Universidad de San Carlos de Guatemala

Facultad de Ingeniería

PROYECTO 1

Nathan Antonio Valdez[2]

202001568

OBJETIVOS

1.1 Objetivos Generales

Aplicar los conocimientos sobre las fases de análisis léxico y sintáctico de un compilador para la construcción de una solución de software.

1.2 Objetivos Específicos

● Aprender a generar analizadores léxicos y sintácticos utilizando las herramientas de JFLEX y CUP.

● Comprenderlos conceptos de token, lexema, patrones y expresiones regulares.

● Realizar correctamente el manejo de errores léxicos.

● Ser capaz de realizar acciones gramaticales usando el lenguaje de programación JAVA

Requisitos del Sistema

Requisitos del Sistema Java 8

La información detallada sobre los requisitos de Java 8 está disponible en Configuraciones de Java 8 soportadas.

Windows

- Windows 10 (8u51 y superiores)
- Windows 8.x (escritorio)
- Windows 7 SP1
- Windows Vista SP2
- Windows Server 2008 R2 SP1 (64 bits)
- Windows Server 2012 y 2012 R2 (64 bits)
- RAM: 128 MB
- Espacio en disco: 124 MB para JRE; 2 MB para Java Update
- Procesador: Mínimo Pentium 2 a 266 MHz
- Exploradores: Internet Explorer 9 y superior, Firefox

Mac OS X

- Mac con Intel que ejecuta Mac OS X 10.8.3+, 10.9+
- Privilegios de administrador para la instalación
- Explorador de 64 bits

Se requiere un explorador de 64 bits (Safari, por ejemplo) para ejecutar Oracle Java en Mac.

Linux

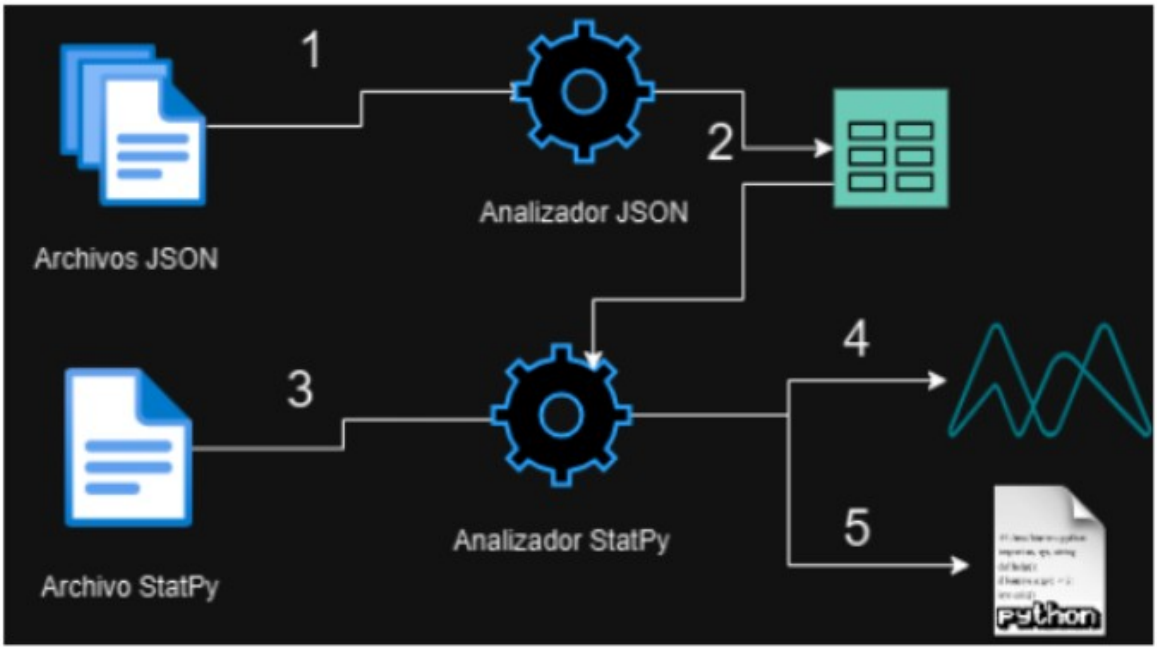
- Oracle Linux 5.5+[1]
- Oracle Linux 6.x (32 bits), 6.x (64 bits)[2]
- Oracle Linux 7.x (64 bits)[2] (8u20 y superiores)
- Red Hat Enterprise Linux 5.5+[1] 6.x (32 bits), 6.x (64 bits)[2]
- Red Hat Enterprise Linux 7.x (64 bits)[2] (8u20 y superiores)
- Suse Linux Enterprise Server 10 SP2+, 11.x
- Suse Linux Enterprise Server 12.x (64 bits)[2] (8u31 y superiores)
- Ubuntu Linux 12.04 LTS, 13.x
- Ubuntu Linux 14.x (8u25 y superiores)
- Ubuntu Linux 15.04 (8u45 y superiores)
- Ubuntu Linux 15.10 (8u65 y superiores)
- Exploradores: Firefox

