Universidad San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y sistemas

Organización de Lenguajes y Compiladores

Manual Técnico

Luciano Isaac Xiquín Ajpop

Carnet: 201800632

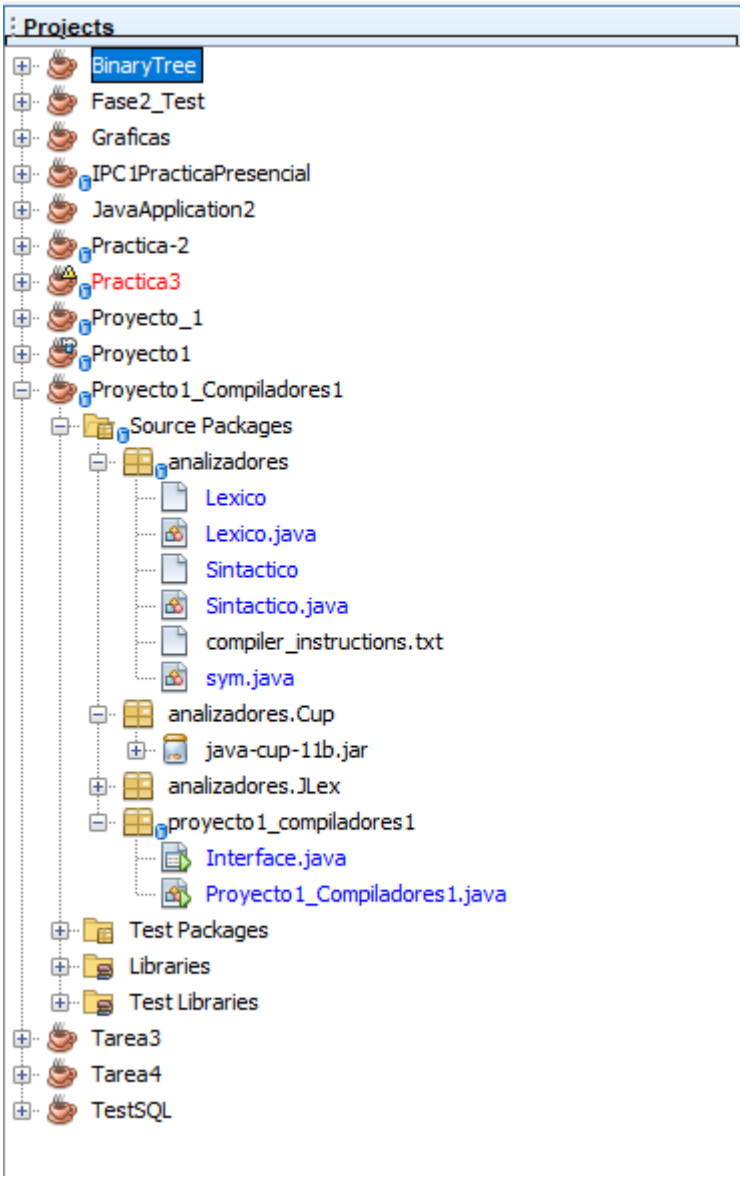
25 de agosto de 2020, Santa Cruz del Quiché, Quiche.

Desarrollo

El siguiente programa fue desarrollado y testeado con las siguientes herramientas y software:

- Lenguaje de Programación Java
- NetBeans 12.4
- Java Cup y JFlex
- Sistema Operativo: Windows 10
- Procesador Intel i-7500U

Estructura del Programa



Lógica del Programa

El programa esta diseñado a partir de los archivos sin extensión que tiene sintaxis de CUP y JLEX para su compilado con el archivo Compiler. El primero de los archivos Léxico contiene las reglas léxicas y expresiones regulares para ser utilizadas luego.

```
L = [a-zA-Z]+
D = [0-9]+
DD = [0-9]+("."[ |0-9]+)?
ASSIGN = ("-"[">"])|"="

LineJump = \n
WhiteSpace = {LineJump} | [\t\f]+

String_ =  {L}|{D}|{WhiteSpace}({L}|{D}|{WhiteSpace})*
Character = [a-zA-Z]|[0-9]|"$""{"({D})+"}"

var_name = "_"{L}({L}|{D})*"_"

OpenCase = "¿"|"¡"
CloseCase = "?"|"!"
```

