Μεταγλωττιστές

Άσκηση Μέρος 2

Συντακτική Ανάλυση

Στο μέρος αυτό ξεκινούμε την υλοποίηση ενός συντακτικού αναλυτή, ο οποίος υποστηρίζει την χρήση βασικών συντακτικών κανόνων που ακολουθούν τους κανόνες της C γλώσσας. Έχοντας κάνει την λεκτική ανάλυση, αλλάζουμε την διαχείριση των αποτελεσμάτων του λεκτικού αναλυτή (εργαλείο JFlex) ώστε να τροφοδοτούνται στον συντακτικό αναλυτή. Ένας LALR(1) συντακτικός αναλυτής δημιουργείται με το εργαλείο CUP. Σε αυτό το μέρος δεν γίνεται ακόμα η παραγωγή του αφηρημένου συντακτικού δέντρου, αλλά κάνουμε μια πλήρη περιγραφή της γλώσσας με βάση:

- Τις λεκτικές μονάδες που έχουμε σαν δεδομένες από το προηγούμενο βήμα, και την σημασία που αυτές έχουν, σε γλώσσες που συντακτικά βασίζονται στη C (όπως π.χ. η JAVA) ως τερματικά σύμβολα μιας γραμματικής.
- Την δεδομένη πλήρη γραμματική της εκφώνησης:
  *<CompilationUnit> ::= (<VariableDefinition> | <FunctionDefinition>)**
  <FunctionDefinition> ::= <TypeSpecifier> <Identifier> ”(” <ParameterList>? ”)” ”{”
  <Statement> ”}”*
  <ParameterList> ::= <ParameterDeclaration>
  | <ParameterList> ”,” <ParameterDeclaration>
  <ParameterDeclaration> ::= <TypeSpecifier> <Identifier>
  | <TypeSpecifier> ”[” ”]” <Identifier>
  <TypeSpecifier> ::= ”char”
  | ”int”
  | ”float”
  | ”void”
- Τις ιδιότητες που έχει κάθε σύμβολο, όπως ο ρόλος του σε εκφράσεις και η προτεραιότητα και η προσεταιριστικότητα που αναμένεται να έχουν σε γλώσσες βασισμένες στη C.

Το project παραμένει βασισμένο στη δομή του εργαστηρίου. Το αρχείο lexer.flex έχει τον κώδικα για το εργαλείο jflex. Το αρχείο grammar.cup έχει τον κώδικα για το εργαλείο CUP. Ο κώδικας στα αρχεία αυτά μεταφράζεται σε κώδικα java αυτόματα από τα εργαλεία.

Αλλαγές στα προηγούμενα βήματα

Στον κώδικα που αντιστοιχεί στο προηγούμενο βήμα (λεκτική ανάλυση) κάναμε πρώτα αλλαγή ώστε ένας χαρακτήρας να αναγνωρίζεται μόνο με την χρήση regular expression, χωρίς να χρειάζεται νέα αρχική κατάσταση σάρωσης.
Έτσι στο αρχείο lexer.flex αφαιρέσαμε τον κώδικα για την αναγνώριση λάθους, στα regular expressions προσθέσαμε ένα για το Char και αφαιρέσαμε το %state CHAR και την διαδικασία σάρωσης με αυτό το αρχικό στάδιο.

Σύνδεση Λεκτικού αναλυτή με συντακτικό αναλυτή

Στο πρώτο και δεύτερο μέρος του lexer, εισάγουμε το Symbol από το εργαλείο cup, δίνουμε directive για σύνδεση του παραγόμενου lexer με τον παραγώμενο parser από το CUP και εισάγουμε την δήλωση μεθόδων για δημιουργία του συμβόλων. Επιπλέον προσθέτουμε και