Requisitos del sistema Solaris

Consulte las configuraciones del sistema Java 8 soportadas para obtener información sobre las plataformas, sistemas operativos, administradores de escritorio y exploradores compatibles.

Funciones del Sistema

Flujo del Sistema



El sistema cuenta con dos analizadores, uno para la entrada de un archivo JSON que define algunas de las variables para la generacion de graficas y otro analizador para la entrada el archivo .sp que esta escrito en el lenguaje Statpy

Al momento de leer el archivo Statpy se traducira automaticamente a lenguaje Python

Si se selecciona la opción "JSON" en el JComboBox, primero borra cualquier dato existente en una estructura de datos llamada "JsonData" dentro de un parser JSON ("Parser_json.Parser.JsonData.clear()"). Luego, llama a un analizador JSON llamado "Json_Analizer" pasando el contenido del área de texto "textArea2" y el nombre del archivo "FileName". Después, crea una cadena de texto que resume las variables y sus valores analizados y la establece en "textArea1" para su visualización.

Si se selecciona la opción "Statpy" en el JComboBox, llama a un analizador llamado "Statpy_Analizer" pasando el contenido del área de texto "textArea2". Luego, establece el resultado del análisis (contenido generado por el analizador) en "textArea1" para su visualización.

Statpy_Analizer()

```
public static void Statpy_Analizer (String entrada){
        try {
            tmplist_ejex.clear();
            tmplist_values.clear();
            Parser.Statpy_Result = "";
            Parser_statpy.Lexer lexer = new Parser_statpy.Lexer(new StringReader(entrada));
            Parser_statpy.Parser parser = new Parser_statpy.Parser(lexer);
            parser.parse();

            Parser.Statpy_Result = Tabulaciones();

        } catch (Exception e) {
            System.out.println("Error fatal en compilación de entrada.");
            System.out.println(e);
        }
    }
```

Inicialización de Listas y Variables: La función comienza limpiando dos listas, tmplist_ejex y tmplist_values, que probablemente se utilizan para almacenar datos temporalmente. Luego, se inicializa la variable Parser.Statpy_Result como una cadena vacía. Esta variable probablemente se utiliza para almacenar el resultado del análisis en el formato final.

