PA06 Expression Tree

Generated by Doxygen 1.8.11

Contents

1 Main Page

This program contains the necessary functions to implement -the Expression Tree ADT using a linked tree structure.

While doing this project, I used various helper functions on top of the functions that were already given and called them recursively, both indirect and direct. In this project, we will learn about preorder, inorder and post order traversals to perform and implement the necessary functions needed for the Expression tree ADT using a linked tree structure.

2 Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

3 File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

4 Class Documentation

4.1 ExprTree< DataType > Class Template Reference

Classes

- class ExprTreeNode

Public Member Functions

- ExprTree ()
- ExprTree (const ExprTree &source)
- ExprTree & operator= (const ExprTree &source)
- ~ExprTree ()
- void build ()
- void expression () const
- DataType evaluate () const throw (logic_error)
- void clear ()
- void commute ()
- bool isEquivalent (const ExprTree &source) const
- void showStructure () const

Private Member Functions

- void insert (ExprTreeNode *&current, ExprTreeNode *source)
- void buildHelper (ExprTreeNode *&node)
- void expressionHelper (const ExprTreeNode *current) const
- DataType evaluateHelper (const ExprTreeNode *current) const
- void clearHelper (ExprTreeNode *&current)
- void commuteHelper (ExprTreeNode *&current)
- bool isEquivalentHelper (ExprTreeNode *current, ExprTreeNode *other) const
- void showHelper (ExprTreeNode *current, int level) const

Private Attributes

- ExprTreeNode * **root**

4.1.1 Constructor & Destructor Documentation

4.1.1.1 template<typename DataType > ExprTree< DataType >::ExprTree ( )

This function is the default constructor for the Expr Tree class

This function will set the root of the ExprTreeNode Class of the ExprTree Class to NULL.

Parameters

none	

Returns

This function does not return anything.

Precondition

The ExprTree is empty and was just created or needs to be modified.

Postcondition

The ExprTree will not have the ExprTreeNode root set to NULL

4.1.1.2 template<typename DataType > ExprTree< DataType >::ExprTree (const ExprTree< DataType > & *source*)

This function is the copy constructor for the ExprTree class.

This function will first set the root to NULL and then call the recursive function insert, with the parameters as root of the class and source.root.

Parameters

ExprTree	&source, which takes a ExprTree by reference so that it can be used to copy the expression tree from the source to this object.

Returns

This function does not return anything.

Precondition

none

Postcondition

This function will create a copy of the parameter of ExprTree& source and make the copy to this object.

4.1.1.3 template<typename DataType > ExprTree< DataType >::~ExprTree ( )

This function is the destructor for the ExprTree class.

This function will call the clear function to dynamically deallocate the memory for the current object.

Parameters

none	

Returns

This function does not return anything.

Precondition

An Exprtree object.

Postcondition

This function will be cleared through dynamically memory allocation.

4.1.2 Member Function Documentation

4.1.2.1 template<typename DataType > void ExprTree< DataType >::build ( )

This function is the build function for the ExprTree class.

This function will call the buildHelper function for to build the ExprTree.

Parameters

none	

Returns

This function does not return anything.

Precondition

An ExprTree object that needs to be built.

Postcondition

This function will build an ExprTree that is inputted to the terminal.

4.1.2.2 template<typename DataType > void ExprTree< DataType >::buildHelper (ExprTreeNode *& *node*) [private]

This function is the buildHelper function for the ExprTree class.

This function will create a location character called temp and cin to the temp. The function will then create a new ExprTreeNode with the parameters of the param constructor as the char temp, NULL and NULL. The function will then check to see if Temp was equal to the operators +,-,* and /. If it is, then the function will recursively call buildHelper function twice with the parameters, as node -> left and node-> right.

Parameters

ExprTreeNode&*	node, an ExprTreeNode pointer by reference so that it can be modified to build.

Returns

This function does not return anything.

Precondition

An ExprTree object that needs to be built.

Postcondition

This function will build an ExprTree that is inputted to the terminal. The function will call itself, if the char from the terminal is an operator, since it assums that operations has to be done, and if there isnt then we assume the char is a number and there is no operation left to do.

