



# **PROYECTO FINAL DE CLASE**

**DEPARTAMENTO ACADÉMICO DE INGENIERÍA EN  
SISTEMAS**

**REALIZADO EN AGOSTO DE 2025**



# UNAH CAMPUS CHOLUTECA

---

## Proyecto Final de Clase



### Fecha de Envío

03 de agosto del 2025

### Asignatura

IS913 – Diseño de Compiladores

### Catedrático

Ph.D. Wilson Octavio Villanueva Castillo

### Integrantes

Nestor Yuvini Ordoñez Baca – 20202300158

Ury Roberto Aguirre Ramirez – 20212320167

Jorlin Fernando Rosa Arrivillaga – 20192330138



INDICE DE CONTENIDO

**I. INTRODUCCIÓN ..... 4**

**II. CONTENIDO ..... 5**

    1. Requisitos Del Entorno De Desarrollo .....5

    2. Preparación Del Entorno De Trabajo.....7

**III. Descripción Del Lenguaje Easy ..... 15**

**IV. Estructura Lexica ..... 18**

**V. Reglas de estilo del lenguaje ..... 20**

**VI. Ejemplos de uso del lenguaje ..... 23**

**VII. Proceso de Ejecución de Código ..... 35**

**VIII. ANEXOS ..... 36**



## I. INTRODUCCIÓN

El presente informe documenta el proceso completo de diseño y desarrollo del compilador **EasyC**, proyecto académico realizado en el marco de la asignatura *Diseño de Compiladores-IS913*. A lo largo del periodo, se ha seguido una ruta formativa progresiva que nos ha permitido comprender y aplicar las distintas etapas involucradas en la construcción de un compilador, desde el análisis léxico hasta la generación de código intermedio.

**EasyC** surge como una propuesta didáctica inspirada en la sintaxis del lenguaje de programación C, pero con una estructura simplificada y accesible. El objetivo principal de este proyecto ha sido facilitar el entendimiento de los conceptos teóricos mediante la creación de un lenguaje propio, permitiendo así experimentar de forma práctica con cada componente del proceso de compilación.

Durante el primer parcial, se trabajó con *Flex* para implementar el analizador léxico, encargado de identificar los tokens definidos por el lenguaje. En el segundo parcial, se utilizó *Bison* para desarrollar el analizador sintáctico, donde se definieron las reglas gramaticales que rigen la estructura del lenguaje **EasyC**. Finalmente, en el tercer parcial, se integró el uso de ANTLR 4, una herramienta moderna y robusta que permite la construcción de analizadores mediante una gramática declarativa, lo que facilitó la visualización de árboles sintácticos y el manejo de estructuras más complejas del lenguaje.

Este informe incluye la documentación detallada de cada una de las fases del proyecto, los archivos fuente utilizados, las pruebas realizadas, las decisiones de diseño tomadas, y los desafíos enfrentados durante el desarrollo del compilador.

**EasyC** no solo representa el producto final de un proceso académico, sino también una herramienta que sintetiza el aprendizaje adquirido en torno al diseño de lenguajes, la teoría de autómatas, y la lógica detrás de los compiladores modernos. Este documento tiene como propósito dejar constancia técnica del trabajo realizado, y a su vez, servir como guía de referencia para estudiantes que en el futuro busquen emprender proyectos similares.



## II. CONTENIDO

### 1. Requisitos Del Entorno De Desarrollo

Esta sección presenta las herramientas y configuraciones necesarias para desarrollar y ejecutar el compilador del lenguaje **Easy**, utilizando ANTLR como base para el análisis léxico y sintáctico.

#### Herramientas necesarias en UBUNTU:

- **Java Development Kit (JDK)**  
Permite compilar y ejecutar los archivos *.java* generados por ANTLR. Se utilizó OpenJDK 17, instalado desde los repositorios oficiales.
- **ANTLR v4**  
Herramienta fundamental para el análisis léxico y sintáctico.  
Versión utilizada: **4.13.2**, descargada manualmente desde el sitio oficial de ANTLR.
- **Editor de código VS Code**  
Utilizado para la edición del código fuente del compilador. Compatible con Ubuntu y permite instalar la extensión oficial de ANTLR.
- **Compilador de Java**  
Viene incluido en el JDK. Se utilizó para compilar los archivos generados por ANTLR, con el classpath configurado de forma manual.
- **Terminal de Ubuntu**  
Se usó para realizar todas las acciones del proceso de compilación y pruebas.

