



Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en ciencias de sistemas.

Manual de técnico

Nombre del estudiante

José Luis Espinoza Jolón - 202202182

Ing.(a). Kevin Lajpop

Guatemala, marzo 10 de 2024

Introducción

• Descripción del programa	3
• Lenguaje para utilizar	3
• Librería a utilizadas.....	3
▪ JFlex :	3
Donde se descargar	3
▪ Cup	4
Donde se descarga:.....	4
▪ JFreeChart:	4
Donde se descarga:.....	5
• Estructura del proyecto	6
❖ Paquete de analizador:	6
Compilador:.....	6
LexerCup:	7
Sintax:.....	7
❖ Paquete de clases:.....	8
nodoArbol:	8
Árbol:.....	8
▪ Run:.....	9
▪ Graficar:.....	9
▪ FuncionEstaditicos:.....	10
❖ Paquete de Listas:	10
.....	11
❖ Paquete de Main:	11

Descripción del programa

El curso de Organización de Lenguajes y Compiladores 1, perteneciente a la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, requiere de usted, como conocedor de la construcción de analizadores Léxico y Sintáctico, crear un sistema que sea capaz de realizar operaciones aritméticas y estadísticas, además de poder generar diversos gráficos a partir de una colección de datos.

Lenguaje para utilizar










Para este tipo de proyecto utilizamos Java.

Librería a utilizadas

- **JFlex** : es un generador de analizadores léxicos escrito en Java. Se utiliza para generar analizadores léxicos (también conocidos como escáneres) que identifican y dividen el texto de entrada en tokens para su procesamiento posterior por un analizador sintáctico.

Donde se descargar: Link para descargar la librería

<https://github.com/jflex-de/jflex/releases/tag/v1.9.1>

▼ Assets 9		
 jflex-1.9.1.tar.gz	4.72 MB	Mar 10, 2023
 jflex-1.9.1.tar.gz.asc	833 Bytes	Mar 10, 2023
 jflex-1.9.1.tar.gz.sha1	61 Bytes	Mar 10, 2023
 jflex-1.9.1.zip	4.84 MB	Mar 10, 2023
 jflex-1.9.1.zip.asc	833 Bytes	Mar 10, 2023
 jflex-1.9.1.zip.sha1	58 Bytes	Mar 10, 2023
 manual.pdf	454 KB	Mar 10, 2023
 Source code (zip)		Mar 10, 2023
 Source code (tar.gz)		Mar 10, 2023

Debes de comprimir el archivo y agregarlo en el proyecto

- **Cup:** es un parser-generator. Es un analizador sintáctico que construye un parser para gramáticas tipo LALR(1), con código de producción y asociación de fragmentos de código JAVA.

Donde se descarga:

link de la pagina <https://www2.cs.tum.edu/projects/cup/index.php>


CUP stands for **Construction of Useful Parsers** and is an **LALR parser generator** for Java. It was developed by **C. Scott Ananian**, Frank Flannery, Dan Wang, **Andrew W. Appel** and **Michael Petter**. It implements **standard LALR(1) parser generation**. As a parser author, you specify the symbols of Your grammar (**terminal** T1,T2; **non terminal** N1, N2;), as well as the productions (**LHS** ::= **RHS1** | **RHS2** ;). If you provide each production alternative with action code ({ : **RESULT** = myfunction(); ; }), the parser will call this action code after performing a reduction with the particular production. You can use these callbacks to assemble an AST (Abstract Syntax Tree) or for arbitrary purposes. You should also have a look at the **scanner generator JFlex**, which is suited particularly well for collaboration with CUP.

⚙ Main features

- > LALR(1) parsing engine with **symbol precedences**
- > **Error recovery**
- > **Syntax completion prerequisites**
- > Optional automatic **parse tree generation**
- > Optional **XML output**
- > **Manual action code**
- > **Grammar Engineering via Eclipse Plugin**
- > **An open licence**

📄 Download

select the version of CUP, you would like to obtain:



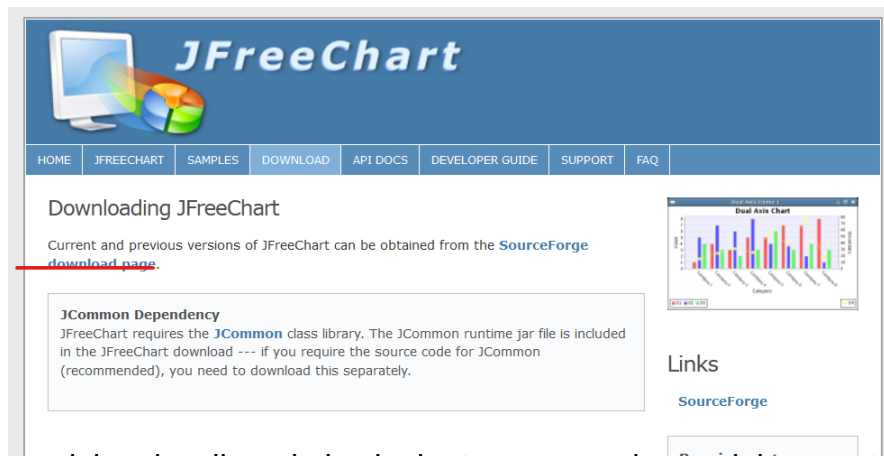
Debes de descomprimirlo y agregar solamente el java-cup-11b.jar en las librerías del proyecto.

- **JFreeChart:**

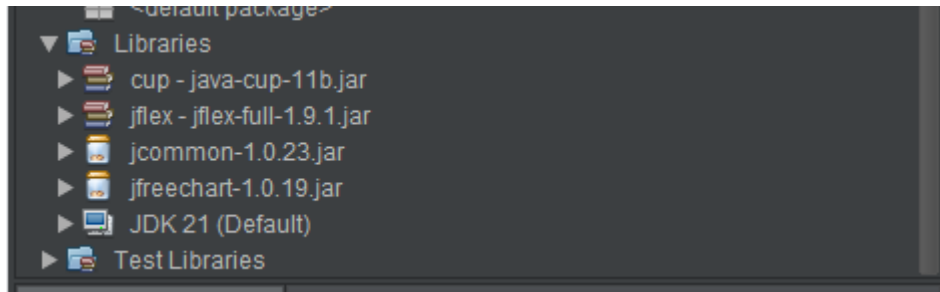
Se utiliza esa librería para realizar las graficas

Donde se descarga:

Link <https://www.jfree.org/jfreechart/download.html>

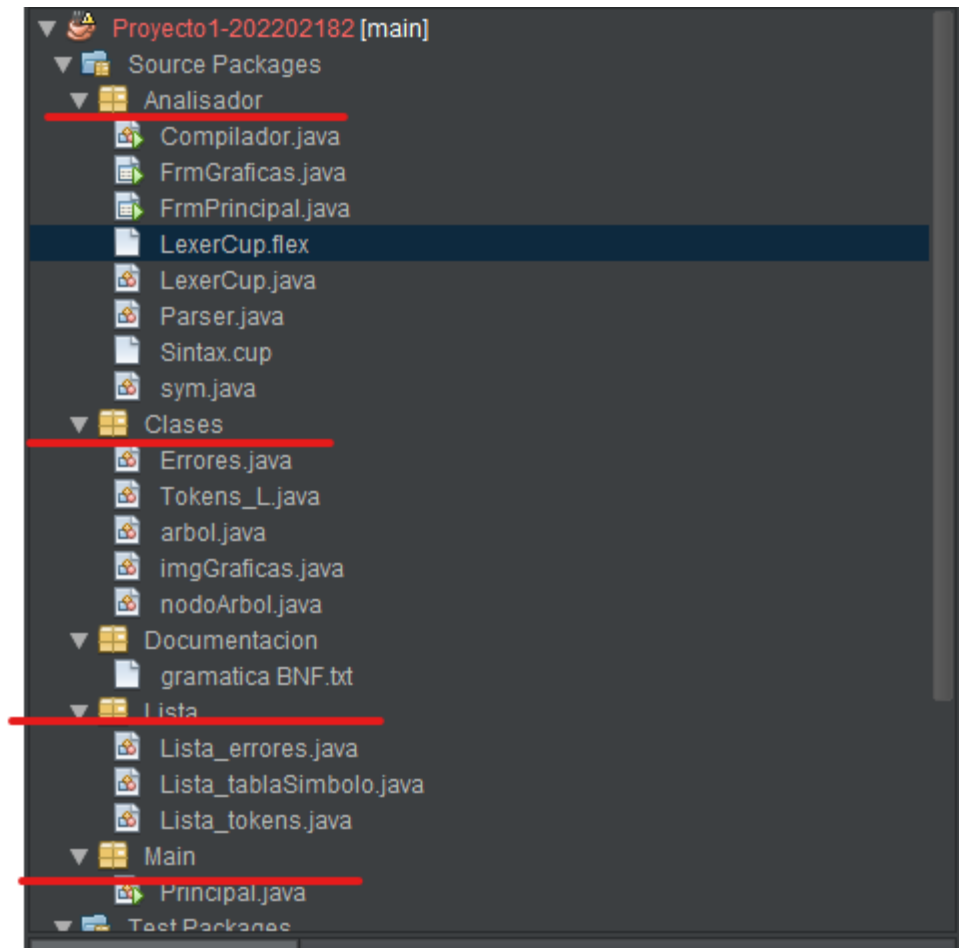


La librería se debe visualizar de la siguiente manera, dentro del proyecto.



🚦 Estructura del proyecto

La manera que organice el proyecto es de la siguiente forma:



❖ Paquete de analizador:

En esta parte realizo todo lo léxico, sintáctico y las vistas, las clases mas importante en este paquete son las siguientes:

Compilador:

Se estableció las rutas para poner manejar los documentos

```

13  /**
14   *
15   * @author Usuario
16   */
17  public class Compilador {
18
19  public static void main(String[] args){
20      String ruta = "./src/Analizador/";
21      //generadorAnalizador(ruta, rutas);
22      generadorAnalizador();
23  }
24
25  private static void generadorAnalizador(){
26
27      try {
28
29          /*File archivo = new File(ruta+"Lexer.flex");
30          JFlex.Main.generate(archivo);
31          java_cup.Main.main(rutas);*/
32
33          String ruta = "src/Analizador/";
34          String[] opcionesJFlex = {ruta+"LexerCup.flex", "-d", ruta};
35          jflex.Main.generate(opcionesJFlex);
36
37          String[] opcionesCup = {"-destin", ruta, "-parser", "Parser", ruta+"Sintax.cup"};
38          java_cup.Main.main(opcionesCup);

```

LexerCup:

Declaramos todas las palabras que usaríamos en el sintáctico, también los errores e ignorar los comentarios.

```

%%
%class LexerCup
%type java_cup.runtime.Symbol
%public
%column
%line
%char
%cup
%unicode
%ignorecase

L=[a-zA-Z_0-9@#%&'!&()*+,-./:;<=>?[\]^_`{|}~" ' \n\r\t]+
D=[0-9]+
espacio[ \t\r\n]+
ESPACIO[ ]
NUMERO = [0-9]+("."+[0-9]+)?
COMULTI = "(-|~|>)"

%%

public static List<Error> lista_E = new ArrayList<>();
public static List<Token> lista_T = new ArrayList<>();

Tokens_L tokens_L;
Errors error;
int conteo_errores = 1;
int conteo_tokens = 1;

%%

%init{
    yyline = 1;
    yycolumn = 1;
    %init;
}

%%

"program" {System.out.println("Inicio: " + yytext() +
    ", en la linea: " + yyline, "en la columna: " + yycolumn); tokens_L = new Tokens_L(conteo_tokens, yytext(), "Reservada", yyline, yycolumn);
    lista_T.add(tokens_L);
    conteo_tokens++;
    return new Symbol(sym.Program, yyline, yycolumn, yytext());
}

```

Sintax:

Se realizo las gramáticas del lenguaje y se agregó a nuestro árbol los datos más importantes, para usarlo.

```

non terminal arbol INICIO, CODIGO, COMPILACION, SENTENCIA, DECLARACION, TARREGLOS, CONTENIDO, TIPCONTENIDO, LISTA,
COMENTARIO, CADENA, ARREGLOSTIP, GRAFIC, COMENTENCIAS, ATRIBUGRAFIC;

start with INICIO;

INICIO ::= Program: p CODIGO: c End: e Program: p1
{
    arbol ini = new arbol("INICIO");
    ini.addHijo(new arbol(p.toString()));
    ini.addHijo(c);
    ini.addHijo(new arbol(e.toString()));
    ini.addHijo(new arbol(p1.toString()));
    RESULT = ini;
};

CODIGO ::= COMPILACION: c
{
    arbol cod = new arbol("CODIGO");
    cod.addHijo(c);
    RESULT = cod;
};

|
{
    arbol cod = new arbol("CODIGO");
    RESULT = cod;
};

COMPILACION ::= SENTENCIA: s
{
    arbol com = new arbol("COMPILACION");
    com.addHijo(s);
    RESULT = com;
};

