

JPS ćwiczenia 9.

Lekser i parser



Cel i funkcje

baza.pracownik where nazwisko = "Zdebel"

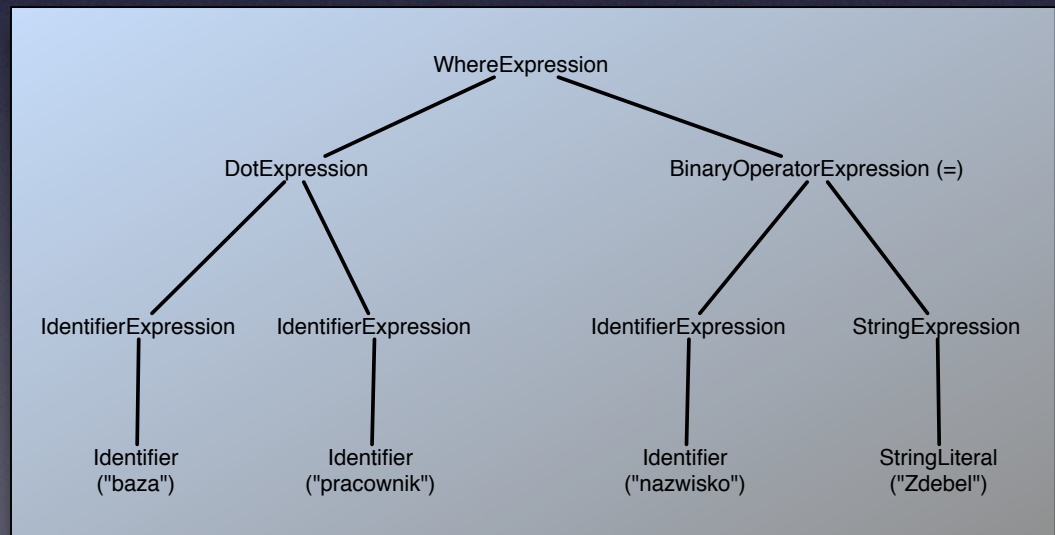


baza . pracownik where nazwisko = "Zdebel"



```
program ::=  
  wyrażenie ;  
  | instrukcja ;  
  ;
```

```
wyrażenie ::=  
  wyrażenie . wyrażenie  
  | wyrażenie where wyrażenie  
  | wyrażenie = wyrażenie  
  | wyrażenie + wyrażenie  
  | identyfikator  
  | literal  
  | ( wyrażenie )  
  ...
```



