BST::printSpiral()** that will print all the nodes in a BST in spiral order. For example, the spiral order of nodes for the BST in Figure3.1 below is **123 65 126 234 125 78**. Function return false for empty tree and true otherwise.



**Figure 3.1: An example of a binary search tree.**

**126**

**125**

**234**

**Figure 3.1(a): An example of the cloned subtree (t2)**

|  |  |  |
| --- | --- | --- |
| tree1  root |  | tree2  root |

**Figure 3.2: Tree1 and Tree2**

(g) Write a function **int menu()** that displays all the above functions (as shown below) for test run.

1. **Read data to BST**
2. **Print deepest nodes**
3. **Display student**
4. **Clone Subtree**
5. **Print Ancestor**
6. **Print Spiral**
7. **Exit**

Although each of the above tasks is to be implemented as a single function, it may be necessary to create additional sub-function(s) to handle portions of the function. This is especially true if the original function is too long or contains more than one functionally related group of statements. When you write a function, remember that this function is to work for all possible inputs. Not on just your test inputs. You must test for all conditions that might possibly arise; print out appropriate messages as needed.

A text file **student.txt** of student records can be downloaded from WBLE. It is created to assist the development and testing of your program. Its content and even the file name can be changed (while still adhering to the same data format) during grading in order to test the robustness of your program. The objective is to ensure that you do not hardcode your program to work exclusively for the given sample student records.

**5.** **ERROR HANDLING**

You can assume that the input is always correct; thus no input validation is required.

**6.** **REPORT**

1. This assignment is due according to the due date posted on WBLE. You must do the submission through WBLE. You should create one directory using your name and then copy all your source code files (\*.h, and \*.cpp) into this directory. You should also provide a readme file (see description below) and all input data files that you test your program. Apart from these, you should not have irrelevant source files in the directory. Compress this directory into a **zip** file and submit only this zip file to WBLE.
2. The readme file should contain the following information:
3. Your name and group member name (if any)
4. Known errors and bugs of your code. (if any)

**7. ACADEMIC HONESTY AND COLLABORATION**

Cooperation is recommended in understanding various concepts and system features. But the actual solution of the assignments, the programming and debugging must be your individual work, except for what you specifically credit to other sources. (Your grade will be based on your own contribution.) For example, copying without attribution any part of someone else’s program is plagiarism, even if you modify it and even if the source is a textbook. You can document the credit to other sources at the start of your program code listing. The University takes acts of cheating and plagiarism very seriously: first time violators may fail the coursework component of UCCD1024. Any wholly (or partly) copied (or being copied) programs will receive zero mark.

**9.** **REFERENCES**

[1] Text and reference books for UCCD1024.

[2] http://msdn.microsoft.com/en-us/visualc/aa336395