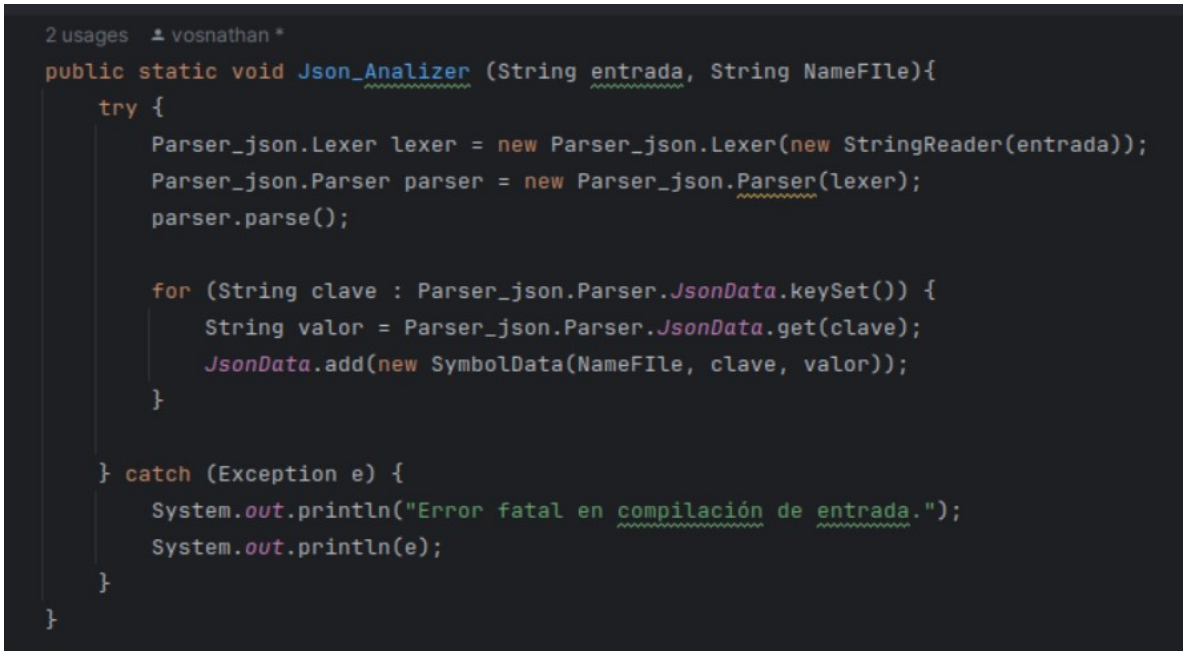
Creación de Lexer y Parser: Luego, se crea una instancia del lexer (Parser_statpy.Lexer) y del parser (Parser_statpy.Parser) necesarios para analizar la entrada. El lexer se encarga de dividir la entrada en tokens, mientras que el parser realiza el análisis sintáctico y semántico del código.

Análisis de la Entrada: La función llama al método parse() del parser, pasando la entrada proporcionada como argumento. Esto inicia el proceso de análisis de la entrada de "Statpy". Durante este proceso, el parser verifica la sintaxis y la semántica del código, identificando estructuras, expresiones y declaraciones válidas.

Generación del Resultado: Después de completar el análisis, la función llama a un método llamado Tabulaciones(). Esto sugiere que la función también se encarga de agregar tabulaciones o formatear el resultado del análisis de manera legible.

Manejo de Errores: La función está rodeada por un bloque try-catch para manejar posibles excepciones. Si ocurre algún error durante la compilación o el análisis de la entrada, se captura la excepción y se muestra un mensaje de error en la consola.

Json_Analizer()



```
2 usages  vosnathan *
public static void Json_Analizer (String entrada, String NameFIle){
    try {
        Parser_json.Lexer lexer = new Parser_json.Lexer(new StringReader(entrada));
        Parser_json.Parser parser = new Parser_json.Parser(lexer);
        parser.parse();

        for (String clave : Parser_json.Parser.JsonData.keySet()) {
            String valor = Parser_json.Parser.JsonData.get(clave);
            JsonData.add(new SymbolData(NameFIle, clave, valor));
        }

    } catch (Exception e) {
        System.out.println("Error fatal en compilación de entrada.");
        System.out.println(e);
    }
}
```

Parámetros de Entrada: La función toma dos parámetros como entrada:
entrada: Una cadena que representa el contenido en formato JSON que se va a analizar.
NameFIle: Una cadena que probablemente contiene el nombre del archivo o fuente de datos asociado a la entrada.
Creación de Lexer y Parser: Dentro de la función, se crea una instancia del lexer (Parser_json.Lexer) y del parser (Parser_json.Parser) necesarios para analizar la entrada JSON. El lexer se encarga de dividir la cadena de entrada en tokens, mientras que el parser realiza el análisis sintáctico y estructural del JSON.