directive για τη δημιουργία του συμβόλου EOF. Για δημιουργία συμβόλων έχουμε δημιουργήσει 2 μεθόδους, ώστε εκτός από τον τύπο συμβόλου να υπάρχει επιλογή δίνεται και τιμή ως ιδιότητα, η οποία είναι χρήσιμη στις περιπτώσεις των identifiers και των literals.

```
import java_cup.runtime.Symbol;

%%

%class Lexer
%unicode
%public
%final
%integer
%line
%column
%cup

%eofval{
   return createSymbol(sym.EOF);
%eofval}

%{
   // user custom code
   StringBuffer sb = new StringBuffer();//Hold a String.

   private Symbol createSymbol(int type) {
      return new Symbol(type, yyline+1, yycolumn+1);
   }

   private Symbol createSymbol(int type, Object value) {
      return new Symbol(type, yyline+1, yycolumn+1, value);
   }

%}

LineTerminator = \r|\n|\r\n
WhiteSpace     = {LineTerminator} | [ \t\f]
Comment        = "/*" [^*] ~"*/" | "/*" "*"+ "/"
Exponent       = [eE][\+\-]?[0-9]+
Float1         = [0-9]+ \. [0-9]+ {Exponent}?
Float2         = \.[0-9]+  {Exponent}?
Float3         = [0-9]+ \. {Exponent}?
Float4         = [0-9]+ {Exponent}
FloatLiteral   = {Float1} | {Float2} | {Float3} | {Float4}
Identifier     = [:jletter:] [:jletterdigit:]*
IntegerLiteral = 0 | [1-9][0-9]*
Char           = '\\[ nt0]' | '.'

%state STRING

%%
```

Στο τελικό μέρος του lexer χρησιμοποιούμε τις μεθόδους δημιουργίας συμβόλων με τον κατάλληλο τρόπο ανάλογα την περίπτωση:

```
//Main initial state
<YYINITIAL> {
    /* reserved keywords */
    "print"                 { return createSymbol(sym.PRINT); }
    "float"                 { return createSymbol(sym.FLOAT); }
    "int"                   { return createSymbol(sym.INT); }
    "char"                  { return createSymbol(sym.CHAR); }
    "while"                 { return createSymbol(sym.WHILE); }
    "do"                    { return createSymbol(sym.DO);}
    "if"                    { return createSymbol(sym.IF); }
    "else"                  { return createSymbol(sym.ELSE); }
    "void"                  { return createSymbol(sym.VOID); }
    "return"                { return createSymbol(sym.RETURN); }
    "break"                 { return createSymbol(sym.BREAK); }
    "continue"              { return createSymbol(sym.CONTINUE); }
    "new"                   { return createSymbol(sym.NEW); }
    "delete"                { return createSymbol(sym.DELETE); }

    /* identifiers */
    {Identifier}            { return createSymbol(sym.IDENTIFIER, yytext()); }

    /* literals */
    {IntegerLiteral}        { return createSymbol(sym.INTEGER_LITERAL, Integer.valueOf(yytext())); }
    {FloatLiteral}          { return createSymbol(sym.DOUBLE_LITERAL, Double.valueOf(yytext())); }
    {Char}                  {  return createSymbol(sym.CHARACTER_LITERAL, yytext().replaceAll("\'", "")); }

    /* String (New Initial State) */
    \"                      { sb.setLength(0); yybegin(STRING); }

    /* operators */
    "="                     { return createSymbol(sym.EQ); }
    ">"                                             { return createSymbol(sym.GREATER); }
    "<"                                             { return createSymbol(sym.LESS); }
    "!="                                        { return createSymbol(sym.NOT_EQUAL); }
    "<="                                        { return createSymbol(sym.LESS_EQ); }
    ">="                                        { return createSymbol(sym.GREATER_EQ); }
    "=="                    { return createSymbol(sym.EQUAL); }
    "+"                     { return createSymbol(sym.PLUS); }
    "-"                     { return createSymbol(sym.MINUS); }
    "*"                     { return createSymbol(sym.MULTI); }
    "/"                                             { return createSymbol(sym.DIV); }
    "%"                                             { return createSymbol(sym.MOD); }
    "&&"                                        { return createSymbol(sym.AND); }
    "||"                                        { return createSymbol(sym.OR); }
    "!"                                             { return createSymbol(sym.NOT); }
    "."                     { return createSymbol(sym.DOT); }
```

```
    /* Separators */
    "{"                                                    { return
createSymbol(sym.LCURBRACKET); }
    "}"                                                    { return
createSymbol(sym.RCURBRACKET);  }
    "["                     { return createSymbol(sym.LBRACKET); }
    "]"                     { return createSymbol(sym.LBRACKET); }
    ";"                     { return createSymbol(sym.SEMICOLON); }
    "("                                                    { return createSymbol(sym.LPAREN); }
    ")"                                                    { return createSymbol(sym.RPAREN); }
    ","                     { return createSymbol(sym.COMMA); }

    /* comments */
    {Comment}                   { /* ignore */ }

    /* whitespace */
    {WhiteSpace}                { /* ignore */ }
}

/*Initial state for String.*/
<STRING> {
    \"                      { yybegin(YYINITIAL);
                              return createSymbol(sym.STRING_LITERAL, sb.toString());
                            }

    [^\n\r\"\\]+                { sb.append(yytext()); }
    \\t                     { sb.append('\t'); }
    \\n                     { sb.append('\n'); }
    \\0                     { sb.append('\0');}
    \\\"                    { sb.append('\"'); }
    \\                      { sb.append('\\'); }
}

/* error fallback */
[^]                         { throw new RuntimeException((yyline+1) + ":" + (yycolumn+1) + ": illegal
character <"+ yytext()+">"); }
```

