202111718

Manual Técnico

Proyecto 1 Compiladores 1

Walter Javier Santizo Mazariegos

Introducción

El proyecto "StatPy Convertor" es una solución de software que consiste en aplicar los conocimientos sobre las fases de análisis léxico y sintáctico de un compilador para ofrecer dos funcionalidades principales: generación de reportes estadísticos y traducción de código de StatPy a Python.

Debido a lo tedioso que puede llegar a ser el sobrescribir una aplicación de un lenguaje a otro, se le solicita a los estudiantes de sistemas que desarrollen un traductor de lenguaje de programación StatPy a Python. Este traductor abarca algunas de las sentencias básicas del lenguaje, como la declaración y asignación de variables, sentencias de retorno, sentencias de condición y repetición, métodos, entre otros.

Además, se solicita que el programa pueda generar reportes estadísticos, con datos que se puedan leer desde archivos JSON (objetos con datos),permitiendo una mejor comprensión de los datos procesados durante el proceso de traducción

Resumen

Un compilador es un programa informático que se encarga de transformar el código fuente de un programa escrito en un lenguaje de programación de alto nivel (como C++, Java o Python) en un código ejecutable en una máquina específica (como un ordenador o un teléfono móvil).

un compilador es un programa que traduce el código de programación escrito en un lenguaje de alto nivel que los humanos pueden entender, a un lenguaje de bajo nivel que la máquina puede entender. Esto se hace para que el código pueda ser ejecutado en la computadora de destino. El proceso de compilación involucra varias fases, como análisis léxico, análisis sintáctico, análisis semántico, generación de código y optimización de código. Cada una de estas fases se encarga de realizar una tarea específica para transformar el código de entrada en código ejecutable.

Los compiladores tienen una amplia variedad de aplicaciones en el campo de la informática. Aquí te presento algunas de las aplicaciones más comunes:

Desarrollo de software: Los compiladores son una herramienta fundamental para los desarrolladores de software, ya que les permiten escribir programas en lenguajes de programación de alto nivel, como C++, Java, Python, entre otros, y luego compilarlos para ser ejecutados en diferentes plataformas.

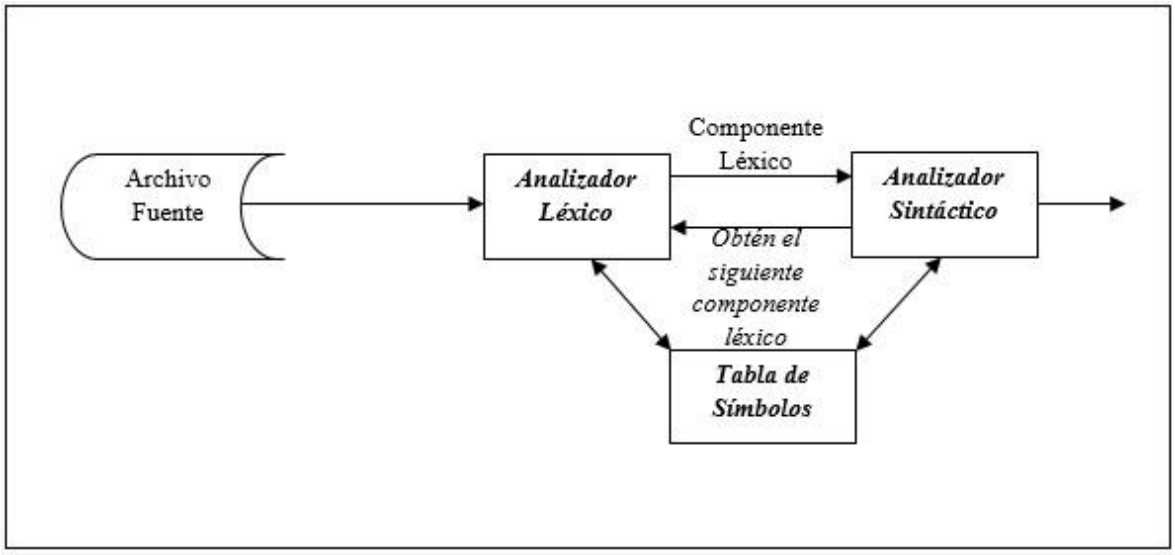
Optimización de código: Los compiladores también se utilizan para optimizar el código fuente de un programa, lo que puede mejorar su rendimiento y velocidad de ejecución.

