operator, whereas, the **info** field of a leaf node contains an operand. In this assignment, we represent all the operands in character form. Even though the binary expression tree could be built on top of the binary tree, in this assignment you will use the binary tree for binary expression trees. Hence, the representation and primary operations of the binary expression trees are the same as the ones given in prog6a.

Enhanced Operations on Binary Tree: You need to declare, specify, and implement the following enhanced operations on the binary expression tree.

```
bool AllConsts (BTree  & T);
int EvalTree (BTree & T);
void ReduceTree (BTree & T); and
```

You MUST USE SwapSubTrees in ReduceTree.

NOTE: You do not have to check for errors in this procedure. In general, when a reduction of the form constant op constant is performed, the result may contain more than one digit. Since the **info** field of a node is of type character, you can assume that such a reduction always results in a single digit.

Files: Copy your folder prog3a. Rename it prog3b.

Input: The program inputs include

1. Name of the input file.
2. File containing values to put into the tree.

Test your program on two input data files. One data file will be emailed to you (prog3b.txt). You should MAKE up another data file to test your program.

Output: All output is to the screen. Output consists of three traversals of the tree to display the items in prefix, fully parenthesized infix, and postfix notation. If the tree contains only operators and constants, the result of evaluating the expression tree is printed. Then the fully parenthesized infix for of the reduced expression is printed.

SEE THE ATTACHED FILE FOR A *SAMPLE* OF INPUT AND OUTPUT FORMAT.

Turn In: Printouts of BTree.cpp (with fixes) and prog3b.cpp files.
Both input files.
Both screen dumps of executions.
All project files on some storage media.