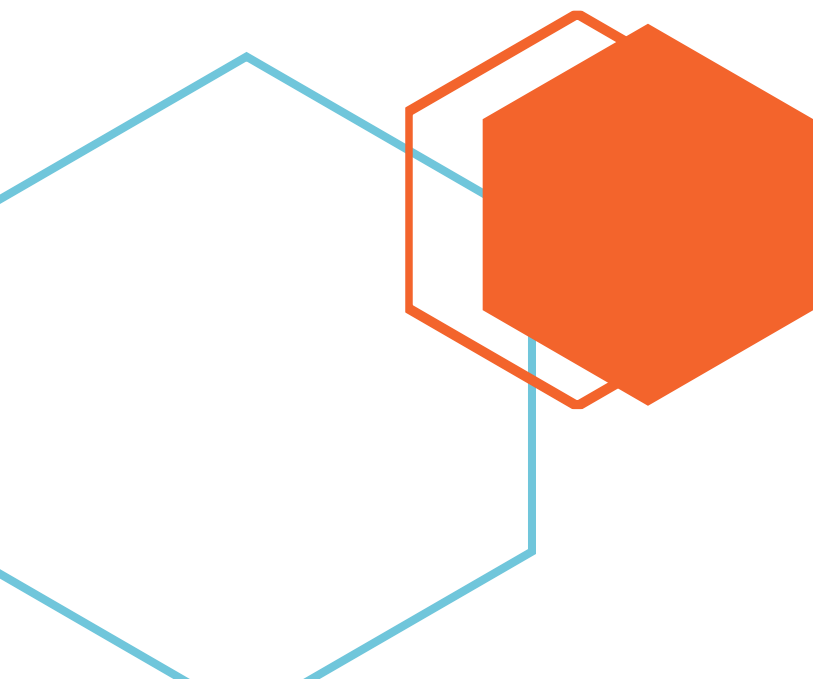




AUGUS

Manual de Usuario

Lenguaje de programación basado en PHP y MIPS



Generalidades

Lenguaje Agus

Descripción

Augus es un lenguaje de programación, basado en PHP y en MIPS. Su principal funcionalidad es ser un lenguaje intermedio, ni de alto nivel como PHP ni de bajo nivel como el lenguaje ensamblador de MIPS.

Generalidades

- El Lenguaje tiene dos restricciones
- cada instrucción es una operación simple
- cada instrucción tiene un máximo de dos operadores y su asignación
- lenguaje débilmente tipado con excepción más adelante explicada
- no hay estructuras de control únicamente IF
- el flujo de control es por medio de etiquetas y saltos

Flujo

Flujo del programa

- Se ingresa el código augus ya sea en un archivo o editado
- Se ejecuta el código (Boton ASC/DESC)
- Si se utiliza la opción ASC se tiene el debugger habilitado
- El resultado de la ejecución se muestra en la consola
- Opcionalmente se pueden acceder a reportes en la sección

Tips



la funcionalidad del código es lineal sin ámbitos.

Sandbox php online debugger ejecuta el mismo código, sin embargo, augus no ejecuta todo el código de php

Tipos de Datos

Punto flotante

Entero

Cadena de caracteres

Arreglos

Definición de registros(variables)

\$t0..\$tn	Temporales
\$a0..\$an	Parámetros
\$v0..\$vn	Valores devueltos por funciones
\$ra	Simulador de dirección de retorno por nivel
\$s0..\$sn	Pilas
\$sp	Puntero de la pila

Cada registro puede almacenar cualquier tipo de dato especificado anteriormente

Instrucciones simples y unarias

En los espacios de registros de las siguientes instrucciones pueden ser ocupados por constantes de cualquier tipo.

main:	Inicio del programa.
label:	Definición del inicio de una etiqueta.
goto label;	Salto incondicional hacia una etiqueta.
\$t1 = 10;	Asignación numérica.
\$t1 = 'hola';	Asignación de una cadena de caracteres.
\$t1 = \$t2;	Copia simple.
\$t1 = - \$t2;	Negativo.
\$t1 = &\$t2;	\$t1 es un puntero a la dirección de \$t2.
unset(\$t1);	Destruye la variable \$t1.
print(\$t1);	Imprime en pantalla el contenido de \$t1.
\$t1 = read();	Lee la entrada del teclado queda en \$t1.
#comment	Comentario de una sola línea.
exit;	Finaliza la ejecución.