Seguridad: Los compiladores también se utilizan para mejorar la seguridad de los programas, ya que pueden detectar vulnerabilidades en el código fuente, como errores de programación y vulnerabilidades de seguridad.

Generación de código para hardware específico: Los compiladores pueden generar código para ser ejecutado en hardware específico, como procesadores de gráficos, sistemas embebidos y dispositivos móviles.

Tecnologías

En esta etapa del proyecto seguimos el esquema de las 2 primeras fases de análisis del compilador las cuales son el análisis léxico y sintáctico y se hizo por medio de este esquema que representa la entrada de caracteres que recibe el analizador y la salida de tokens que luego analizara el analizador sintáctico siempre utilizando la tabla de símbolos en las dos fases como medida de prevención de errores y de uso posterior



Jflex 1.7

JFlex es una herramienta de generación de analizadores léxicos (scanners) escritos en Java. Un analizador léxico es una parte importante de un compilador, ya que su función es leer el código fuente y separar los distintos elementos léxicos, como palabras clave, identificadores, números y símbolos, para que el compilador pueda entenderlos y convertirlos en un programa ejecutable.

JFlex se encarga de generar un analizador léxico a partir de una especificación en un archivo de texto que describe los patrones de caracteres que deben ser reconocidos y las acciones que deben ser ejecutadas cuando se encuentran esos patrones. La especificación se realiza utilizando expresiones regulares, que permiten definir patrones de búsqueda complejos y flexibles.

```
//package e importaciones
%%
//configuraciones para el análisis
%%
//reglas lexicas
```

Cup 11b

CUP (Constructor of Useful Parsers) es una herramienta de generación de analizadores sintácticos (parsers) para Java. Un analizador sintáctico es una parte importante de un compilador, ya que su función es analizar la estructura del código fuente y verificar si cumple con la sintaxis definida en el lenguaje de programación.

CUP se encarga de generar un analizador sintáctico a partir de una especificación en un archivo de texto que describe las reglas gramaticales del lenguaje. La especificación se realiza utilizando una notación llamada "gramática libre de contexto", que permite definir la estructura del lenguaje de programación mediante la definición de reglas gramaticales.

La herramienta CUP utiliza el algoritmo de análisis sintáctico LR (left-to-right, rightmost derivation), que es capaz de procesar gramáticas más complejas y ambigüedades en la definición de la estructura del lenguaje.

```
//package e importaciones

parser code
{:
:}
//terminales
terminal String ENTERO;
//no terminales
non terminal instrucciones;
//precedencias
precedence left MAS,MENOS;
//producción de inicio
start with ini;
//producciones
ini::=instrucciones;
```

Flujo de Caracteres

Este será un flujo de caracteres estándar y convertirá en tokens mediante la siguiente tabla que reconoce ciertos patrones y mediante el análisis léxico del archivo de jflex los convertirá en tokens que luego el parser se encargara de convertir en el caso del archivo json a una tabla de variables y en cuanto al lenguaje statpy se encargara de traducir el código al lenguaje Python

Json

```
/* ejemplo de carga

   de objeto json */


{
    "Titulo":"titulo Pie",
    "variable1": 4.5,
    "Titulo2": "titulo barras",
    "variable2": 5.0

}
```

