UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
Centro Universitario de Occidente CUNOC
Organización de lenguajes y compiladores 2

Manual Técnico

José Carlos Soberanis Ramírez
201730246

MANUAL TÉCNICO

- ¿Qué es un compilador?
    - Es un tipo de traductor que transforma un programa entero de un lenguaje de programación a otro.
    - Podemos mencionar que está compuesto por tres fases básicas
        - Análisis léxico
            - Se encarga de identificar en la cadena de entrada los caracteres aceptados por el lenguaje, para posteriormente agruparlas y retornarnos estos valores de caracteres agrupados en forma de tokens.
                - Un token es una agrupación de caracteres reconocidos por el analizador léxico que constituyen los símbolos con los que se forman las sentencias del lenguaje y también se les denomina componentes léxicos.
        - Análisis sintáctico
            - Se encarga de verificar que el orden de los tokens sea el correcto, para este proceso existen dos tipos de analizadores sintácticos.
                - Ascendentes
                - Descendentes
        - Análisis semántico
            - Se encarga de verificar que todos los tokens que ya han sido reconocidos y validados de forma sintáctica tengan coherencia en el contexto en el que se encuentran.
            - Ejemplo, podríamos mencionar que tenemos dos variables una almacena una cadena y la otra un número, si tenemos la cadena “a * b”, sintácticamente estaría correcto en un lenguaje que acepte multiplicaciones de identificadores, pero claro, si uno de esos identificadores almacena una cadena, habría un error semántico.
- ¿Qué es Flex&Cup?

  Es una aplicación java que mediante un archivo de entrada, realiza los tres procesos específicados anteriormente para poder crear un analizador léxico y sintáctico que se usará posteriormente para evaluar cadenas de entrada que el usuario desee, siguiendo los siguientes pasos:

    - Ingresamos un archivo .len para cargar el lenguaje
    - Seleccionamos el lenguaje en el menú de lenguajes
    - En el menú de acciones, usamos la opción compilar
        - Esta opción nos permite analizar una cadena de entrada con el lenguaje que se encuentre seleccionado, que deberemos haber cargado con anterioridad.

- Estructura del código fuente

    - Estructura definida para las reglas léxicas
        - "%%" { retorna el Token(SEPARADOR) }
        - "nombre" { retorna el Token(PR_NOMBRE) }
        - "autor" { retorna el Token(PR_AUTOR) }
        - "version" { retorna el Token(PR_VERSION) }
        - "lanzamiento" { retorna el Token(PR_LANZAMIENTO) }
        - "extension" { retorna el Token(PR_EXTENSION) }
        - ":" { retorna el Token(ASIGNACION_INF) }
        - "entero" { retorna el Token(PR_ENTERO) }
        - "real" { retorna el Token(PR_REAL) }
        - "cadena" { retorna el Token(PR_CADENA) }
        - "no" { retorna el Token(PR_NO) }
        - "terminal" { retorna el Token(PR_TERMINAL) }
        - ";" { retorna el Token(FIN_DE_LINEA) }
        - ({Lmin}|("_"))({Lmin}|{Digito}|("_"))* { retorna el Token(ID_T) }
        - ({Lmay}|("_"))({Lmay}|{Digito}|("_"))* { retorna el Token(ID_NT) }
        - ({L}|("_"))({L}|{Digito}|("_"))* { retorna el Token(ID) }
        - {IntegerLiteral} { retorna el Token(ENTERO) }
        - ({IntegerLiteral}("\."{Digito})*) { retorna el Token(VERSION) }
        - (["{L}"-"{L}"]") { retorna el Token(RANGO_LETRAS) }
        - (["{Digito}"-"{Digito}"]") { retorna el Token(RANGO_NUMEROS) }
        - {Cadena} { retorna el Token(CADENA) }
        - "=" { retorna el Token(ASIGNACION_ER) }
        - "+" { retorna el Token(UNA_O_MAS_VECES) }
        - "*" { retorna el Token(CERO_O_MAS_VECES) }
        - "?" { retorna el Token(PUEDE_O_NO_PUEDE) }
        - "|" { retorna el Token(O) }
        - "[" { retorna el Token(COR_A) }
        - "]" { retorna el Token(COR_C) }
        - "(" { retorna el Token(PAR_A) }
        - ")" { retorna el Token(PAR_C) }
        - "\\n" { retorna el Token(SALTO_DE_LINEA) }
        - "\\t" { retorna el Token(TABULACION) }
        - "\\b" { retorna el Token(RETORNO) }
        - "&" { retorna el Token(IGNORAR) }
        - "," { retorna el Token(COMA) }
        - "::" { retorna el Token(ASIGNACION_GRAMA) }
        - {JavaCode} { retorna el Token(JAVA_CODE) }
        - {Comment} { ignora los comentarios }
        - {LineTerminator} { ignora los saltos de linea }
        - {WhiteSpace} { ignora los espacios en blanco, retornos, tabulaciones }