Conversiones

\$t1 = (int) \$t2;	Si \$t2 tiene decimales son eliminados. Si \$t2 es un carácter se toma su código ASCII. Si \$t2 es una cadena se toma el ASCII del primer carácter. Si \$t2 es un arreglo se aplica al primer elemento las reglas anteriores.
\$t1 = (float) \$t2;	Si \$t2 es entero se agrega “.0”. Si \$t2 es un carácter se toma su código ASCII con decimal. Si \$t2 es una cadena se toma el ASCII del primer carácter más el decimal. Si \$t2 es un arreglo se aplica al primer elemento las reglas anteriores.
\$t1 = (char) \$t2;	Si \$t2 es un número de 0 a 255 se convierte en el carácter basado en ASCII. Si \$t2 es un número mayor a 255 entonces se aplica el módulo 256 para extraer el ASCII. Si \$t2 es un decimal, se quitan los decimales y se aplican las reglas anteriores. Si \$t2 es una cadena, se almacena solo el primer carácter. Si \$t2 es un arreglo, se almacena solo el primer valor aplicando las reglas anteriores.

Instrucciones Lógicas

\$t1 = !\$t2;	Not, si \$t2 es 0 \$t1 es 1, si \$t2 es 1 \$t1 es 0.
\$t1 = \$t2 && \$t3;	And, 1 para verdadero, 0 para falso.
\$t1 = \$t2 \$t3;	Or, 1 para verdadero, 0 para falso.
\$t1 = \$t2 xor \$t3;	Xor, 1 para verdadero, 0 para falso.

Instrucciones bit a bit

\$t1 = ~\$t2;	Not.
\$t1 = \$t2 & \$t3;	And.
\$t1 = \$t2 \$t3;	Or.
\$t1 = \$t2 ^ \$t3;	Xor.
\$t1 = \$t2 << \$t3;	Shift de \$t2, \$t3 pasos a la izquierda.
\$t1 = \$t2 >> \$t3;	Shift de \$t2, \$t3 pasos a la derecha.

Instrucciones Relacionales

\$t1 = \$t2 == \$t3;	\$t1 = 1 si \$t2 es igual a \$t3, sino 0.
\$t1 = \$t2 != \$t3;	\$t1 = 1 si \$t2 no es igual a \$t3, sino 0.
\$t1 = \$t2 >= \$t3;	\$t1 = 1 si \$t2 es mayor o igual a \$t3, sino 0.
\$t1 = \$t2 <= \$t3;	\$t1 = 1 si \$t2 es menor o igual a \$t3, sino 0.
\$t1 = \$t2 > \$t3;	\$t1 = 1 si \$t2 es mayor a \$t3, sino 0.
\$t1 = \$t2 < \$t3;	\$t1 = 1 si \$t2 es menor a \$t3, sino 0.

Instrucciones aritméticas

\$t1 = \$t2 + \$t3;	Suma.
\$t1 = \$t2 - \$t3;	Resta.
\$t1 = \$t2 * \$t3;	Multipliación.
\$t1 = \$t2 / \$t3;	División.
\$t1 = \$t2 % \$t3;	Residuo.
\$t1 = abs(\$t2);	Valor absoluto.

Arreglos, cadenas y structs

Existen dos tipos de arreglo: numérico y asociativo

El numérico funciona como los arreglos convencionales, asignando un índice del arreglo a un valor en específico. El asociativo funciona como un struct en C, como una clase en c++(solo datos).

