```
/****
 *
 * 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);
    }
:}


/*-*
 * 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;

/*-* 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 TreeNode  type;
nonterminal TreeNode  identtype;
nonterminal TreeNode  arraytype;
nonterminal TreeNodeList  vars;
nonterminal TreeNode  var;
nonterminal TreeNode  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 = new TreeNode1(sym.PROGRAM, b); :}
                ;

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

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

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

identtype       ::= identifier:i
                        {: RESULT = new TreeNode1(sym.IDENT, i); :}
                ;

arraytype       ::= ARRAY LEFT_BRKT integer:i RT_BRKT OF type:t
                        {: RESULT = new TreeNode2(sym.ARRAY, i, t); :}
                ;

vardecl         ::= VAR vars:vs COLON type:t
                        {: RESULT = new TreeNode2(sym.VAR, 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; :}
                ;

prochdr         ::= PROCEDURE identifier:i LEFT_PAREN formals:fs RT_PAREN
                        {: RESULT = new TreeNode4(sym.PROCEDURE, i, fs,
                                                  null, null); :}
                           // NOTE: the parent procdecl rule sets child4 to the
                                    procedure block
                | PROCEDURE identifier:i
                             LEFT_PAREN formals:fs RT_PAREN COLON identtype:it
                        {: RESULT = new TreeNode4(sym.PROCEDURE, i, fs,
                                                  it, null); :}
                ;

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); :}
                ;

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); :}
                | 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); :}
                ;

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

```
225                 ;
226
227   proccallstmt    ::= identifier:i LEFT_PAREN exprlist:el RT_PAREN
228                         {: RESULT = new TreeNode2(sym.PROCEDURE, i, el); :}
229                 ;
230
231   compoundstmt    ::= BEGIN stmts:ss END
232                         {: RESULT = ss; :}
233                 ;
234
235   expr            ::= integer:i
236                         {: RESULT = i; :}
237                 | real:r
238                         {: RESULT = r; :}
239                 | character:c
240                         {: RESULT = c; :}
241                 | designator:d
242                         {: RESULT = d; :}
243                 | var:v LEFT_PAREN exprlist:el RT_PAREN
244                         {: RESULT = new TreeNode2(sym.PROCEDURE, v, el); :}
245                 | expr:e1 relop:op expr:e2
246                         {: RESULT = op; op.child1 =
247                            e1; op.child2 = e2; :}                    %prec EQ
248                 | expr:e1 addop:op expr:e2
249                         {: RESULT = op;
250                            op.child1 = e1; op.child2 = e2; :}        %prec PLUS
251                 | expr:e1 multop:op expr:e2
252                         {: RESULT = op;
253                            op.child1 = e1; op.child2 = e2; :}        %prec TIMES
254                 | unyop:op expr:e
255                         {: RESULT = op; op.child = e; :}             %prec NOT
256                 | LEFT_PAREN expr:e RT_PAREN
257                         {: RESULT = e; :}
258                 ;
259
260   designator      ::= var:v
261                         {: RESULT = v; :}
262                 | designator:d LEFT_BRKT expr:e RT_BRKT
263                         {: RESULT = new TreeNode2(sym.LEFT_BRKT, d, e); :}
264                 ;
265
266   exprlist        ::= expr:e
267                         {: RESULT = new TreeNodeList(e, null); :}
268                 | expr:e COMMA exprlist:el
269                         {: RESULT = new TreeNodeList(e, el); :}
270                 ;
271
272   relop           ::= LESS
273                         {: RESULT = new TreeNode2(sym.LESS, null, null); :}
274                            // NOTE: the parent expr rule sets child1 and child2
275                            //       to the operand values
276                 | GTR
277                         {: RESULT = new TreeNode2(sym.GTR, null, null); :}
278                 | EQ
279                         {: RESULT = new TreeNode2(sym.EQ, null, null); :}
280                 | LESS_EQ
281                         {: RESULT = new TreeNode2(sym.LESS, null, null); :}
282                 | GTR_EQ
283                         {: RESULT = new TreeNode2(sym.GTR, null, null); :}
284                 | NOT_EQ
285                         {: RESULT = new TreeNode2(sym.NOT, null, null); :}
286                 ;
287
288   addop           ::= PLUS
289                         {: RESULT = new TreeNode2(sym.PLUS, null, null); :}
290                 | MINUS
291                         {: RESULT = new TreeNode2(sym.MINUS, null, null); :}
292                 | OR
293                         {: RESULT = new TreeNode2(sym.OR, null, null); :}
294                 ;
295
296   multop          ::= TIMES
297                         {: RESULT = new TreeNode2(sym.TIMES, null, null); :}
298                 | DIVIDE
299                         {: RESULT = new TreeNode2(sym.DIVIDE, null, null); :}
300                 | AND
301                         {: RESULT = new TreeNode2(sym.AND, null, null); :}
302                 ;
303
304   unyop           ::= PLUS
305                         {: RESULT = new TreeNode1(sym.UNY_PLUS, null); :}
306                 | MINUS
307                         {: RESULT = new TreeNode1(sym.UNY_MINUS, null); :}
308                 | NOT
309                         {: RESULT = new TreeNode1(sym.NOT, null); :}
310                 ;
311
312   identifier      ::= IDENT:i
313                         {: RESULT = new LeafNode(sym.IDENT, i); :}
314                 ;
315
316   real            ::= REAL:r
317                         {: RESULT = new LeafNode(sym.REAL, r); :}
318                 ;
319
320   integer         ::= INT:i
321                         {: RESULT = new LeafNode(sym.INT, i); :}
322                 ;
323
324   character       ::= CHAR:c
325                         {: RESULT = new LeafNode(sym.CHAR, c); :}
326                 ;
```