#### Herramientas necesarias en WINDOWS:

- **Java Development Kit (JDK)**  
Indispensable para compilar y ejecutar el código fuente generado por ANTLR. Se utilizó OpenJDK 17, instalado desde Adoptium o SDKMAN.
- **ANTLR v4**  
Herramienta utilizada para construir el analizador léxico y sintáctico del lenguaje.  
Versión empleada: **4.13.2**, descargada desde el sitio oficial de ANTLR.



- **Editor de código VS Code**  
Seleccionado para editar archivos fuente del compilador. Se recomienda agregar la extensión *ANTLR4 grammar syntax support*.
- **Compilador de Java**  
Incluido en el JDK, permite compilar los archivos .java generados por ANTLR.
- **Símbolo del sistema / PowerShell**  
Ambiente de línea de comandos desde el cual se ejecutaron los procesos de compilación y prueba.

### Herramientas necesarias en macOS:

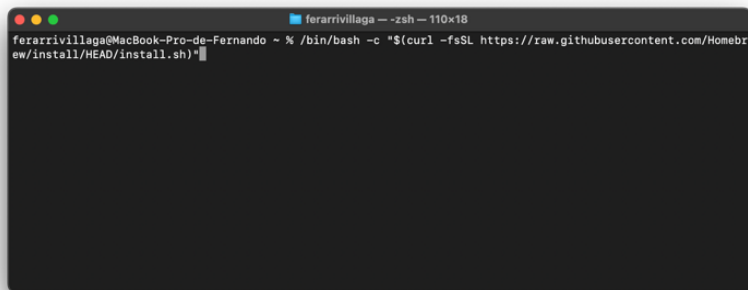
- **Java Development Kit (JDK)**  
Se requiere para compilar y ejecutar los archivos .java generados por ANTLR. Se recomienda usar la versión OpenJDK 24.0.1 disponible desde Homebrew.
- **ANTLR v4**  
Herramienta principal para generar analizadores léxicos y sintácticos a partir de gramáticas.  
Versión utilizada: **4.13.1**
- **Editor de código VS Code**  
Recomendado para editar archivos fuente del compilador (.g4, .java, etc.). Se sugiere instalar la extensión *ANTLR4 grammar syntax support* para facilitar la navegación y validación de gramáticas.
- **Compilador de Java**  
Incluido en el JDK, se utiliza para compilar los archivos generados por ANTLR. Se emplea el comando javac desde terminal.
- **Terminal de macOS**  
Utilizada para ejecutar los comandos de instalación, compilación y pruebas del compilador.

## 2. Preparación Del Entorno De Trabajo

A continuación, se detallan los pasos necesarios para instalar ANTLR 4.13.1 y configurar el entorno de desarrollo en macOS (Sequoia):

- macOS

### Paso 1: Instalar Homebrew

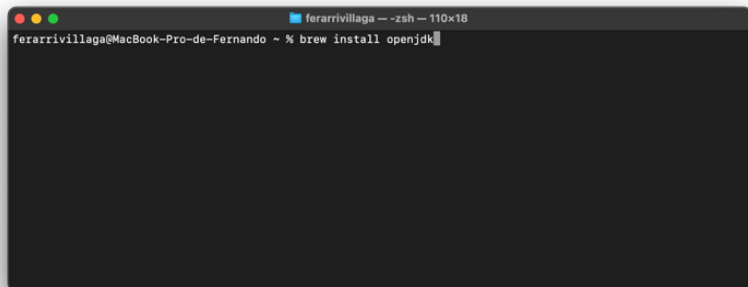


```
ferarrivillaga@MacBook-Pro-de-Fernando ~ % /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

#### ¿Qué hace?

Instala **Homebrew**, el gestor de paquetes más utilizado en macOS. Te permitirá instalar fácilmente Java y ANTLR sin tener que descargarlos manualmente.

### Paso 2: Instalar Java (OpenJDK)



```
ferarrivillaga@MacBook-Pro-de-Fernando ~ % brew install openjdk
```

#### ¿Qué hace?

Instala **Java Development Kit (JDK)**. ANTLR está programado en Java, por lo que es necesario tenerlo para poder ejecutar el compilador y las herramientas asociadas.

### Paso 3: Agregar Java al PATH (si aún no funciona)



