Universidad San Carlos de Guatemala
Organización de Lenguajes y Computadores 1

Manual técnico

Nataly Saraí Guzmán Duarte

Carnet: 202001570

Índice

Introducción

El presente documento se adjuntan las características, funcionamiento y estructuras que fueron requeridas para el desarrollo del sistema presentado el cual consta de un sistema que traduce archivos Json a una tabla de símbolos y traducción de codigo statpy a Python, tambien grafica las funciones.

Lugar de realización: Guatemala

Fecha: 16/09/2023

Responsable de elaboración: Nataly Saraí Guzmán Duarte

Información destacada

El programa fue realizado utilizando como plataforma visual studio code, la realización de la fase fue de aproximadamente 4 semanas, cuya realización fue separada en partes del menú, cada parte se realizo en aproximadamente 5 días su realización completa.

Objetivos y alcances del sistema

- Aprender a Generar Analizadores Léxicos y Sintácticos utilizando las herramientas JFLEX y CUP.
- Comprender Conceptos Clave como token, lexema, patrones y expresiones regulares. Estos conceptos son fundamentales para el análisis léxico.
- Manejar correctamente los errores léxicos que puedan surgir durante el proceso de análisis del código fuente.
- Realizar Acciones Gramaticales en JAVA.

Requerimientos

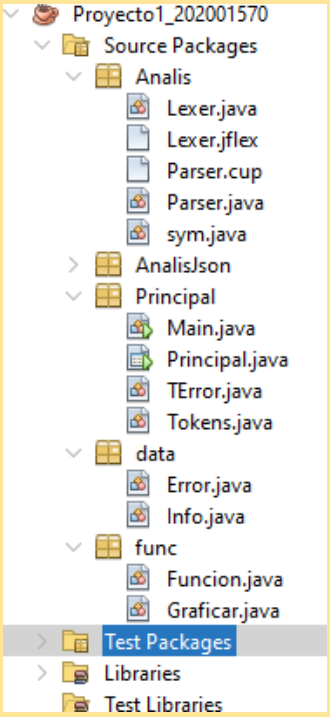
- RAM: 1 GB (mínimo).
- ROM: 250 MB (mínimo).
- Arquitectura x32 bits o x64 bits.
- Sistema operativo: Windows, Linux, MacOS.
- Lenguaje de programación: Java.
- Plataforma IDE: visual studio code.

Datos técnicos para la realización del sistema

- Windows 10 Home Single Language.
- Procesador: Intel Core i5.
- RAM: 8.00 GB
- Plataforma IDE: visual studio code.

Lógica del programa

El programa está constituido por intefaz grafica creada en el lenguaje de programación java es simple e intuitivo con el usuario mostraremos a continuación las distintas clases que se usaron para su creación.



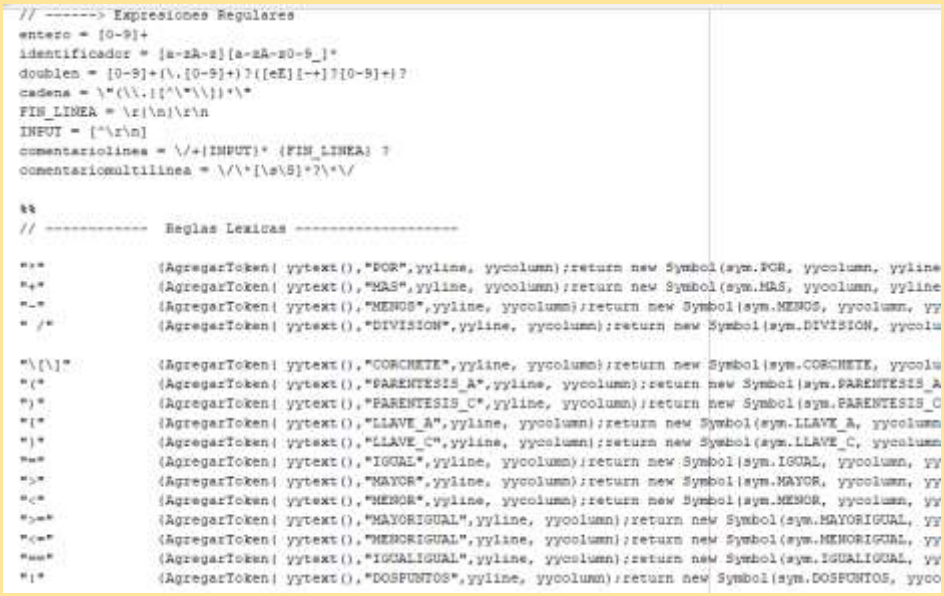
Carpeta Analis: En esta carpeta se realizaron los analizadores léxico y sintáctico correspondiente a la traducción statpy a Python, tambien se incluyeron las funciones pero solo fueron analizadas y no traducidas.

