```
/****
 *
 * This file defines a simple tree-generating parser for a subset of the Pascal
 * programming language, with the following features:
 *
 *    -- declarations for variables, types, and procedures
 *    -- built-in types for identifiers and 1-dimensional arrays
 *    -- statements for assignment, if-then-else, and procedure call
 *    -- expressions for arithmetic, boolean relations, array reference, and
 *       function call
 *
 * Semantic action methods are defined in the companion files Tree*.java.
 *
 */

import java_cup.runtime.*;

parser code {:
    public void syntax_error(Symbol cur_token) {
        report_error("Sytax error at line " + (cur_token.left+1) +
            ", column " + cur_token.right, null);
    }

    public void initSymbolTable(int size) {
        symtab = new SymbolTable(size);
    }

    public SymbolTable getSymbolTable() {
        return symtab;
    }

    public SymbolTable symtab;
    public FunctionEntry entry;
    public final int PROC_SYMTAB_SIZE = 25;

    public void enterVars(TreeNodeList vars, TypeNode type) {
        TreeNode node;
        TreeNodeList rest;
        boolean done  = false;
        for (node = vars.node, rest = vars.siblings; !done; ) {
            symtab.enter(new VariableEntry(
                (String) (((LeafNode) node).value), type, false,
                    symtab.memorySize++, symtab.level));
            if (rest == null) {
                done = true;
            }
            else {
                node = rest.node;
                rest = rest.siblings;
            }
        }
    }

    public void enterFormals(TreeNodeList formals) {
        TreeNode2 node;
        TreeNodeList rest;
        boolean done  = false;
        if (formals == null) return;
        for (node = (TreeNode2)formals.node, rest = formals.siblings; !done; ) {
            symtab.enter(new VariableEntry(
                (String) (((LeafNode) node.child1).value),
                    (TypeNode) node.child2, false, symtab.memorySize++,
                        symtab.level));
            if (rest == null) {
                done = true;
            }
            else {
                node = (TreeNode2) rest.node;
                rest = rest.siblings;
            }
        }
    }

:}


/*-*
 * SYMBOL DEFINITIONS
 */

/*-* Terminal symbols */
terminal AND;
terminal ARRAY;
terminal BEGIN;
terminal ELSE;
terminal END;
terminal IF;
terminal NOT;
terminal OF;
terminal OR;
terminal PROGRAM;
terminal PROCEDURE;
terminal THEN;
terminal TYPE;
terminal VAR;
terminal TIMES;
terminal PLUS;
terminal MINUS;
terminal DIVIDE;
terminal UNY_PLUS;
terminal UNY_MINUS;
terminal SEMI;
terminal COMMA;
terminal LEFT_PAREN;
terminal RT_PAREN;
terminal LEFT_BRKT;
terminal RT_BRKT;
terminal EQ;
terminal GTR;
terminal LESS;
terminal LESS_EQ;
terminal GTR_EQ;
```

```
terminal NOT_EQ;
terminal COLON;
terminal ASSMNT;
terminal DOT;
terminal IDENT;
terminal INT;
terminal REAL;
terminal CHAR;

/*-* Hacks to have Pascal and EJay co-exist. */
terminal FLOAT;
terminal STRING;
terminal BOOLEAN;
terminal VOID;

/*-* Non-non terminal symbols **/
nonterminal TreeNode  program;
nonterminal TreeNode  block;
nonterminal TreeNodeList  decls;
nonterminal TreeNode  decl;
nonterminal TreeNode  typedecl;
nonterminal TreeNode  vardecl;
nonterminal TreeNode  procdecl;
nonterminal TypeNode  type;
nonterminal TypeNode  identtype;
nonterminal TypeNode  arraytype;
nonterminal TreeNodeList  vars;
nonterminal TreeNode  var;
nonterminal LeafNode  identifier;
nonterminal TreeNode  prochdr;
nonterminal TreeNodeList  formals;
nonterminal TreeNode  formal;
nonterminal TreeNodeList  stmts;
nonterminal TreeNode  stmt;
nonterminal TreeNode  assmntstmt;
nonterminal TreeNode  designator;
nonterminal TreeNode  ifstmt;
nonterminal TreeNode  proccallstmt;
nonterminal TreeNode  compoundstmt;
nonterminal TreeNodeList  exprlist;
nonterminal TreeNode  expr;
nonterminal TreeNode2  relop;
nonterminal TreeNode2  addop;
nonterminal TreeNode2  multop;
nonterminal TreeNode1  unyop;
nonterminal TreeNode  real;
nonterminal TreeNode  integer;
nonterminal TreeNode  character;

/*-* Operator Precedences **/
precedence right ASSMNT;
precedence left EQ, LESS, GTR, LESS_EQ, GTR_EQ, NOT_EQ;     /* RelOperator */
precedence left PLUS, MINUS, OR;                            /* AddOperator */
precedence left TIMES, DIVIDE, AND;                         /* MultOperator */


```

```
/*-*
 * GRAMMAR RULES
 */

program         ::= PROGRAM block:b DOT
                        {: RESULT = b; :}
                ;

block           ::= decls:d BEGIN stmts:s END
                        {: RESULT = new TreeNode2(
                               sym.BEGIN, d, s, dleft, dright); :}
                ;

decls           ::= /* empty */
                        {: RESULT = null; :}
                | decl:d
                        {: RESULT = new TreeNodeList(d, null); :}
                | decl:d SEMI decls:ds