```
ferarrivillaga@MacBook-Pro-de-Fernando ~ % echo 'export PATH="/opt/homebrew/opt/openjdk/bin:$PATH"' >> ~/.zprofile
file
source ~/.zprofile
```

### ¿Qué hace?

Agrega la ruta de instalación de Java al PATH del sistema. Esto permite que puedas ejecutar java desde cualquier carpeta en la terminal.

### Paso 4: Verificar que Java esta correctamente instalado

```
ferarrivillaga@MacBook-Pro-de-Fernando ~ % java -version
```

### ¿Qué hace?

Muestra la versión de Java instalada. Deberías ver algo como *openjdk version "24.0.1"*.

### Paso 5: Instalar ANTLR 4

```
ferarrivillaga@MacBook-Pro-de-Fernando ~ % brew install antlr
```

### ¿Qué hace?

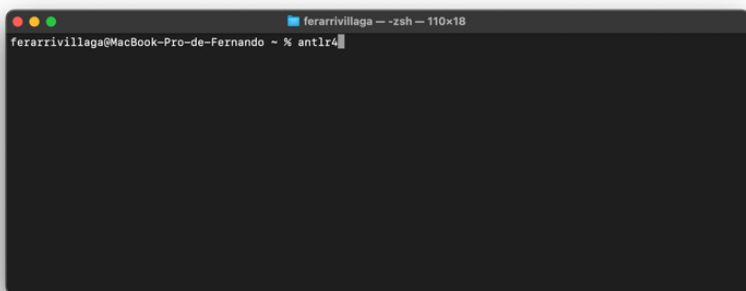




Descarga e instala ANTLR 4 (versión más reciente) junto con sus herramientas (antlr4 y grun).

## Paso 6: Verificar que ANTLR 4 se instaló correctamente

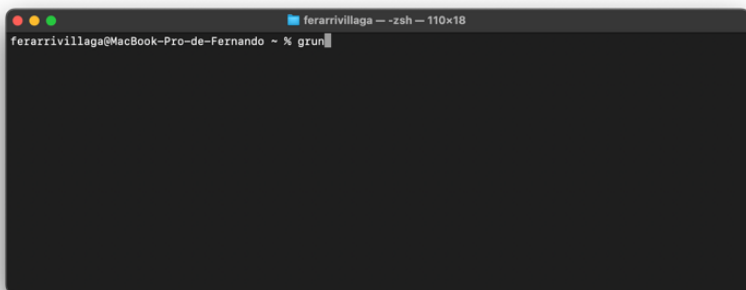
### a. Verificar el comando ANTLR4



### ¿Qué hace?

Ejecuta el generador de analizadores ANTLR. Si se instaló correctamente, mostrará un mensaje con las instrucciones de uso.

### b. Verificar el comando *grun* (TestRig)



### ¿Qué hace?

Ejecuta el visualizador de árboles de análisis de ANTLR (útil para probar tu gramática).

- Windows

A continuación, se detallan los pasos necesarios para instalar ANTLR 4.13.2 y configurar el entorno de desarrollo en Windows:

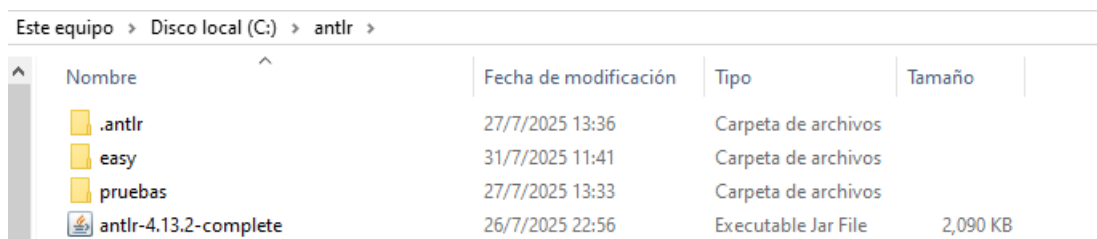
1. descargar el JDK de Java [https://download.oracle.com/java/24/latest/jdk-24\\_windows-](https://download.oracle.com/java/24/latest/jdk-24_windows-)

[x64\\_bin.exe](#) ([sha256](#))

2. descargar el binario de ANTLR. Al momento de la realización de este documento la última versión del binario es 4.13.2 <https://www.antlr.org/download.html>



3. instalar el JDK
4. creas una carpeta dentro del disco C:/ con el nombre antlr, ahi guardas el binario que descargaste.



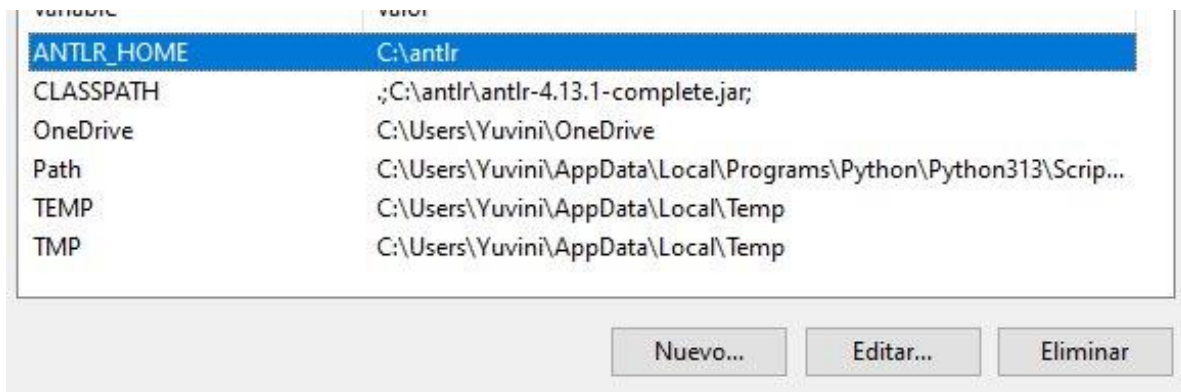
5. Ir a la sección de Variables de Entorno para crear la variables necesarias para el funcionamiento de ANTLR.



6. Creamos la primera variable que se llamara ANTLR\_HOME a este la damos el valor de la ruta donde esta la carpeta que creamos en el disco C:/



La segunda variable se llamara CLASPATH que es este permitirá que podamos acceder al jar de java y le asignamos el siguiente valor `.;%ANTLR_HOME%\antlr-4.13.2-complete.jar`;



7. Visualizamos via consola CMD si todo se instaló correctamente con el siguiente comando `java -cp "C:\antlr\antlr-4.13.2-complete.jar" org.antlr.v4.Tool` debe aparecer el mensaje que se muestra en la siguiente imagen.

