Expression Tree

1.0

Generated by Doxygen 1.8.8

Wed Oct 8 2014 22:57:49

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 ExprTree< DataType > Class Template Reference

`#include <ExpressionTree-1.h>`

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 isEmpty () const
- bool isEquivalent (const ExprTree &source) const
- void showStructure () const
- 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 isEmpty () const
- bool isEquivalent (const ExprTree &source) const
- void showStructure () const

Private Member Functions

- void showHelper (ExprTreeNode ∗p, int level) const
- void buildHelper (ExprTreeNode ∗&top)
- void expressionHelper (ExprTreeNode ∗ptr) const
- void clearHelper (ExprTreeNode ∗&ptr)
- DataType evaluateHelper (ExprTreeNode ∗ptr) const throw (logic_error)
- void commuteHelp (ExprTreeNode ∗ptr)
- bool isEquivalentHelp (const ExprTreeNode ∗home, const ExprTreeNode ∗source) const
- void copyHelper (ExprTreeNode ∗&ptr, ExprTreeNode ∗source)
- void showHelper (ExprTreeNode ∗p, int level) const
- void buildHelper (ExprTreeNode ∗&top)
- void expressionHelper (ExprTreeNode ∗ptr) const
- void clearHelper (ExprTreeNode ∗&ptr)
- DataType evaluateHelper (ExprTreeNode ∗ptr) const throw (logic_error)
- void commuteHelp (ExprTreeNode ∗ptr)
- bool isEquivalentHelp (const ExprTreeNode ∗home, const ExprTreeNode ∗source) const

Private Attributes

- ExprTreeNode ∗ root

3.1.1 Constructor & Destructor Documentation

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

This is the tree constructor.

Parameters

none	

Returns

constructor, no return

Precondition

no pre conditions

Postcondition

the root will be null

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

This is the tree copy constructor. Will create another tree just like the initial one. Just uses the copyHelper function.

Parameters

ExprTree,the	tree to copy to

Returns

constructor, no return

Precondition

there should be another tree initialized already

Postcondition

there will be two identical trees

initialize

if the source is not empty proceed to copying

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

This is the tree destructor.

frees all memory by using clear function

Parameters

none	

Returns

void

Precondition

there should be an initialized tree

Postcondition

all memory will be freed and the tree will be destroyed

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

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

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

3.1.2 Member Function Documentation

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

This is the build function.

just calls the buildhelper

Parameters

none	

Returns

void

Precondition

there should be an initialized tree

Postcondition

a tree with valid data will be created

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

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

This is the build helper

creates the tree recursively basically creates a new node if it is not a digit will call the function again to create the children nodes

Parameters

the	pointer to work on, initially the root

Returns

void

Precondition

there should be an initialized tree

Postcondition

an expression tree will be created

read in

create new node

if it is a digit stop, if not make more

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

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

This is the clear.

calls the clear helper

Parameters

none	

Returns

void

Precondition

there should be a built tree

Postcondition

all memory will be freed

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

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

This is the clear helper.

deallocates all memory recursively, if the pointer is not null it will call itself to the pointers left and right then will delete

Parameters

ptr,initially	root

Returns

void

Precondition

there should be a built tree

Postcondition

all memory will be freed

recall to left and right of node

delete the node

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

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

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

This is the commute function.

calls commute help function

Parameters

none	

Returns

void

Precondition

there should be a built tree

Postcondition

the tree will be flipped

3.1.2.11 template<typename DataType > void ExprTree< DataType >::commuteHelp (ExprTreeNode * *ptr*) [private]

This is the commute helper function.

swaps every single left and right child pointer down the tree recursively

Parameters

ptr,intially	the root

Returns

void

Precondition

there should be a built tree

Postcondition

the tree will be flipped

if at the end, do nothing (i believe this stuff isnt needed but too lazy to test without

if not at the end, swap left and right and recall for new pointers

3.1.2.12 template<typename DataType > void ExprTree< DataType >::commuteHelp (ExprTreeNode * *ptr*) [private]

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

This is the function that will recursively copy two trees to eachother

Checks if the current item is a character or digit, calls itself two more timess then creates the new node if it is a digit, it will only create a new node and stop

Parameters

ExprTree,the	tree to copy to and copy from

Returns

void

Precondition

there should two initialized trees

Postcondition

there will be two identical trees

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

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

This is the evaluate.

calls the evaluation helper to get the output

Parameters

none	

Returns

the datatype, most likely a float

Precondition

there should be a built tree

Postcondition

the function will be solved

3.1.2.16 template<typename DataType > DataType ExprTree< DataType >::evaluateHelper (ExprTreeNode * *ptr*) const throw logic_error) [private]

This is the evaluate helper.

evaluates the tree recusively, works its way to the bottom of the tree then solves the left and right of each individual expression

Parameters

the	current ptr, initially the root

Returns

the datatype, most likely a float

Precondition

there should be a built tree

Postcondition

the function will be solved

if it is a digit convert and return

if an expression get the left and right values by recalling, then check for char and do operation

3.1.2.17 template<typename DataType > DataType ExprTree< DataType >::evaluateHelper (ExprTreeNode * *ptr*) const throw logic_error) [private]

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

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

This is the expression function.

calls the expression helper to show the function with parenthesis

Parameters

none	

Returns

void

Precondition

there should be a built tree

Postcondition

the function will be displayed on the screen

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

This is the expression helper function.

prints out the function recursively, if current item is a digit will print out digit if the current item is not a digit, will print the beginning parenthesis and recalls itself for left and right

Parameters

the	ptr, initially the root

Returns

void

Precondition

there should be a built tree

Postcondition

the function will be displayed on the screen

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

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

3.1.2.23 template<typename Datatype > bool ExprTree< Datatype >::isEmpty ( ) const

This is the isempty funtion.

