Simple Tree in VC++

Posted by Alejandro G. Carlstein Ramos Mejia on October 15, 2010 November 12, 2010 About Programming / Algorithms / C++

Simple Tree in Visual C++

This code was compiled and tested on a Windows.

NOTIFICATION: These examples are provided for educational purposes. Using this code is under your own responsibility and risk. The code is given ‘as is’. I do not take responsibilities of how they are used.

mainTree.cpp:

```
#include 'standard.h'
#include 'tree.h'

using namespace std;

//Print the divider to the specified file
void PrintDivider ( ofstream& fout,
        char symbol,
     int numOfSymbols,
     int section );

//Print the count to the specified file
void PrintCount ( ofstream& fout,
      int count );

int main(void)
{
 //declare the files
 ifstream fin;
 ofstream fout;

 //Set the task number to -1
 int section = -1;

 //Open the files
 fin.open('tree.txt');
 fout.open('treeOut.txt');

 //Create an instance of the class
 TreeClass tree;

 //Declare variables
 int numOfNodesInTree;
```

```
 //print empty tree message
 tree.PrintTree ( fout );
 PrintDivider ( fout,
             '*',
       63,
       section++);

 //Insert values from the input file
 tree.CreateATree ( fin );

 //Print the tree nodes on one line
 tree.PrintTree ( fout );
 PrintDivider ( fout,
             '*',
       63,
       section++);

 //Calculate the number of nodes in the tree
 numOfNodesInTree = tree.CountNodesInTree();

 //Print the count
 PrintCount ( fout,
           numOfNodesInTree );
 fout << endl;

 //Close the files
 fin.close();
 fout.close();

 return 0;
}

//Print the divider to the specified file
void PrintDivider ( ofstream& fout,
        char symbol,
     int numOfSymbols,
     int section )
{
 fout << endl << endl;
 fout << '#' << setw(2)<< section << BLANK;
 fout.fill( symbol );
 fout << setw( numOfSymbols ) << BLANK;
 fout << endl;
 fout.fill( BLANK );
 fout << setw(3)<<BLANK;

}

//Print the count to the specified file
void PrintCount ( ofstream& fout,
      int count )
{
 fout << ' Number of Nodes currently in the tree are: '
```

```
   << count << endl;
}
```

standard.h:

```
#ifndef standard_h
#define standard_h
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;
const char BLANK = ' ';
#endif
```

tree.h:

```
#ifndef tree_h
#define tree_h
#include 'standard.h'
struct nodeType
{
 int value;
};
struct treeType
{
 nodeType  data;
    treeType* left;
 treeType* right;
};
typedef treeType* treePtr;
class TreeClass
```

```
{
 private:
  treePtr root;
  //Insert the node in ascending order
  void InsertNode ( treePtr& ptr,
                    nodeType aNode );
  //Create memory, insert data into a tree node
  treePtr GetANode ( nodeType ANode );
  //Delete a leaf node from a tree, free the memory
  void DeleteLeaf( treePtr location,
          treePtr& parent );
  //Delete a node with one left child only
  void DeleteLeft ( treePtr location,
           treePtr& parent );
  //Delete a node with one right child only
  void DeleteRight ( treePtr location,
         treePtr& parent );
  //Delete a node that has two children
  void DeleteTwoChildren( treePtr& location );
  //Called recursively to visit each node in the tree InOrder
  void InOrder ( ofstream& fout,
              treePtr ptr );
  //Called recursively to visit each node in the tree PreOrder
  void PreOrder ( ofstream& fout,
               treePtr ptr );
  //Called recursively to visit each node in the tree PostOrder
```

```
 void PostOrder ( ofstream& fout,
                  treePtr ptr );
 //Count the nodes in the tree
 void CountNodes ( treePtr ptr,
                   int& count );
public:
 //Constructor
 TreeClass( void );
 //Create a tree from an input file
 void CreateATree ( ifstream& fin );
 //Return a bool based on value of root
 bool EmptyTree() const;
 //Print Tree content to an output file
 void PrintTree ( ofstream& fout );
 //Count the nodes in a tree
 int CountNodesInTree ( void );
 //Delete a node in the tree, return success of delete
 bool Delete ( int num );
 //Insert a node in Order in a tree
 void Insert ( nodeType node );
 //Print tree in PreOrder
 void PrintPreOrder ( ofstream& fout );
 //Print tree in PostOrder
 void PrintPostOrder ( ofstream& fout );
 //Search the tree for a num, return the node, the parent and a flag
 void Search ( treePtr& ptr,
               treePtr& parent,
```

```
       int num,

       bool& success );

  //The destructor, free the memory

  ~TreeClass ( void );

};

#endif
```