Para generar estos analizadores se utilizo

```
public static void main(String[] args) {
    analizadores("src/Analis/", jflexFile: "Lexer.jflex", cupFile: "Parser.cup");
    //analizadores("src/AnalisJson/", "Lexer.jflex", "Parser.cup");
    //Principal m = new Principal();
    /*m.setVisible(true);
    /*double [] valores = {12.1,19.1,91.2};
    String [] ejes = {"Dos", "Jua","JOA"};
    func.Graficar.barras("Prueba","Titulos", "Titulos", valores, ejes);*/
}
public static void analizadores(String ruta, String jflexFile, String cupFile){
    try {
        String opcionesJflex[] =  {ruta+jflexFile,"-d",ruta};
        jflex.Main.generate(...opcionesJflex);

        String opcionesCup[] =  {"-destdir", ruta,"-parser","Parser",ruta+cupFile};
        java_cup.Main.main(...opcionesCup);

    } catch (Exception e) {
        System.out.println(x: "No se ha podido generar los analizadores");
        System.out.println(x: e);
    }
}
```

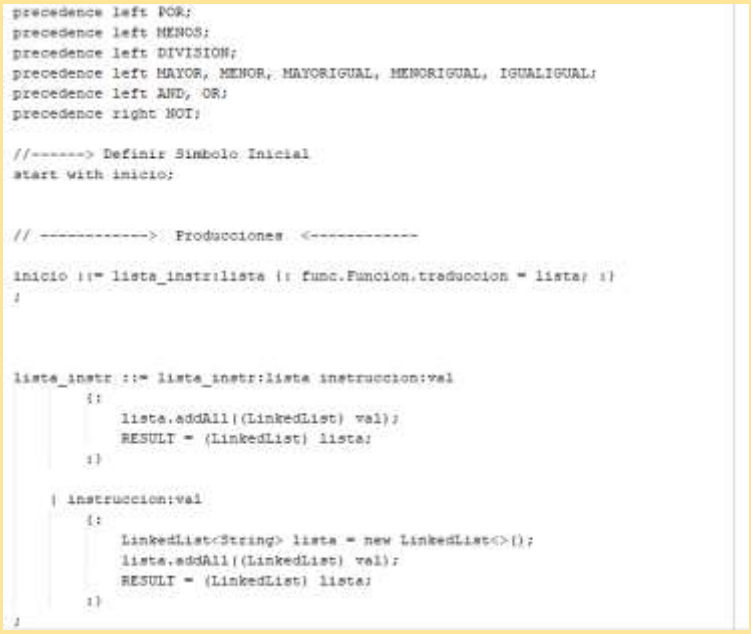
Para el analizador léxico se crearon expresiones regulares y todas estás fueron declaradas con return para poder ser utilizadas en el analizador sintactico, tambien