Las cadenas de caracteres también son tomadas como arreglos

<code>\$t1 = array();</code>	Define \$t1 como un arreglo o un struct, para diferenciarlos se utiliza ya sea el valor numérico o el nombre asociativo.
<code>\$t1[4] = 1;</code>	Asignación de un valor numérico (1) a un índice del arreglo (4).
<code>\$t1['nombre'] = 'carlos';</code>	Asignación de un valor cadena (carlos) a un componente del struct (nombre).
<code>\$t1 = \$t1[4];</code>	Acceso a un índice del arreglo.
<code>\$t1 = \$t2['nombre'];</code>	Acceso a un componente del struct.
<code>\$t1 = 'hola'; print(\$t1[0]); #imprime h</code>	Acceder a un carácter de una cadena.

Instrucciones de control

<code>if (\$t1) goto label;</code>	Salto condicional, si \$t1 es 1 salto, sino sigue la siguiente instrucción, \$t1 puede ser relacional. Las estructuras de control se implementan utilizando este salto condicional.
------------------------------------	---

Funciones y procedimientos

En vez de utilizar los conocidos CALL Y RETURN, se utilizarán los registros de parámetros y el de valor de retorno, junto con los GOTO para simular la llamada como se hace en MIPS con jal y jr

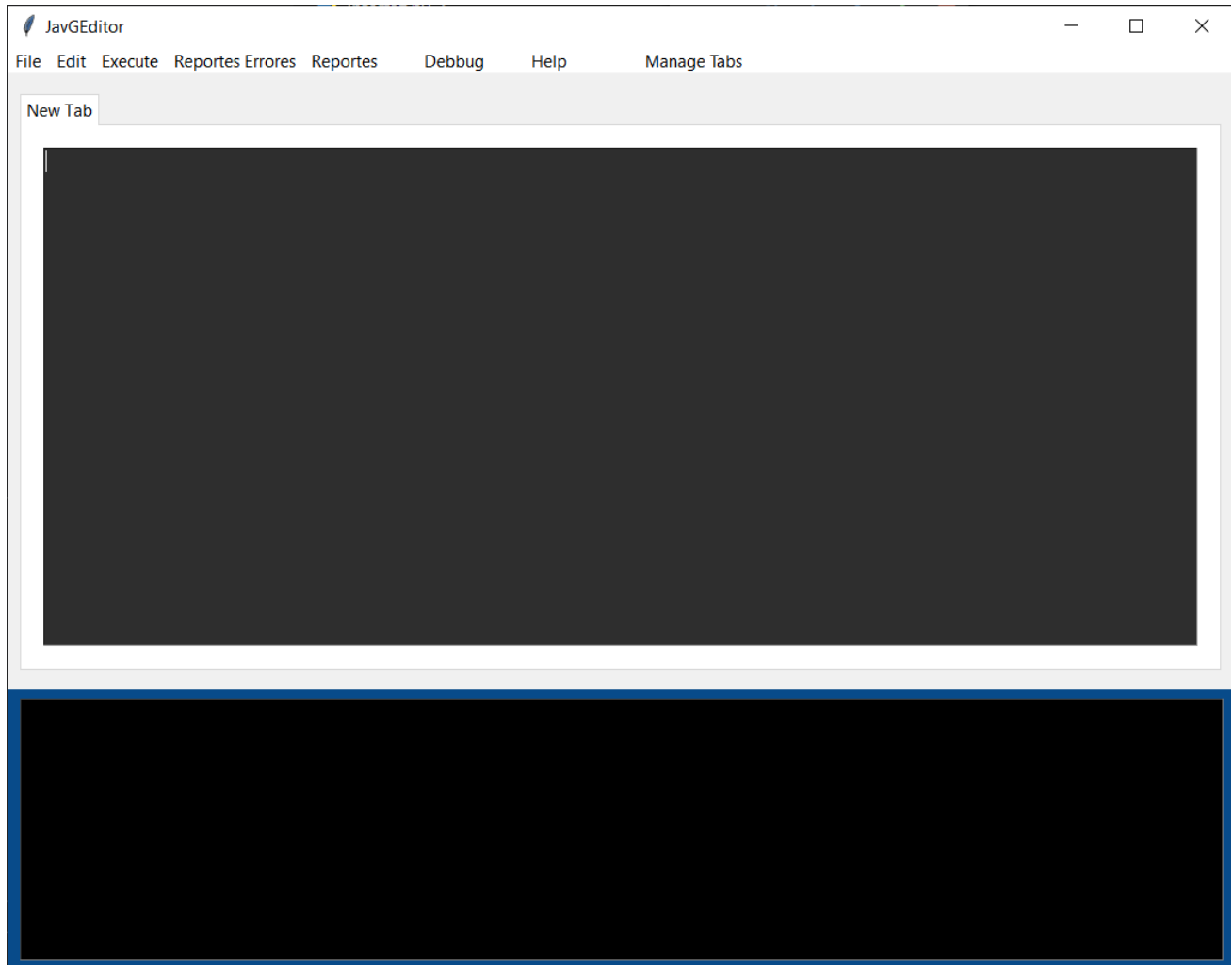
5. Ejemplos de alto nivel (C) y de medio nivel (Augus)