Luego el archivo tiene reglas léxicas para enviarlas como símbolos al archivo sintáctico a continuación. La mayoría de los símbolos que aquí se encuentran serán utilizados en el análisis sintáctico.

```
"Numero" { return new Symbol(sym.INTVALUE,yyline,yychar,yytext());}
"Cadena" { return new Symbol(sym.STRING,yyline,yychar,yytext());}

"Boolean" { return new Symbol(sym.BOOLEAN,yyline,yychar,yytext());}
"Booleana" { return new Symbol(sym.BOOLEAN,yyline,yychar,yytext());}

"falso" { return new Symbol(sym.FALSE,yyline,yychar,yytext());}
"verdadero" { return new Symbol(sym.TRUE,yyline,yychar,yytext()); }
"not" { return new Symbol(sym.INVERSE,yyline,yychar,yytext());}

"Caracter" { return new Symbol(sym.CHAR,yyline,yychar,yytext());}

"+" { return new Symbol(sym.SUMA,yyline,yychar,yytext()); }
"-" { return new Symbol(sym.RESTA,yyline,yychar,yytext()); }
"*" { return new Symbol(sym.MULT,yyline,yychar,yytext()); }
"/" { return new Symbol(sym.DIVD,yyline,yychar,yytext()); }
"potencia" { return new Symbol(sym.POTENCIA,yyline,yychar,yytext()); }
"mod" { return new Symbol(sym.MODULO,yyline,yychar,yytext()); }
"(" { return new Symbol(sym.PARENABRE,yyline,yychar,yytext()); }
")" { return new Symbol(sym.PARENCIERRA,yyline,yychar,yytext()); }
"<" { return new Symbol(sym.MENORQUE,yyline,yychar,yytext()); }
```

El archivo Sintáctico contiene las funciones de errores que serán útiles para encontrar errores a la hora de la ejecución de nuestro programa. La LinkedList es para añadir los tokens en orden para la traducción a Golang y Python

```
package analizadores;

import java_cup.runtime.*;
import java.util.LinkedList;

parser code {:
    public LinkedList<String> tokens = new LinkedList<String>();

    public void syntax_error(Symbol s){
        System.out.println("Error linea: " + s.left + " columna: " +s.right+" Value:   " + s.value);
    }
    public void unrecovered_syntax_error(Symbol s) throws java.lang.Exception{
        System.out.println("Error linea irrecuperable: " + s.left + "columna: " +s.right+"value:" + s.value);
    }
:}
```

La siguiente lista describe la lista de terminales y no terminales, como se puede observar la lista de terminales es la lista de tokens del archivo Léxico que hemos definido con anterioridad.

```
terminal String NUMERO, DECIMAL, INTVALUE, STRING, BOOLEAN, CHAR, UMENOS;
terminal String SUMA, RESTA, MULT, DIVD, POTENCIA, MODULO, PARENABRE, PARENCIERRA;
terminal String COMA, PUNTOCOMA, CORCABRE, CORCCIERRA, INICIO, FIN, ASSIGN;
terminal String IDENTIFICADOR, INGRESAR, COMO, CON_VALOR, IF, SO, ELIF, ELSE, IFCLOSE, SWITCH, DO, SWITCHCLOSE;
terminal String FOR, UNTIL, FORCLOSE, INCREMENT, WHILE, WHILECLOSE, REPEAT, UNTIL_CONDITION, RETURN;
terminal String METOD, METODCLOSE, FUN, FUNCLOSE, PARAMETERS, EXECUTE, PRINT, PRINTLN;
terminal String TRUE, FALSE, STRING_CONTENT, CHAR_CONTENT, MENORQUE, MAYORQUE, MENORIGUAL, MAYORIGUAL, ESIGUAL, NOESIGUAL, INVERSE;
terminal String OPENCASE, CLOSECASE;

non terminal ini;
non terminal instrucciones;
non terminal instruccion;
non terminal printables;
non terminal assignables;
non terminal id;
non terminal only_type;
non terminal logic;
non terminal operators;
non terminal elif_statements;
non terminal else_statements;
non terminal bool_test;
non terminal s_cases;
non terminal args;
non terminal pars;
non terminal D_Tipo;
non terminal Double math;
```