en esta clase se crearon lista de errores y de lexemas para la realización de reportes.



```
// ------> Expresiones Regulares
entero = [0-9]+
identificador = [a-zA-z][a-zA-z0-9_]*
doublen = [0-9]+(\.[0-9]+)?([eE][-+]?[0-9]+)?
cadena = \"(\\.|[^\"\\])*\"
FIN_LINEA = \r|\n|\r\n
INPUT = [^\r\n]
comentariolinea = \/+(INPUT)* (FIN_LINEA) ?
comentariomultilinea = \/\*[\s\S]*?\*\/

%%
// ------------  Reglas Lexicas --------------------

"*"             {AgregarToken( yytext(),"POR",yyline, yycolumn);return new Symbol(sym.POR, yycolumn, yyline
"+"             {AgregarToken( yytext(),"MAS",yyline, yycolumn);return new Symbol(sym.MAS, yycolumn, yyline
"-"             {AgregarToken( yytext(),"MENOS",yyline, yycolumn);return new Symbol(sym.MENOS, yycolumn, yy
"/"             {AgregarToken( yytext(),"DIVISION",yyline, yycolumn);return new Symbol(sym.DIVISION, yycolu

"\{\]"          {AgregarToken( yytext(),"CORCHETE",yyline, yycolumn);return new Symbol(sym.CORCHETE, yycolu
"("             {AgregarToken( yytext(),"PARENTESIS_A",yyline, yycolumn);return new Symbol(sym.PARENTESIS_A
")"             {AgregarToken( yytext(),"PARENTESIS_C",yyline, yycolumn);return new Symbol(sym.PARENTESIS_C
"{"             {AgregarToken( yytext(),"LLAVE_A",yyline, yycolumn);return new Symbol(sym.LLAVE_A, yycolumn
"}"             {AgregarToken( yytext(),"LLAVE_C",yyline, yycolumn);return new Symbol(sym.LLAVE_C, yycolumn
"="             {AgregarToken( yytext(),"IGUAL",yyline, yycolumn);return new Symbol(sym.IGUAL, yycolumn, yy
">"             {AgregarToken( yytext(),"MAYOR",yyline, yycolumn);return new Symbol(sym.MAYOR, yycolumn, yy
"<"             {AgregarToken( yytext(),"MENOR",yyline, yycolumn);return new Symbol(sym.MENOR, yycolumn, yy
">="            {AgregarToken( yytext(),"MAYORIGUAL",yyline, yycolumn);return new Symbol(sym.MAYORIGUAL, yy
"<="            {AgregarToken( yytext(),"MENORIGUAL",yyline, yycolumn);return new Symbol(sym.MENORIGUAL, yy
"=="            {AgregarToken( yytext(),"IGUALIGUAL",yyline, yycolumn);return new Symbol(sym.IGUALIGUAL, yy
":"             {AgregarToken( yytext(),"DOSPUNTOS",yyline, yycolumn);return new Symbol(sym.DOSPUNTOS, yyco
```

Para el analizador sintáctico se crearon terminales, no terminales y precedencias para ser utilizadas en las gramáticas, en la gramática se visualizó como deberían de ir las instrucciones y asi poder ser traducidas correctamente. Y al igual que el analizador sintáctico se creo la lista de errores para ser utilizado en los reportes, se observo si el error puede ser recuperado o no tiene recuperación.



```
precedence left POR;
precedence left MENOS;
precedence left DIVISION;
precedence left MAYOR, MENOR, MAYORIGUAL, MENORIGUAL, IGUALIGUAL;
precedence left AND, OR;
precedence right NOT;

//------> Definir Simbolo Inicial
start with inicio;


// ------------>  Producciones  <------------

inicio ::= lista_instr:lista {: func.Funcion.traduccion = lista; :}
;



lista_instr ::= lista_instr:lista instruccion:val
        {:
            lista.addAll((LinkedList) val);
            RESULT = (LinkedList) lista;
        :}

    | instruccion:val
        {:
            LinkedList<String> lista = new LinkedList<>();
            lista.addAll((LinkedList) val);
            RESULT = (LinkedList) lista;
        :}
;
```

Carpeta AnalisJson: En esta carpeta se realizaron los analizadores léxico y sintáctico correspondiente a la lectura del archivo Json para agregarlo a la tabla de símbolos.

Para generar estos analizadores se utilizo

```
public static void main(String[] args) {
    //analizadores("src/Analis/", "Lexer.jflex", "Parser.cup");
    analizadores("src/AnalisJson/", "Lexer.jflex", "Parser.cup");
    //Principal m = new Principal();
    //m.setVisible(true);
    /*double [] valores = {12.1,18.1,81.2};
    String [] ejes = {"Hoa", "Jua","JOA"};
    func.Graficar.barras("Prueba","Titulox", "Tituloy", valores, ejes);*/
}
public static void analizadores(String ruta, String jflexFile, String cupFile){
    try {
        String opcionesJflex[] =  {ruta+jflexFile,"-d",ruta};
        jflex.Main.generate(opcionesJflex);

        String opcionesCup[] =  {"-destdir", ruta,"-parser","Parser",ruta+cupFile};
        java_cup.Main.main(opcionesCup);

    } catch (Exception e) {
        System.out.println("No se ha podido generar los analizadores");
        System.out.println(e);
    }
}
```