Análisis de la Entrada JSON: Luego, la función llama al método parse() del parser, pasando la cadena entrada como argumento. Esto inicia el proceso de análisis de la entrada JSON. Durante este proceso, el parser verifica la estructura y sintaxis del JSON.

Extracción de Claves y Valores: Después de completar el análisis, la función recorre el mapa de datos (Parser_json.Parser.JsonData) que probablemente se utiliza para almacenar las claves y valores extraídos del JSON. Por cada par clave-valor en el mapa, crea una instancia de SymbolData y la agrega a una lista llamada JsonData. Esto significa que se está transformando la información JSON en objetos que pueden ser utilizados o procesados posteriormente.

Manejo de Errores: La función está protegida por un bloque try-catch para manejar posibles excepciones. Si ocurre algún error durante la compilación o el análisis del JSON, se captura la excepción y se muestra un mensaje de error en la consola.

Gramática utilizada

<inicio> ::= RVOID RMAIN PARENTESIS_O PARENTESIS_C CURLY_O <lista_instr> CURLY_C

<lista_instr> ::= <lista_instr> <instruccion>
| <instruccion>

<instruccion> ::= <definirglobales>
| <graficabarras>
| <graficapie>
| <print>
| <assignedvalues>
| <if>
| <switch_>
| <for>
| <while>
| <dowhile>
| BREAK SEMICOLON

<print> ::= RCONSOLE DOT RWRITE PARENTESIS_O <expresion> PARENTESIS_C SEMICOLON

<assignedvalues> ::= <type> VARIABLE EQUALS <expresion> SEMICOLON
| <type> VARIABLE EQUALS <expresion> SEMICOLON
| VARIABLE EQUALS <expresion> SEMICOLON

<switch_> ::= RSWITCH PARENTESIS_O VARIABLE PARENTESIS_C CURLY_O <cases> RDEFAULT COLON <lista_instr> CURLY_C

<cases> ::= <cases> <case>
| <case>

<case> ::= RCASE ENTERO COLON <lista_instr>

<if> ::= RIF PARENTESIS_O <expresion> PARENTESIS_C CURLY_O <lista_instr> CURLY_C
| RIF PARENTESIS_O <expresion> PARENTESIS_C CURLY_O <lista_instr> CURLY_C RELSE CURLY_O <lista_instr> CURLY_C
| RIF PARENTESIS_O <expresion> PARENTESIS_C CURLY_O <lista_instr> CURLY_C <elif_list>
| RIF PARENTESIS_O <expresion> PARENTESIS_C CURLY_O <lista_instr> CURLY_C <elif_list> RELSE CURLY_O <lista_instr> CURLY_C

<elif_list> ::= <elif_list> <_elif>
        | <_elif>

<_elif> ::= RELIF PARENTESIS_O <expresion> PARENTESIS_C CURLY_O <lista_instr>
      CURLY_C

<for> ::= RFOR PARENTESIS_O RINT VARIABLE EQUALS ENTERO SEMICOLON
            VARIABLE MINOR ENTERO SEMICOLON VARIABLE PLUS PLUS
            PARENTESIS_C CURLY_O <lista_instr> CURLY_C
   | RFOR PARENTESIS_O RINT VARIABLE EQUALS ENTERO SEMICOLON VARIABLE
      MINOREQUAL ENTERO SEMICOLON VARIABLE PLUS PLUS PARENTESIS_C
      CURLY_O <lista_instr> CURLY_C

<while> ::= RWHILE PARENTESIS_O <expresion> PARENTESIS_C CURLY_O
      <lista_instr> CURLY_C