TreeFunctions.cpp:

```
#include 'tree.h'
///Add your methods HERE

///Private Methods ----------------------------------------------------------------------
--///

//Insert the node in ascending order
void TreeClass::InsertNode ( treePtr& ptr,
                 nodeType aNode )
{
 //if this is the first node
 if ( ptr == NULL )
 {

  ptr = GetANode ( aNode );

 }
 //is the value less than current ptr value, insert to the left
 else if ( aNode.value < ptr->data.value )
 {
  InsertNode ( ptr->left, aNode );
 }
 //if the value greater than the current ptr value,insert to the right
 else
 {
  InsertNode ( ptr->right, aNode );
 }
}

//Create memory, insert data into a tree node
treePtr TreeClass::GetANode ( nodeType aNode )
{
 treePtr ptr;

 //Create a new tree node
 ptr = new treeType;

 //Put the data into the tree node
```

```
 ptr->data = aNode;

 //Set the pointers of this new leaf node
 ptr->left = NULL;
 ptr->right = NULL;

 return ptr;
}

//Delete a leaf node from a tree, free the memory
void TreeClass:eleteLeaf( treePtr location,
          treePtr& parent )
{
 //Check to see if it is the root
    if ( parent == NULL )
 {
  root = NULL;
 }
 else if (parent->right == location)
 {
  //right child, sever connection
  parent->right = NULL;
 }
    else
 {
  //left child, sever connection
        parent->left = NULL;
 }

 //free the memory
 delete location;
}

//Delete a node with one left child only
void TreeClass:eleteLeft ( treePtr location,
           treePtr& parent )
{
 //Check to see if it is the root
 if ( parent == NULL )
 {
  //Make left child the root
  root = location->left;
 }
 else if ( parent->left == location )
 {
  //left child, re-connect to it's left
  parent->left = location->left;
 }
 else
 {
  //right child, re-connect to it's right
  parent->right = location->left;
 }
```

```
 //Set left pointer to NULL
 location->left = NULL;

 //Free the memory
 delete location;
}

//Delete a node with one right child only
void TreeClass:eleteRight( treePtr location,
           treePtr& parent )
{
 //Check to see if it is the root
 if ( parent == NULL )
 {
  //Set root to the right child
  root = location->right;
 }
 else if ( parent->right == location )
 {
  //right child, re-connect right
  parent->right = location->right;
 }
    else
 {
  //left child, re-connect left
  parent->left = location->right;
 }

 //Set right pointer to NULL
 location->right = NULL;

 //Free the memory
 delete location;
}

//Delete a node that has two children
void TreeClass:eleteTwoChildren( treePtr& location )
{

 treePtr place;
 treePtr walker;

 //Go left once
 place = location->left;

 //Check right, if NULL stop
 if ( place->right == NULL )
 {// found a place
  //move the data into location
  location->data = place->data;

  //Set the pointer of location left to place's left
```

```
  location->left = place->left;

 }// found a place

 else
 {// walk through tree
  do
  {
   // find node without right child
   walker = place;

   //Keep going right
   place = place->right;
  }while ( place->right!=NULL );

  //move the data into location
  location->data = place->data;

  //connect left child of place to right of walker
  walker->right = place->left;

 }// walk through tree

 //set place's pointer to NULL
 place->left = NULL;

 //Free the memory
 delete place;
}

//Called recursively to visit each node in the tree InOrder
void TreeClass::InOrder( ofstream& fout,
          treePtr ptr )
{
 //There is still a node in the tree
 if ( ptr != NULL )
 {
  //Go left
  InOrder( fout,
           ptr->left );

  //Print this node
  fout << setw(3) << ptr->data.value;

  //Go right
  InOrder( fout,
           ptr->right );
   }
}

//Called recursively to visit each node in the tree PreOrder
void TreeClass:😜reOrder ( ofstream& fout,
         treePtr ptr )
```

```
{
 //There is still a node in the tree
 if ( ptr != NULL )
 {
  //Print this node
  fout << setw(3) << ptr->data.value;

  //Go left
  PreOrder( fout,
            ptr->left );

  //Go right
  PreOrder( fout,
            ptr->right );

   }
}

//Called recursively to visit each node in the tree PostOrder
void TreeClass:ostOrder ( ofstream& fout,
          treePtr ptr )
{
 //There is still a node in the tree
 if ( ptr != NULL )
 {
  //Go left
  PostOrder( fout,
             ptr->left );

  //Go right
  PostOrder( fout,
             ptr->right );

  //Print the node