Potrzebne pliki

Program generatora parserów

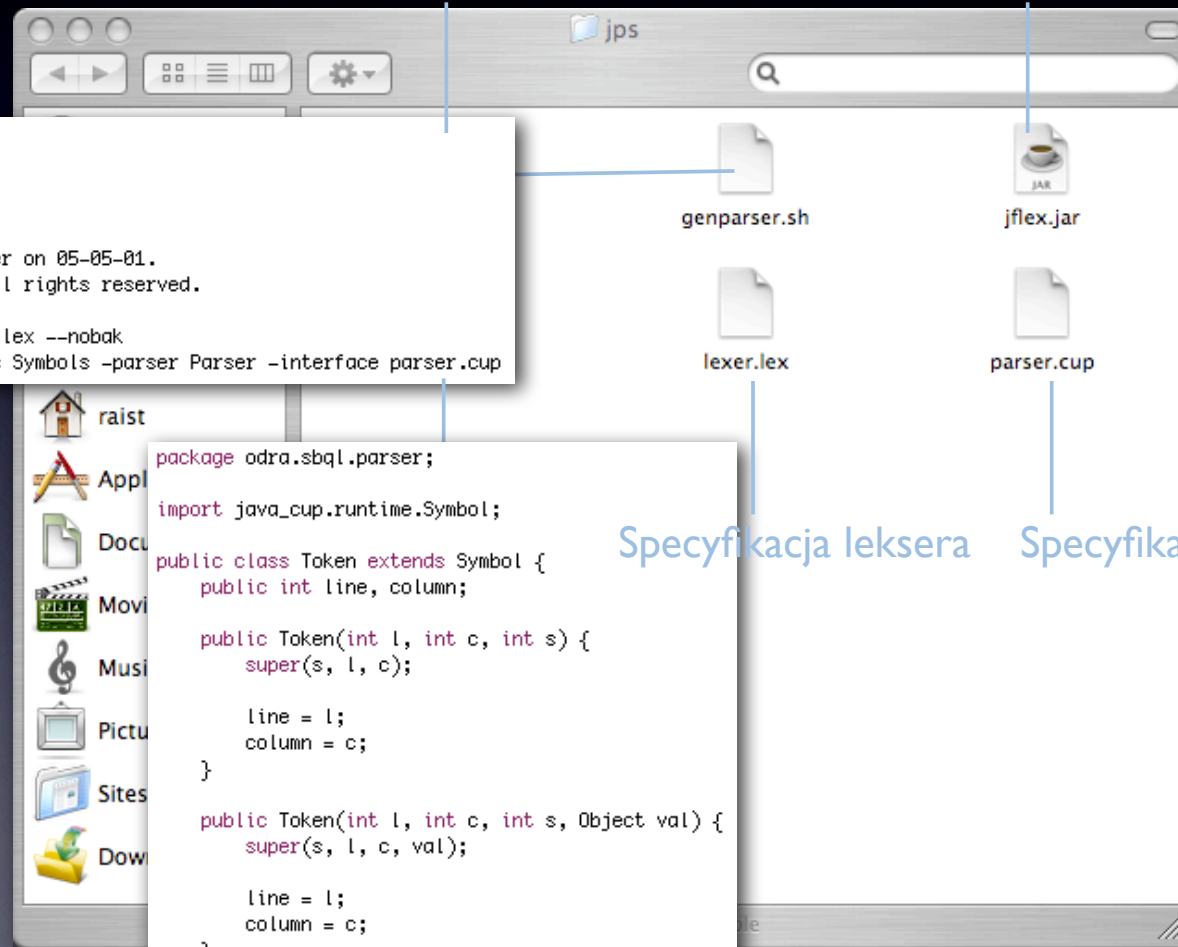
Program generatora lekserów

```
#!/bin/sh  
  
# genparser.sh  
# Odra  
#  
# Created by Michal Lentner on 05-05-01.  
# Copyright 2005 PJIIT. All rights reserved.  
  
java -jar jflex.jar lexer.lex --nobak  
java -jar cup.jar -symbols Symbols -parser Parser -interface parser.cup
```

```
package odra.sqql.parser;  
  
import java_cup.runtime.Symbol;  
  
public class Token extends Symbol {  
    public int line, column;  
  
    public Token(int l, int c, int s) {  
        super(s, l, c);  
  
        line = l;  
        column = c;  
    }  
  
    public Token(int l, int c, int s, Object val) {  
        super(s, l, c, val);  
  
        line = l;  
        column = c;  
    }  
}
```

Specyfikacja leksera

Specyfikacja parsera



```

import java_cup.runtime.Symbol;

import odra.sbql.ast.*;
import odra.sbql.ast.expressions.*;
import odra.sbql.ast.statements.*;
import odra.sbql.ast.terminals.*;

terminal Integer      INTEGER_LITERAL;
terminal String       STRING_LITERAL;
terminal String       IDENTIFIER;

terminal              SEMICOLON;
terminal              PLUS;
terminal              MINUS;
terminal              TIMES;
terminal              DIVIDE;

non terminal SBQLProgram      goal;
non terminal ExpressionStatement  expr_stmt;
non terminal Expression      expr;
non terminal IntegerLiteral    integer_literal;
non terminal StringLiteral     string_literal;
non terminal Identifier        simple_name;

precedence left PLUS, MINUS;
precedence left TIMES, DIVIDE;

start with goal;

goal ::=
    expr_stmt:s      {: RESULT = new SBQLProgram(s); :}
    ;

expr_stmt ::=
    expr:e SEMICOLON      {: RESULT = new ExpressionStatement(e); :}
    ;

expr ::=
    expr:e1 PLUS expr:e2  {: RESULT = new BinaryExpression(e1, e2, Operator.opPlus); :}
    | expr:e1 MINUS expr:e2  {: RESULT = new BinaryExpression(e1, e2, Operator.opMinus); :}
    | expr:e1 TIMES expr:e2  {: RESULT = new BinaryExpression(e1, e2, Operator.opMultiply); :}
    | expr:e1 DIVIDE expr:e2  {: RESULT = new BinaryExpression(e1, e2, Operator.opDivide); :}
    | integer_literal:l     {: RESULT = new IntegerExpression(l); :}
    | string_literal:l      {: RESULT = new StringExpression(l); :}
    | simple_name:i         {: RESULT = new IdentifierExpression(i); :}
    ;

integer_literal ::=
    INTEGER_LITERAL:l     {: RESULT = new IntegerLiteral(l.intValue()); :}
    ;

string_literal ::=
    STRING_LITERAL:l      {: RESULT = new StringLiteral(l); :}
    ;

simple_name ::=
    IDENTIFIER:i          {: RESULT = new Identifier(i); :}
    ;