```
C:\Users\Yuvini>java -cp "C:\antlr\antlr-4.13.2-complete.jar" org.antlr.v4.Tool
ANTLR Parser Generator Version 4.13.2
-o ____ specify output directory where all output is generated
-lib ____ specify location of grammars, tokens files
-atn ____ generate rule augmented transition network diagrams
-encoding ____ specify grammar file encoding; e.g., euc-jp
-message-format ____ specify output style for messages in antlr, gnu, vs2005
-long-messages show exception details when available for errors and warnings
-listener generate parse tree listener (default)
-no-listener don't generate parse tree listener
-visitor generate parse tree visitor
-no-visitor don't generate parse tree visitor (default)
-package ____ specify a package/namespace for the generated code
-depend generate file dependencies
-D<option>=value set/override a grammar-level option
-Werror treat warnings as errors
-XdbgST launch StringTemplate visualizer on generated code
-XdbgSTwait wait for STviz to close before continuing
-Xforce-atn use the ATN simulator for all predictions
-Xlog dump lots of logging info to antlr-timestamp.log
-Xexact-output-dir all output goes into -o dir regardless of paths/package
```

8. La estructura de árbol de la carpeta debe quedar así

```
antlr-project/  
├── antlr-4.13.2-complete.jar  
├── Expr.g4  
└── Main.java
```

- Linux (Ubuntu)

A continuación, se detallan los pasos necesarios para instalar ANTLR 4.13.2 y configurar el entorno de desarrollo en Linux (Ubuntu):

**Paso 1** actualizamos los paquetes de nuestro sistema en esta ocasión Ubuntu

```
ubuntu@ubuntu:~$ sudo apt update && upgrade  
Ign:1 cdrom://Ubuntu 24.04.2 LTS _Noble Numbat_ - Release amd64 (20250215) noble InRelease  
Hit:2 cdrom://Ubuntu 24.04.2 LTS _Noble Numbat_ - Release amd64 (20250215) noble Release  
Hit:3 http://archive.ubuntu.com/ubuntu noble InRelease  
Get:4 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]  
Get:5 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]  
Get:6 http://archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]  
Get:8 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [1,054 kB]  
Get:9 http://security.ubuntu.com/ubuntu noble-security/main i386 Packages [316 kB]  
Get:10 http://security.ubuntu.com/ubuntu noble-security/main Translation-en [183 kB]  
Get:11 http://security.ubuntu.com/ubuntu noble-security/main amd64 Components [21.6 kB]  
Get:12 http://security.ubuntu.com/ubuntu noble-security/main Icons (48x48) [13.4 kB]  
Get:13 http://security.ubuntu.com/ubuntu noble-security/main Icons (64x64) [20.0 kB]
```



**Paso 2** instalamos JDK de Java

