SWE Übung 07

Melanie Gaßner

Personenkennzeichen: 2010458007

Stundenausmaß: ca. 15 Stunden

BEISPIEL 1: AUSDRUCKSBAUM

1.1 LÖSUNGSIDEE

`class StNode = BASISIKLASSE`
diese Klasse soll zur Darstellung eines Ausdrucksbaums dienen. Ein Ausdrucksbaum ist eine Baumdarstellung eines Ausdrucks, wobei jeder Knoten des Baums eine Operation oder einen Wert darstellt.
Die Klasse enthält drei Methoden, um den Baum in Preorder-, Inorder- und Postorder zu printen. Diese Funktionen durchlaufen den Baum rekursiv und geben die in den einzelnen Knoten gespeicherten Werte aus.

```
virtual T evaluate(NameList<SyntaxTree<T>*>* name_list) const = 0;
```
Diese Methode soll den Wert des Ausdrucks berechnen, der durch den Syntaxbaum dargestellt wird. Die Funktion nimmt als Eingabe einen Zeiger auf ein NameList-Objekt, das die Werte der im Ausdruck verwendeten Variablen enthält.

`class StNodeValue : public StNode<T> = ABGELEITETE KLASSE`
Mit dieser abgeleiteten Klasse sollen die Werte gespeichert werden und mittels `evaluate() und print()` dann die gespeicherten Wert wieder zurückzugeben bzw. ausgegeben werden.

`class StNodeOperator : public StNode<T> = ABGELEITETE KLASSE`
StNodeOperator speichert einen Operator und implementiert die Funktionen `evaluate() und print()` um die Berechnung des Operators durchzuführen bzw. ihn auszugeben. Die verfügbaren Operatoren sind Addition, Subtraktion, Multiplikation, Division und Potenzierung.

`class SyntaxTree`
Der Syntaxbaum wird verwendet, um einen mathematischen oder logischen Ausdruck in einer baumartigen Struktur darzustellen. Der Syntaxbaum besteht aus Knoten, wobei jeder Knoten eine Instanz der Klasse `StNode` ist. Die Klasse `SyntaxTree` fungiert als Wurzelknoten für den Baum und enthält einen Zeiger auf die Wurzel des Baums, die ebenfalls eine Instanz von `StNode` ist.

Die Klasse bietet mehrere Methoden zur Auswertung des im Baum gespeicherten Ausdrucks, zum Ausdrucken des Baums und zum Ausdrucken des Baums in verschiedenen Formaten. Zudem wird der `SyntaxTree` auch in Pre-Order-, In-Order- und Post-Order-Notation ausgegeben werden.
`void print_2d_upright(std::ostream& os) const`
gibt den Baum in einem 2D-Format aus

`void print_2d(std::ostream& os) const`
gibt den Baum in einem 2D-Format aus

`T evaluate(NameList<SyntaxTree<T>*>* name_list)`
Die evaluate-Methode berechnet den Wert des im Baum gespeicherten Ausdrucks unter Verwendung der in einem NameList-Objekt gespeicherten Namen.

`void print(std::ostream& out) const`
Die print-Methode gibt den Baum in den Notationen pre-order, in-order und post-order aus.

`class NameList = virtuelle Klasse`
Diese Klasse ist eine Schnittstelle zum Suchen von Variablen nach Namen, zum Registrieren von Variablen mit ihren Namen und Werten und zum ausgeben der Liste der registrierten Variablen bietet.

`class NameListMap : public NameList<T>`
Hier soll die konkrete Implementierung der Klasse NameList stattfinden. Dazu wird eine std::map verwendet, um die Namen und Werte der registrierten Variablen zu speichern. Die Klasse ist in der Lage, Variablen nach Namen zu suchen, Variablen mit ihren Namen und Werten zu registrieren und die Liste der registrierten Variablen zu drucken.

`class Parser = virtuelle Klasse`
Diese Klasse dient wieder nur Schnittstelle für die parse Methoden und die print Funktion um die Namensliste der Map auszugeben

`class ParseSyntaxTree : public Parser<T>`
ist von der Klasse `Parser` abgeleitet und implementiert die Methoden parse und print_name_list. Die parse-Methode kann entweder einen gegebenen string oder eine Datei parsen.

Dazu wird folgende Grammatik verwendet:

Programm = { Ausgabe | Zuweisung } .
Ausgabe = „**print**“ „(“ Ausdruck „)“ „;“ .
Zuweisung = „**set**“ „(“ Identifier „,“ Ausdruck „)“ „;“ .

Ausdruck = Term { AddOp Term } .
Term = Faktor { MultOp Faktor } .
Faktor = [AddOp] UFaktor .

UFaktor = Monom | KAusdruck .
Monom = WMonom [Exponent] .
KAusdruck = „(“ Ausdruck „)“ .
WMonom = Identifier | Real .
Exponent = „^“ [AddOp] Real .
AddOp = „+“ | „−“ .
MultOp = „*“ | „/“ .

`void print_name_list()`
wird verwendet, um den Inhalt der NameListMap auszugeben.