Parameters

none	

Returns

true if empty, false if not

Precondition

an initialized tree

Postcondition

same

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

This is the is eqeuivalent function.

compared two trees to see if they are the same by calling its helper

Parameters

the	tree to compare with

Returns

bool, true if the same, false if not

Precondition

there should be 2 built trees

Postcondition

nothing will change

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

3.1.2.26 template<typename DataType > bool ExprTree< DataType >::isEquivalentHelp (const ExprTreeNode * *home,* const ExprTreeNode * *source*) const [private]

This is the is eqeuivalent helper function.

compares two trees to see if they are the same recursively the 2 stopping conditions are that 1, a pointer is null (at the end) so it compares it to the other and if the same returns true and 2, if they are not the same will return false. if niether is met then calls itself twice for the left and right

Parameters

the	tree to compare with

Returns

bool, true if the same, false if not

Precondition

there should be 2 built trees

Postcondition

nothing will change

if pointers are null compare them, should both be null

check for equivalency, if not return false

if passed other 2, then call for left and right

3.1.2.27 template<typename DataType > bool ExprTree< DataType >::isEquivalentHelp (const ExprTreeNode * *home*, const ExprTreeNode * *source*) const [private]

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

This is the overloaded assignment operator, will set two trees equal to eachother Just uses the copyHelper function.

Parameters

ExprTree,*the*	tree to copy to

Returns

returns the copied tree

Precondition

there should be a tree to copy to and copy from already created

Postcondition

there will be two identical trees

check to see if the trees are the same, if they are return it

if not empty, make empty

if the source is empty return empty tree

copy

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

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

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

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

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

3.1.3 Member Data Documentation

3.1.3.1 template<typename DataType > ExprTreeNode * ExprTree< DataType >::root [private]

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

- ExpressionTree-1.h
- ExpressionTree.h
- ExpressionTree-1.cpp
- ExpressionTree.cpp
- show8.cpp

3.2 ExprTree< DataType >::ExprTreeNode Class Reference

Public Member Functions

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

Public Attributes

- char dataItem
- ExprTreeNode * left
- ExprTreeNode * right

3.2.1 Constructor & Destructor Documentation

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

This is the tree node constructor. Will just create the node.

Parameters

char	elem (the data), 2 node pointers (will be the children of the created node

Returns

constructor, no return

Precondition

no pre conditions

Postcondition

there will be a node created with the data

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

3.2.2 Member Data Documentation

3.2.2.1 template<typename DataType > char ExprTree< DataType >::ExprTreeNode::dataItem

3.2.2.2 template<typename DataType > ExprTreeNode * ExprTree< DataType >::ExprTreeNode::left

3.2.2.3 template<typename DataType > ExprTreeNode * ExprTree< DataType >::ExprTreeNode::right

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

- ExpressionTree-1.h
- ExpressionTree.h
- ExpressionTree-1.cpp
- ExpressionTree.cpp

Chapter 4

File Documentation

4.1 config.h File Reference

Macros

- #define LAB8_TEST1 0
- #define LAB8_TEST2 1
- #define LAB8_TEST3 1

4.1.1 Macro Definition Documentation

4.1.1.1 #define LAB8_TEST1 0

Expression Tree class (Lab 8) configuration file. Activate test #N by defining the corresponding LAB8_TESTN to have the value 1.

4.1.1.2 #define LAB8_TEST2 1

4.1.1.3 #define LAB8_TEST3 1

4.2 ExpressionTree-1.cpp File Reference

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <sys/time.h>
#include "ExpressionTree.h"
#include "show8.cpp"
```

4.3 ExpressionTree-1.h File Reference

```
#include <stdexcept>
#include <iostream>
```

Classes

- class ExprTree< DataType >
- class ExprTree< DataType >::ExprTreeNode

4.4 ExpressionTree.cpp File Reference

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <sys/time.h>
#include "ExpressionTree.h"
#include "show8.cpp"
```

4.5 ExpressionTree.h File Reference

```
#include <stdexcept>
#include <iostream>
```

Classes

- class ExprTree< DataType >
- class ExprTree< DataType >::ExprTreeNode

4.6 show8.cpp File Reference

4.7 test8.cpp File Reference

```
#include <iostream>
#include <stdexcept>
#include "ExpressionTree.cpp"
#include "config.h"
```

Functions

- template<typename DataType >
  void dummy (ExprTree< DataType > copyTree)
- int main ()

4.7.1 Function Documentation

4.7.1.1 template<typename DataType > void dummy (ExprTree< DataType > *copyTree*)

4.7.1.2 int main ( )