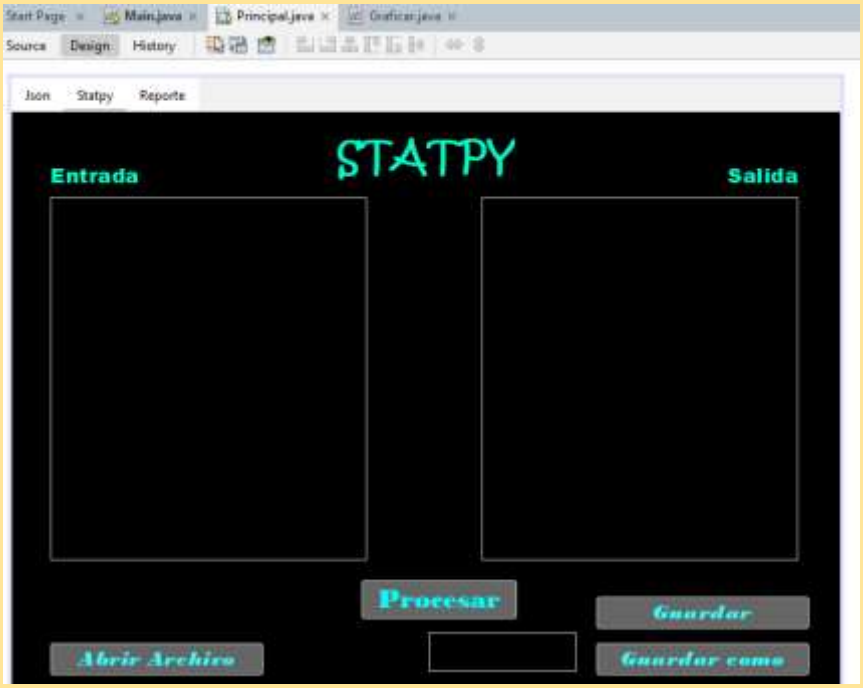
Para el analizador léxico se crearon expresiones regulares y todas estás fueron declaradas con return para poder ser utilizadas en el analizador sintactico, tambien en esta clase se crearon lista de errores y de lexemas para la realización de reportes.

```
// ------> Expresiones Regulares
nombrear = [a-zA-Z0-9_\-]+\.(json)
double = [0-9]+(\.[0-9]+)?([eE][-+]?[0-9]+)?
cadena = \"[\w\d]+\"
caden = \"(\\.|[^\"\\])*\"
FIN_LINEA = \r|\n|\r\n
INPUT = [^\r\n]
comentariolinea = \/+(INPUT)* |FIN_LINEA) ?
comentariomultilinea = \/\*[\s\S]*?\*\/

%%
// ------------  Reglas Lexicas --------------------


"{"             {AgregarToken( yytext(),"LLAVE_A",yyline, yycolumn);return new Symbol(sym.LLAVE_A, yycolumn,
"}"             {AgregarToken( yytext(),"LLAVE_C",yyline, yycolumn);return new Symbol(sym.LLAVE_C, yycolumn,
":"             {AgregarToken( yytext(),"DOSPUNTOS",yyline, yycolumn);return new Symbol(sym.DOSPUNTOS, yycol

","             {AgregarToken( yytext(),"COMA",yyline, yycolumn);return new Symbol(sym.COMA, yycolumn, yylin

{double}   {AgregarToken( yytext(),"DOUBLEM",yyline, yycolumn); return new Symbol(sym.DOUBLEM, yycolumn, yyl
{cadena}   {AgregarToken( yytext(),"CADENA",yyline, yycolumn); return new Symbol(sym.CADENA, yycolumn, yyline
{caden}    { AgregarToken( yytext(),"CADEN",yyline, yycolumn);return new Symbol(sym.CADEN, yycolumn, yyline, y
{comentariolinea}   {AgregarToken( yytext(),"COMENLI",yyline, yycolumn); return new Symbol(sym.COMENLI, yycol
{comentariomultilinea}   {AgregarToken( yytext(),"COMENMULTILI",yyline, yycolumn); return new Symbol(sym.COME
{nombrear}   { AgregarToken( yytext(),"NOMBREAR",yyline, yycolumn);return new Symbol(sym.NOMBREAR, yycolumn,

//------> Ingorados
[ \t\r\n\f]     {/* Espacios en blanco se ignoran */}
```

Para el analizador sintáctico se crearon terminales y no terminales para ser utilizadas en las gramáticas, en la gramática se visualizó como deberían de ir las instrucciones correspondientes a lectura del archivo Json. Y al igual que el analizador sintáctico se creo la lista de errores para ser utilizado en los reportes, se observo si el error puede ser recuperado o no tiene recuperación.

```
non terminal asignacion;
non terminal lista_instr;
non terminal instruccion;
non terminal expresion;
non terminal instrucjson;
non terminal instruog;

//------> Definir Simbolo Inicial
start with inicio;


// ------------>  Producciones  <------------

inicio ::= lista_instr
;

lista_instr ::= lista_instr instruccion
    | instruccion
;

instruccion ::=  LLAVE_A instrucjson LLAVE_C
    | COMENMULTILI
    | COMENLI
;

instrucjson ::= instrucjson COMA asignacion
    | asignacion
;
```

En la carpeta Principal contiene la clase main donde se generaron los analizadores y se manda a llamar la clase principal, en la clase principal se encuentra la interfaz grafica y las instrucciones que realiza cada botón, como abrir archivos, guardar, analizar etc.