- Folgende Methoden werden verwendet, um zu prüfen, ob das aktuelle Symbol im Ausdruck von einem bestimmten Typ ist.
```
bool is_tb_Program() const;
bool is_tb_Output() const;
bool is_tb_Assignment() const;
bool is_tb_Expression() const;
bool is_tb_Term() const;
bool is_tb_Factor() const;
bool is_tb_UFactor() const;
bool is_tb_Monom() const;
bool is_tb_Exponent() const;
bool is_tb_AddOp() const;
bool is_tb_MultOp() const;
bool is_tb_PExpression() const;
bool is_tb_Identifier() const;
```

`double parse_Assignment();`
Die Methode parst einen Ausdruck und überprüft dabei ob es sich um eine Zuweisung handelt. Zusätzlich erstellt sie einen neuen Syntaxbaum mit dem ausgewerteten Ausdruck und registriert die Variable und ihren NamesMap.
`double parse_Program();`
delegiert entweder an `parse_Output()` oder `parse_Assignment();` weiter
`double parse_Output();`
Die Methode überprüft ob es sich um eine Output-Anweisung handelt, indem sie auf Schlüsselwörter achtet

`double parse_Expression();`
Die Methode parst einen Ausdruck, indem sie zunächst prüft, ob das aktuelle Symbol der Anfang eines Ausdrucks ist, und dann einen Term analysiert.
`double parse_Term();`
Die Methode parst einen Term, indem sie zunächst prüft, ob das aktuelle Symbol der Anfang eines Terms ist, und dann einen Faktor parst.
`double parse_Factor();`
Die Methode analysiert einen Faktor, indem sie zunächst prüft, ob das aktuelle Symbol der Anfang eines Faktors ist, und dann entweder eine vorzeichenlose Zahl, einen Klammerausdruck oder ein Zeichen und einen Faktor analysiert.
`double parse_AddOp();`
Die Methode parst eine Additionsoperationen.

`double parse_PExpression();`
Die Methode analysiert einen Klammerausdruck, indem sie prüft, ob das aktuelle Symbol der Anfang eines Klammerausdrucks ist, und dann parse_Expression aufruft, um den Ausdruck innerhalb der Klammer zu analysieren.

`double parse_Monom();`
parst ein Monom. Der monomiale Ausdruck kann entweder eine Zahl oder ein Bezeichner sein und kann einen Exponenten haben. Wenn der monomiale Ausdruck einen Exponenten hat, erstellt die Funktion einen neuen StNodeOperator-Knoten mit dem Operatortyp EXP.

`double parse_Exponent();`
pars einen Exponenten. Der Exponentenausdruck ist ein Ausdruck, der ein Exponentensymbol (^) und eine reelle Zahl enthält. Die Funktion prüft auf das Vorhandensein des Exponentensymbols und erstellt, falls gefunden, einen neuen StNodeOperator-Knoten, dessen Operatortyp auf EXP gesetzt ist.

1.2 TESTFÄLLE

- `void test_evaluate_add();`

```
TESTING EVALUATE ADD:

Root: +
left Node: 13
right Node: x
value for x: 8
Print tree in pre-order notation: + x 13
Print tree in in-order notation: x + 13
Print tree in post-order notation: x 13 +
=21
```

- `void test_evaluate_sub();`

```
TESTING EVALUATE SUB:

Root: -
left Node: 13
right Node: x
value for x: 8
Print tree in pre-order notation: - x 13
Print tree in in-order notation: x - 13
Print tree in post-order notation: x 13 -
=-5
```

- `void test_evaluate_mul();`

```
TESTING EVALUATE MUL:

Root: *
left Node: 13
right Node: x
value for x: 8
Print tree in pre-order notation: * x 13
Print tree in in-order notation: x * 13
Print tree in post-order notation: x 13 *
=104
```

- `void test_evaluate_div();`

```
TESTING EVALUATE DIV:

Root: /
left Node: 13
right Node: x
value for x: 8
Print tree in pre-order notation: / x 13
Print tree in in-order notation: x / 13
Print tree in post-order notation: x 13 /
=0.615385
```

- void test_evaluate_exp();

```
TESTING EVALUATE EXP:

Root: ^
left Node: 13
right Node: x
value for x: 8
Print tree in pre-order notation: ^ x 13
Print tree in in-order notation: x ^ 13
Print tree in post-order notation: x 13 ^
=5.49756e+11
```

- void test_print_tree();

```
TESTING PRINT TREE:

Root: *
second Root: -
left Node: y
right Node: 13
second right Node: 8
value for y: 6

Print tree 2d:
     8
-
          13
     *
           y



Print tree 2d_upright:
        -
        *            8
        y           13



Print name_list:
y: Print tree in pre-order notation: 6
Print tree in in-order notation: 6
Print tree in post-order notation: 6
```

➔ 2d_upright funktioniert leider nicht richtig

TESTFÄLLE DIE NICHT FUNKTIONIEREN
(Testfunktionen sind in der Test.cpp)

- void test_valid_input();
- void test_invalid_input();
- void test_division_by_zero();