DEPARTMENT OF COMPUTER SCIENCE, TECHNISCHE UNIVERSITÄT KAISERSLAUTERN
COMPILERS AND LANGUAGE PROCESSING TOOLS - SS17

A Parser for Mathematical Expressions

Exercise 1

Group 03
Daniele Gadler, Gopal Praveen
Alka Scaria, Stephen Banin Payin

April 30, 2017

OVERVIEW

In the present report, we describe the Parser we built for parsing and interpreting mathematical expressions through a general 'Overview' and through a more in-depth 'Technical Description'. The constructed Parser passes both default and group-designed tests.

FEATURES

- **Support for parentheses-enclosed and non-parentheses-enclosed expressions**: The resulting Parser is capable of parsing and evaluating both parentheses-enclosed expressions. e.g., (2+2) or (4+2) as well as non-parentheses-enclosed expressions (e.g:, 1+4 + 2*3). In the latter case, the Parser is able to disambiguate expressions through precedence rules and adds parentheses when and where needed. e.g:, 1+4 + 2*3 becomes ((1 + 4) + (2 * 3)).
- **Addition, subtraction, multiplication and division support**: The Parser we constructed supports the mentioned operations respectively through the following operands: "+", "–", "*", "/" and can handle complicated Results are always output as integers.
- **Redundant parentheses' and nested expressions support**: The built Parser accepts and evaluates expressions with redundant brackets. e.g: (((3))) is evaluated to 3 or (((2+1))) is evaluated as (2 + 1). Furthermore, it can also handle complex nested expressions such as ((2+1) * (5*3)) + 1 through the evaluation as Abstract Syntax Tree.

TECHNICAL DESCRIPTION

TASK 2: AST AND BINDING STRENGTH

We solved the binding strength for addition/subtraction and multiplication/division by setting adequate priority for operands (from bottom to top), as explained in the official CUP's docu-

mentation [1]. We also set priority for parentheses, as they need to be parsed into an AST when computing expressions containing redundant brackets (e.g: (((1+2)))).

```
precedence left RPAREN;
precedence left LPAREN;
precedence left  PLUS, MINUS;
precedence left  TIMES, DIV;
```

Particular attention needed to be paid to expressions preceded by a minus sign (e.g: - (1 / 4)) and negative integers (e.g: -1).

- **Negative Expressions**: To distinguish between positive and negative expressions, we implemented an instance attribute named 'sign' in ExprBinary (the parent class of all expressions). If an expression is preceded by a "-", the attribute 'sign' is set to 1. Otherwise, in case of a positive expressions, it is set to 0. In the evaluation, if the attribute 'sign' is 1, the expression will be multiplied by -1 so that will yield a negative result.

- **Negative Integers**: After all non-terminal expressions are evaluated, terminal nodes (numbers) are parsed. We distinguish between positive and negative numbers. If a number is preceded by a "-" sign, it is recognized to be negative and it is passed into the new instance of ExprNumber with a '-' in front of it. This way, it will be considered a negative integer.

ASTs are built automatically by parsing the grammar for all recognized expressions defined in the grammar in the file

TASK 3 AND 4 - VISITOR PATTERN

We implemented the Visitor Pattern by creating an interface called 'ExprVisitor' that defines methods' signatures for accessing the pretty printing and evaluation functions for the different expressions (e.g., for addition, visitAdd for printing or visitEvaluateAdd for evaluating an expression). Methods defined in this interface take the corresponding expression (e.g: ExprAdd) as input and return respectively a string for printing or an integer for evaluating the passed expression.
The class 'ExprPrinter' implements the interface 'ExprVisitor' and consequently overrides the 'visit' methods' bodies declared in 'ExprVisitor' defining methods for the different operations.
In the abstract class 'Expr', 'accept' methods are defined: this class is extended by the class 'ExprBinary'. All expressions' classes (e.g: ExprAdd, ExprSub, ExprMult, ExprDiv) extend this class and implement the 'accept' method . The only purpose of accept methods is calling the corresponding 'visit' method in the class implementing visit methods (hence, in 'ExprPrinter').
Methods for carrying out expressions (e.g: visitAdd for addition) can be thought to work in a 'recursive' manner, accessing the left-hand-side or the right-hand-side of an expression until a number is found. If a number is found, its value is returned and these numbers are processed in the manner defined by the expression itself (e.g: for visitAdd, as an addition).
For further details on the processing of negative numbers, please refer to Task 2.

BIBLIOGRAPHY

REFERENCES

[1] Official CUP Documentation from TU München. *http://www2.cs.tum.edu/projects/cup/*. Accessed on 26th April 2017.