6.2 BST Pretty Print (graded assignment)

Due 15 Oct 2020 by 23:59 **Points** 10 **Submitting** an external tool

For this assignment, you extend your BST from Assignment 6.1 by a function *prettyPrint()*. Admittedly, beauty is in the eye of the beholder, so "*pretty*" might be debatable. In any case, your program will display the structure of the BST that is built up when inserting nodes.

Start with your implementation from 6.1, the simple BST. Extend its interface by a function *prettyPrint()* as follows:

```
class BST{
    public:
        BST();
        ~BST();
        void insertKey(int newKey);
        bool hasKey(int searchKey);
        std::vector<int> inOrder();
        int getHeight();
        void prettyPrint();
    private:
        // your implementation goes here
};
```

prettyPrint() writes to *std::cout* a table that represents the tree structure. The table has as many columns as there are nodes in the BST. The number of rows is equal to the height of the BST. The root node is printed in the top most row. There is no overlapping of subtrees, so the left subtree of the root is shown completely to the left of the root. And the right subtree is shown completely to the right of the root. This property holds recursively for all nodes in the BST. The direct children of a node in row n are shown in row $n+1$ of the table. All table elements are surrounded by lines (made up by the characters '|' and '-'). If the BST is empty, *prettyPrint()* should print nothing at all. This is best illustrated by the following examples of correct runs of the program:

```
Enter the numbers to be stored: 0 4 -4 6 -6 -3 3 -7 9 q
The numbers in sorted order: -7 -6 -4 -3 0 3 4 6 9
---------------------------------------------
|    |    |    |    |   0|    |    |    |    |
---------------------------------------------
|    |    |  -4|    |    |    |   4|    |    |
---------------------------------------------
|    |  -6|    |  -3|    |   3|    |   6|    |
---------------------------------------------
|  -7|    |    |    |    |    |    |    |   9|
---------------------------------------------


Enter the numbers to be stored: 1 2 3 4 5 q
```

```
The numbers in sorted order: 1 2 3 4 5
-------------------------
|   1|    |    |    |    |
-------------------------
|    |   2|    |    |    |
-------------------------
|    |    |   3|    |    |
-------------------------
|    |    |    |   4|    |
-------------------------
|    |    |    |    |   5|
-------------------------

Enter the numbers to be stored: 8 6 4 2 99 26 30 31 q
The numbers in sorted order: 2 4 6 8 26 30 31 99
---------------------------------------
|    |    |    |   8|    |    |    |    |
---------------------------------------
|    |    |   6|    |    |    |    |  99|
---------------------------------------
|    |   4|    |    |  26|    |    |    |
---------------------------------------
|   2|    |    |    |    |  30|    |    |
---------------------------------------
|    |    |    |    |    |    |  31|    |
---------------------------------------
```

All table cells are equally wide (in the examples above, 4 characters wide). To achieve this, you can look up *std::setw(width)* in the zyBook, Chapter 13.2.

Before you begin, you should think about what kind of information you need from the BST in order to print this kind of table. Also think how to store this information. It is fine if you add more data items to the nodes of the BST. You may also want to use something like the Matrix class, the example developed in class of module 5. It is OK (=**not** considered plagiarism) to use the code of this Matrix class if you prefer so. You can submit your program either in one or in many files.

For retrieving information from the BST, you should think of possible traversals of the nodes (pre-order, in-order, post-order, zyBook Chapter7.4) and how they can build up the information. It is fine of your *prettyPrint()* traverses the tree more than once.

One important thing: your BST must **no**t create **memory leaks**. But your code from Assignment 6.1 that you reuse here should already take care of this.

Like with Assignment 6.1, the keys stored in the BST must be stored in node objects that have been dynamically created using *new*. Next to the tree, you might use additional storage, like a Matrix. (And the use of arrays for this assignment will be punished by public whipping!)

This tool needs to be loaded in a new browser window

Load 6.2 BST Pretty Print (graded assignment) in a new window