**Instituto Tecnológico de Costa Rica**

**Escuela de Ingeniería en Computación**

**Proyecto programado #1**

Greivin Mauricio Fernández Brizuela c.2022437510

Daniel Alonso Garbanzo Carvajal c.2022117129 

IC-5701 – Compiladores e Interpretes

16 de diciembre de 2023

Contenido

[Manual de usuario 3](#_Toc153622930)

[Pruebas de funcionalidad 4](#_Toc153622931)

[Descripción del problema 5](#_Toc153622932)

[Diseño del programa 6](#_Toc153622933)

[Librerías usadas 7](#_Toc153622934)

[Análisis de resultados 8](#_Toc153622935)

# Manual de usuario

1. Construcción inicial del proyecto:
   * Realiza un "build" del proyecto para inicializar todas las librerías necesarias.
   * Asegúrate de que los plugins definidos en el archivo pom.xml de la carpeta programa estén correctamente configurados. Esto es crucial para una depuración adecuada.
2. Estructura del proyecto:
   * El proyecto se compone de tres paquetes principales: Analizadores, Generadores y Programa.
   * Los archivos jflex y cup en el paquete Analizadores son esenciales para generar los archivos .java correspondientes.
3. Generación de código:
   * Ejecuta los archivos GeneradorLexer y GeneradorParser para generar el código necesario.
   * Se debe tener en cuenta los siguientes detalles durante este proceso:
4. Modificaciones en App.java:
   * + En la carpeta Programa, dentro del archivo App.java, encontrarás algunos errores.
     + Comenta los imports import Analizadores.Lexer e import Analizadores.sym.
     + También comenta las líneas del 32 al 62.

b. Ejecución de GeneradorParser:

* + - Después de ejecutar GeneradorParser y generar Parser.java, comenta las líneas del 100 al 112.
    - Esto es necesario porque el Lexer aún no ha sido creado.

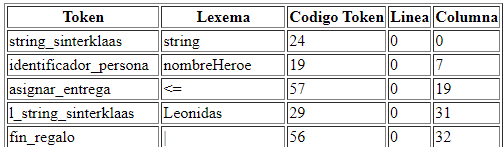
c. Ejecución de GeneradorLexer:

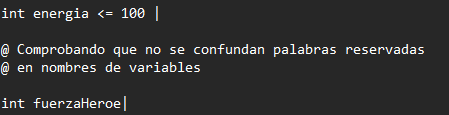
* + - Una vez que GeneradorParser haya finalizado, ejecuta GeneradorLexer.

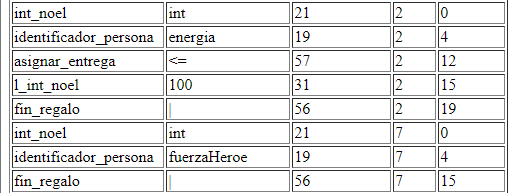
1. Finalización y ejecución:
   * Descomenta todas las líneas que previamente comentaste.
   * Ejecuta el archivo App.java del paquete Programa para utilizar la aplicación.
2. Resultado de la ejecución:
   * Al correr la aplicación, se debe crear un archivo .html.
   * Este archivo contendrá un análisis de todas las líneas del archivo test.txt, mostrando el token, lexema, código del token, línea y columna.

# Pruebas de funcionalidad

string nombreHeroe <= "Leonidas"| @ Inicialización de cadena

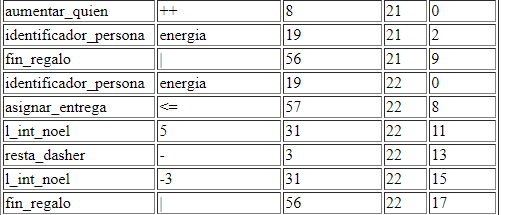






++energia|

energia <= 5 - -3|



float agilidad <= 20.5 - 3.1 + -5|

float resistencia <= 2.3|

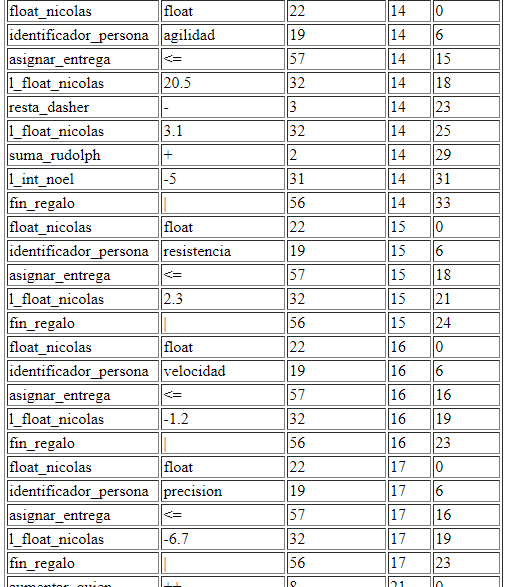
float velocidad <= -1.2|

float precision <= -6.7|

/\_

Comentario multilinea sobre las estadísticas

\_/



do {

--energia|

print("Energía actual: ", energia)|

} until (energia == 0)