Nótese que se permiten los comentarios multilínea y de una sola línea

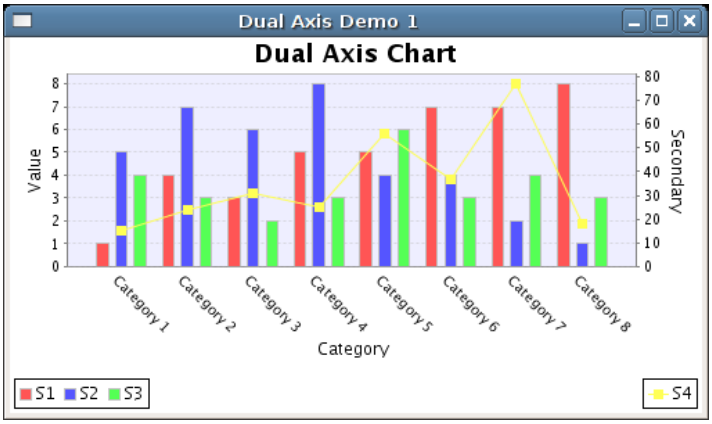
```
void main ( ){
    int b = 2;
    int a =1;
    int var1 = 5+8*9;
    if (b > a){
        Console.Write("b mayor que a");
    }else if(a == b){
        Console.Write("a y b son iguales");
    }
    void DefinirGlobales(){
        string reporte1 = "Reporte 1";
        double pe1 = 0.8;
        double pe2 = 0.5;
        double pe3 = 0.2;
        double po1 = ${ NewValor, "archivo2.json", "valor1"};
        double po2 = ${ NewValor, "archivo1.json", "valor1"};
        string vart = "Valor Obtenido";
        string var2 = "Valor Esperado clase 1";
        string var22 = "Valor Obtenido clase 1";
        string var3 = "Valore Esperado clase 2";
```

```
        string var3 = "Valor Obtenido clase 2";
    }
    void GraficaBarras(){
        string Titulo= reporte1;
        string [] Ejex= { "Probabilidad Esperada clase 1", "Probabilidad
Obtenida Clase 1", var2, var22, var3, var33};
        double [] Valores= { pe1, po1, pe2, po2, pe3, ${ NewValor,
"archivo1.json", "valor2"} };
        string TituloX= "Atributo";
        string TituloY= "Puntaje";
    }
    while(a < 10){
        Console.Write("el valor de a es: " + a);
    }

}
```

Aquí también se permiten los comentarios multilínea

JfreeChart 1.0.19

JFreeChart es una biblioteca de código abierto en Java utilizada para crear gráficos y visualizaciones en aplicaciones Java. Proporciona una amplia gama de tipos de gráficos, como gráficos de barras, gráficos circulares, gráficos de líneas, gráficos de dispersión, entre otros. JFreeChart es altamente configurable y permite personalizar la apariencia y el estilo de los gráficos generados. Se utiliza comúnmente en aplicaciones de informes, análisis de datos y representación visual de información numérica. Ofrece una API sencilla para crear y personalizar gráficos, lo que lo convierte en una opción popular para desarrolladores que desean agregar capacidades de visualización a sus aplicaciones Java.



