Project 8 - Expression Tree ADT

AUTHOR

Version 1.000

10/08/2014

Table of Contents

Project 8 (Expression Tree ADT)

This program will create an implmenetation of an Expression Tree using a linked tree structure.

Author:

Saharath Kleips

Version:

1.00

The specifications of this project match those of the book C++ Data Structures - A Laboratory Course (3rd Edition) Project 8. Mathematical expressions form a hierarchy of operations built upon the addition, subtraction, multiplication, and division operators. This hieracrhy of operations to preform on an expression can be explicitly expressed using a binary tree, specifically called a expression tree.

Notes:

Infix notation is when each operator is placed between its operands: (1 + 3) * (6 - 4)

Prefix notation is when each operator is placed immediately before its operands: * + 1 3 - 6 4

Class Index

Class List

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

File Index

File List

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

Class Documentation

ExprTree< DataType > Class Template Reference

Classes

class ExprTreeNode

Public Member Functions

ExprTree ()

The default constructor that creates an empty expression tree.

ExprTree (const ExprTree &source)

The copy constructor that initializes this ExprTree to be equivalent to the other ExprTree object parameter.

ExprTree & operator= (const ExprTree &source)

The overloaded assignment operator that sets this ExprTree to be equivalent to the other ExprTree object parameter and returns a reference to this object.

~ExprTree ()

The destructor that deallocates memory used to store the ExprTree.

void build ()

Reads an arithmetic expression in prefix form from the keyboard and builds the corresponding expression tree.

void expression () const

Outputs the expression corresponding to the value of the tree in fully parenthesized infix form.

DataType evaluate () const throw (logic_error)

Returns the value of the corresponding arithmetic expression.

void clear ()

Removes all the data items in the expression tree.

void commute ()

Commutes the operates for every arithmetic operator in the expression tree.

bool isEquivalent (const ExprTree &source) const

Compares the expression tree to another expression tree for equivalence.

void showStructure () const

Outputs an expression tree with its branches orientated from left (root) to right (leaves).

Private Member Functions

void showHelper (ExprTreeNode *p, int level) const

Recursive helper for showStructure() function.

DataType evalHelper (ExprTreeNode *p) const

Recursive helper function for evaluate() function.

void buildHelper (ExprTreeNode *&node)

Recursive helper function for the build() function.

void copyHelper (ExprTreeNode *&dest, ExprTreeNode *source)

Recursive helper function for operator=().

void expressionHelper (ExprTreeNode *p) const

Recursive helper function for expression() function.

void clearHelper (ExprTreeNode *&p)

Recursive helper function for clear() function.

void commuteHelper (ExprTreeNode *&p)

Recursive helper function for commute() function.

bool isEquivHelper (ExprTreeNode *dest, ExprTreeNode *source) const

Recursive helper function for isEquivalent() function.

Private Attributes

ExprTreeNode * **root**

Detailed Description

template<typename DataType>class ExprTree< DataType >

Definition at line 24 of file ExpressionTree.h.

Constructor & Destructor Documentation

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

The default constructor that creates an empty expression tree.

Postcondition:

This expression tree will be a valid empty expression tree.

Definition at line 37 of file ExpressionTree.cpp.

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

The copy constructor that initializes this ExprTree to be equivalent to the other ExprTree object parameter.

Precondition:

source is a valid ExprTree.

Postcondition:

This ExprTree will be a deep copy of the source ExprTree.

Parameters:

source	is the ExprTree that this ExprTree will be made equivalent to.

operator=()

Definition at line 51 of file ExpressionTree.cpp.

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

The destructor that deallocates memory used to store the ExprTree.

Postcondition:

This ExprTree will be an empty, deallocated, ExprTree.

See also:

clear()

Definition at line 101 of file ExpressionTree.cpp.

Member Function Documentation

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

Reads an arithmetic expression in prefix form from the keyboard and builds the corresponding expression tree.

Precondition:

The expression from the keyboard must be a valid arithmetic expression.

Postcondition:

This ExprTree will accurately represent the arithmetic expression entered.

See also:

buildHelper()

Definition at line 114 of file ExpressionTree.cpp.

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

Recursive helper function for the build() function.

It takes a reference to a pointer to an expression tree node so that if a new node should be added to the tree, it can be allocated and added by assigning the newly allocated node to the node parameter. node is the node to put data into.

See also:

build()

Definition at line 127 of file ExpressionTree.cpp.

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

Removes all the data items in the expression tree.

Postcondition:

This ExprTree will be an empty, deallocated, ExprTree.

See also:

clearHelper()

Definition at line 226 of file ExpressionTree.cpp.

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

Recursive helper function for clear() function.