Έτσι υλοποιείται η σύνδεση των δύο εργαλείων. Σε κάθε αναγνώριση λεξήματος, ο λεκτικός αναλυτής δημιουργεί σύμβολο,και στη συνέχεια ο συντακτικός αναλυτής διαχειρίζεται τα παραγόμενα σύμβολα.

Περιγραφή γλώσσας στον συντακτικό αναλυτή

Το αρχείο με τον κώδικα για το εργαλείο CUP στην αρχή έχει τον κώδικα για το logging λαθών που έχει ο parser που παράγεται, με τη χρήση του Simple Logging Facade για JAVA. Ο κώδικας αυτός είναι ίδιος με του εργαστηρίου.
Στη συνέχεια δηλώνουμε τερματικά σύμβολα, τα οποία είναι είτε σταθερές, είτε identifier είτε literals:

```
//Main Terminal Symbols
terminal java.lang.String IDENTIFIER;
```

```
terminal PRINT, IF, ELSE, RETURN, NEW, DELETE;
terminal DO , WHILE, BREAK, CONTINUE;
terminal INT, FLOAT, CHAR, VOID;
terminal LPAREN, RPAREN, SEMICOLON, COMMA;
terminal EQ;
terminal LESS, LESS_EQ, GREATER, GREATER_EQ, EQUAL, NOT_EQUAL;
terminal MULTI, DIV, MOD;
terminal AND, OR, NOT, DOT;
terminal PLUS, MINUS, UMINUS;
terminal LBRACKET, RBRACKET, LCURBRACKET, RCURBRACKET;

//Literals
terminal java.lang.Integer INTEGER_LITERAL;
terminal java.lang.Double  DOUBLE_LITERAL;
terminal java.lang.String  STRING_LITERAL;
terminal java.lang.Char CHARACTER_LITERAL;
```

Τα μη τερματικά ομαδοποιούν τερματικά και μη τερματικά με βάση τους κανόνες που θα δούμε στη γραμματική. Κάθε μη τερματικό έχει συγκεκριμένες ιδιότητες, τις οποίες υποδηλώνει και το όνομα που δίνουμε στο καθένα. Τα μη τερματικά είναι:

```
//Non Terminals
non terminal Expr;              Expressions (εκφράσεις συχνά χρησιμοποιούμενες και ως παράμετροι)
non terminal Stmt;              Statements (βασικές προτάσεις π.χ. κλήση συνάρτησης ή ένα while )
non terminal CompStmt;          Compound Statement ( Ένα block κώδικα με brackets {} )
non terminal StmtList;          Λίστα από Statements
non terminal VarFuncDef;        Λίστα από δηλώσεις συναρτήσεων και global μεταβλητών στο αρχείο
non terminal FuncDef;           Δήλωση συνάρτησης
non terminal ParamDecl;         Παράμετρος σε δήλωση συνάρτησης
non terminal ParamDeclList;     Λίστα από παραμέτρους σε δήλωση συνάρτησης
non terminal TypeSpec;          Τύπος μεταβλητής ή τύπος επιστρεφόμενης τιμής από συνάρτηση
non terminal VarDef;            Δήλωση μεταβλητής
non terminal CompUnit;          "Ξεκίνημα ανάλυσης", μη τερματικό - στόχος
non terminal FuncCall;          Κλήση συνάρτησης
non terminal ObjId;             Ταυτοποίηση μεταβλητής ή πεδίου/συνάρτησης ενός αντικειμένου
non terminal DefAssign;         Αρχικοποίηση ή Ανάθεση
non terminal Assign;            Απλή ανάθεση ή νέο αντικείμενο (με χρήση constructor method)
non terminal ParamList;         Λίστα από παραμέτρους σε δήλωση συνάρτησης.
non terminal RetStmt;           Return Statement
non terminal Param;             Οποιαδήποτε δυνατή παράμετρος
```