4.1.2.3 template<typename DataType > void ExprTree< DataType >::clear ( )

This function is the clear function for the Expr Tree class

The function checks to see that the root is not NULL, so that it can clear the expression tree. This function will call the clearHelper function with root passed as parameter to dynamically deallocate the expression tree. The function then sets the root equal to NULL, just to double check.

Parameters

none	

Returns

This function does not return anything.

Precondition

An expression tree object.

Postcondition

The ExprTree will be dynamically deallocated and cleared.

4.1.2.4 template<typename DataType > void ExprTree< DataType >::clearHelper (ExprTreeNode *& *current*) [private]

This function is the clearHelper function for the Expr Tree class

The function checks to make sure that the current ExprTreeNode pointer passed by reference's left is not NULL. If it is not Null, the function will call itself with the parameters of current -> left. The function repeats the same process for current -> right. The function will then delete current and then set it to NULL.

Parameters

ExprTreeNode&*	current, ExprTreeNode pointer passed by reference to modify so that it may be dynamically deleted.

Returns

This function does not return anything.

Precondition

An expression tree object.

Postcondition

The ExprTree will be dynamically deallocated and cleared and call itself.

4.1.2.5 template<typename DataType > void ExprTree< DataType >::commute ( )

This function is the commute function for the Expr Tree class

The ExprTree will call the commuteHelper function with root passed as the parameter.

Parameters

none	

Returns

This function does not return anything.

Precondition

none

Postcondition

The ExprTree will call the commuteHelper function with root passed as the parameter. This is to commute the oeprands for every arithmetic operating in the expression tree.

4.1.2.6 template<typename DataType > void ExprTree< DataType >::commuteHelper (ExprTreeNode *& *current*) [private]

This function is the commuteHelper function for the Expr Tree class

The ExprTree will first check to see if the passed parameter current is not NULL. If it is not, the function will then declare and initialize a ExprTreeNode pointer called temp to current -> left. The function will then set current -> left to current -> right. The function will then set current -> right to temp. The function will then recursively call itself with current -> left and current -> right as the parameters.

Parameters

ExprTreeNode&*	current, which is the current tree node that the function will go through to commute.

Returns

This function does not return anything.

Precondition

none

Postcondition

The ExprTree will call the commuteHelper function with root passed as the parameter. This is to commute the oeprands for every arithmetic operating in the expression tree.

4.1.2.7 template<typename DataType > DataType ExprTree< DataType >::evaluate ( ) const throw logic_error)

This function is the evaluate function for the ExprTree class.

This function will first check to see if the root of the object is NULL, if it is, we throw a logic error. If the root is not NULL , then we call the evaluateHelper function with root as the passed parameter.

Parameters

none	

Returns

This function returns the evaluateHelper function with root as the passed parameter.

Precondition

The object should not be an empty tree.

Postcondition

This function will recursively evaluate the expression of the ExprTree.

4.1.2.8 template<typename DataType > DataType ExprTree< DataType >::evaluateHelper (const ExprTreeNode * *current*) const [private]

This function is the evaluateHelper function for the ExprTree class.

This function will create a local char temp which will have current node -> dataItem. The function will then create a DataType holder variable. The function will switch the char temp to evaluate the expression.If temp is a number between 0 and 9, the function will hold the datatype in the holder and convert it by subtractin '0' from it and then return the holder variable. If the operator is a '+', the function will call itself with the current -> left + current -> right passed in as parameters to evaluate it. If the operator is a '-', the function will call itself with the current-> left - current -> right passed in as parameters to evaluate it. If the operator is a '*', the function will call itself with the current-> left * current -> right passed in as parameters to evaluate it. If the operator is a '/', the function will call itself with the current-> left / current -> right passed in as parameters to evaluate it.

Parameters

const	ExprTreeNode* current, which is the current TreeNode that will be observed, such as if it an operator or a number and so on.

Returns

This function returns the evaluated expression based on the case statement that switches the char temp, which is the dataItem of the current parameter.

Precondition

The object should not be an empty tree.

Postcondition

This function will recursively evaluate the expression of the ExprTree.

4.1.2.9 template<typename DataType > void ExprTree< DataType >::expression ( ) const

This function is the expression function for the ExprTree class.

This function will check to see if the root of the ExprTree is not NULL.If the ExprTree is not NULL, the function will call the expressionHelper function with the parameter passed as the root.