En esta carpeta tambien se encuentra la clase Terror que es el objeto para la creación de reportes de errores, ya sean sintácticos o léxicos.

```
package Principal;

/**
 *
 * @author Natal
 */
public class TError {
    public String lexema, descripcion;
    public int linea, columna;

    public TError( String desc,String lex, int lin, int col){
        this.descripcion=desc;
        this.lexema=lex;
        this.linea=lin;
        this.columna=col;

    }
    public String get(){
        return "[ "+this.lexema+ ", "+this.descripcion+" ]";
    }
    public int getLinea(){
        return this.linea;
    }

}
```

En esta carpeta tambien se encuentra la clase Tokes que es el objeto para la creación de reportes de tokens léxicos.

```java
package Principal;

/**
 *
 * @author Natal
 */
public class Tokens {
    public String lexema, token;
    public int linea, columna;

    public Tokens(String lex, String tok, int lin, int col){
        this.lexema=lex;
        this.token = tok;
        this.linea=lin;
        this.columna=col;

    }
    public String get(){
        return "[ "+this.lexema+ ", "+this.token+" ]";
    }
    public int getLinea(){
        return this.linea;
    }

}
```

En la carpeta data se encuentra la clase Info que fue creada para guardar los linkenlist que se crearon en el analizador sintáctico para demostrar la traducción a Python.

```java
package data;

import java.util.HashMap;
import java.util.LinkedList;
/**
 *
 * @author Natal
 */
public class Info {
    public static LinkedList<Error> listaErrores = new LinkedList<>();
    public static HashMap<String, String> listaVariables = new HashMap<>();
    public static HashMap<String, HashMap<String, String>> hashMapPrincipal = new HashMap<>();
}
```

En la carpeta func se encuentra la clase Función la cual guarda la traducción y hace la tabulación necesaria para que se traduzca correctamente a Python.

```java
package func;

import java.util.HashMap;
import java.util.LinkedList;
import javax.swing.JOptionPane;
/**
 *
 * @author Natal
 */
public class Funcion {
    public static LinkedList<String> funciones = new LinkedList<>();//Lista de funciones estadisticas
    public static int contador = 0; //Contador de tabulaciones
    public static LinkedList<String> traduccion = new LinkedList<>();//Lista de traduccion

    // Agregar Tabulaciones
    public static LinkedList<String>  tabulaciones(LinkedList<String> lista){
        String tabs = "";
        for (int i = 0; i < contador; i++) {
            tabs = "\t"+tabs;
        }

        for (int i = 0; i < lista.size(); i++) {
            lista.set(i, tabs+lista.get(i));
        }

        return lista;
    }


}
```

Tambien en esta carpeta se encuentra la clase grafica en la cual se obtienen los datos de las funciones para poder realizar la grafica de barras y la gráfica de pie.

```java
    public static double[] valores ;
    public static void AgregarS(String nombre,String valor){
        nombre = valor;
        nombre = nombre.replaceAll("\"", "");
        System.out.println("Aqui");
        System.out.println(nombre);
    }
    public static void AgregarDo(String nombre,String valor){
        nombre = valor;
        System.out.println("Aqui");
        System.out.println(nombre);
    }
    public static void AgregarD(String nombre,String valor){
        nombre = valor;
        System.out.println("Aqui");
        System.out.println(nombre);

    }
    public static void pasareje(){
        // Crear un arreglo de String del mismo tamaño que el LinkedList
        ejex = new String[ejexs.size()];

        // Pasar los datos del LinkedList al arreglo
        ejex = ejexs.toArray(ejex);

        // Imprimir el contenido del arreglo
        for (String elemento : ejex) {
            System.out.println(elemento);
        }
```

Diagrama general de la aplicación:

Muestra el sistema de organización que se utilizo para desarrollar este sistema en una estructura UML de clases con ello podemos validar la fácil adaptación en caso de ser requerido a un nuevo sistema de lenguaje.