  fout << setw(3) << ptr->data.value;
   }
}

//Count the nodes in the tree
void TreeClass::CountNodes ( treePtr ptr,
           int& count )
{
 //There is still a node in the tree
 if ( ptr !=NULL )
 {
  //Increment the count
  count++;

  //Go left
  CountNodes ( ptr->left,
            count );

  //Go right
```

```
  CountNodes ( ptr->right,
             count );
 }
}
///Public Methods ------------------------------------------------------------------------
-///
//Constructor
TreeClass::TreeClass()
{
 root = NULL;
}

//Create a tree from an input file
void TreeClass::CreateATree ( ifstream& fin )
{
 nodeType node;

 //while there is data in the input file
 while ( !fin.eof() )
 {
  //read a value
  fin >> node.value;

  //Insert the value into the tree
  Insert( node );
 }
}

//Return a bool based on value of root
bool TreeClass::EmptyTree() const
{
 return ( root == NULL );
}

//Print Tree content to an output file
void TreeClass:🤔rintTree ( ofstream& fout)

{
 //Set a node to the root
 treePtr ptr = root;

 //if null, tree is empty
 if (ptr == NULL)
 {
  fout<< 'Tree is empty' << endl;
 }
 else
 {
  //Call Inorder Print of tree
  InOrder ( fout,
               ptr );
 }
```

```
}

//Count the nodes in a tree
int TreeClass::CountNodesInTree ( void )
{
 //Set a pointer to the root
 treePtr ptr = root;

 //Initialize the count
 int count = 0;

 //Call the recursive method to count the nodes in the tree
 CountNodes ( ptr,
           count );

 return count;
}

//Delete a node in the tree, return success of delete
bool TreeClass:elete ( int num )
{
 //Set a pointer to the root
 treePtr ptr = root;

 //Parent of root is null
 treePtr parent = NULL;

 bool success ;

 //Call recursive search method
 Search ( ptr,
       parent,
    num,
    success );

 //If it was successful, node was found
 if ( success )
 {
  //Is it a leaf node?
  if ( ptr->left == NULL && ptr->right == NULL )
  {
   //Delete the leaf
   DeleteLeaf ( ptr,
             parent );
  }
  //This is a node with only a left child
  else if ( ptr->left !=NULL && ptr->right == NULL )
  {
   //Delete a node with a left child
   DeleteLeft ( ptr,
             parent );
  }
  //This is a node with a right child
```

```
  else if ( ptr->left == NULL && ptr->right != NULL )
  {
   //Delete a node with a right child
   DeleteRight ( ptr,
               parent );
  }
  //it must be a node with two children
  else
  {
   //Delete a node with two children
   DeleteTwoChildren ( ptr );
  }
 }

 return success;
}

//Insert a node in Order in a tree
void TreeClass::Insert ( nodeType node )
{
 //Set a pointer to the root
 treePtr ptr = root;

 //Call the recursive method to insert a node
 InsertNode ( ptr,
           node );

 //if this is the first node,set the root to the new node
 if ( EmptyTree())
 {
  root = ptr;
 }
}

//Print tree in PreOrder
void TreeClass:🤔rintPreOrder ( ofstream& fout )
{
 //Call the recursive method for PreOrder
 PreOrder ( fout,
          root );
}

//Print tree in PostOrder
void TreeClass:🤔rintPostOrder ( ofstream& fout )
{
 //Call the recursive method for PostOrder
 PostOrder ( fout,
           root );
}

//Search the tree for a num, return the node, the parent and a flag
void TreeClass::Search ( treePtr& ptr,
       treePtr& parent,
```

```
       int num,
       bool& success )
{
 //Node was not found yet
 success = false;

 //while there are still nodes to search and node value was not found
 while ( ptr !=NULL && !success )
 {
  //If the node value in the tree is the value you are searching for
  if ( ptr->data.value == num )
  {
   //you found it
   success = true;
  }
  //if the node value in the tree is less than the value you are searching for
  else if ( num < ptr->data.value )
  {
   //Set the parent to the current ptr in the tree
   parent = ptr;

   //Move to the left
   ptr = ptr->left;
  }
  else
  {
   //Set the parent to the current ptr in the tree
   parent = ptr;

   //Move to the right
   ptr = ptr->right;
  }
 }
}

//The destructor, free the memory
TreeClass::~TreeClass ( void )
{
 //while the tree is not empty
 while ( !EmptyTree())
 {
  //Delete the node at the root
  Delete ( root->data.value );
 }
}
```

tree.txt:

```
48
12
56
2
9
18
52
49
10
-5
55
5
59
57
58
9
65
41
18
42
```

If you encounter any problems or errors, please let me know by providing an example of the code, input, output, and an explanation. Thanks.