```

❖ Paquete de clases:

Se establecieron algunas clases que se utilizó para guardar datos como los errores y token

```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5  package Clases;
6
7  /**
8   *
9   * @author Luis Jolon
10  */
11  public class Tokens_L {
12
13      public int conteo;
14      public String lexema;
15      public String tipo;
16      public int Linea;
17      public int Columna;
18
19
20      public Tokens_L(int conteo, String lexema, String tipo, int Linea, int Columna) {
21          this.conteo = conteo;
22          this.lexema = lexema;
23          this.tipo = tipo;
24          this.Columna = Columna;
25          this.Linea = Linea;
26      }
27  }
```

nodoArbol:

Se establecieron los atributos y el constructo para guardar los datos de mi árbol.

```
4  /*
5  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
6  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
7  */
8  package Clases;
9
10  /**
11   *
12   * @author Usuario
13   */
14  public class nodoArbol {
15
16      public String nombre;
17      public String rol;
18      public String tipo;
19      public String entorno;
20      public String pertenece_a;
21      public String valor;
22      public int conteo;
23
24      public nodoArbol(int conteo, String nombre, String rol, String tipo, String entorno, String pertenece_a, String valor) {
25          this.conteo = conteo;
26          this.nombre = nombre;
27          this.rol = rol;
28          this.tipo = tipo;
29          this.entorno = entorno;
30          this.pertenece_a = pertenece_a;
31          this.valor = valor;
32      }
33  }
```

Árbol:

Esta clase es la mas importante, fue donde se generó toda la lógica.


```

public class arbol {

    public String lex;
    public ArrayList<arbol>hijos;
    public Map<String, String> hashMap = new HashMap<>();
    public static int imageCounter = 0;
    public static int imageCounter1 = 0;
    public static int imageCounter2 = 0;
    public static int imageCounter3 = 0;
    private String result;
    public int conteo = 1;

    public arbol(String lex){
        this.lex = lex;
        this.hijos = new ArrayList();
    }

    public void addHijo(arbol hijo){
        this.hijos.add(hijo);
    }
}

```

Las funciones más importantes son las siguientes.

- **Run:**

Esta función fuimos recorriendo todo el árbol atreves de sus hijos , para ir obteniendo los datos, cada if y elf if , significar que vamos subiendo nuestro árbol, comenzamos de la producción mas baja de nuestra gramática a la mas alta.

```

public void run(arbol raiz, ArrayList<nodoArbol> Ts, JTextArea txtconsola) throws IOException{

    for(arbol hijo : raiz.hijos){
        run(hijo,Ts, txtconsola);
    }

    if(raiz.lex == "GRAFIC" && raiz.hijos.size() == 2){
        Graficar(raiz.hijos.get(0).lex,hashMap,txtconsola);
        //System.out.println(raiz.hijos.get(0).lex);
        //System.out.println(raiz.hijos.get(1).result);
        raiz.result = raiz.hijos.get(0).lex;
        raiz.result = raiz.hijos.get(1).result;
    }else if(raiz.lex == "CONSENTENCIAS" && raiz.hijos.size() == 2){
        //System.out.println("2");
        raiz.result = raiz.hijos.get(0).result + raiz.hijos.get(1);
        //System.out.println(raiz.hijos.get(0).result + raiz.hijos.get(1));
    }else if(raiz.lex == "CONSENTENCIAS" && raiz.hijos.size() == 1){
        //System.out.println("3");
        raiz.result = raiz.hijos.get(0).result;
        //System.out.println(raiz.hijos.get(0).result);
    }else if(raiz.lex == "ATRIBUGRAFIC" && raiz.hijos.size()==3){
        //obtengo todo los datos
    }
}

