A Recursively-Defined Tree Class*

Jeffrey D. Oldham

2000 Apr 05

We explain how to use the recursively defined `Tree` class. The implementation file and example use code are available.

1 The Tree Class

Using the templatized `Tree` class requires specifying the type of the items stored in the tree. In this document, we will use trees of `int`s.

A *tree* is

- either an empty tree
- or an item with two subtrees.

1.1 Empty Trees

To create an empty tree, use

```
Tree<int>()
```

To check if a tree `T` is empty, use

```
T.empty()
```

which returns the boolean value true if the tree has no items and otherwise false. Using `T` in a place where a boolean is expected yields true if the tree is not empty and otherwise false. For example, `if (T) cout << "tree is not empty\n";`.

1.2 Nonempty Trees

To construct a tree with one item, e.g., 3, use

```
Tree<int>(3,Tree<int>(),Tree<int>())
```

To construct a tree with one item, e.g., -17, and two tree children `L` and `R`, use

```
Tree<int>(-17,L,R)
```

To obtain a tree `T`'s item, which has `int` type, use

```
T.item()
```

To obtain the left subtree, which has `Tree<int>` type, use

T.left()

To obtain the right subtree, which has `Tree<int>` type, use

T.right()

To check if a tree is nonempty, negate the result of checking for an empty tree.

1.3 Example

The example use code illustrates using `Trees`.

```
// Oldham, Jeffrey D. and Berna L. Massingill
// 2000 April 04
// CS1321

// Simple example of use of Tree class.

// (c) 2000 Berna L. Massingill and Jeffrey D. Oldham
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
#include <iostream.h>
#include <assert.h>
#include "tree.h"

typedef Tree<int> iTree;
typedef Tree<char> cTree;

// Function to print contents of tree -- root, then left subtree,
//   then right subtree, indented to show nesting.
// Pre:  indents is the number of tabs to print at the start of each
//         line (in addition to the ones used to show nesting).
// Post:  tree "t" printed to output stream "out".
template <typename ItemType>
void printTree(ostream & out, const Tree<ItemType> &t,
               const int indents = 0);

// Main program.
int main(void)
{
  iTree it0;                    // empty tree of integers.
```

```
assert (!it0);

iTree it1(1);                  // tree with a single node (1).
assert (it1.left().empty());   // its left subtree is empty.
assert (it1.right().empty());  // its right subtree is empty.
assert (it1.item() == 1);      // its root item is 1.

cout << "A simple tree of ints:\n";
printTree(cout, it1);
cout << endl;

iTree itNonTriv(0,
                iTree(1, iTree(2), iTree(3)),
                iTree(4,
                      iTree(5, iTree(6), it0),
                      iTree(7, it0, iTree(8))
                      )
                );             // moderately complicated tree.

cout << "A moderately complicated tree of ints:\n";
printTree(cout, itNonTriv);
cout << endl;

cout << "Its root item again:\n";
cout << itNonTriv.item() << endl;
cout << "Its left subtree again:\n";
printTree(cout, itNonTriv.left());
cout << "Its right subtree again:\n";
printTree(cout, itNonTriv.right());
cout << endl;

cTree ct0;                     // empty tree of characters.
assert (ct0.empty());

cTree ct1('a');                // tree with a single node ('a').
assert (ct1.left().empty());   // its left subtree is empty.
assert (ct1.right().empty());  // its right subtree is empty.
assert (ct1.item() == 'a');    // its root item content is 'a'.

cout << "A simple tree of chars:\n";
printTree(cout, ct1);
cout << endl;

cTree ctNonTriv('a',
                cTree('b', cTree('c'), cTree('d')),
                cTree('e',
                      cTree('f', cTree('g'), ct0),
```

```
                      cTree('h', ct0, cTree('i'))
                      )
                );            // moderately complicated tree.

  cout << "A moderately complicated tree of chars:\n";
  printTree(cout, ctNonTriv);
  cout << endl;

  cout << "Its root item again:\n";
  cout << ctNonTriv.item() << endl;
  cout << "Its left subtree again:\n";
  printTree(cout, ctNonTriv.left());
  cout << "Its right subtree again:\n";
  printTree(cout, ctNonTriv.right());
  cout << endl;

  cout << "\nThat's all, folks!\n";
  return 0;
}

// Function definitions.

// Helper function.
void printTabs(ostream & out, const int n)
{
  for (int i = 0; i < n; i++)
    out << '\t';
  return;
}

// Function to print tree.
template <typename ItemType>
void printTree(ostream & out, const Tree<ItemType> &t,
               const int indents)
{
  if (t.empty()) {
    printTabs(out, indents);
    cout << "empty tree\n";
  }
  else {
    printTabs(out, indents);
    cout << "root = " << t.item() << endl;
    printTabs(out, indents);
    cout << "left subtree = \n";
    printTree(out, t.left(), indents+1);
    printTabs(out, indents);
    cout << "right subtree = \n";
```

```
    printTree(out, t.right(), indents+1);
  }
  return;
}
```

2 Logistics of Using the Code

To use the linked tree class with a program you wrote, copy the implementation file to the directory containing the program's C++ code. One way to do this is to use the "Save As..." item on a WWW browser's file menu.

In your C++ program, add the line

```
#include "tree.h"
```

near the other header inclusions. See also the sample program.