```
ubuntu@ubuntu:~$ sudo apt-get install default-jdk
Reading package lists... 58%
```

**Paso 3** instalación del curl .

```
ubuntu@ubuntu:~$ sudo apt-get install curl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  curl
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 226 kB of archives.
After this operation, 534 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 curl amd64 8.5.0-2ubuntu10.6 [226 kB]
Fetched 226 kB in 1s (213 kB/s)
```


**Paso 4** Descargar el archivo .jar de ANTLR:

```
ubuntu@ubuntu:~$ curl -O https://www.antlr.org/download/antlr-4.13.2-complete.jar
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed
100 2089k  100 2089k    0     0  849k      0  0:00:02  0:00:02 --:--:--  849k
ubuntu@ubuntu:~$
```

**Paso 5** Mover el archivo .jar a una ruta comúnmente usada para bibliotecas compartidas

```
ubuntu@ubuntu:~$ sudo mv antlr-4.13.2-complete.jar /usr/local/lib/
ubuntu@ubuntu:~$
```

**Paso 6** Configuramos las variables de entorno:



```

ubuntu@ubuntu:~$ sudo mv antlr-4.13.2-complete.jar /usr/local/lib/
ubuntu@ubuntu:~$ echo 'export CLASSPATH=".:usr/local/lib/antlr-4.13.2-complete.jar:$CLASSPATH"' >> ~/.bashrc
ubuntu@ubuntu:~$ echo 'alias antlr4="java -jar /usr/local/lib/antlr-4.13.2-complete.jar"' >> ~/.bashrc
ubuntu@ubuntu:~$ echo 'alias grun="java org.antlr.v4.gui.TestRig"' >> ~/.bashrc
ubuntu@ubuntu:~$

```

**Paso 7** Mediante el comando antlr4 verificamos que antlr está funcionando correctamente

```

Fredycatelllon@fredycatelllon-VMware-Virtual-Platform:~$ antlr4
ANTLR Parser Generator Version 4.13.2
-o ____ specify output directory where all output is generated
-lib ____ specify location of grammars, tokens files
-atn generate rule augmented transition network diagrams
-encoding ____ specify grammar file encoding; e.g., euc-jp
-message-format ____ specify output style for messages in antlr, gnu, vs2005
-long-messages show exception details when available for errors and warnings
-listener generate parse tree listener (default)
-no-listener don't generate parse tree listener
-visitor generate parse tree visitor
-no-visitor don't generate parse tree visitor (default)
-package ____ specify a package/namespace for the generated code
-depend generate file dependencies
-D<option>=value set/override a grammar-level option
-Werror treat warnings as errors
-XdbgST launch StringTemplate visualizer on generated code
-XdbgSTWait wait for STViz to close before continuing
-Xforce-atn use the ATN simulator for all predictions
-Xlog dump lots of logging info to antlr-timestamp.log
-Xexact-output-dir all output goes into -o dir regardless of paths/package
(arg: 1) antlr4 ^C
Fredycatelllon@fredycatelllon-VMware-Virtual-Platform:~$ antlr4 |head -n 1
ANTLR Parser Generator Version 4.13.2
Fredycatelllon@fredycatelllon-VMware-Virtual-Platform:~$

```



### III. Descripción Del Lenguaje Easy

**EASY** es un lenguaje de programación de alto nivel, multiparadigma (imperativo y estructurado) diseñado con fines educativos y de prototipado rápido. Su objetivo es simplificar el proceso de aprendizaje de la programación y la construcción de compiladores, combinando la familiaridad de C con una sintaxis más amigable y moderna.

Su extensión es **.ec**

- **Sintaxis clara** y simplificada, inspirada en C.
- **Tipado estático**, pero con inferencia básica de tipos.
- **Orientado a principiantes**, pero suficientemente robusto para proyectos medianos.
- **Biblioteca estándar mínima**, enfocada en entrada/salida, matemáticas y estructuras básicas.
- **Soporte para estructuras de control clásicas** (if, while, for, switch).
- **Modularidad** mediante funciones.

Actualmente Easy va evolucionando permitiendo estructuras de programación más complejas tales como Arreglos, Funciones y Clases (POO), aunque estas estructuras no son tan robustas como los lenguajes de programación tradicionales, si cumplen con la idea de Easy de ser un lenguaje de programación de uso educativo.

Una mención importante es que Easy es un lenguaje multiplataforma, lo que significa que es compatible con Windows, MAC y Linux, en la parte de anexos de este informe técnico se incluye la evidencia de los antes mencionado.



## **Tipos de datos primitivos**

- entero: Números enteros (ej: 42, -10)
- flotante: Números con decimales (ej: 3.14, -0.5)
- booleano: Valores de verdad (representados por vdo y fso)
- Cadenas de texto: Delimitadas por comillas dobles (ej: "Hola mundo")]

## **Forma de declaración de variables**

- `tipo_dato nombre_variable;`      // Declaración sin inicialización
- `tipo_dato nombre_variable : valor;` // Declaración con inicialización
- `tipo_dato arreglo[tamaño];`      // Declaración de arreglo


## **Ejemplos:**

- `entero edad : 25;`
- `flotante precio;`
- `booleano activo : vdo;`
- `entero numeros[10];`

## **Operadores soportados**

- Asignación: `:`
- Igualdad: `::`
- Mayor que: `>`
- Menor que: `<`
- Mayor o igual que: `>=`
- Menor o igual que: `<=`
- Suma: `+`
- Resta: `-`



- 
- Multiplicación: \*
  - División: /
  - Marcador de caso: = (usado en seleccionar-caso)

## Estructuras de control de flujo

*Todas las estructuras son en español:*

### Condicional (if-else):