Parameters

none	

Returns

This function does not return anything.

Precondition

An ExprTree that is not empty.

Postcondition

This function will call the expressionHelper function to print out the expression of the ExprTree.

4.1.2.10 template<typename DataType > void ExprTree< DataType >::expressionHelper (const ExprTreeNode * *current*) const [private]

This function is the expressionHelper function for the ExprTree class.

This function will create a local char variable called temp. Temp will be equal to the current -> dataItem. The function then checks, to see if the char contains a number between 0-9. If it is, it the prints out the char. If the char is not a number, then the function prints out the open parantheses first amd then recurssively calls itself twice with the parameters current -> left and then cout the current -> dataItem and t the parameter current -> right and then cout a close parantheses.

Parameters

const	ExprTreeNode* current, a ExprTreeNode pointer that is used to start the recursion to print out the expression of the object.

Returns

This function does not return anything.

Precondition

none.

Postcondition

This function will recursively printout the expression of the ExprTree.

4.1.2.11 template<typename DataType > void ExprTree< DataType >::insert (ExprTreeNode *& *current,* ExprTreeNode * *source*) [private]

This function is the insert function for the ExprTree class, which is the helper function for the constructors for me.

This function will take in a pointer by reference and a pointer and take the source and modify the current to be the same as the soure. This is done by checking if the source is NULL, if it is, then current is set to NULL. If Source is not NULL, then, we set the current with the ExprTreeNode param constructor with source's dataItem and its left and right set to NULL. I then recursively call the function till there is nothing left to insert, with current -> left, source -> left and current -> right and source -> right passed as the parameters.

Parameters

ExprTreeNode	*& current, which takes a ExpTreeNode pointer by reference so that it can be modified
ExprTreeNode	*source, which takes a ExprTreeNode pointer so that it can be used to set the other pointer passed to be reference as.

Returns

This function does not return anything.

Precondition

The ExprTreeNode*&current is empty and was just created or needs to be copied from the ExprTreeNode* Source

Postcondition

The ExprTreeNode*& current will have a copy of the function of ExprTreeNode*source.

4.1.2.12 template<typename DataType > bool ExprTree< DataType >::isEquivalent (const ExprTree< DataType > & *source*) const

This function is the isEquivalent function for the Expr Tree class

The function will check to see if both the root of the current object and the source object are NULL. If they are, the function will then check to see if they are equal. If they are, the function returns true, other wise it returns false. The function then checks to see if the current root->dataItem is equal to source.root -> dataItem, and if they are equal the function returns isEquivalentHelper with root and source.root passed as the parameters. Otherwise the function will return false.

Parameters

const	ExprTree& source, which is the other tree that the current object will be compared to.

Returns

This function returns either true of false, depending on if the expression tree computations are equal.

Precondition

none

Postcondition

The ExprTree will call the commuteHelper function with root passed as the parameter. This is to commute the oeprands for every arithmetic operating in the expression tree.

4.1.2.13 template<typename DataType > bool ExprTree< DataType >::isEquivalentHelper (ExprTreeNode * *current,* ExprTreeNode * *other*) const [private]

This function is the isEquivalentHelper function for the Expr Tree class

The function first declares and intializes a local char temp variable to current -> dataItem. The function then creates local char variables called left and right. The function will check to see if the temp is a number between 0 and 9, and then if it is, it makes to sure that the two nodes passed, have the same dataItem and then return true if that is the case and return false if that is not the case. The function assigns left and right to current -> left -> dataItem and current -> right ->dataItem. The function then checks to see if either left or right is an operator. If so, it goes

to make sure that left and right are equal to other -> left -> dataItem and other -> right -> dataItem. If that is true, then the function will return itself with current -> left, other -> left, current -> right and other-> right passed as parameters. The function also checks to make sure that the left of the current is equal to the right of the other and vice versa and does the same return call but with, current -> left, other -> right, current ->right and other -> left respectively. The function then checks to see if temp is either a - or /, since 1-2 and 2-1, and 1/2 and 2/1 are different for example. The property wont apply to the two operator. So the function checks that and returns itself with the left and right data item of current and other to make sure that it is equal. Otherwise, it does the same check but with two cases, where (current -> left -> dataItem, other -> left -> dataItem, current -> right -> dataItem and other -> right dataItem, are equal) or (current -> left -> dataItem, other -> right -> dataItem, current -> right -> dataItem and other -> left dataItem, are equal

Parameters

ExprTreeNode∗	current, which is the current node that the function will look at and compare to the other.
ExprTreeNode∗	other, which is the other node that the function will compare with to check if they are equal.

