UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1
PRIMER SEMESTRE 2024

DataForge
Proyecto 1
Manual Técnico

Samuel Isaí Muñoz Pereira - 202100119
Auxiliar: Walter Alexander Guerra Duque
Ingeniero: Mario Bautista

Analizador Léxico

Estructura del Proyecto

El analizador léxico se encuentra dentro del paquete analizadores. El código fuente se divide en dos partes principales: la especificación del analizador léxico en el archivo .flex y la implementación en el archivo Java.

Especificación del Analizador Léxico

La especificación del analizador léxico se encuentra en el archivo Lexico.flex. Utiliza la sintaxis de Java Cup para definir patrones de tokens y acciones asociadas. Algunos elementos importantes de la especificación son:

- *Definición de patrones regulares para tokens como palabras clave, identificadores, números enteros, números decimales, cadenas de texto, caracteres, etc.*
- *Asociación de acciones con cada patrón para construir los tokens correspondientes y manejar errores léxicos.*

Implementación en Java

El analizador léxico se implementa en la clase Lexico.java. Esta clase contiene el código generado a partir de la especificación en Lexico.flex. Algunos aspectos destacados de la implementación son:

- *Uso de la biblioteca Java Cup para generar el analizador léxico a partir de la especificación Flex.*
- *Manejo de listas para almacenar tokens y errores encontrados durante el análisis léxico.*
- *Definición de tokens utilizando la clase Symbol de Java Cup.*

Uso del Analizador Léxico

Para utilizar el analizador léxico en un proyecto Java, se debe instanciar la clase Lexico y llamar al método yylex(), que escanea el código fuente y devuelve los tokens encontrados. Los tokens y errores se almacenan en listas accesibles a través de métodos públicos.

java

```
BLANCOS = [ \r\t]+
ENTERO  = [0-9]+
DECIMAL = [0-9]+(\.[0-9]+)?
LETRA   = [a-zA-ZÑñ]
ID      = {LETRA}({LETRA}|{ENTERO}|"_")*
SPACE   = [\ \r\t\f\t]
STRING = \"(([^\"\\]?|\\.)*)\"
CHAR = \'(([^\"\\]?|\\.))\'
COM_SIMP = "//"([^\r\n]*)
COM_MULT = [/][*][^*]*[*]+([^/*][^*]*[*]+)*[/]



%%
"void"      {listaTokens.add(new Tokens(yytext(), "PR_VOID", yyline+"", yychar+""));       return new Symbol(sym.PR_VOID, yyline, yychar, yytext()); }
"main"      {listaTokens.add(new Tokens(yytext(), "PR_MAIN", yyline+"", yychar+""));       return new Symbol(sym.PR_MAIN, yyline, yychar, yytext()); }
"int"       {listaTokens.add(new Tokens(yytext(), "PR_INT", yyline+"", yychar+""));        return new Symbol(sym.PR_INT, yyline, yychar, yytext()); }
"double"    {listaTokens.add(new Tokens(yytext(), "PR_DOUBLE", yyline+"", yychar+""));     return new Symbol(sym.PR_DOUBLE, yyline, yychar, yytext()); }
"char"      {listaTokens.add(new Tokens(yytext(), "PR_CHAR", yyline+"", yychar+""));       return new Symbol(sym.PR_CHAR, yyline, yychar, yytext()); }
"bool"      {listaTokens.add(new Tokens(yytext(), "PR_BOOL", yyline+"", yychar+""));       return new Symbol(sym.PR_BOOL, yyline, yychar, yytext()); }
"String"    {listaTokens.add(new Tokens(yytext(), "PR_STRING", yyline+"", yychar+""));     return new Symbol(sym.PR_STRING, yyline, yychar, yytext()); }
"Console"   {listaTokens.add(new Tokens(yytext(), "PR_CONSOLE", yyline+"", yychar+""));    return new Symbol(sym.PR_CONSOLE, yyline, yychar, yytext()); }
"Write"     {listaTokens.add(new Tokens(yytext(), "PR_WRITE", yyline+"", yychar+""));      return new Symbol(sym.PR_WRITE, yyline, yychar, yytext()); }
"for"       {listaTokens.add(new Tokens(yytext(), "PR_FOR", yyline+"", yychar+""));        return new Symbol(sym.PR_FOR, yyline, yychar, yytext()); }
"while"     {listaTokens.add(new Tokens(yytext(), "PR_WHILE", yyline+"", yychar+""));      return new Symbol(sym.PR_WHILE, yyline, yychar, yytext()); }
"if"        {listaTokens.add(new Tokens(yytext(), "PR_IF", yyline+"", yychar+""));         return new Symbol(sym.PR_IF, yyline, yychar, yytext()); }
"else"      {listaTokens.add(new Tokens(yytext(), "PR_ELSE", yyline+"", yychar+""));       return new Symbol(sym.PR_ELSE, yyline, yychar, yytext()); }
"do"        {listaTokens.add(new Tokens(yytext(), "PR_DO", yyline+"", yychar+""));         return new Symbol(sym.PR_DO, yyline, yychar, yytext()); }
"break"     {listaTokens.add(new Tokens(yytext(), "PR_BREAK", yyline+"", yychar+""));      return new Symbol(sym.PR_BREAK, yyline, yychar, yytext()); }
"case"      {listaTokens.add(new Tokens(yytext(), "PR_CASE", yyline+"", yychar+""));       return new Symbol(sym.PR_CASE, yyline, yychar, yytext()); }
"switch"    {listaTokens.add(new Tokens(yytext(), "PR_SWITCH", yyline+"", yychar+""));     return new Symbol(sym.PR_SWITCH, yyline, yychar, yytext()); }
"default"    {listaTokens.add(new Tokens(yytext(), "PR_DEFECTO", yyline+"", yychar+""));     return new Symbol(sym.PR_DEFECTO, yyline, yychar, yytext()); }
"True"    {listaTokens.add(new Tokens(yytext(), "PR_TRUE", yyline+"", yychar+""));     return new Symbol(sym.PR_TRUE, yyline, yychar, yytext()); }
"False"    {listaTokens.add(new Tokens(yytext(), "PR_FALSE", yyline+"", yychar+""));     return new Symbol(sym.PR_FALSE, yyline, yychar, yytext()); }
"DefinirGlobales"    {listaTokens.add(new Tokens(yytext(), "PR_DEFGLOBAL", yyline+"", yychar+""));     return new Symbol(sym.PR_DEFGLOBAL, yyline, yychar, yytext()); }
"NewValor"    {listaTokens.add(new Tokens(yytext(), "PR_NEWVALOR", yyline+"", yychar+""));     return new Symbol(sym.PR_NEWVALOR, yyline, yychar, yytext()); }
"GraficaBarras"    {listaTokens.add(new Tokens(yytext(), "PR_GBARRAS", yyline+"", yychar+""));     return new Symbol(sym.PR_GBARRAS, yyline, yychar, yytext()); }
"GraficaPie"    {listaTokens.add(new Tokens(yytext(), "PR_GPIE", yyline+"", yychar+""));     return new Symbol(sym.PR_GPIE, yyline, yychar, yytext()); }
```

