Expression Tree

Tim Kwist

Version1.0

Wednesday, October 8, 2014

Table of Contents

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

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
- bool **isEmpty** () const
- void **showStructure** () const

Constructor & Destructor Documentation

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

Constructor for Expression Tree

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

Creates an empty expression tree.

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

Copy constructor for Expression Tree

Parameters:

source	Expression tree that this expression tree will become a copy of

Returns:

None

Precondition:

None

Postcondition:

Initializes the expression tree to be equivalent to the other **ExprTree** object parameter.

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

Deconstructor for Expression Tree

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

Deallocates the memory used to store this expression tree

Member Function Documentation

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

Builds the Expression Tree from user input

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

Builds an expression tree according to keyboard input

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

Deallocate memory assigned to this tree and remove its contents.

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

All the data items in this expression tree will be removed.

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

Apply the commutative property to the expression tree

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

The contents of the expression tree will be swapped around according to the commutative property

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

Evaluate the expression contained within the expression tree.

Parameters:

None	

Returns:

Datatype value of the corresponding arithmetic expression contained within this Expression Tree

Precondition:

Expression tree is not empty.

Postcondition:

The contents of this expression tree will not be changed. The value of the corresponding arithmetic expression will be returned.

Exceptions:

logic_error	Throws error if tree is empty

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

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

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

The contents of this expression tree will not be changed. An expression corresponding to the value of the tree in fully parenthesized infix form will be outputted.

template<typename DataType > bool ExprTree< DataType >::isEmpty () const

Checks if the current Exprssion Tree is empty

Parameters:

None	

Returns:

True if tree is empty, false otherwise

Precondition:

None

Postcondition:

The contents of this expression tree will not be changed

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

Checks whether a given expression tree is equivalent to the current expression tree. This means that the overall value of the tree is equivalent, and the children nodes are equal. The order of child nodes may be different if the overall expression is not changed. IE 1 + 3 is equivalent to 3 + 1, but 3 - 1 is not equivalent to 1 - 3.

Parameters:

source	Another expression tree to which this expression tree is being compared to

Returns:

True if both trees are equivalent, otherwise false.

Precondition:

None

Postcondition:

The contents of both trees will not be changed.

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

Assignment operator overload for Expression Tree

Parameters:

source	Expression tree that this expression tree will become a copy of

Returns:

ExprTree& Reference to this object

Precondition:

None

Postcondition:

Sets the expression tree to be equivalent to the other **ExprTree** object parameter. Will not change any part of source parameter

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

Outputs an express tree with its branches oriented from left (root) to right (leaves) - that is, the tree output is rotated counterclockwise ninety degrees from its conventional orientation. If the tree is empty, outputs "Empty Tree". Note that this operation is intended for testing/debugging purposes only. It assumes that arithmetic expressions contain only single-digit, nonnegative integers and the arithmetic operators for addition, subtraction, mutiplication, and division.

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

The contents of this expression tree will not be modified. The expression tree will be outputted.

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

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

File Documentation

ExpressionTree.cpp File Reference

This program will implement an Expression Tree.
```
#include <ExpressionTree.h>
#include <stdlib.h>
```

Detailed Description

This program will implement an Expression Tree.

Author:

Tim Kwist

Version:

1.0

The specifications of this program are defined by C++ Data Structures: A Laboratory Course (3rd edition) by Brandle, JGeisler, Roberge, Whittington, exercise 8.

Date:

Wednesday, October 8, 2014

Index

INDEX