                        {: RESULT = new TreeNodeList(d, ds); :}
                ;

decl            ::= typedecl:td
                        {: RESULT = td; :}
                | vardecl:vd
                        {: RESULT = vd; :}
                | procdecl:pd
                        {: RESULT = pd; :}
                ;

typedecl        ::= TYPE:ty identifier:i EQ type:t
                        {: RESULT = new TreeNode2(
                               sym.TYPE, i, t, tyleft, tyright); :}
                ;

type            ::= identtype:it
                        {: RESULT = it; :}
                | arraytype:at
                        {: RESULT = at; :}
                ;

identtype       ::= identifier:i
                        {: RESULT = new TypeNode(
                               sym.IDENT, i, ileft, iright); :}
                ;

arraytype       ::= ARRAY:a LEFT_BRKT integer:i RT_BRKT OF type:t
                        {: RESULT = new TypeNode(
                               sym.ARRAY, i, t, aleft, aright); :}
                ;

vardecl         ::= VAR:v vars:vs COLON type:t
                        {: RESULT = new TreeNode2(
                               sym.VAR, vs, t, vleft, vright);
                           parser.enterVars(vs, t); :}
                ;

```

```
vars            ::= var:v
                        {: RESULT = new TreeNodeList(v, null); :}
                | var:v COMMA vars:vs
                        {: RESULT = new TreeNodeList(v, vs); :}
                ;

var             ::= identifier:i
                        {: RESULT = i; :}
                ;

procdecl        ::= prochdr:ph SEMI block:b
                        {: RESULT = ph; ((TreeNode4) RESULT).child4 = b;
                           parser.entry.body = b;
                           parser.symtab = parser.symtab.ascend(); :}
                ;

prochdr         ::= PROCEDURE:p identifier:i LEFT_PAREN formals:fs RT_PAREN
                        {: RESULT = new TreeNode4(
                              sym.PROCEDURE, i, fs, null, null, pleft, pright);
                           // NOTE: the parent procdecl rule sets child4 to the
                           //       procedure block
                          parser.symtab = parser.symtab.newLevel(parser.entry =
                              new FunctionEntry((String) i.value, null, fs,
                                  null, null), parser.PROC_SYMTAB_SIZE);
                          parser.enterFormals(fs); :}
                | PROCEDURE:p identifier:i
                              LEFT_PAREN formals:fs RT_PAREN COLON identtype:it
                        {: RESULT = new TreeNode4(
                               sym.PROCEDURE, i, fs, it, null, pleft, pright);
                          parser.symtab = parser.symtab.newLevel(parser.entry =
                              new FunctionEntry((String) i.value, it, fs,
                                  null, null), parser.PROC_SYMTAB_SIZE);
                          parser.enterFormals(fs); :}
                ;

formals         ::= /* empty */
                        {: RESULT = null; :}
                | formal:f
                        {: RESULT = new TreeNodeList(f, null); :}
                | formal:f SEMI formals:fs
                        {: RESULT = new TreeNodeList(f, fs); :}
                ;

formal          ::= var:v COLON identtype:it
                        {: RESULT = new TreeNode2(
                               sym.COLON, v, it, vleft, vright); :}
                ;

stmts           ::= stmt:s
                        {: RESULT = new TreeNodeList(s, null); :}
                | stmt:s SEMI stmts:ss
                        {: RESULT = new TreeNodeList(s, ss); :}
                ;

stmt            ::= /* empty */
                        {: RESULT = new LeafNode(
                               0, null, 0, 0); :}
                | assmntstmt:as
                        {: RESULT = as; :}
                | ifstmt:is
                        {: RESULT = is; :}
                | proccallstmt:ps
                        {: RESULT = ps; :}
                | compoundstmt:cs
                        {: RESULT = cs; :}
                ;

assmntstmt      ::= designator:d ASSMNT expr:e
                        {: RESULT = new TreeNode2(
                               sym.ASSMNT, d, e, dleft, dright); :}
                ;

ifstmt          ::= IF:i expr:e THEN stmt:s
                        {: RESULT = new TreeNode3(
                               sym.IF, e, s, null, ileft, iright); :}
                | IF:i expr:e THEN stmt:s1 ELSE stmt:s2
                        {: RESULT = new TreeNode3(
                               sym.IF, e, s1, s2, ileft, iright); :}
                ;

proccallstmt    ::= identifier:i LEFT_PAREN exprlist:el RT_PAREN
                        {: RESULT = new TreeNode2(
                               sym.PROCEDURE, i, el, ileft, iright); :}
                ;

compoundstmt    ::= BEGIN:b stmts:ss END