Tenemos las reglas gramaticales para el análisis sintáctico de nuestro archivo

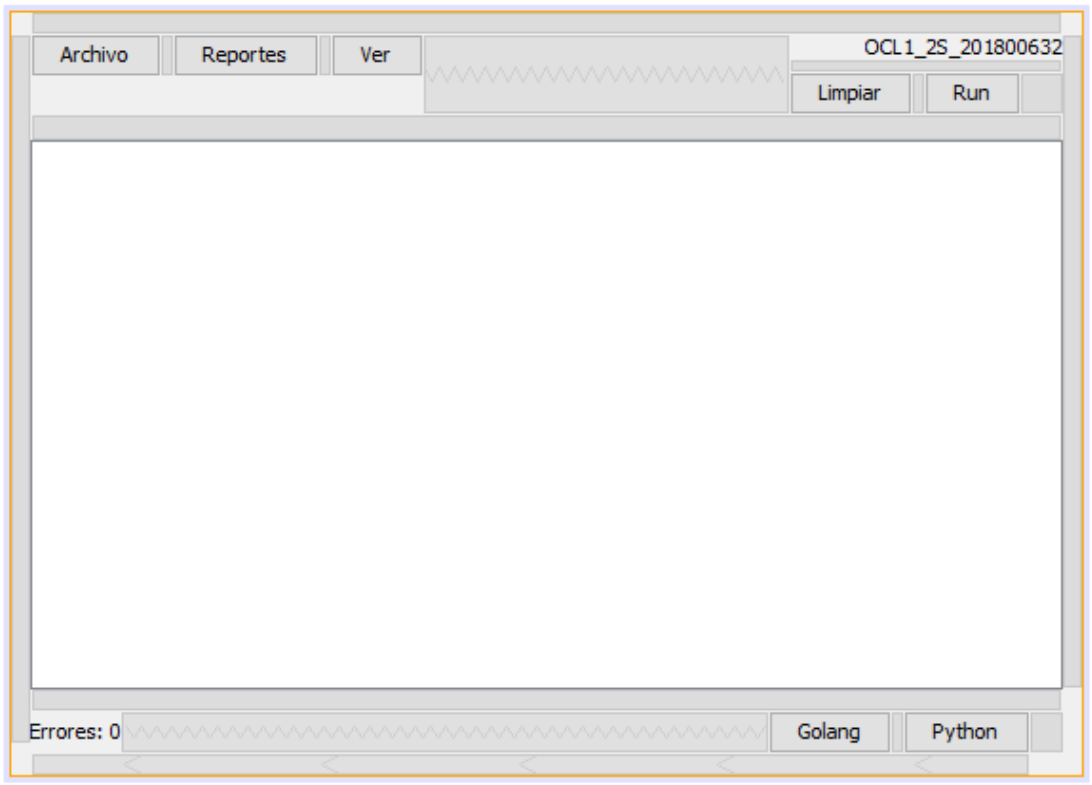
```
precedence left SUMA, RESTA, POTENCIA, MODULO;
precedence left MULT,DIVD;
precedence left MENORQUE, MAYORQUE, MENORQUE, MAYORQUE, MENORIGUAL, MAYORIGUAL, ESIGUAL, NOESIGUAL, PARENCIERRA, CLOSECASE, CORCCIERRA, error;
precedence left FIN, IFCLOSE, SWITCHCLOSE, FORCLOSE, WHILECLOSE, METODCLOSE, FUNCLOSE;
precedence left PUNTOCOMA;
precedence right UMENOS, INICIO, INGRESAR, IF, SWITCH, FOR, WHILE, REPEAT, METOD, FUN, EXECUTE, PRINT, PRINTLN, CON_VALOR, UNTIL;
precedence right PARENABRE, IDENTIFICADOR, OPENCASE, CORCABRE, COMA, RETURN, UNTIL_CONDITION;

start with ini;

ini ::= INICIO instrucciones FIN
    | INICIO FIN
;

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

Entonces esto se conecta desde la instancia de interface para la interacción directa con el analizador sintáctico y léxico.



Link del repositorio: https://github.com/guy-devoir/OLC1-201800632