Analizador Sintáctico

Paquete y importaciones: *El código está en el paquete "Analizadores" y hace uso de algunas clases y funciones definidas en otros paquetes y archivos.*

Definiciones de terminales y no terminales: *Se definen los símbolos terminales y no terminales utilizados en el lenguaje que se está analizando.*

Código del parser: *La sección entre {} contiene el código Java que acompaña al parser generado. En este código, se definen métodos para manejar errores sintácticos y otros aspectos del análisis sintáctico.*

Reglas de producción: *A partir de la sección start with ini;, se definen las reglas de producción para el lenguaje que se está analizando. Esto especifica cómo se deben combinar los símbolos terminales y no terminales para formar las diferentes construcciones sintácticas permitidas en el lenguaje.*

Acciones semánticas: *En algunas reglas de producción, se incluyen bloques de código Java entre {}. Estas son acciones semánticas que se ejecutan cuando se reconoce esa regla de producción. Por ejemplo, se pueden realizar acciones como la declaración de variables, la inicialización de estructuras de datos, la generación de código intermedio, etc.*

Comentarios: *Se incluyen reglas para manejar comentarios tanto de una sola línea como de múltiples líneas.*

```
non terminal intruccionGrafica;
non terminal instruccionesGraficas;

precedence left MENOS;

start with ini;

ini::= codigo;

codigo::=
    codigo comentarios
    | codigo PR_PROGRAM instrucciones PR_END PR_PROGRAM
    | PR_PROGRAM instrucciones PR_END PR_PROGRAM
    | comentarios
;

instrucciones::=
    instruccion instrucciones
    | instruccion
    | error instrucciones
;

instruccion::=
    PR_VAR DOSPUNTOS tiposVariables:TipoVar DOSPUNTOS DOSPUNTOS ID:clave MENOR_QUE MENOS valores:valor PR_END PTCOMA
    {:variablesDeclaradas.put(clave, valor);   listaSimbolos.add(new Simbolos(clave, "Variable " + TipoVar.toString(), valor.toString(), claveleft, claveright));:}

    | PR_ARR DOSPUNTOS tiposVariables:TipoVar DOSPUNTOS DOSPUNTOS ARROBA ID:clave MENOR_QUE MENOS COR_IZQ valoresArreglo COR_DER PR_END PTCOMA
    {:variablesDeclaradas.put(clave, Funciones.copiaLista());  listaSimbolos.add(new Simbolos(clave, "Arreglo " + TipoVar.toString(), Funciones.textoArrelgo, claveleft, claveright));:}

    | PR_CONSOLE DOSPUNTOS DOSPUNTOS PR_PRINT IGUAL valoresArreglo PR_END PTCOMA  {:Funciones.ImprimirConsola();:}
    | PR_CONSOLE DOSPUNTOS DOSPUNTOS PR_COLUMN IGUAL valores:nombre MENOS MAYOR_QUE COR_IZQ valoresArreglo COR_DER PR_END PTCOMA   {:Funciones.ImprimirColumna(nombre);:}
    | PR_CONSOLE DOSPUNTOS DOSPUNTOS PR_COLUMN IGUAL valores:nombre MENOS MAYOR_QUE ARROBA ID:val PR_END PTCOMA   {:Funciones.ImprimirColumnaArreglo(Funciones.busquedaLista(val), nombre);:}
    | GraficaLineas
    | GraficaBarras
    | GraficaPie
    | GraficaHistograma
    | comentarios
;
```

