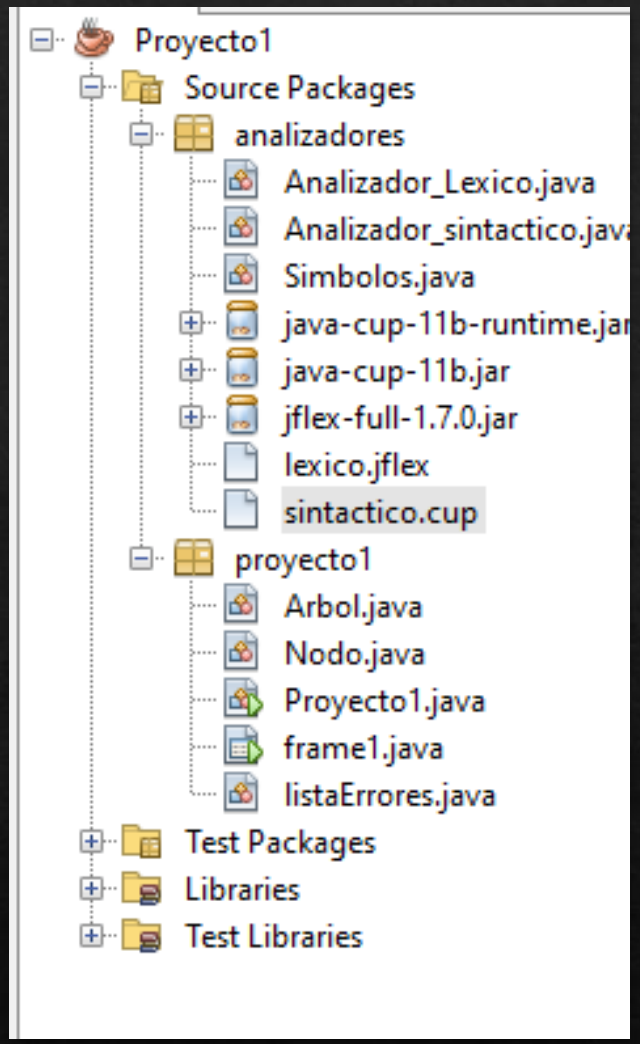
Manual Técnico

Proyecto 1

CLASES DEL PROGRAMA

Se utilizaron 2 paquetes, uno para los analizadores y otro para el resto del programa, cada uno con sus clases respectivas para mantener el orden y la funcionalidad del proyecto al máximo posible



```
"+"  {
                System.out.println("Reconocio token:<SUMA> lexema:"+yytext());
                return new Symbol(Simbolos.sum, yycolumn, yyline, yytext());
                }
"-"  {
                System.out.println("Reconocio token:<RESTA> lexema:"+yytext());
                return new Symbol(Simbolos.res, yycolumn, yyline, yytext());
                }
"*"  {
                System.out.println("Reconocio token:<MULT> lexema:"+yytext());
                return new Symbol(Simbolos.mult, yycolumn, yyline, yytext());
                }
"/"  {
                System.out.println("Reconocio token:<DIV> lexema:"+yytext());
                return new Symbol(Simbolos.div, yycolumn, yyline, yytext());
                }
"potencia"  {
                System.out.println("Reconocio token:<POT> lexema:"+yytext());
                return new Symbol(Simbolos.pot, yycolumn, yyline, yytext());
                }
"modulo"  {
                System.out.println("Reconocio token:<MOD> lexema:"+yytext());
                return new Symbol(Simbolos.mod, yycolumn, yyline, yytext());
                }
"("  {
                System.out.println("Reconocio token:<CARACTER> lexema:"+yytext());
                return new Symbol(Simbolos.abrir_par, yycolumn, yyline, yytext());
                }
")"  {
```

ARCHIVO FLEX

- Se utilizó la herramienta jflex para el análisis léxico, en esta etapa se reconocen todos los tokens que se volverán nuestros Terminales en el Sintactico

