Lab 11

6 November 2014

In today's lab, you'll write some code that builds, and then traverses, binary trees. For simplicity, we'll write trees that only hold characters.

Tree nodes

First, create a class `BinaryNode`, capable of representing any of the nodes in a tree. It will have three instance variables: a `char`, holding the value that is stored at a particular node, and two pointers to `BinaryNode` (one to the left child, if any, and one to the right child, if any).

By now you should be getting comfortable with writing your own classes, so I won't recap those instructions here; look back at previous labs to help you remember how. Don't forget these parts:

- A private section with the three instance variables
- At least one public constructor (in this case, possibly more than one)
- Methods to get/set values of the instance variables

Examples

In a notebook, draw out the following three trees:

- `emptyTree`, which is simply set to `nullptr`
- `simple`, which points to a node containing `'Q'` whose left child contains `'X'` and right child contains `'Z'` (and no further descendants)
- `tree5`, pointing to a node that is the root of a small tree that contains the five letters `'A'` through `'E'` and is relatively balanced (i.e. not just a line)

and start a unit test file whose `fixture` includes at least those examples. Note that functions that work on a `BinaryNode*` should, in general, work on `emptyTree`, since it's a perfectly valid example of a (empty) tree.

Functions

In a separate file `BinaryNodeFunctions.cpp`, write three functions `prePrint`, `postPrint`, and `inPrint`, each of which takes a `const BinaryNode*` argument and an `ostream&` argument, and prints the given tree to the given stream. The three functions should each recursively print the tree contents (if any) to the stream, left-to-right, in the correct traversal order.

Write the functions' prototypes in `BinaryTreeFunctions.h`, and include that in your `.u` file. When you test the traversals, use an `ostringstream` to check the output (as we did with `Maze` and `Card`).

Two other functions

Write and test the following recursive functions also:

- `count` counts the total number of nodes in the subtree rooted at a given `const BinaryNode*` (including the node itself, if any).
- `contains` determines whether the subtree rooted at a given `const BinaryNode*` includes a given character. (Note that it does not rely on binary search order—`simple`, for instance, isn't in that order.)

Handing in

Hand in your work electronically as `lab11`, by 4pm on Wednesday.

RUBRIC

1 Present in lab
1 Readme with all required information
Class and examples
1 General class definition, instance variables ♣
1 Constructor(s) defined and used ♣
1 Specified trees created
Function definitions and tests
1 All five required functions have correct test cases (fail ok) ♣
1 One recursive traversal is implemented correctly ♣
1 All three traversals are implemented correctly ♣
1 `count` ♣
1 `contains` ♣