    - Estructura definida para las reglas sintácticas, para la cual vamos a utilizar la siguiente nomenclatura:

        - SIMBOLO_GRAMATICAL PRODUCE A
            - PRODUCCION_1
            - PRODUCCION_2
            - PRODUCCION_N

Las reglas sintácticas especificadas comienzan aquí

- S
  - seccionInformacion SEPARADOR SEPARADOR seccionER
- seccionInformacion
  - seccionInformacion dato
  - dato
- dato
  - PR_NOMBRE ASIGNACION_INF idCompuesto FIN_DE_LINEA
  - PR_AUTOR ASIGNACION_INF idCompuesto FIN_DE_LINEA
  - PR_LANZAMIENTO ASIGNACION_INF ENTERO FIN_DE_LINEA
  - PR_VERSION ASIGNACION_INF VERSION FIN_DE_LINEA
  - PR_EXTENSION ASIGNACION_INF
- SeccionER
  - ExpresionesRegulares SEPARADOR seccionSimbolos
- ExpresionesRegulares
  - expresionesRegulares declaracionExpresionRegular
  - declaracionExpresionRegular
- declaracionExpresionRegular
  - ID_T ASIGNACION_ER expresionRegular FIN_DE_LINEA
  - IGNORAR ASIGNACION_ER expresionRegular FIN_DE_LINEA
- ExpresionRegular
  - expresionRegular PUEDE_O_NO_PUEDE
  - expresionRegular CERO_O_MAS_VECES
  - expresionRegular UNA_O_MAS_VECES
  - expresionRegular expresionRegular
  - RANGO_LETRAS_MIN
  - RANGO_NUMEROS
  - SALTO_DE_LIENA
  - TABULACION
  - RETORNO
  - COR_A expresionRegular COR_C
  - PAR_A expresionRegular PAR_C
  - CARÁCTER_EXPLICITO
  - CADENA
  - ENTERO
  - VERSION
  - ID_T
- seccionSimbolos
  - declaracionesSimbolos SEPARADOR seccionGramatica
- declaracionesSimbolos
  - declaracionesSimbolos declaracionSimbolo
  - declaracionSimbolo
- declaracionSimbolo
  - PR_TERMINAL tipoSimbolo listadoIdsMinus FIN_DE_LINEA
  - PR_NO PR_TERMINAL tipoSimbolo listadoIdsMayus FIN_DE_LINEA
- SeccionGramatica
  - seccionGramatica producción
  - producción
- producción
  - ID ASIGNACION_GRAMA listadoSimbolos codigoJava FIN_DE_LINEA

        - ID ASIGNACION_GRAMA FIN_DE_LINEA
    - listadoSimbolos
        - listadoSimbolos simboloProduccion
        - simboloProduccion
    - simboloProduccion
        - ID ASIGNACION_INF ID
        - ID
    - IdCompuesto
        - idCompuesto ID
        - ID
    - TipoSimbolo
        - PR_ENTERO
        - PR_CADENA
        - PR_REAL
    - ListadoIdsMinus
        - listadoIdsMinus ID_T
        - ID_T
    - ListadoidsMayus
        - listadoidsMayus ID_NT
        - ID_NT
    - CodigoJava
        - JAVA_CODE
        -

- Documentación de todas las clases generadas y utilizadas

- Información del equipo de desarrollo

    - SO: Windows 10
    - Ram: 8 GB
    - Procesador: Intel core i7-6500u a 2.5 Ghz
    - Lenguaje: Java 13 (JDK13)
    - Librerias Utilizadas/dependencias
        - Cup versión 11
        - Jflex versión 1.8.2
    - IDE utilizado: Netbeans 11