While

<pre>int main() { int x = 0; int a = 0; while(x<4) { a = a + x; x = x + 1; } printf("%d",a); }</pre>	<pre>main: \$t1 = 0; \$t2 = 0; while: if (\$t1>=4) goto end; \$t2 = \$t2 + \$t1; \$t1 = \$t1 + 1; goto while; end: print(\$t2);</pre>
---	--

IDE

El IDE que utiliza Augus en JavGEditor que cuenta con múltiples opciones par el usuario y una interfaz amigable y entendible además de ser muy poderosa y ligera

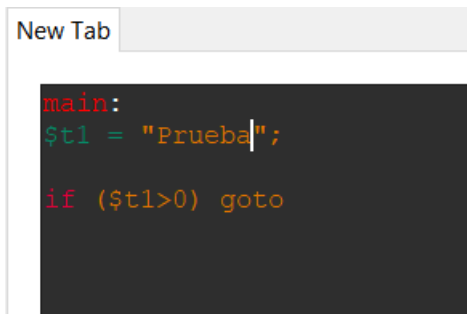


La interfaz es muy simple he intuitiva:

Componentes

Editor de código

El editor de código identificado con el color gris es el área en donde se coloca el código a trabajar ya sea un archivo abierto o edición a mano, el editor cuenta con syntax highlighting para poder diferenciar los diferentes componentes del código.

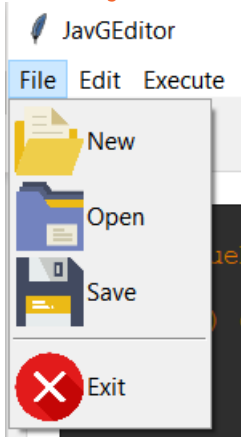


```
main:
$tl = "Prueba";

if ($tl>0) goto
```

El editor cuenta con múltiples pestañas por lo que para hacer el focus sobre que código trabajar basta con darle clic al editor de la pestaña requerida

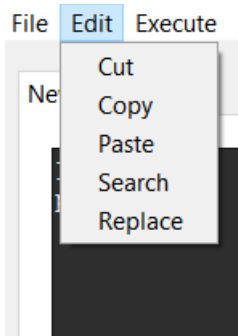
Manejo de archivos



La opción File nos despliega una lista de con las siguientes opciones:

- New crear un nuevo archivo de texto vacío
- Open: abre el explorador de archivos para colocar la información en el actual are de edición
- Save: Guarda los cambios en el archivo actualmente abierto
- EXIT: cierra el programa

Edicion

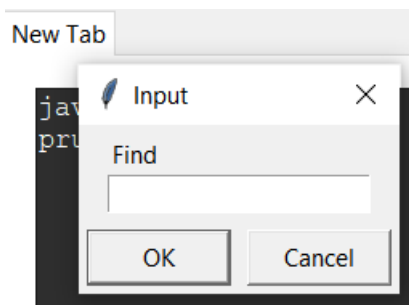


Cut: corta el texto seleccionado del actual área de edicion

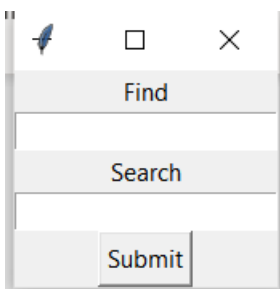
Copy: copia el texto del actual área de edicion