```
si (condición) {  
    // código  
} sino {  
    // código alternativo  
}
```

### Bucle mientras (while):

```
mientras (condición) {  
    // código a repetir  
}
```

### Estructura de selección (switch):

```
seleccionar (expresión) {  
    caso valor1 = {  
        // código  
        romper;  
    }  
    caso valor2 = {  
        // código  
        romper;  
    }  
    defecto = {  
        // código por defecto  
    }
```



romper;

}

}

### Entrada y salida estándar

- Salida: imprimir(expresión);
- Entrada: leer(variable);

## IV. Estructura Lexica

Token Léxico	Equivalente	Propósito
inicio	---	Marca el inicio de un bloque de código principal
fin	---	Marca el final de un bloque de código principal
si	if	Ejecuta un bloque de código si la condición es verdadera
sino	else	Bloque si ninguna condición previa es verdadera
mientras	while	Bucle que se ejecuta mientras la condición sea verdadera
para	for	Bucle controlado por contador
retornar	return	Devuelve un valor desde una función
leer	input / scanner	Entrada de datos
imprimir	print / system.out.print	Imprime datos en consola
seleccionar	switch	Selección entre múltiples opciones
caso	case	Opción específica dentro de seleccionar
defecto	default	Opción por defecto si ninguna coincide
romper	break	Finaliza una estructura como seleccionar o bucle
clase	class	Define una clase
super	super	Accede a métodos o atributos de la clase padre
metodo	method	Define un método dentro de una clase
funcion	function	Declaración de función



entero	int	Valor entero
flotante	float	Valor decimal
booleano	boolean	Valor verdadero o falso
cadena	string	Valor tipo string
VERDADERO	true	Valor booleano verdadero
FALSO	false	Valor booleano falso
::	==	Comparación de igualdad
>:	>=	Comparación mayor o igual que
<:	<=	Comparación menor o igual que
::!	!=	Comparación de desigualdad
:	=	Asignación
=	=	Asignación en casos
&	&&	Operador lógico AND
??		Operador lógico OR
>	>	Comparación mayor que
<	<	Comparación menor que
+	+	Operador de suma
-	-	Operador de resta
*	*	Operador de multiplicación
/	/	Operador de división
{	{	Delimita bloques de código
}	}	Cierre de bloques de código
(	(	Inicio de lista de argumentos o expresiones
)	)	Fin de lista de argumentos o expresiones
[	[	Inicio de arreglo o lista
]	]	Fin de arreglo o lista
;	;	Final de instrucción
,	,	Separación de elementos
.	.	Acceso a propiedades o métodos



## V. Reglas de estilo del lenguaje

- **Sangrías:** Se utilizan 4 espacios por nivel de indentación (no tabulaciones).
- **Posición de llaves:** Las llaves de apertura se colocan en la misma línea que la estructura de control o declaración de función, seguidas por un salto de línea. Las llaves de cierre van en una nueva línea.
- **Espaciado:** Se debe colocar un espacio después de palabras clave y antes de abrir paréntesis o llaves.
- **Operadores:** Se separan por un espacio a cada lado.
- **Punto y coma:** Todas las sentencias deben terminar con punto y coma.

### # Convenciones de nombres

- **Variables:** Notación camelCase (primera palabra en minúscula, siguientes palabras con la primera letra en mayúscula).
  - entero contadorTotal : 0;
  - flotante precioUnitario : 10.5;
- **Constantes:** Todas las letras en mayúsculas con palabras separadas por guión bajo.
  - entero MAX\_INTENTOS : 3;
- **Funciones:** Notación camelCase, con verbo que indica acción.
  - entero calcularTotal(entero a, entero b) { ... }

### # Longitud de identificadores

- Mínima: 2 caracteres (excepto variables de iteración como 'i', 'j', 'k').
- Máxima recomendada: 30 caracteres para mantener la legibilidad.
- Preferencia: Nombres descriptivos pero concisos, evitando abreviaturas poco claras.

### # Estilo de comentarios



- **Comentarios de bloque:** Delimitados por ``/# ... #/``. Se utilizan para documentación extensa o para explicar secciones de código.