- Esta etapa acperta coincidencias de strings y expresiones regulares

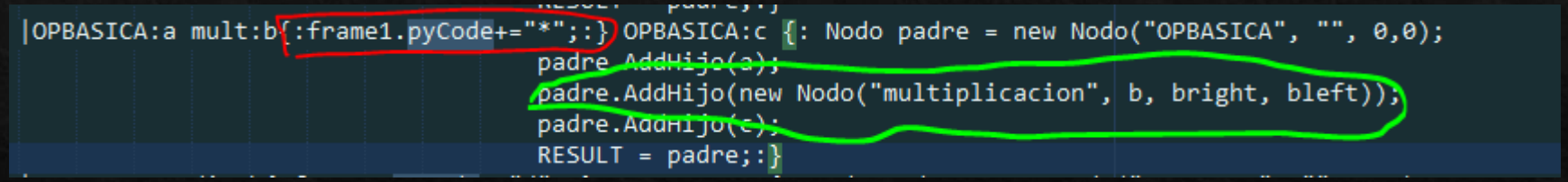
```
//definicion de terminales
terminal String sum,res,mult,div,pot,mod,abrir_par,cerrar_par,abrir_cor,cerrar_cor,igual,coma,mayor,menor,mayor_igual,
menor_igual,es_igual,es_dif,or,and,not,inicio,fin,ingresar,como,con_val,asignar,
si,entonces,fin_si,elif,els,segun,hacer,fin_segun,para,hasta,incremento,fin_para,
mientras,fin_mientras,repetir,hasta_que,retornar,metodo,param,fin_metodo,
func,fin_func,ejecutar,imprimir,imprimir_nl,numero,booleano,caracter,variable,cadena,
comentario_ln,puntoycoma,abrir_inte,cerrar_inte,tipo_cadena,tipo_caracter,tipo_booleano,tipo_numero;

//definicion de no terminales
non terminal Nodo INICIO, E, TODO, OPBASICA, RELACION, DECLARACION, TIPODATO, LISTAVAR, ASIGNACION, CONDICIONALSI, SINO,
SELECCION, SELECTRECURSIVO, CICLOPARA, CICLOMIENTRAS, CICLOREPETIR, METODO, LISTAPARAM, FUNCION, RETORNO, EJECUTAR, LISTAPARAM2,
IMPRESION;
```

ARCHIVO CUP

- Se utilizó la herramienta cup para el análisis sintactico, en esta etapa se reciben tokens como T y NT, se le da un sentido y orden a la gramatica

Gramática CUP

- Se ingresa la gramática y a su vez se mandan valores al árbol para realizar el AST y la traducción hacia los lenguajes, utilizando nodos hijos y padres para dar forma y jerarquía al arbol



