Instituto Tecnológico de Costa Rica

Escuela de Computación

Compiladores e Intérpretes - IC5701

Cómo compilar el parser
Y
Código del parser

Estudiantes

Jasson Josué Moya Álvarez – 2014082335

Crisia Piedra Chaves - 201238058

I Semestre, 2016

¿Cómo compilar el parser?

La librería de cup necesita dos archivos para poder compilar el parser, los cuales son el archivo con extensión .lex y el archivo con extensión .cup.

El lexer ya se tenía desde el proyecto pasado, pero para que fuera compatible, se necesitaba modificar las reglas para que tuvieran un identificador utilizable en el parser.

El archivo que parsea tiene una estructura de código que se escribe en Java para ver los errores. Luego, tiene la definición de terminales y no terminales. Más adelante ya vienen las expresiones que conforman la gramática del lenguaje.

Para compilarlo, se necesita llamar al método adecuado (Es este caso está en la clase "Scanner", y se llama miParser.parse()), con la dirección del archivo con extensión .cup. Cuando se llama a este método, se crean dos archivos, uno que representa la clase que maneja la gramática, y otro llamado "sym", que guarda un identificador único para cada terminal definido en el lexer y el parser.

```
Parser miParser = new Parser(lexer);
miParser.parse();
```

Ilustración 1 - Código para compilar el parser

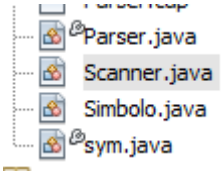

Ilustración 2 - Archivos creados por la compilación

```
------- CUP v0.11b 20140808 (SVN rev 54) Parser Generation Summary -------
  0 errors and 33 warnings
  96 terminals, 57 non-terminals, and 131 productions declared,
  producing 243 unique parse states.
  33 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 "sym.java".
```

