Brown University ENGN2912B Fall 2017

Scientific Computing in C++

Assignment 07

Refactoring `AlgebraicTreeNode` and `AlgebraicTreeExpression` using Inheritance

The design of the `AlgebraicTreeNode` class that you implemented is not quite right. The `_type` variable allows you to use instances of the same class to implement different types of nodes. As a result, the implementation is inefficient in terms of memory use, and complex, since the functions must behave differently for the different data types. The proper way to implement this program is using inheritance. Your task is to re-implement the classes using the following public interfaces. You have to decide what private variables and functions are needed in the different classes and subclasses. But your old main() functions should work with the new classes without any significant changes. Your new implementation should be shorter, cleaner, more memory efficient, and a lot easier to read. But you should be able to reuse most of your old code. Also, feel free to switch from C-style strings to C++ strings in your new implementation, although it would be OK if you decide not to do so.

This process of restructuring existing computer code – changing the factoring – without changing its external behavior is called **Code Refactoring**. After your first implementation of a project is completed and debugged, you should always analyze your design, and consider refactoring your code.

```
////////////////////////////////////////////////////////////////////////////
// file AlgebraicTreeExpression.hpp
#ifndef _AlgebraicTreeExpression_hpp_
#define _AlgebraicTreeExpression_hpp_

#include "AlgebraicTreeNode.hpp"
#include "AlgebraicTreeNumber.hpp"
#include "AlgebraicTreeOperation.hpp"

class AlgebraicTreeExpression {

public:

  ~AlgebraicTreeExpression();

  AlgebraicTreeExpression();
  AlgebraicTreeExpression(const char* expression);

  bool   isEmpty() const;
  double evaluate() const;
  char*  toString() const;

  void setRoot(AlgebraicTreeNode* root);

protected:

  // your choice ...

};

#endif // _AlgebraicTreeExpression_hpp_
```

```
////////////////////////////////////////////////////////////////////////////
// file AlgebraicTreeNode.hpp
#ifndef _AlgebraicTreeNode_hpp_
#define _AlgebraicTreeNode_hpp_

class AlgebraicTreeNode {

public:

  virtual ~AlgebraicTreeNode();

  AlgebraicTreeNode();

  const bool isRoot() const;
  const bool isInvalid() const;

  virtual const bool isNumber() const;
  virtual const bool isOperation() const;

  virtual double   evaluate() const =0;
  virtual unsigned toStringLength() const =0;
  virtual unsigned toString(char*& str) const =0;

  void setInvalid(bool value);
  void setParent(AlgebraicTreeNode* parent);

protected:

  // your choice ...

};

#endif // _AlgebraicTreeNode_hpp_

////////////////////////////////////////////////////////////////////////////
// file AlgebraicTreeNumber.hpp
#ifndef _AlgebraicTreeNumber_hpp_
#define _AlgebraicTreeNumber_hpp_

#include "AlgebraicTreeNode.hpp"

class AlgebraicTreeNumber : public AlgebraicTreeNode {

public:

  virtual ~AlgebraicTreeNumber();

  AlgebraicTreeNumber(const double value=0.0);

  virtual const bool isNumber() const;
  virtual const bool isOperation() const;

  virtual double     evaluate() const;
  virtual unsigned   toStringLength() const;
  virtual unsigned   toString(char& str) const;

  void setValue(double value);

private:

  // your choice ...

};

#endif // _AlgebraicTreeNumber_hpp_
```

```
////////////////////////////////////////////////////////////////////////////
// file AlgebraicTreeOperation.hpp
#ifndef _AlgebraicTreeOperation_hpp_
#define _AlgebraicTreeOperation_hpp_

#include "AlgebraicTreeNode.hpp"

enum AlgebraicTreeOperationType {
  ZERO,ADD,SUBTRACT,MULTIPLY,DIVIDE
};

class AlgebraicTreeOperation : public AlgebraicTreeNode {

public:

  virtual ~AlgebraicTreeOperation();

  AlgebraicTreeOperation(AlgebraicTreeOperationType type=ZERO);

  virtual const bool isNumber() const;
  virtual const bool isOperation() const;

  virtual double     evaluate() const;
  virtual unsigned   toStringLength() const;
  virtual unsigned   toString(char& str) const;

  void setType(AlgebraicTreeOperationType type);
  void setChildLeft(AlgebraicTreeNode* childLeft);
  void setChildRight(AlgebraicTreeNode* childRight);

  AlgebraicTreeNode* getChildLeft();
  AlgebraicTreeNode* getChildRight();

  AlgebraicTreeNode** getChildLeftPtr();
  AlgebraicTreeNode** getChildRightPtr();

private:

  // your choice ...

};

#endif // _AlgebraicTreeOperation_hpp_
```

Extra Credit

This is optional. Implement a new subclass `AlgebraicTreeUnary.h` to support unary operations, in such a way that expressions such as “sqrt(3*4)” and “cos((sqrt(0.35)*0.4)+sin(tan(0.45)))” become valid. You can decide which unary functions to support, but you should include the identity function as a unary operation, to make expressions with unnecessary parentheses, such as “(7.3)” or “((((6*7))+(8*(-6))))” valid. Extend your evaluate() and toString() functions so that they work after these changes.