Precedencia de operadores

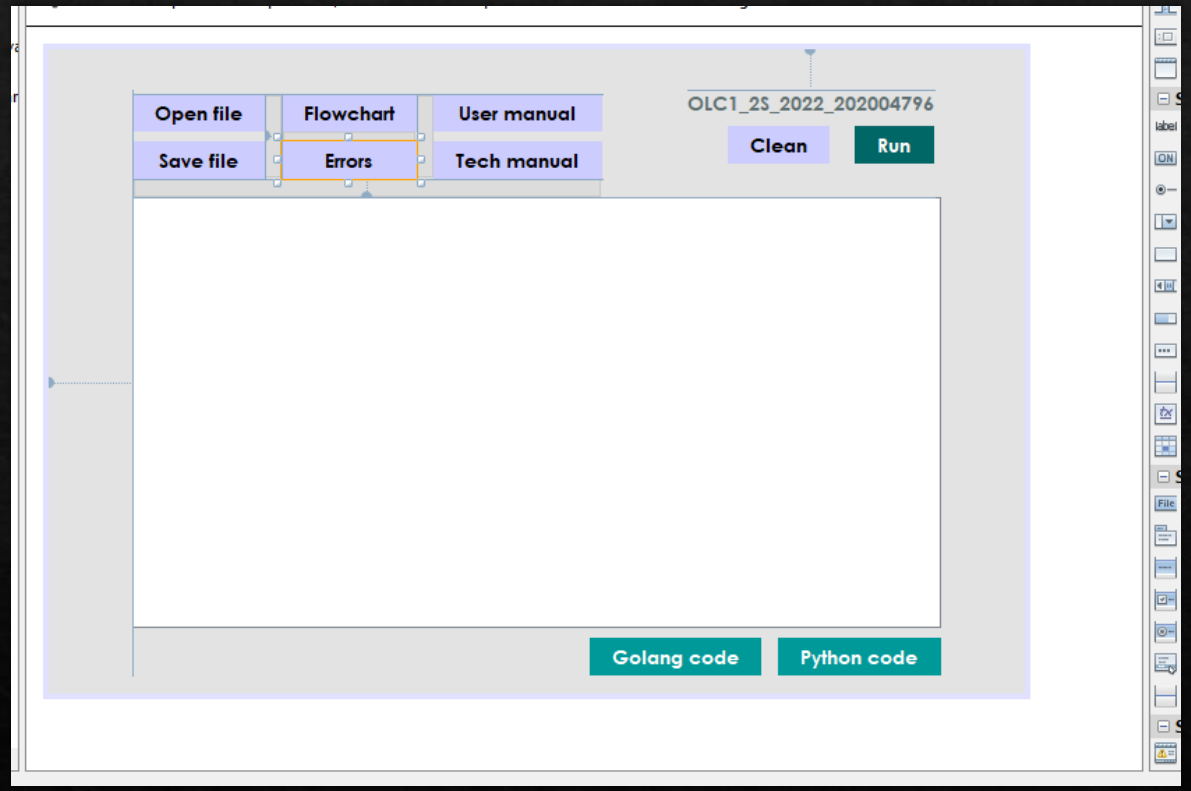
```
precedence left coma;
precedence left sum, res;
precedence left mult, div;
precedence left pot, mod;
precedence left and, or;
precedence left puntoycoma;
precedence right not;
precedence left abrir_inte, cerrar_inte;
precedence left abrir_par, cerrar_par;
precedence left abrir_cor, cerrar_cor;
precedence left si;
```

Clase arbol

Clase árbol para realizar el AST con los símbolos obtenidos durante el análisis sintactico del archivo de entrada

```
public void GraficarSintactico(){
    String grafica = "Digraph Arbol_Sintactico{\n\n" + GraficaNodos(this.raiz, "0") + "\n\n}";
    GenerarDot(grafica);

}

private String GraficaNodos(Nodo nodo, String i){
    int k=0;
    String r = "";
    String nodoTerm = nodo.token;
    nodoTerm = nodoTerm.replace("\"", "");
    r= "node" + i + "[label = \"" + nodoTerm + "\"];\n";

    for(int j =0 ; j<=nodo.hijos.size()-1; j++){
        r = r + "node" + i + " -> node" + i + k + "\n";
        r= r + GraficaNodos(nodo.hijos.get(j), ""+i+k);
        k++;
    }

    if( !(nodo.lexema.equals("")) ){
        String nodoToken = nodo.lexema;
        nodoToken = nodoToken.replace("\"", "");
        r += "node" + i + "c[label = \"" + nodoToken + "\"];\n";
        r += "node" + i + " -> node" + i + "c\n";
    }

    return r;
}
```

Frame de entrada de datos



Clase lista de errores

Para reconocer, enlistar y mostrar los datos en una salida html junto con un css para una presentación mas estetica

```
 */
public class listaErrores {
    String lexema;
    int linea;
    int columna;
    String tipo;

    public listaErrores(String lexema, int linea, int columna, String tipo){

        this.lexema=lexema;
        this.linea=linea;
        this.columna=columna;
        this.tipo=tipo;
    }


    public String getLexema() {
        return lexema;
    }

    public int getLinea() {
        return linea;
    }

    public int getColumna() {
        return columna;
    }
```

```css
body{
    background-color: #fdc8c3;
    font-family: Arial;
}

#main-container{
    margin: 150px auto;
    width: 600px;
}

table{
    background-color: white;
    text-align: left;
    border-collapse: collapse;
    width: 100%;
}

th, td{
    padding: 20px;
}

thead{
    background-color: #E63B2A;
    border-bottom: solid 5px #750a00;
    color: white;
}

tr:nth-child(even){
    background-color: #dac4c4;
}
```

Lista de Errores

Creado por: Gerhard Benjamin Ardon Valdez 202004796

Tipo	Lexema	Linea
lexico	@	1
sintactico	con_valor	4
lexico	_	5
lexico	p	5