Paste: Pega del corta papeles al área de edicion actual

Search: abre una dialogo donde insertamos la palabra a buscar la cual se resaltara en color rojo, para quitar el resaltado es necesario buscar y cancelar la búsqueda

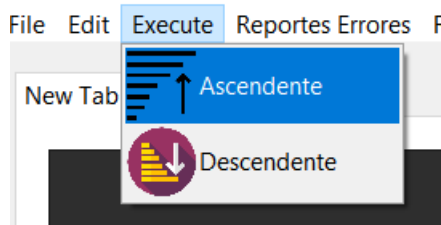


Replace: Abre una ventana de dialogo donde ingresamos la palabra a buscar en el texto y la palabra que hará el reemplazo en todas sus ocurrencias en el texto

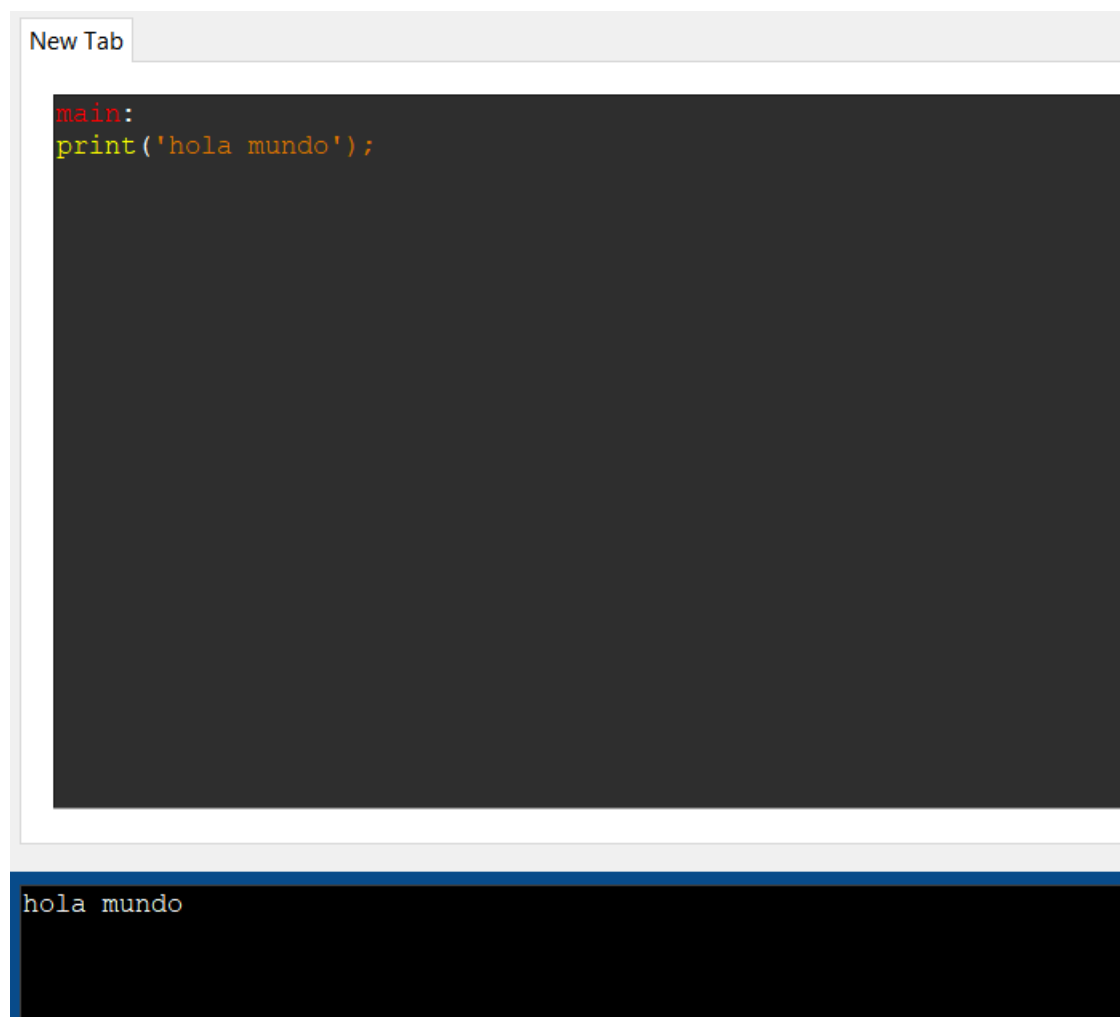


Execute:

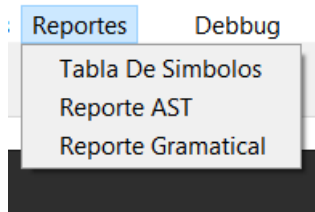
El área mas importante, aquí tenemos dos opciones de forma ascendente o de forma descendente, en cualquier de las dos opciones los resultados serán los mismos, sin embargo el reporte gramatical será completamente diferente



Al ejecutar veremos en la parte de la consola el resultado de nuestro Código, ya sea con mensajes de error o salida esperada



Una vez interpretado nuestro lenguaje podremos observar los reportes



Reporte Tabla de Simbolos

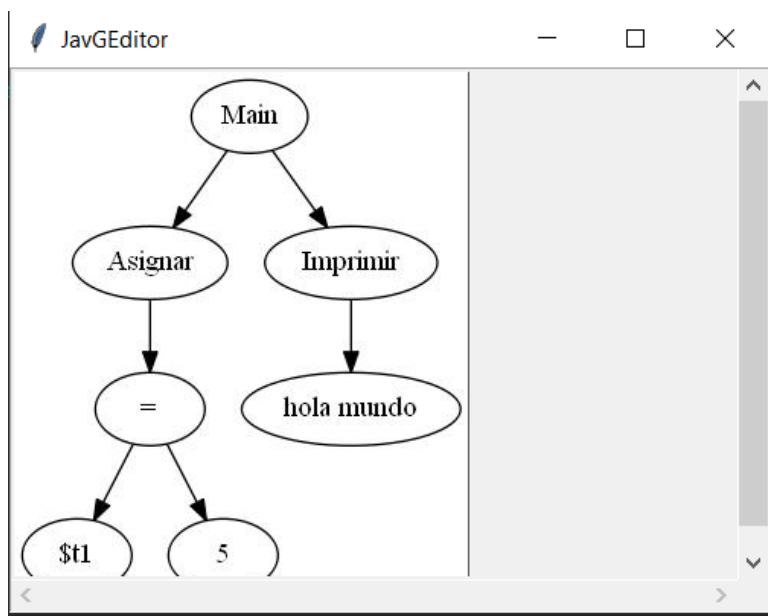
Muestra las variables y estructuras utilizadas en el código

A screenshot of the 'Reporte Tabla de Simbolos' window in the JavGEditor application. The window title is 'Reporte Tabla de Simbolos'. It displays a table with the following data:

Referencia	Nombre	Tipo	Valor	Ambito	Dimension
1	\$t1	TIPO_DATO.NUMERO	5	main	1

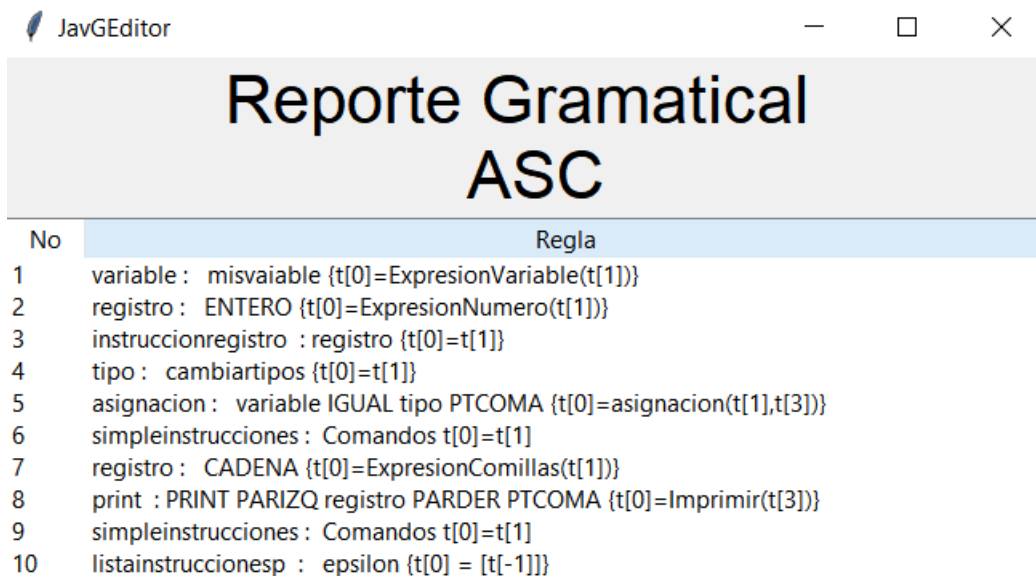
Reporte AST

Representa el árbol ordenado y enlazado que representa la estructura sintáctica del archivo a ejecutar



Reporte Gramatical

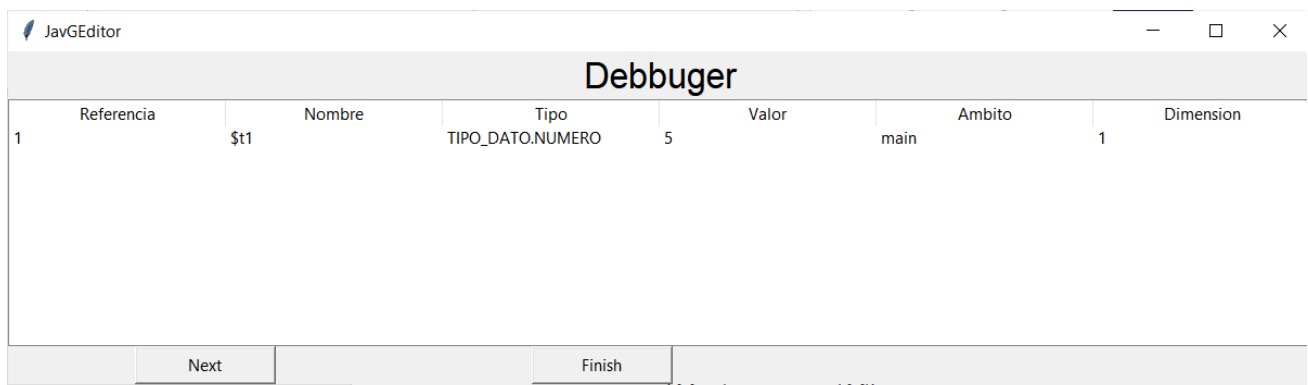
Es la definicion dirigida por la sintaxis, indicando que expresiones se utilizaron, símbolos terminales, y no terminales



No	Regla
1	variable : misvaible {t[0]=ExpresionVariable(t[1])}
2	registro : ENTERO {t[0]=ExpresionNumero(t[1])}
3	instruccionregistro : registro {t[0]=t[1]}
4	tipo : cambiartipos {t[0]=t[1]}
5	asignacion : variable IGUAL tipo PTCOMA {t[0]=asignacion(t[1],t[3])}
6	simpleinstrucciones : Comandos t[0]=t[1]
7	registro : CADENA {t[0]=ExpresionComillas(t[1])}
8	print : PRINT PARIZQ registro PARDER PTCOMA {t[0]=Imprimir(t[3])}
9	simpleinstrucciones : Comandos t[0]=t[1]
10	listainstruccionesp : epsilon {t[0] = [t[-1]]}

Reporte Gramatical

El debugger es la ejecucion paso a paso de nuestro codigo, se va mostrando el estado de la tabla de símbolos



	Referencia	Nombre	Tipo	Valor	Ambito	Dimension
1	\$t1	TIPO_DATO.NUMERO	5		main	1

Next Finish

- Next: ejecuta la siguiente instrucción
- Finish: ejecuta de forma normal el Código

AUGUS 2020