Ilustración 3 - Resultado de la compilación

Código de la gramática

```
/* GRAMATICA PRINCIPAL */

start with INICIO ;

INICIO ::= DECLARACION_CLASE EXPRESION_BLOQUE | DECLARACION_FUNCIONES DECLARACION_VARIABLES EXPRESION_BLOQUE ;

EXPRESION_BLOQUE ::= BLOQUE_PRINCIPAL ;

EXPRESION_PRINCIPAL ::= SENTENCIA_IF | SENTENCIA_WHILE | SENTENCIA_FOR | SENTENCIA_TRY_EXCEPT;

/* Declaración de variables */
DECLARACION_VARIABLES ::= VARIABLE DECLARACION_VARIABLES | ;
VARIABLE ::= TIPOS identificador VARIABLES_JUNTAS ASIGNACION_DEFINICION salto_linea | error salto_linea;
VARIABLES_JUNTAS ::= coma identificador VARIABLES_JUNTAS | ;
ASIGNACION_DEFINICION ::= igual EXPRESION | ;
TIPOS ::= int_ |float_ | boolean_ | string | char_ | list;

/* Expresiones - ERRORES: Si viene algo después del paréntesis de cierre - Si viene unaria y luego otra cosa */
EXPRESION ::= EXPRESION_IZQUIERDA EXPRESION_PRIMA
            | OPERACIONES_UNARIAS EXPRESION_IZQUIERDA EXPRESION_PRIMA
            | parentesis_abre EXPRESION parentesis_cierra EXPRESION_PRIMA;

EXPRESION_PRIMA ::= OPERACIONES_BINARIAS EXPRESION | OPERACIONES_CONDICIONALES EXPRESION | ;

EXPRESION_IZQUIERDA ::= EXPRESION_NUMERICA | identificador | true_ | false_ | break_ | continue_ | string_literal | LISTAS | SENTENCIA_LLAMADA | FUNCION_PRINT | FUNCION_INPUT ; /* A

EXPRESION_NUMERICA ::= decimal_literal | decimal_long | hexa_literal | hexa_long | octal_literal |
                       octal_long | binario_literal | binario_long | flotante_literal | double_literal;

OPERACIONES_BINARIAS ::= multiplicacion | division | division_doble | modulo | potencia | suma | resta;
OPERACIONES_UNARIAS ::= suma | resta | not;
OPERACIONES_CONDICIONALES ::= not_igual | menor_mayor | mayor | menor_ | mayor_igual | menor_igual | igual_igual | and | or ;

/* Listas */
LISTAS ::= LISTAS_PARAMETROS ;
LISTAS_PARAMETROS ::= cuadrado_abre EXPRESION PARAMETROS_EXPRESIONES_AUX cuadrado_cierra | cuadrado_abre cuadrado_cierra ;

/* Sentencia IF */
SENTENCIA_IF ::= if_ parentesis_abre EXPRESION parentesis_cierra dos_puntos BLOQUE_PRINCIPAL punto_coma salto_linea OPCIONES_IF | if_ error punto_coma ;
BLOQUE_IF ::= elif parentesis_abre EXPRESION parentesis_cierra dos_puntos BLOQUE_PRINCIPAL punto_coma OPCIONES_ELIF | else_ dos_puntos BLOQUE_PRINCIPAL punto_coma BLOQUE_PRINCIPAL;
OPCIONES_IF ::= BLOQUE_IF | BLOQUE_PRINCIPAL;
OPCIONES_ELIF ::= BLOQUE_IF | ;
```

```
/* Sentencia Llamada */
SENTENCIA_LLAMADA ::= OPCIONES_LLAMADA PARAMETROS_EXPRESIONES ;
OPCIONES_LLAMADA ::= identificador punto identificador | identificador ;
PARAMETROS_EXPRESIONES ::= parentesis_abre EXPRESION PARAMETROS_EXPRESIONES_AUX parentesis_cierra | parentesis_abre parentesis_cierra;
PARAMETROS_EXPRESIONES_AUX ::= coma EXPRESION PARAMETROS_EXPRESIONES_AUX | ;

/* Sentencia While */
SENTENCIA_WHILE ::= while_ parentesis_abre EXPRESION parentesis_cierra dos_puntos BLOQUE_PRINCIPAL punto_coma BLOQUE_PRINCIPAL;

/* Sentencia For */
SENTENCIA_FOR ::= for_ identificador in range parentesis_abre EXPRESION FOR_AUX parentesis_cierra dos_puntos BLOQUE_PRINCIPAL punto_coma BLOQUE_PRINCIPAL;
FOR_AUX ::= coma EXPRESION | ;

/* Sentencia Try Except */
SENTENCIA_TRY_EXCEPT ::=  try_ dos_puntos BLOQUE_PRINCIPAL punto_coma SENTENCIA_EXCEPT;
SENTENCIA_EXCEPT ::= except identificador dos_puntos BLOQUE_PRINCIPAL punto_coma OPCIONES_EXCEPT;
SENTENCIA_FINALLY ::= finally_ dos_puntos BLOQUE_PRINCIPAL punto_coma BLOQUE_PRINCIPAL;
OPCIONES_EXCEPT ::= SENTENCIA_FINALLY | BLOQUE_PRINCIPAL;

/* Funcion Print */
FUNCION_PRINT ::= print_ FUNCION_PRINT_AUX ;
FUNCION_PRINT_AUX ::= parentesis_abre  EXPRESION_PRINT parentesis_cierra | parentesis_abre  parentesis_cierra;
EXPRESION_PRINT ::=  EXPRESION | EXPRESION_CASCADA ;
EXPRESION_CASCADA ::= EXPRESION coma EXPRESION_PRINT ;

/* Funcion Input */
EXPRESION_INPUT_RETURN ::= input | return_ ;
FUNCION_INPUT ::= EXPRESION_INPUT_RETURN FUNCION_INPUT_AUX ;
FUNCION_INPUT_AUX ::= parentesis_abre EXPRESION_INPUT parentesis_cierra;
EXPRESION_INPUT ::= EXPRESION | ;

/* Declaración Función */
DECLARACION_FUNCIONES ::= def identificador parentesis_abre DECLARACION_FUNCIONES_AUX parentesis_cierra dos_puntos
                          DECLARACION_VARIABLES BLOQUE_PRINCIPAL punto_coma salto_linea DECLARACION_FUNCIONES | salto_linea | ;
DECLARACION_FUNCIONES_AUX ::= | DECLARACION_PARAMETROS;
DECLARACION_PARAMETROS ::=   DECLARACION_PARAMETROS_AUX | DECLARACION_VARIABLE_FUNCION ;
DECLARACION_PARAMETROS_AUX ::= DECLARACION_VARIABLE_FUNCION coma DECLARACION_PARAMETROS ;
DECLARACION_VARIABLE_FUNCION ::= TIPOS identificador ;

/* DECLARACION DE CLASES*/
DECLARACION_CLASE ::= class_ identificador dos_puntos DECLARACION_VARIABLES DECLARACION_FUNCIONES BLOQUE_PRINCIPAL punto_coma salto_linea;

/* Bloque Principal */
BLOQUE_PRINCIPAL ::= identificador IGUALES EXPRESION salto_linea BLOQUE_PRINCIPAL | EXPRESION salto_linea BLOQUE_PRINCIPAL | EXPRESION_PRINCIPAL salto_linea BLOQUE_PRINCIPAL
                   | identificador error salto_linea BLOQUE_PRINCIPAL | EXPRESION error salto_linea BLOQUE_PRINCIPAL | EXPRESION_PRINCIPAL error salto_linea BLOQUE_PRINCIPAL | ;
IGUALES ::= igual | mas_igual | menos_igual | multiplicacion_igual |  division_igual | potencia_igual | division_doble_igual ;

/* Con el bloque de esta manera, se va a animadar el salto linea linealmente con cada bloque principal que se escriba en el código */
```

Ilustración 4 - Gramática (El archivo original se encuentra adjunto a la entrega)