<dowhile> ::= <assignedvalues> RDOWHILE CURLY_O <lista_instr> CURLY_C
      RWHILE PARENTESIS_O <expresion> PARENTESIS_C SEMICOLON

<type> ::= RSTRING
     | RINT
     | RDOUBLE
     | RBOOL
     | RCHAR

<expresion> ::= CADENA
        | DECIMAL
        | ENTERO
        | VARIABLE
        | TRUE
        | FALSE
        | NOT <expresion>
        | <expresion> PLUS <expresion>
        | <expresion> LESS <expresion>
        | <expresion> BY <expresion>
        | <expresion> DIVIDED <expresion>
        | <expresion> MINOR <expresion>
        | <expresion> GREATER <expresion>
        | <expresion> MINOREQUAL <expresion>
        | <expresion> GREATEREQUAL <expresion>
        | <expresion> EQUALEQUAL <expresion>
        | <expresion> DIFERENT <expresion>

```
        |  <expresion> RAND <expresion>
        |  <expresion> ROR <expresion>


<definirglobales> ::= RVOID RGLOBALES PARENTESIS_O PARENTESIS_C CURLY_O
                        <var_list> CURLY_C

<var_list> ::= <var_list> <var>
       |  <var>

<var> ::= RSTRING VARIABLE EQUALS CADENA SEMICOLON
   |  RDOUBLE VARIABLE EQUALS DECIMAL SEMICOLON
   |  RSTRING VARIABLE EQUALS DOLLAR CURLY_O RNEWVAL COMMA CADENA
      COMMA CADENA CURLY_C SEMICOLON
   |  RDOUBLE VARIABLE EQUALS DOLLAR CURLY_O RNEWVAL COMMA CADENA
      COMMA CADENA CURLY_C SEMICOLON
;

<graficapie> ::= RVOID RPIE PARENTESIS_O PARENTESIS_C CURLY_O <pie_list>
                 CURLY_C

<pie_list> ::= <pie_list> <pie>
       |  <pie>

<pie> ::= RSTRING RTITULO EQUALS <_string> SEMICOLON
  |  RSTRING BRACKET_O BRACKET_C REJEX EQUALS CURLY_O <array_ejex>
      CURLY_C SEMICOLON
  |  RDOUBLE BRACKET_O BRACKET_C RVALORES EQUALS CURLY_O <array_val>
      CURLY_C SEMICOLON
;

<graficabarras> ::= RVOID RBARRAS PARENTESIS_O PARENTESIS_C CURLY_O
                    <barras_list> CURLY_C


<barras_list> ::= <barras_list> <barras>
        | <barras>

<barras> ::=  RSTRING RTITULO EQUALS <_string> SEMICOLON
     |  RSTRING BRACKET_O BRACKET_C REJEX EQUALS CURLY_O <array_ejex>
      CURLY_C SEMICOLON
     |  RDOUBLE BRACKET_O BRACKET_C RVALORES EQUALS CURLY_O <array_val>
      CURLY_C SEMICOLON
```

```
    |  RSTRING RTITULOX EQUALS <_string> SEMICOLON
    |  RSTRING RTITULOY EQUALS <_string> SEMICOLON
;

<_string> ::=  CADENA
    |  VARIABLE
    |  DOLLAR CURLY_O RNEWVAL COMMA CADENA COMMA CADENA CURLY_C
;

<array_ejex> ::=  <array_ejex> COMMA <_ejex>
    |     <_ejex>
;

<_ejex> ::=  CADENA
    |  VARIABLE

    |  DOLLAR CURLY_O RNEWVAL COMMA CADENA COMMA CADENA CURLY_C

;

<array_val> ::=  <array_val> COMMA <_val>
       |  <_val>
;

<_val> ::=   DECIMAL
    |  VARIABLE

     |  DOLLAR CURLY_O RNEWVAL COMMA CADENA COMMA CADENA CURLY_C
```