See also:

clear()

Definition at line 235 of file ExpressionTree.cpp.

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

Commutes the operates for every arithmetic operator in the expression tree.

Postcondition:

Every operator in ExprTree is commuted.

See also:

commuteHelper()

Definition at line 253 of file ExpressionTree.cpp.

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

Recursive helper function for commute() function.

See also:

commute()

Definition at line 262 of file ExpressionTree.cpp.

template<typename DataType > void ExprTree< DataType >::copyHelper (ExprTreeNode *& *dest*, ExprTreeNode * *source*) [private]

Recursive helper function for operator=().

Postcondition:

The nodes will be set equal to eachother.

Parameters:

See also:

dest	is the node to copy the values to.
source	is the node to copy the values from.

operator=()

Definition at line 83 of file ExpressionTree.cpp.

template<typename DataType > DataType ExprTree< DataType >::evalHelper (ExprTreeNode * *p*) const [private]

Recursive helper function for evaluate() function.

See also:

evaluate()

Definition at line 192 of file ExpressionTree.cpp.

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

Returns the value of the corresponding arithmetic expression.

Precondition:

This ExprTree is not empty.

Returns:

The value of the arithmetic expression.

Exceptions:

See also:

The	ExprTree is empty.

evalHelper()

Definition at line 181 of file ExpressionTree.cpp.

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

Outputs the expression corresponding to the value of the tree in fully parenthesized infix form.

Precondition:

This ExprTree represents a valid arithmetic expression.

Postcondition:

The expression will be outputed to the console.

See also:

expressionHelper()

Definition at line 148 of file ExpressionTree.cpp.

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

Recursive helper function for expression() function.

See also:

expression()

Definition at line 157 of file ExpressionTree.cpp.

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

Compares the expression tree to another expression tree for equivalence.

If the two trees are equivalent, then returns true. Otherwise returns false.

Precondition:

Both trees are valid expression trees

Parameters:

See also:

source	is the ExprTree to compare to this ExprTree

True if the trees are equivalent. False if they are not equivalent.

See also:

isEquivHelper()

Definition at line 287 of file ExpressionTree.cpp.

template<typename DataType > bool ExprTree< DataType >::isEquivHelper (ExprTreeNode * *dest*, ExprTreeNode * *source*) const `[private]`

Recursive helper function for isEquivalent() function.

See also:

isEquivalent()

Definition at line 296 of file ExpressionTree.cpp.

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

The overloaded assignment operator that sets this ExprTree to be equivalent to the other ExprTree object parameter and returns a reference to this object.

Precondition:

source is a valid ExprTree.

Postcondition:

This ExprTree will be a deep copy of the source ExprTree.

Parameters:

Returns:

source	is the ExprTree that this ExprTree will be made equivalent to.

The reference to this ExprTree.

See also:

copyHelper()

Definition at line 66 of file ExpressionTree.cpp.

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

Recursive helper for showStructure() function.

Outputs the subtree whose root node is pointed to by p. Parameter level is the level of this node within the expression tree.

Parameters:

Returns:

p	is the root node of the subtree.
level	is the current depth of the subtree.

The subtree will be outputed to the console.

Definition at line 360 of file ExpressionTree.cpp.

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

Outputs an expression tree with its branches orientated from left (root) to right (leaves).

The tree output is rotated counterclockwise ninty degrees from the conventional orientation. If the tree is empty, it outputs "Empty tree." This function is for testing purposes only and assumes arithmetic expressions contain only single-digit, non-negative integers, and the arithmetic operators for addition, subtraction, multiplication, and division.

Postcondition:

The expression tree will be outputed to the console.

Definition at line 340 of file ExpressionTree.cpp.

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

1. ExpressionTree.h
2. ExpressionTree.cpp
3. show8.cpp

ExprTree< DataType >::ExprTreeNode Class Reference

Public Member Functions

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

The parameterized constructor that sets ExprTreeNode's data item, left pointer, and right pointer.

Public Attributes

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

Detailed Description

template<typename DataType>class ExprTree< DataType >::ExprTreeNode

Definition at line 49 of file ExpressionTree.h.

Constructor & Destructor Documentation

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

The parameterized constructor that sets ExprTreeNode's data item, left pointer, and right pointer.

Postcondition:

This ExprTreeNode will be a valid ExprTreeNode.

Parameters:

Postcondition:

elem	is the character data to store within the node.
leftPtr	is the left branch within the node.
rightPtr	is the right branch within the node.

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

4 ExpressionTree.h
5 ExpressionTree.cpp

File Documentation

ExpressionTree.cpp File Reference

```
#include "ExpressionTree.h"
#include <ctype.h>
```

Detailed Description

Definition in file ExpressionTree.cpp.

Index

INDE