Στη συνέχεια δηλώνουμε τις ιδιότητες προτεραιότητας και προσεταιριστικότητας:

```
//Precedence and Associativity
precedence right NOT;
precedence right EQ;
precedence left AND, OR;
precedence left EQUAL, NOT_EQUAL;
precedence left LESS , GREATER, LESS_EQ, GREATER_EQ;
```

```
precedence left PLUS, MINUS;
precedence left MULTI, DIV, MOD;
precedence right UMINUS;
```

Το μη τερματικό που είναι και ο στόχος της shift-reduce συντακτικής ανάλυσης:

```
//Start or the goal non-terminal for Shift-Reduce parsing
start with CompUnit;
```

Έχοντας όλα τα παραπάνω υπ'όψη η γραμματική που περιγράφει την γλώσσα μας είναι η εξής:

```
CompUnit ::=
        | VarFuncDef;

VarFuncDef ::= VarDef
        | FuncDef
        | VarFuncDef VarDef
        | VarFuncDef FuncDef;

VarDef ::= TypeSpec IDENTIFIER SEMICOLON
      | TypeSpec LBRACKET RBRACKET IDENTIFIER SEMICOLON
      ;

FuncDef ::= TypeSpec IDENTIFIER LPAREN ParamDeclList RPAREN CompStmt
        ;

ParamDeclList ::=
        | ParamDecl
        | ParamDeclList COMMA ParamDecl
        ;

TypeSpec ::= INT
        | FLOAT
        | CHAR
        | VOID
        ;

ParamDecl ::= TypeSpec IDENTIFIER
        | TypeSpec LBRACKET RBRACKET IDENTIFIER
        ;


CompStmt ::= LCURBRACKET RCURBRACKET
        | LCURBRACKET StmtList RCURBRACKET
        ;

StmtList ::= Stmt
        | StmtList Stmt
        ;

Stmt ::= VarDef
     | FuncCall SEMICOLON
```

```
        | DefAssign SEMICOLON
        | DELETE ObjId SEMICOLON
        | PRINT LPAREN Expr RPAREN SEMICOLON
        | WHILE LPAREN Expr RPAREN CompStmt
        | DO CompStmt WHILE LPAREN Expr RPAREN SEMICOLON
        | IF LPAREN Expr RPAREN CompStmt
        | IF LPAREN Expr RPAREN CompStmt ELSE CompStmt
        | RetStmt
        | BREAK SEMICOLON //Semantics: for while clauses only
        | CONTINUE SEMICOLON //Semantics: for while clauses only
        ;

RetStmt ::= RETURN SEMICOLON
        | RETURN Param SEMICOLON
        ;

DefAssign ::= Assign
        | TypeSpec Assign
        ;

Assign ::= ObjId EQ Expr //anything else...
        | ObjId EQ NEW FuncCall //New Object with constructor...
        ;

ObjId  ::= IDENTIFIER
        | ObjId DOT IDENTIFIER
        ;

Expr ::= INTEGER_LITERAL
        | DOUBLE_LITERAL // same as FLOAT_LITERAL
        | STRING_LITERAL
        | CHARACTER_LITERAL
        | ObjId
        | FuncCall
        | LPAREN Expr RPAREN
        | Expr PLUS Expr
        | Expr MINUS Expr
        | Expr MOD Expr
        | Expr MULTI Expr
        | Expr DIV Expr
        | Expr EQUAL Expr
        | Expr NOT_EQUAL Expr
        | Expr LESS Expr
        | Expr LESS_EQ Expr
        | Expr GREATER Expr
        | Expr GREATER_EQ Expr
        | Expr OR Expr
        | Expr AND Expr
        | NOT Expr
        | MINUS Expr %prec UMINUS
        ;
```

```
FuncCall ::= ObjId LPAREN ParamList RPAREN
    ;

ParamList ::=
    | Param
    | ParamList COMMA Param
    ;

Param ::= ObjId
    | INTEGER_LITERAL
    | DOUBLE_LITERAL
    | STRING_LITERAL
    | CHARACTER_LITERAL
    | FuncCall
    ;
```

Για έλεγχο του project, στο αρχικό directory του project:

```
mvn package
java -jar target/compiler-0.0.2.jar input1.txt
```