2-3-4 Tree Class Implementation
Cpt S 223 Homework Assignment
by Evan Olds

Assignment Instructions (read all instructions *carefully* before you write any code):

In this assignment you will implement a 2-3-4 tree. You will need to process commands that allow to you add items to the tree and get information about the tree in various ways. The next homework assignment may build on top of this one and require you to do deletion, but that is not guaranteed at this point. It can vary from semester to semester whether or not this assignment is 2 parts.

You will implement the 2-3-4 tree as a template class, but for this assignment you will always use **Student** as the template type. Implement a simple Student class or structure that consists of a string for a student name and an integer for an ID number. As is mentioned a few more times later on in this assignment, the ID is what you use to compare one student structure to another. Implement overloaded comparison operators in the Student structure so that the 2-3-4 tree code can compare them.

You MUST use the preemptive split scheme that we discussed in class. Your output must EXACTLY match the expected output for the application. There is no variation expected across implementations if you correctly followed the preemptive split rules.

Set up a main function to read commands from stdin:

In this assignment you will read simple commands from standard input. You will display your name and ID number on a single line to start with and then enter the command processing loop. There will be one command per line, so your main function (pseudo-code) will be something like:

```
cout << "Student Name, ID#\n";
while (true)
{
  string cmd = ReadLine();
  if (cmd.StartsWith("cmd1name")) { HandleCmd1(); }
  else if (cmd.StartsWith("cmd2name")) { HandleCmd2(); }
  else if (cmd.StartsWith("quit")) { break; }
}
cout << "Done\n";
```

Command strings will be much like C function calls in terms of syntax. You must support the following commands:

add(student name,student_id) : Adds a student structure to the tree. The name string may consist of any characters except for commas, parentheses, and non-displayable characters. The student ID is an integer that is used as the key for comparing one student structure to another as you build the tree.

- The is no space after the comma and before the student ID
- Do not display any output when processing this command
- Do NOT allow addition of two students with the same student ID
- DO allow addition of two students with the same name and different ID numbers

countLevels() : Displays an integer value indicating the number of levels in the tree. An empty tree has zero levels, a tree with only node, which implicitly is the root and the root is a leaf in this case, has 1 level, and so on.

displayInOrder() : Displays the contents of the entire tree, in order by student ID numbers. For each student the name is displayed, followed by a comma, then the student ID, and then a semicolon. There is a semicolon after every name,ID pair including the very last one on the line.

- If the student name contained spaces then those spaces will be displayed, but other than that there must not be any spaces on the line

getName(student_id) : Does a search within the tree for a student with the specified ID number and displays their name if found. If no student with the specified ID exists within the tree then then **(student not found)** is displayed.

getLevel(student_id) : Finds a node within the tree that contains a student with the specified ID number, then displays the level of the tree that this node is on. The root node is on level 0, its immediate children are on level 1 and so on. If the tree does not contain a student with the specified ID, then **-1** is displayed.

countValuesInNode(student_id) : Finds a node within the tree that contains a student with the specified ID number, then displays an integer value for the number of values that reside within that node. This will be **1**, **2**, or **3** if the node is found, or **-1** if the tree does not contain a student with the specified ID.

quit() : Breaks the command processing loop and terminates your program. Display the line “Done” as the very last thing before returning from main.

Notes:

- Put comments in your code! Give a brief explanation for each function at a minimum.
- Look at the sample input/output files in the zip to get an idea of what the commands and their corresponding outputs look like. As usual, develop your own additional test cases on top of these, since they are samples only and not meant to rigorously test your code.
- Your code must compile with g++ and run on Linux.