```
/#  
# Esta función calcula el promedio de dos números  
# y retorna el resultado como valor flotante  
#/
```

- **Comentarios de línea:** Se utilizan dobles barras ``//`` seguidas de un espacio.

- entero contador : 0; // Inicializa el contador

- **Documentación de funciones:** Debe incluir descripción, parámetros y valor de retorno.

```
/#  
# Calcula el promedio de dos valores  
# @param a Primer valor  
# @param b Segundo valor  
# @return Promedio de los dos valores  
#/
```

# Estilo de escritura para estructuras

- Estructuras condicionales:

```
si (condición) {  
    // código  
} sino {  
    // código alternativo  
}
```



➤ **Bucles:**

```
mientras (condición) {  
    // código a repetir  
}
```

➤ **Estructura de selección:**

```
seleccionar (expresión) {  
    caso 1 = {  
        // código  
        romper;  
    }  
    caso 2 = {  
        // código  
        romper;  
    }  
    defecto = {  
        // código por defecto  
        romper;  
    }  
}
```

## VI. Ejemplos de uso del lenguaje

```
easyC1.txt
1  /*Ejemplo de operaciones aritméticas en Easy#*/
2
3  inicio {
4      /* Declaración de variables */
5      entero a : 10;
6      entero b : 5;
7      entero suma : 0;
8      entero resta : 0;
9      entero multiplicacion : 0;
10     flotante division : 0.0;
11
12     /* Operaciones aritméticas */
13     suma : a + b;
14     resta : a - b;
15     multiplicacion : a * b;
16     division : a / b;
17
18     /* Mostrar resultados */
19     imprimir("Valores iniciales: a = ");
20     imprimir(a);
21     imprimir(", b = ");
22     imprimir(b);
23     imprimir("\n");
24
25     imprimir("Suma: ");
26     imprimir(suma);
27     imprimir("\n");
28
29     imprimir("Resta: ");
30     imprimir(resta);
31     imprimir("\n");
32
33     imprimir("Multiplicacion: ");
34     imprimir(multiplicacion);
35     imprimir("\n");
36
37     imprimir("Division: ");
38     imprimir(division);
39     imprimir("\n");
40
41     /* Operación con condición */
42     si (suma > multiplicacion) {
43         imprimir("La suma es mayor que la multiplicación\n");
44     } sino {
45         imprimir("La multiplicación es mayor o igual que la suma\n");
46     }
47 } fin
```

*Ejemplo 1: Operaciones Aritméticas*





```
easyC.txt
1  /*ejemplo de codigo en Easy#*/
2
3  inicio {
4      entero numero : 5; /* definición de un entero con valor 5#/
5      flotante numero1 : 5.0; /* definición de un flotante con valor 5.0#/
6      booleano valor : vdo;
7      booleano valor2 : fso;
8
9      si (numero == numero1) {
10         imprimir("El entero y el flotante son iguales.\n");
11     } sino {
12         imprimir("El entero y el flotante son diferentes.\n");
13     }
14 } fin
15
16
```

*Ejemplo 2: Declaración de variables*

