Manual Técnico

Language Code Editor

VERSIÓN 1.0
FECHA: 03/09/2020
DEVELOPER: SOFIA GUTIERREZ
ORGANIZACIÓN: CUNOC-USAC

1 Descripción

La aplicación consiste en un editor de código que tenga la capacidad de reconocer y compilar una serie de lenguajes, para ello la aplicación deberá tener un repositorio en donde se almacenarán los archivos que contienen la definición de las expresiones regulares, gramática y reglas semánticas de los lenguajes soportados por la aplicación.

2 Datos de Desarrollo

El Software fue desarrollado en un computador con las siguientes características:

- ★ OS: Linux Mint 19.3 ‘Tricia’ MATE
- ★ RAM: 6 GB
- ★ Procesador: Intel core i5
- ★ Lenguaje: Java 11
- ★ IDE: NetBeans 12
- ★ Dependencias utilizadas:
  - JAVA CUP version 11.
  - JFLEX versión 1.7.0

3 Analizadores

3.1 Analizadores Léxicos

Los analizadores léxicos constituyen la primera fase de un compilador, consiste en una clase que recibe como entrada el ‘código fuente’ de un programa, a través de una secuencia de caracteres y produce una salida en forma de tokens.

En este caso el analizador se trabajó a través de estados, los cuales fueron:

- ❑ **Estados**
  - ★ JAVA_CODE
  - ★ REG_EX
  - ★ SYM
  - ★ GRAMMAR

- ❑ **Reglas léxicas**
  - ★ Int = ({Number})({Number})*
  - ★ Version = ({Int})(("."){Int}))*
  - ★ Id = ({Letter})({Letter} | {Number})+
  - ★ TraditionalCommentary = "/*" [^*] ~"*/"
  - ★ BasicCommentary = "//" {InputCharacter}* {LineTerminator}?
  - ★ Commentary = {TraditionalCommentary} | {BasicCommentary}

- **Palabras reservadas**
  - ★ nombre
  - ★ version
  - ★ autor
  - ★ lanzamiento
  - ★ extension
  - ★ terminal
  - ★ no
  - ★ entero
  - ★ real
  - ★ cadena
  - ★ RESULT

- **Otros**
  - ★ :
  - ★ ;
  - ★ ,
  - ★ ::
  - ★ *
  - ★ %%
  - ★ &
  - ★ +
  - ★ ?
  - ★ =
  - ★ {
  - ★ }
  - ★ [
  - ★]
  - ★ (
  - ★)
  - ★ \n
  - ★ \t
  - ★ \b
  - ★ “

3.2 Analizador Sintáctico

Parte media del compilador, transforma la entrada de un árbol de derivación. El análisis sintáctico convierte los tokens recibidos por el analizador léxico en otras estructuras, entre las cuales la más común son los árboles, ya que son útiles para el análisis posterior y capturar la jerarquía implícita en la entrada.

- ★ **G** = {N,T,P,S}
- ★ **N** = { name, version, autor, extension, extended_id, java_code, terminal_id, non_terminal_id, semantic_rules, code_j, release, character, language_info,

lexical_exp, symbols_list, symbol_type, expression, str, bi_operation, uni_operation, grammar, id_asig }

- **T** = { NOMBRE, VERSION, AUTOR, LANZAMIENTO, EXTENSION, SEMICOLON, COLON, SEPARATOR, ID, REGEX,
- JAVAC, QUESTION_MARK, ASTERISK, PLUS, PIPE, ROUNDB_O, ROUNDB_C, SQUAREB_O, SQUAREB_C, TAB,
- NEW_LINE, SPACE, NUMBER_RANK, LETTER_RANK, EQUAL, AMPERSAND, CHAR, TERMINAL,
- NO, ENTERO, REAL, CADENA, COMMA, LOWER_C, UPPER_C, RESULT, DOUBLE_COLON, CURLYB_O, CURLYB_C,
- VER_RES, QUOTE, INT }

3.3 Analizador Semántico

Un analizador semántico es el encargado de verificar la compatibilidad entre un operador y sus operandos, que el flujo sea adecuado, que no haya duplicidad entre nombres, etc

Hay dos tipos, uno estático, en tiempo de compilación y otro dinámico en tiempo de ejecución.

El análisis semántico dentro del código se realizó mediante un árbol de expresiones, en el caso del archivo pintura, mientras que en el resto de archivos se realizó mediante una tabla de símbolos.