```

```

lexer.lex
package odra.sqql.parser;

%%
%{
    private StringBuffer str;
}%

%public
%class Lexer
%cup
%line
%column
%state STRING
%eofval{
    return new Token(yyline, yycolumn, Symbols.EOF);
}%eofval}

INTEGER      = [0-9]+
IDENTIFIER   = [a-zA-Z_][a-zA-Z_0-9]*
TEXT        = [^\\r\\n\\\"\\\]
WHITESPACE  = {\LINETERM} | [ \t\\r\\f]
LINETERM    = \\r|\\n|\\r\\n

%%

<YYINITIAL> {
    ";" { return new Token(yyline, yycolumn, Symbols.SEMICOLON); }
    "+" { return new Token(yyline, yycolumn, Symbols.PLUS); }
    "-" { return new Token(yyline, yycolumn, Symbols.HYPHEN); }
    "*" { return new Token(yyline, yycolumn, Symbols.TIMES); }
    "/" { return new Token(yyline, yycolumn, Symbols.DIVIDE); }

    {INTEGER} {
        int val;
        try {
            val = Integer.parseInt(yytext());
        }
        catch (Exception e) {
            throw new Exception(e.getMessage());
        }
        return new Token(yyline, yycolumn, Symbols.INTEGER_LITERAL, new Integer(val));
    }

    {IDENTIFIER} {
        return new Token(yyline, yycolumn, Symbols.IDENTIFIER, yytext());
    }

    "\"" { str = new StringBuffer(); yybegin(STRING); break; }

    {WHITESPACE} { break; }

    "." { throw new Exception("Illegal character: " + yytext(), yyline + 1, yycolumn + 1); }
}

<STRING> {
    "\"" { yybegin(YYINITIAL); return new Token(yyline, yycolumn, Symbols.STRING_LITERAL, str.toString()); }
    {TEXT} { str.append(yytext()); break; }
}