Returns

This function returns either true of false, depending on if the expression tree computations are equal.

Precondition

Two ExprTreeNode pointers must be passed so that the entire tree traversal can be compared.

Postcondition

The function will go and traverse through the trees of both of Node representatives and check to see if they are equal and return true of false based on that. The function will call itself to check if certain cases are true or false and then return that.

4.1.2.14 template<typename DataType > ExprTree< DataType > & ExprTree< DataType >::operator= (const ExprTree< DataType > & *source*)

This function is the overloaded assignment operator for the ExprTree class.

This function will check if the current object is not the source parameter. The function will first clear the object, and then the root to NULL and then call the recursive function insert, with the parameters as root of the class and source.root.

Parameters

Const	ExprTree &source, which takes a ExprTree by reference so that it can be used to copy the expression tree from the source to this object.

Returns

This function does not return anything.

Precondition

An object that is not the same as the object that was passed as the parameter for the copy constructor.

Postcondition

This function will create a copy of the parameter of ExprTree& source and make the copy to this object.

4.1.2.15 template<typename DataType > void ExprTree< DataType >::showHelper (ExprTreeNode * *p,* int *level*) const [private]

This function is the showHelper function for the Expr Tree class

The function will iterate through the loop, and print out the leaves and branches of the expression tree.

Parameters

ExpTreeNode	*p, which is the node to start the printing at.
int	level, the level to start the printing of the trees and branches at.

Returns

This function does not return anything.

Precondition

none

Postcondition

The contents of the ExpressionTree will be printed to the terminal in the way that it should be.

4.1.2.16 template<typename DataType > void ExprTree< DataType >::showStructure ( ) const

This function is the showStructure function for the Expr Tree class

The function will make sure that the tree is not empty. If it is, the function will print empty tree. Other wise it will call the showHelper function with root, and 1 passed as the parameters.

Parameters

none	

Returns

This function does not return anything.

Precondition

none

Postcondition

The contents of the ExpressionTree will be printed to the terminal in the way that it should be.

The documentation for this class was generated from the following files:

- ExpressionTree.h
- ExpressionTree.cpp
- show8.cpp

4.2 ExprTree< DataType >::ExprTreeNode Class Reference

Public Member Functions

- ExprTreeNode (char elem, ExprTreeNode *leftPtr, ExprTreeNode *rightPtr)

Public Attributes

- char **dataItem**
- ExprTreeNode * **left**
- ExprTreeNode * **right**

4.2.1 Constructor & Destructor Documentation

4.2.1.1 template<typename DataType > ExprTree< DataType >::ExprTreeNode::ExprTreeNode (char *elem,* ExprTreeNode * *leftPtr,* ExprTreeNode * *rightPtr*)

This function is the ExprTreeNode param constructoror the Expr Tree class

This function will initialize the ExprTreeNode to hold a certain dataItem, and point to the left and right of its branch/leaf, depending on if they exist.

Parameters

char	elem, which will be the dataItem that the node holds.
ExprTreeNode	*leftPtr, which will be the node's left branch/leaf
ExprTreeNode	*rightPtr, which will be the node's right branch/leaf

Returns

This function does not return anything.

Precondition

none

Postcondition

modifies the TreeNode to hold a dataitem and point to the left and right based on the parameters passed.

The documentation for this class was generated from the following files:

- ExpressionTree.h
- ExpressionTree.cpp

5 File Documentation

5.1 ExpressionTree.cpp File Reference

This program will implement an Expression Tree ADT using a linked tree structure.

```
#include "ExpressionTree.h"
#include "iostream"
```

5.1.1 Detailed Description

This program will implement an Expression Tree ADT using a linked tree structure.

Author

Kripash Shrestha

Version

1.0

The specifications of the program are instructed and documented on Lab 8 Queue ADT of C++ Data Structures: A Laboratory Course Third Edition by Brandle, Geisler, Roberge and Whittington

Date

Wednesday, October 11, 2017

Index