Gramática

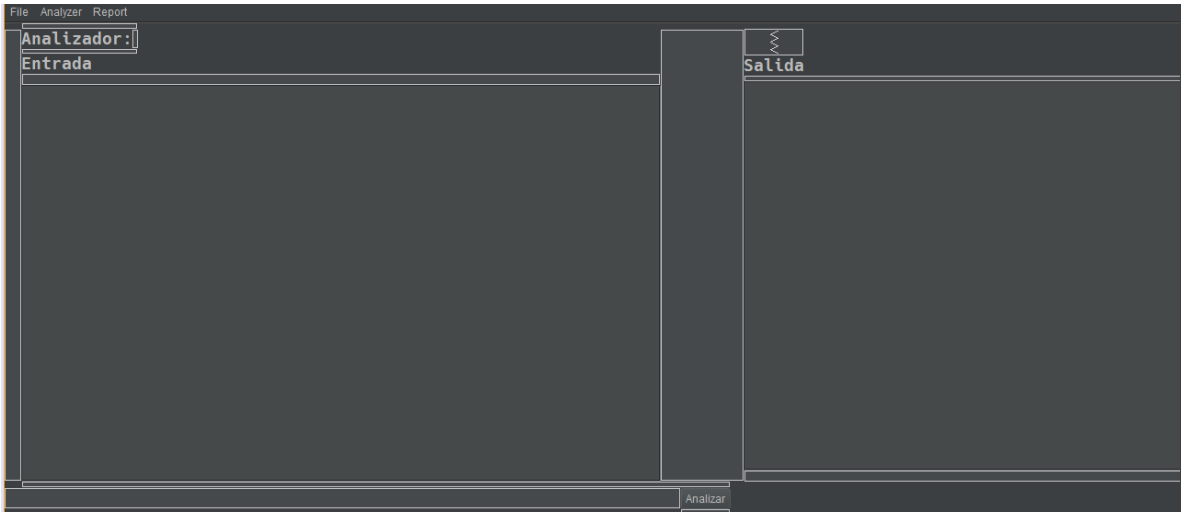
Una gramática libre de contexto es un conjunto de reglas que describe la estructura de un lenguaje. Estas reglas definen cómo se pueden combinar diferentes elementos (llamados símbolos) para formar secuencias válidas en el lenguaje. Cada regla consta de un símbolo no terminal (que puede reemplazarse por otras secuencias de símbolos) y una secuencia de símbolos, que pueden ser terminales (símbolos finales del lenguaje) o no terminales. Las gramáticas libres de contexto son ampliamente utilizadas en la teoría de la computación y la programación, especialmente en la descripción de lenguajes de programación y en el análisis sintáctico de texto en procesamiento de lenguaje natural.

```
<Inicio> ::= BRAIZQ <listainstrucciones> BRADER
;

<listainstrucciones> ::= <listainstrucciones> COMA <instruccion>
                       | <instruccion>
;

<instruccion> ::= <declaracion>
;

<declaracion> ::= CADENA DOSPTO CADENA
                | CADENA DOSPTO DECIMAL
;
```