```

```
1  /* The following
2
3  package odra.sbc
4
5
6  /**
7  * This class is
8  * <a href="http
9  * on 09.06.05
10 * <tt>file:/Us
11 */
12 public class Lex
13
14 /** This char
15 final public
16
17 /** initial s
18 final private
19
20 /** lexical st
21 final public
22 final public
23
24 /**
25 * Translates
26 */
27 final private
28 0, 0, 0,
29 0, 0, 0,
30 5, 0, 13,
31 1, 1, 1,
32 0, 2, 2,
33 2, 2, 2,
34 0, 2, 2,
35 2, 2, 2,
36 };
37
38 /**
39 * Translates
40 */
41 final private
42 0, 1,
43
```

```
Parser.java
Parser.java:137:9  CUP$Parser$do_action0
144     case 9: // integer_literal ::= INTEGER_LITERAL
145     {
146         IntegerLiteral RESULT = null;
147     int lleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-0)).left;
148     int lright = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-0)).right;
149     Integer l = (Integer)((java_cup.runtime.Symbol) CUP$Parser$stack.elementAt(CUP$Parser$top-0)).value;
150     RESULT = new IntegerLiteral(l.intValue());
151     CUP$Parser$result = new java_cup.runtime.Symbol(4/*integer_literal*/, ((java_cup.runtime.Symbol)CUP$Parser$
152     }
153     return CUP$Parser$result;
154
155     /*. . . . .*/
156     case 8: // expr ::= simple_name
157     {
158         Expression RESULT = null;
159     int ileft = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-0)).left;
160     int iright = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-0)).right;
161     Identifier i = (Identifier)((java_cup.runtime.Symbol) CUP$Parser$stack.elementAt(CUP$Parser$top-0)).value;
162     RESULT = new IdentifierExpression(i);
163     CUP$Parser$result = new java_cup.runtime.Symbol(3/*expr*/, ((java_cup.runtime.Symbol)CUP$Parser$stack.eleme
164     }
165     return CUP$Parser$result;
166
167     /*. . . . .*/
168     case 7: // expr ::= integer_literal
169     {
170         Expression RESULT = null;
171     int lleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-0)).left;
172     int lright = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-0)).right;
173     IntegerLiteral l = (IntegerLiteral)((java_cup.runtime.Symbol) CUP$Parser$stack.elementAt(CUP$Parser$top-0)).value
174     RESULT = new IntegerExpression(l);
175     CUP$Parser$result = new java_cup.runtime.Symbol(3/*expr*/, ((java_cup.runtime.Symbol)CUP$Parser$stack.eleme
176     }
177     return CUP$Parser$result;
178
179     /*. . . . .*/
180     case 6: // expr ::= expr DIVIDE expr
181     {
182         Expression RESULT = null;
183     int e1left = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
184     int e1right = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
185     Expression e1 = (Expression)((java_cup.runtime.Symbol) CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
186     int e2left = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-0)).left;
187     int e2right = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-0)).right;
188     Expression e2 = (Expression)((java_cup.runtime.Symbol) CUP$Parser$stack.elementAt(CUP$Parser$top-0)).value;
189     RESULT = new ExpressionDivide(e1, e2);
190     CUP$Parser$result = new java_cup.runtime.Symbol(6/*expr*/, ((java_cup.runtime.Symbol)CUP$Parser$stack.eleme
191     }
```

Jak z tego korzystać ?

```
Terminal — bash — 105x34
Last login: Thu Jun  9 11:38:26 on ttyt2
Welcome to Darwin!
Ogryzek:~ raist$ cd ~/Desktop/jps
Ogryzek:~/Desktop/jps raist$ ./genparser.sh
Reading "lexer.lex"
Constructing NFA : 48 states in NFA
Converting NFA to DFA :
.....
21 states before minimization, 15 states in minimized DFA
Writing code to "Lexer.java"
Opening files...
Parsing specification from standard input...
Checking specification...
Building parse tables...
  Computing non-terminal nullability...
  Computing first sets...
  Building state machine...
  Filling in tables...
  Checking for non-reduced productions...
Writing parser...
Closing files...
----- CUP v0.10k Parser Generation Summary
  0 errors and 0 warnings
 10 terminals, 7 non-terminals, and 13 productions
 producing 20 unique parse states.
  0 terminals declared but not used.
  0 non-terminals declared but not used.
  0 productions never reduced.
  0 conflicts detected (0 expected).
 Code written to "Parser.java", and "SymbolTable.java"
-----
Ogryzek:~/Desktop/jps raist$ javac -cp jflex.jar Parser.java
Ogryzek:~/Desktop/jps raist$ java -cp cup.jar Parser
Ogryzek:~/Desktop/jps raist$
```

```
Program.java
Program.java:18:1  <No selected symbol>
1  import java.io.*;
2
3  import odra.sbql.ast.*;
4
5  public class Program {
6      public void begin() throws Exception {
7          String prgstr = "1 + 2 + 3;";
8
9          SBQLProgram obj = (SBQLProgram) new Parser(new Lexer(new StringReader(prgstr))).parse().value;
10
11         // obj.accept(new SBQLInterpreter(), null);
12     }
13
14     public static void main(String[] args) throws Exception {
15         new Program().begin();
16     }
17 }
18
```

Ćwiczenia