                        {: RESULT = new TreeNode1(
                               sym.BEGIN, ss, bleft, bright); :}
                ;

expr            ::= integer:i
                        {: RESULT = i; :}
                | real:r
                        {: RESULT = r; :}
                | character:c
                        {: RESULT = c; :}
                | designator:d
                        {: RESULT = d; :}
                | var:v LEFT_PAREN exprlist:el RT_PAREN
                        {: RESULT = new TreeNode2(
                               sym.PROCEDURE, v, el, vleft, vright); :}
                | expr:e1 relop:op expr:e2
                        {: RESULT = op; op.child1 =
                           e1; op.child2 = e2; :}                  %prec EQ
                | expr:e1 addop:op expr:e2
                        {: RESULT = op;
                           op.child1 = e1; op.child2 = e2; :}      %prec PLUS
                | expr:e1 multop:op expr:e2
                        {: RESULT = op;
                           op.child1 = e1; op.child2 = e2; :}      %prec TIMES
                | unyop:op expr:e
                        {: RESULT = op; op.child = e; :}           %prec NOT
```

```
                | LEFT_PAREN expr:e RT_PAREN
                        {: RESULT = e; :}
                ;

designator      ::= var:v
                        {: RESULT = v; :}
                | designator:d LEFT_BRKT expr:e RT_BRKT
                        {: RESULT = new TreeNode2(
                               sym.LEFT_BRKT, d, e, dleft, dright); :}
                ;

exprlist        ::= expr:e
                        {: RESULT = new TreeNodeList(e, null); :}
                | expr:e COMMA exprlist:el
                        {: RESULT = new TreeNodeList(e, el); :}
                ;

relop           ::= LESS:op
                        {: RESULT = new TreeNode2(
                               sym.LESS, null, null, opleft, opright); :}
                           // NOTE: the parent expr rule sets child1 and child2
                           //       to the operand values
                | GTR:op
                        {: RESULT = new TreeNode2(
                               sym.GTR, null, null, opleft, opright); :}
                | EQ:op
                        {: RESULT = new TreeNode2(
                               sym.EQ, null, null, opleft, opright); :}
                | LESS_EQ:op
                        {: RESULT = new TreeNode2(
                               sym.LESS, null, null, opleft, opright); :}
                | GTR_EQ:op
                        {: RESULT = new TreeNode2(
                               sym.GTR, null, null, opleft, opright); :}
                | NOT_EQ:op
                        {: RESULT = new TreeNode2(
                               sym.NOT, null, null, opleft, opright); :}
                ;

addop           ::= PLUS:op
                        {: RESULT = new TreeNode2(
                               sym.PLUS, null, null, opleft, opright); :}
                | MINUS:op
                        {: RESULT = new TreeNode2(
                               sym.MINUS, null, null, opleft, opright); :}
                | OR:op
                        {: RESULT = new TreeNode2(
                               sym.OR, null, null, opleft, opright); :}
                ;

multop          ::= TIMES:op
                        {: RESULT = new TreeNode2(
                               sym.TIMES, null, null, opleft, opright); :}
                | DIVIDE:op
                        {: RESULT = new TreeNode2(
                               sym.DIVIDE, null, null, opleft, opright); :}
                | AND:op
                        {: RESULT = new TreeNode2(
                               sym.AND, null, null, opleft, opright); :}
                ;

unyop           ::= PLUS:op
                        {: RESULT = new TreeNode1(
                               sym.UNY_PLUS, null, opleft, opright); :}
                | MINUS:op
                        {: RESULT = new TreeNode1(
                               sym.UNY_MINUS, null, opleft, opright); :}
                | NOT:op
                        {: RESULT = new TreeNode1(
                               sym.NOT, null, opleft, opright); :}
                ;

identifier      ::= IDENT:i
                        {: RESULT = new LeafNode(
                               sym.IDENT, i, ileft, iright); :}
                ;

real            ::= REAL:r
                        {: RESULT = new LeafNode(
                               sym.REAL, r, rleft, rright); :}
                ;

integer         ::= INT:i
                        {: RESULT = new LeafNode(
                               sym.INT, i, ileft, iright);
                        :}
                ;

character       ::= CHAR:c
                        {: RESULT = new LeafNode(
                               sym.CHAR, c, cleft, cright); :}
                ;
```