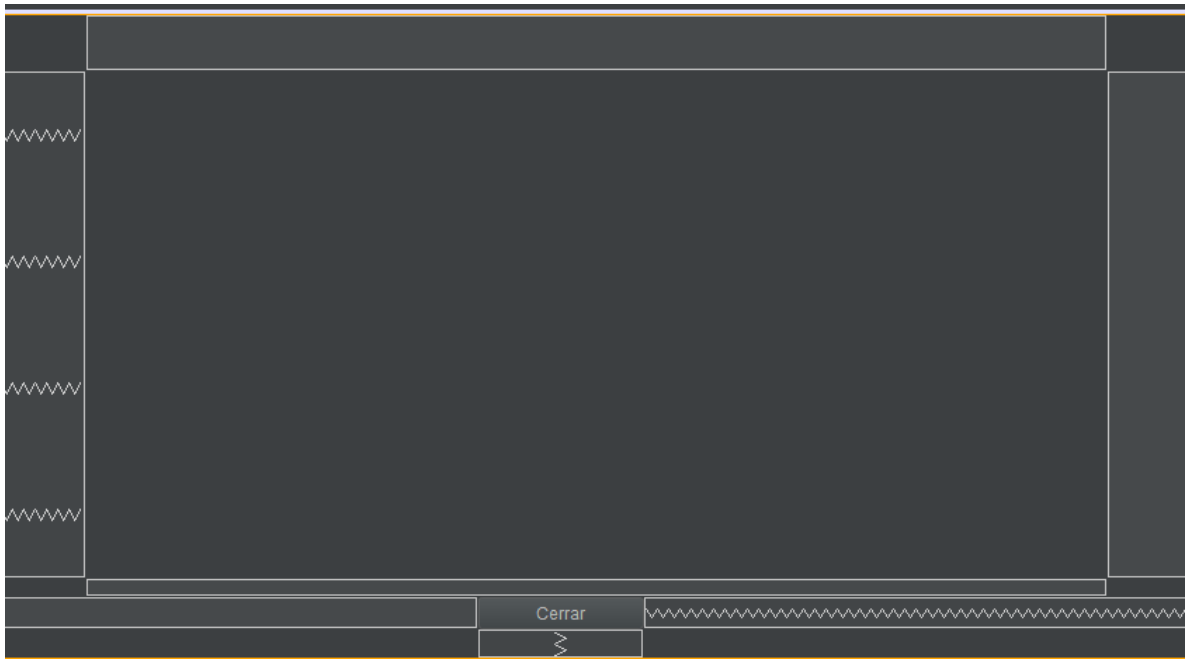
Frontend

Para la elaboración del frontend se uso el drag and drop de java swing que nos permite utilizar el programa netbeans en esta caso su versión 16

Ventana Main



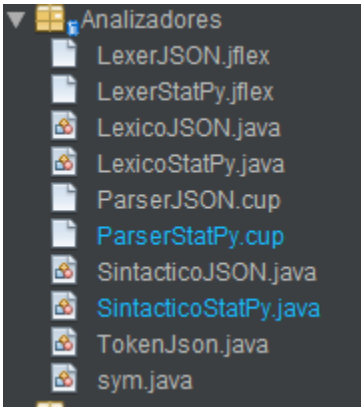
Diseño simple de la ventana principal del cual se desprenden dos ventanas hijas en donde se mostrara el reporte de las funciones estadísticas



Un panel y un botón que permite cerrar la ventana sin necesidad de acabar con la ejecución

Scanners y Parsers

Estas están representados por una clase que representa el scanner de flujo de caracteres que se encargara de convertirlos en tokens que a su vez serán usados en el parser que se encargara de armar el arbol de análisis sintactico donde realizaremos nuestras acciones gramaticales



Los archivos jflex contendrán las expresiones regulares de nuestro lenguaje y los cup las reglas gramaticales de nuestro lenguaje para generar los analizadores nos ayudamos de 2 clases

Parser

Se encarga de que el archivo. cup sea leído y transformado en el parser de tokens y que los tokens sean reconocidos por medio de una clase auxiliar llamada symbol

```
public class ParserStatPy {

    public static void main(String[] args) {
        String opciones[] = new String[7];

        //Seleccionamos la opción de dirección de destino
        opciones[0] = "-destdir";

        //Le damos la dirección, carpeta donde se va a generar el
parser.java & el simbolosxxx.java
        opciones[1] = "src/main/java/Analizadores/";

        //Seleccionamos la opción de nombre de archivo simbolos
        opciones[2] = "-symbols";

        //Le damos el nombre que queremos que tenga
        opciones[3] = "sym";
```

```
        //Seleccionamos la opcion de clase parser
        opciones[4] = "-parser";

        //Le damos nombre a esa clase del paso anterior
        opciones[5] = "SintacticoStatPy";

        //Le decimos donde se encuentra el archivo .cup
        opciones[6] = "src/main/java/Analizadores/ParserStatPy.cup";
        try
        {
            java_cup.Main.main(opciones);
        }
        catch (Exception ex)
        {
            System.out.print(ex);
        }
    }
}
```

Mensaje después de ejecutar el programa:

```
------- CUP v0.11b 20160615 (GIT 4ac7450) Parser Generation Summary -------
  0 errors and 0 warnings
  62 terminals, 37 non-terminals, and 118 productions declared,
  producing 352 unique parse states.
  0 terminals declared but not used.
  0 non-terminals declared but not used.
  0 productions never reduced.
  0 conflicts detected (0 expected).
  Code written to "SintacticoStatPy.java", and "sym.java".
---------------------------------------------------- (CUP v0.11b 20160615 (GIT 4ac7450))
```