```

- **Graficar:**

Se realizo un switch para verificar que tipo de graficas era , para poder generar , los parámetros son tipo de grafica, hasMap se utilizo para obtener los datos dentro de mi árbol y de ultimo mi txtConsolas

```

}
public static void Graficar(String tipo, Map<String, String> hashMap, JTextArea txtconsola) throws IOException {
    System.out.println(tipo);
    switch (tipo) {
        case "graphbar":
            System.out.println("obtengo una grafica graphbar");
            System.out.println("-----");
            String valor = hashMap.get("titulo");
            String valor1 = hashMap.get("ejeX");
            String[] categorias = valor1.split(",");
            String valor2 = hashMap.get("ejeY");
            String[] valoresStr = valor2.split(",");
            double[] valores = new double[valoresStr.length];

            // Convertir elementos de valoresStr a double y guardarlos en valores[]
            for (int i = 0; i < valoresStr.length; i++) {
                valores[i] = Double.parseDouble(valoresStr[i]);
            }

            String valor3 = hashMap.get("tituloX");
            String valor4 = hashMap.get("tituloY");
    }
}

```

▪ FuncionEstaditicos:

De la misma forma con un switch se realizó, par verificar que tipo era, los parámetros fueron el tipo de función y los valores de nuestro arreglo, Ese arreglo se tuvo que parsear a double para poder trabajar.

```

public static double FuncionEstaditico(String tipo, String arreglo) {
    String[] arregloN = arreglo.split(",");
    double[] valores = new double[arregloN.length];

    // Convertir elementos de arregloN a double y guardarlos en valores[]
    for (int i = 0; i < arregloN.length; i++) {
        valores[i] = Double.parseDouble(arregloN[i]);
    }

    System.out.println(valores);

    switch (tipo) {
        case "media":
            double sumaMedia = 0;
            for (double valor : valores) {
                sumaMedia += valor;
            }
            return sumaMedia / valores.length;
        case "mediana":
            Arrays.sort(valores);
            int longitud = valores.length;
            if (longitud % 2 == 0) {

```

❖ Paquete de Listas:

La lista se utilizo para generar los reportes y hacer métodos para limpiar dichas listas

```

//
// lista.add(tokens);
//
}

public void recorrer() {
    for (int i = 0; i < lexemCap.lista_T.size(); i++) {
        Tokens_t tokens = lexemCap.lista_T.get(i);
        System.out.println("i: " + tokens.conteo + " lexem: " + tokens.lexema + " tipo: " + tokens.tipo);
    }
}

public void limpiar() {
    lexemCap.lista_T.clear();
}

public void reporte() {
    StringBuilder sb = new StringBuilder();
    sb.append("<!DOCTYPE html>");
    sb.append("<html lang='en'>");
    sb.append("<head>");
    sb.append("<meta charset='UTF-8'>");
    sb.append("<meta http-equiv='X-UA-Compatible' content='if=edge'>");
    sb.append("<meta name='viewport' content='width=device-width, initial-scale=1.0'>");
    sb.append("<title>Reporte Tokens</title>");
    sb.append("</head>");
    sb.append("<body>");
    sb.append("<div>Reporte de tokens</div>");
    sb.append("<div>");
    sb.append("<table border='1'>");
    sb.append("<thead>");
    sb.append("<tr>");
    sb.append("<th style='background-color: #f2f2f2; color: #ffffff;'>No.</th>");
    sb.append("<th style='background-color: #f2f2f2; color: #ffffff;'>lexema</th>");
    sb.append("<th style='background-color: #f2f2f2; color: #ffffff;'>tipo</th>");
    sb.append("<th style='background-color: #f2f2f2; color: #ffffff;'>linea</th>");
    sb.append("<th style='background-color: #f2f2f2; color: #ffffff;'>columna</th>");
    sb.append("</tr>");
    sb.append("</thead>");
    sb.append("</div>");
    sb.append("</body>");
    sb.append("</html>");
}

```

❖ Paquete de Main:

Solamente es donde se debe de ejecutar el programa, para que todo funcione.

```

1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4   */
5  package Main;
6
7  import Analisador.FrmPrincipal;
8
9
10 /**
11  *
12  * @author Usuario
13  */
14 public class Principal {
15
16     public static void main(String[] args) {
17         FrmPrincipal vista = new FrmPrincipal();
18         vista.setVisible(true);
19     }
20
21 }

```