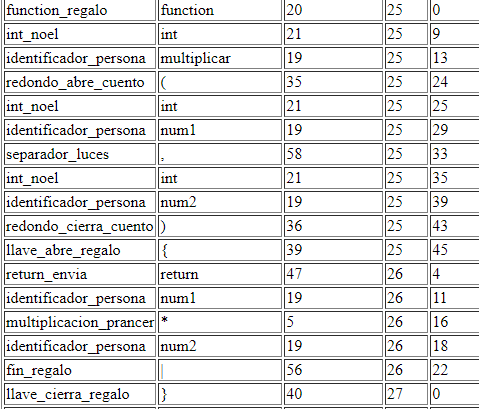


@ Definiendo nuevas funciones

function int multiplicar(int num1, int num2) {

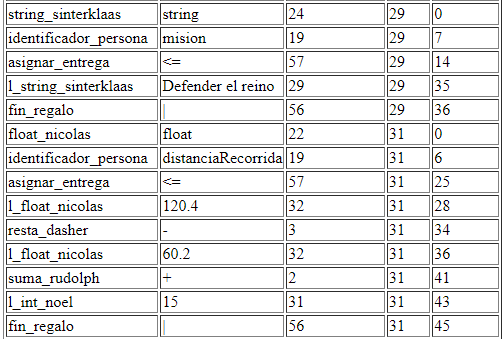
return num1 \* num2|

}



string mision <= "Defender el reino"| @ Definiendo el objetivo

float distanciaRecorrida <= 120.4 - 60.2 + 15|

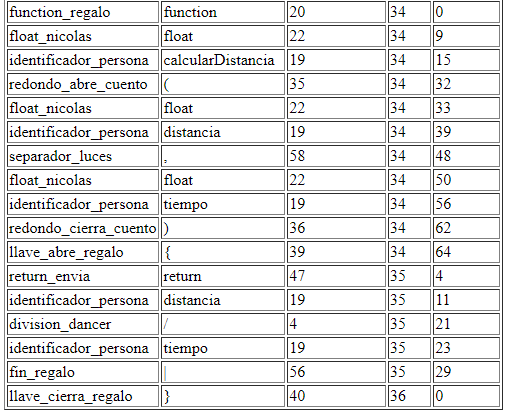


@ Función para cálculo de distancias

function float calcularDistancia(float distancia, float tiempo) {

return distancia / tiempo|

}



# Descripción del problema

El avance del proyecto a diseñar es la fase de Análisis Léxico para gramática. Algunos aspectos relevantes del problema son los siguientes: leer un archivo fuente (.txt), escribir en un archivo todos los tokens encontrados (.html), identificador asociado con el lexema y debe de reportar y manejar los errores léxicos encontrados.

La herramienta que se tiene que usar es JFlex para realizar el análisis léxico junto con la librería java-cup. Se deben crear las definiciones regulares correspondientes para identificar los diferentes tokens con la siguiente información: tipo de token, lexema, línea de aparición y columna.

# Diseño del programa

La solución propuesta para el programa es la siguiente:

Se contará con 3 paquetes diferentes, los cuales son: Analizadores, Generadores y Programa. Analizadores cuenta con los archivos .jflex y .cup, adicionalmente en este paquete se crearán los archivos .java para las clases sym, Parser (java-cup) y Lexer (JFlex). Generadores cuenta con 2 archivos .java, los cuales se encargan de generar las diferentes clases para el funcionamiento correcto del programa. Por último, Programa cuenta con la clase App.java, para correr el programa (leer archivo fuente, guardar resultados en archivo .html y mostrar errores de tokens en consola), adicionalmente, se cuenta con la clase HTMLTable.java para generar el archivo .html.

El archivo lexer.jflex cuenta con todas las definiciones regulares para identificar los diferentes tokens. Al mismo tiempo, el archivo parser.cup cuenta con todos los nombres de los diferentes terminales.

Por último, al ser un proyecto creado en Maven todas las instalaciones de las dependencias se almacenan en un archivo pom.xml. De esta manera, se lleva un control exacto de las versiones de las librerías que se están utilizando y el compilar es más amigable.

# Librerías usadas

Las librerías que se utilizaron fueron JFlex para realizar el análisis léxico, adicionalmente se utilizó java-cup para hacer el parsing del archivo y trabajar junto con JFlex. Las librerías mencionadas anteriormente se instalaron con Maven, por lo tanto, se cuenta con un pom.xml para mantener un control de las diferentes dependencias y simplificar la compilación al usuario.

# Análisis de resultados

* Operadores aritméticos binarios: nombre de renos 
* Operadores aritméticos unarios: grinch, quien
* Operadores relacionales: elfos de santa
* Operadores lógicos: reyes magos
* identificador: persona
* tipos: nombres de papá Noel
* literales (valorar caso especial bool): nombres de papá noel con un prefijo l\_
* paréntesis: abrecuento, cierrecuento
* paréntesis cuadrado: abreempaque, cierraempaque
* llaves: abreregalo, cierraregalo
* lexemas de estructuras de control (if, elif, else, for, do, until, return, break): elfo, hada, duende, envuelve, hace, revisa, envia, corta
* print, read: narra, escucha
* lexema de fin de expresión: finregalo
* lexema de asignar: entrega
* lexema separador(coma): 