Función para Operaciones Matematicas

El método comienza por convertir los valores de tipo Object a double utilizando el método Double.parseDouble(). Luego, utiliza una estructura switch para determinar la operación a realizar según el valor de la cadena operacion. Dependiendo del caso, realiza la operación correspondiente (SUM para suma, RES para resta, MUL para multiplicación, DIV para división y MOD para módulo).

Si la operación especificada no coincide con ninguna de las opciones válidas, se lanza una excepción de tipo IllegalArgumentException indicando que la operación no es válida.

Si ocurre algún error durante la conversión de los valores o durante la operación aritmética, se lanza una excepción de tipo RuntimeException con el mensaje "Error al sumar los valores".

Funcion para Operaciones Estadisticas

El método comienza convirtiendo los elementos del ArrayList original a double y los almacena en una nueva lista temporal (ArrayList<Double> listaTemp). Esto se hace para garantizar que los cálculos estadísticos se realicen sobre valores numéricos.

Luego, se utiliza una estructura switch para determinar qué medida estadística calcular según el valor de la cadena funcion. Las medidas estadísticas implementadas incluyen:

- **Media (MEDIA)**: Se calcula sumando todos los elementos de la lista y dividiendo la suma por el número total de elementos.
- **Mediana (MEDIANA)**: Se ordena la lista y se selecciona el valor central si el tamaño de la lista es impar, o se calcula el promedio de los dos valores centrales si es par.
- **Moda (MODA)**: Se identifica el valor que más se repite en la lista.
- **Varianza (VARIANZA)**: Se calcula la varianza de la lista.
- **Máximo (MAX)**: Se determina el valor máximo en la lista.
- **Mínimo (MIN)**: Se determina el valor mínimo en la lista.

Si ocurre algún error durante los cálculos, se imprime un mensaje de error en la salida estándar de error. Si la función especificada no coincide con ninguna de las opciones válidas, no se realiza ningún cálculo y el método devuelve 0.

Manejo de Gráficas

1. histograma(): Este método genera un histograma a partir de una lista de valores numéricos. Calcula la frecuencia de cada valor,

la frecuencia acumulada y la frecuencia relativa. Luego, crea un gráfico de barras 3D utilizando JFreeChart y lo muestra en la interfaz gráfica.

2. graficaPie(): Este método genera un gráfico de pastel a partir de una lista de valores numéricos y una lista de etiquetas. Utiliza estos datos para crear un gráfico de pastel utilizando JFreeChart y lo muestra en la interfaz gráfica.
3. graficaBarras(): Este método genera un gráfico de barras a partir de una lista de valores numéricos y una lista de etiquetas para el eje X. Utiliza estos datos para crear un gráfico de barras 3D utilizando JFreeChart y lo muestra en la interfaz gráfica.
4. graficaLineas(): Este método genera un gráfico de líneas a partir de una lista de valores numéricos y una lista de etiquetas para el eje X. Utiliza estos datos para crear un gráfico de líneas utilizando JFreeChart y lo muestra en la interfaz gráfica.
5. GuardarGrafica(): Este método guarda una gráfica como un archivo PNG en la carpeta "Graficas" en el directorio actual. Toma el nombre del archivo y el objeto JFreeChart de la gráfica como parámetros.

La clase también contiene una variable estática GraficasAlmacenadas que mantiene un mapa de nombres de gráficas y los objetos JFreeChart correspondientes. Esto con la finalidad de que el usuario pueda cambiar entre las graficas utilizando el combo box, y que de esa manera pueda tener una grafica mas interactiva

Cada método está diseñado para manejar posibles errores mediante el manejo de excepciones y proporciona retroalimentación a través de la consola en caso de falla. Además, estos métodos están diseñados para trabajar con una interfaz gráfica de usuario (GUI) en algún contexto, ya que muestran las gráficas generadas en dicha interfaz.

Interfaz Gráfica (GUI)

La interfaz gráfica también conocida como GUI es la forma en la que podemos utilizar un programa haciendo uso de ventanas, de este modo se agregan funcionalidades extras que permite a los usuarios navegar dentro del programa de una manera mas cómoda y accesible comparado con un programa por terminal

en este caso el programa cuenta con las opciones básicas para el manejo de archivos, el uso del programa, la generación de reportes y la documentación correspondiente