Lexer

Se encarga de devolver los patrones de la cadena de caracteres en estructuras de datos llamadas tokens que tienen ciertos atributos y que serán usados en el análisis léxico del parser

```
import java.io.File;


public class LexerStatPy {
    public static void generarLexer(String ruta){
        File archivo=new File(ruta);
        jflex.Main.generate(archivo);

    }

    public static void main(String[] args) {
```

```
        String ruta="src/main/java/Analizadores/LexerStatPy.jflex";
        generarLexer(ruta);
    }
}
```

Mensaje después de ejecutar el programa

```
285 states before minimization, 265 states in minimized DFA
Old file "src\main\java\Analizadores\LexicoStatPy.java" saved as "src\main\java\Analizadores\LexicoStatPy.java~"
Writing code to "src\main\java\Analizadores\LexicoStatPy.java"
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  3.495 s
Finished at: 2023-08-29T17:14:09-06:00
------------------------------------------------------------------------
```

Errores

```
public class ErrorLexico {
    public String Lexema;
    public String Descripcion="Caracter No Reconocido por el Lenguaje";
    public int Linea;
    public int Columna;

    public ErrorLexico(String Lexema, int Linea, int Columna) {
        this.Lexema = Lexema;
        this.Linea = Linea;
        this.Columna = Columna;
    }

}
```

Con esta clase manejamos los errores léxicos de nuestro programa guardándolos desde acciones gramaticales en una lista como esta estructura y luego recorriendo cada error si es que existiesen

Acciones gramaticales

En cup una acción gramatical es la acción que realiza el analizador sintactico que se encarga de ejecutar alguna instrucción la forma correcta de ejecutar estas acciones es de la siguiente forma

```
{:
    /* Accion gramatical
    codigo java en su defecto*/

:}
```

Por ejemplo en esta acción asignamos el titulo obtenido de la gramática a nuestra variable java

```
instruccionesbarras::=STRING TITULO  IGUAL IDENT:a
PTOCOMA{:Titulo_Barras=t_variables.get(a).toString();:}
```

de esta forma se realiza todo el funcionamiento del código por medio de estas acciones

Archivos

Como bien se sabe los archivos se leen y se puede escribir en ellos ayudados de esto fue que conseguimos manipular los archivos de entrada con estos dos métodos que uno se encarga de leer contenido y el otro de escribir contenido

```
public static String leer(String path){
    File archivo = null;
    FileReader fr = null;
    BufferedReader br = null;
    String texto="";
    try {
       // Apertura del fichero y creacion de BufferedReader para poder
       // hacer una lectura comoda (disponer del metodo readLine()).
       archivo = new File (path);
       fr = new FileReader (archivo);
       br = new BufferedReader(fr);
       // Lectura del fichero
       String linea;
       while((linea=br.readLine())!=null)
          texto+=linea+"\n";
    }
    catch(Exception e){
       e.printStackTrace();
    }finally{

       try{
          if( null != fr ){
             fr.close();
          }
       }catch (Exception e2){
          e2.printStackTrace();
       }
    }
```

```
  return texto;
  }

  public static void escribir(String path,String TextField2){
      FileWriter fichero = null;
      PrintWriter pw = null;
      try
      {
          fichero = new FileWriter(path);
          pw = new PrintWriter(fichero);
          String texto=TextField2;

          pw.println(texto);
      } catch (Exception e) {
          e.printStackTrace();
      } finally {
         try {
         // Nuevamente aprovechamos el finally para
         // asegurarnos que se cierra el fichero.
         if (null != fichero)
            fichero.close();
         } catch (Exception e2) {
            e2.printStackTrace();
         }
      }
}
```

Diagrama de Clases