Języki formalne i kompilatory

Projekt

Program do upraszczania wyrażeń algebraicznych

Autorzy

Ciałowicz Robert robcial@student.agh.edu.pl

Szpila Magdalena mszpila@student.agh.edu.pl

Opis

Program służy do upraszczania wyrażeń algebraicznych. Obsługuje operacje `+, -, *, /, (, )` oraz dowolną symboliczną nazwę zmiennej. Domyślnym zachowaniem programu jest pobieranie wyrażeń z pliku wsadowego `/example/example.txt` linia po linii oraz zwracanie wyników do `example/example_result.txt`. Lokalizację pliku wsadowego można zmienić poprzez zmianę marametru `inputFilePath` w klasie `main`.

Uruchamianie

1. Uruchom projekt w IntelliJ
2. Zainstaluj plugin Antlr4
3. Uruchom mvn clean package
4. Uruchom metodę `main()` klasy `Main`

Architektura i implementacja

Wykorzystane technologie:

- Java 8
- Antrl4 i Antlr4 plugin

Etapy przetwarzania wyrażeń algebraicznych:

Główna klasa programu do upraszczania wyrażeń algebraicznych składa się z następujących kroków:

Czytanie linii z pliku

```
val stream = CharStreams.fromString(polynomial);
```

Powyższa instrukcja odpowiada za podzielenie linii wczytanej z pliku jako łańcuch znaków do tablicy znaków.

Lexer

```
val lexer = new calculatorLexer(stream);
```

Tokeny

```
val tokens = new CommonTokenStream(lexer);
```

Parser

```
val parser = new calculatorParser(tokens);
```

Budowanie drzewa

```
val tree = parser.expression();
```

Ewaluacja wyrażeń

```
val result = new CalculatorVisitorImpl().visit(tree);
```

Visitor przechodzi po drzewie i zwraca wynik jako `PolynomialSum`.

Drukowanie wyniku

```
ResultParser.polynomialSumToString(result)
```

Powyższa metoda klasy `ResultParser` odpowiada za nadpisanie metody `toString()` na obiekcie typu `PolynomialSum`.

W tej metodzie zawarta jest również logika dodatkowa logika uwzględniająca poniższe przypadki: - x^1 -> x - x^0 -> 1 - 0*x -> 0 - x^-1 -> x^(-1)

Gramatyka

```
grammar calculator;

expression
   : multiplyingExpression PLUS expression # Plus
   | multiplyingExpression MINUS expression # Minus
   | multiplyingExpression # toMultiplyingExpression
   ;

multiplyingExpression
   : powExpression TIMES multiplyingExpression # Times
   | powExpression DIV multiplyingExpression # Div
   | powExpression # toPowExpression
   ;

powExpression
   : signedAtom POW signedAtom # Pow
   | signedAtom # toSignedAtom
   ;

signedAtom
   : atom # PositiveAtom
   | MINUS atom # NegativeAtom
   ;

atom
   : FLOAT # Number
   | VARIABLE # Variable
   | LPAREN expression RPAREN # Parens
   ;

LPAREN : '(' ;
RPAREN : ')' ;
PLUS : '+' ;
MINUS : '-' ;
TIMES : '*' ;
DIV : '/' ;
COMMA : ',' ;
POINT : '.' ;
POW : '^' ;

VARIABLE
   : ('a' .. 'z') | ('A' .. 'Z')
   ;

FLOAT
   : ('0' .. '9') + ('.' ('0' .. '9') +)?
   ;

fragment SIGN
   : ('+' | '-')
   ;

WS
   : [ \r\n\t] + -> skip
   ;
```

Przykład działania

- (a + 2) * 2 --> 2.0a+4.0
- (a + 2) / (a) --> 2.0(a^(-1))+1

- (a + b) / ((a + b) ^ 2) --> a(b^(-2))+b(a^(-2))+(a^(-1))+0.5(b^(-1))
- (2 * a ^ 2 + 3 * a * b) * a * b + a * b - a * b --> 2.0b(a^3)+3.0(b^2)(a^2)
- a * a --> (a^2)
- a + a --> 2.0a
- (a + 2) ^ 3 --> 12.0a+6.0(a^2)+(a^3)+8.0
- 1 + 2 --> 3.0
- (a + 2) * a --> 2.0a+(a^2)
- ((a + 2) ^ 2) / (4 * a) --> (a^(-1))+0.25a+1
- (a + 2) / (4) --> 0.25a+0.5
- (a + 2) ^ 2 --> 4.0a+(a^2)+4.0
- (a * b) / (a ^ 3) --> b(a^(-2))
- a + b + a b + 7 + a ^ 3 + b ^ 2 + 1 + 0 --> 2.0a+b
- (a + b) ^ 2 --> (b^2)+2.0ba+(a^2)
- a * (b ^ (2)) --> (b^2)a