Grundlagen der C++-Programmierung

Assignment due Wednesday May 16 (23:59)

Assignment 6 - Expression Tree

In diesem Beispiel steht der Begriff *expression tree* für einen Syntax-Baum für einfache Terme. Wir wollen solche Bäume konstruieren und den Wert von Termen bestimmen. Dafür implementieren wir ausgehend von der Basisklasse `Node` Rechenoperationen in abgeleiteten Klassen. Außerdem erlauben wir Variablen, deren Werte in den Berechnungen verwendet werden können. Die Werte setzen wollen wir mit Hilfe des *Zuweisungsoperators*.

Hinweis: Das Beispiel ist ähnlich zum `ExpressionTree` aus der Vorlesung *Algorithmen und Datenstrukturen*. (Es ist hier allerdings kein Parser nötig!)

Die Klasse `Variable` soll Werte vom Typ `double` darstellen können.

- Für eine Instanz `var` soll eine Anweisung der Form `variable=123.0;` den Wert setzen.
- Und `double(var)` soll den Wert als `double` liefern.

Ziel des Aufgabenteils ist es, Operatoren zu überladen.

- Aus den unterschiedlichen `Node`-Klassen soll sich durch geeignete *Konstruktoren* ein Baum bauen lassen.
- Der Wert des Terms, den dieser Baum beschreibt, soll dann berechnet werden können (`evaluate`), indem die Werte aus einer Liste von `Variable`s in den Ausdruck eingesetzt werden.

Dabei hat ein Knoten, der einen *binären Operator* beschreibt (`BinOpNode`), zwei Kinder. In den Blättern des Baums stehen Knoten, die Variablen bezeichnen (`VarNode`). Siehe `main.cpp` für ein Beispiel, wie die Klassen zu benutzen sein sollen.

Ziel des Aufgabenteils ist es, Vererbung und virtuelle Funktionen zu nutzen.

⚠ **Beachte**, dass die Vorgabe zwar Schnittstellen definiert aber unvollständig ist.

Versuche, das C++11 Schlüsselwort override sinnvoll zu verwenden. Warum ist das hilfreich?

Optional

Schau Dir die folgenden Regeln aus den C++ Core Guidelines an und versuche, sie beim Implementieren Deiner Lösung umzusetzen.

- C.35: A base class destructor should be either public and virtual, or protected and nonvirtual
- C.52: Use inheriting constructors to import constructors into a derived class that does not need further explicit initialization
- C.121: If a base class is used as an interface, make it a pure abstract class
- C.126: An abstract class typically doesn't need a constructor

- C.128: Virtual functions should specify exactly one of virtual, override, or final
- C.145: Access polymorphic objects through pointers and references