```
easyC2.txt
1  /* Programa que utiliza una estructura de selección en Easy# */
2
3  inicio {
4      /* Declaramos una variable para el día de la semana */
5      entero dia : 3;
6
7      /* Estructura de selección */
8      seleccionar (dia) {
9          caso 1 = {
10             imprimir("Lunes");
11             romper;
12         }
13         caso 2 = {
14             imprimir("Martes");
15             romper;
16         }
17         caso 3 = {
18             imprimir("Miércoles");
19             romper;
20         }
21         caso 4 = {
22             imprimir("Jueves");
23             romper;
24         }
25         caso 5 = {
26             imprimir("Viernes");
27             romper;
28         }
29         caso 6 = {
30             imprimir("Sábado");
31             romper;
32         }
33         caso 7 = {
34             imprimir("Domingo");
35             romper;
36         }
37         defecto = {
38             imprimir("Día no válido");
39             romper;
40         }
41     }
42
43     /* Mostramos un mensaje final */
44     imprimir("Fin del programa");
45 } fin
```

*Ejemplo 3: Estructura de Selección*



```

easyC3.txt
1  /* Programa para realizar una búsqueda lineal en un arreglo */
2
3  inicio {
4      /* Declaramos un arreglo de 10 elementos */
5      entero numeros[10];
6      entero i : 0;
7      entero buscar : 42;
8      entero encontrado : 0; /* 0 = falso, 1 = verdadero */
9      entero posicion : -1;
10
11     /* Inicializamos el arreglo */
12     numeros[0] : 15;
13     numeros[1] : 23;
14     numeros[2] : 8;
15     numeros[3] : 42;
16     numeros[4] : 16;
17     numeros[5] : 4;
18     numeros[6] : 89;
19     numeros[7] : 31;
20     numeros[8] : 77;
21     numeros[9] : 42;
22
23     /* Realizamos la búsqueda lineal usando ciclo mientras */
24     mientras (i < 10) {
25         si (numeros[i] == buscar) {
26             encontrado : 1;
27             posicion : i;
28             romper; /* Salimos del ciclo al encontrar el primer resultado */
29         }
30         i : i + 1;
31     }
32
33     /* Mostramos el resultado */
34     si (encontrado == 1) {
35         imprimir("Número encontrado en la posición: ");
36         imprimir(posicion);
37     } sino {
38         imprimir("Número no encontrado en el arreglo");
39     }
40 } fin

```

*Ejemplo 4: Arreglos*



```
$ easyC4.txt
1  /*# Calculadora simple en EasyC */
2
3  /*# Declaración de funciones */
4  flotante sumar(floatante a, flotante b) {
5      retornar a + b;
6  }
7
8  flotante restar(floatante a, flotante b) {
9      retornar a - b;
10 }
11
12 flotante multiplicar(floatante a, flotante b) {
13     retornar a * b;
14 }
15
16 flotante dividir(floatante a, flotante b) {
17     si (b == 0) {
18         imprimir("Error: División por cero");
19         retornar 0;
20     }
21     retornar a / b;
22 }
23
24 inicio {
25     flotante num1 : 0;
26     flotante num2 : 0;
27     entero operación : 0;
28     flotante resultado : 0;
29
30     /* Solicitar entrada del usuario */
31     imprimir("Ingrese el primer número: ");
32     leer(num1);
33
34     imprimir("Ingrese el segundo número: ");
35     leer(num2);
36
37     imprimir("Seleccione una operación:");
38     imprimir("1 - Suma");
39     imprimir("2 - Resta");
40     imprimir("3 - Multiplicación");
41     imprimir("4 - División");
42     leer(operación);
43
44     /* Realizar la operación seleccionada */
45     seleccionar (operación) {
46         caso 1 = {
47             resultado : sumar(num1, num2);
48             romper;
49         }
50         caso 2 = {
51             resultado : restar(num1, num2);
52             romper;
53         }
54         caso 3 = {
55             resultado : multiplicar(num1, num2);
56             romper;
57         }
58         caso 4 = {
59             resultado : dividir(num1, num2);
60             romper;
61         }
62         defecto = {
63             imprimir("Operación no válida");
64             romper;
65         }
66     }
67
68     /* Mostrar resultado */
69     imprimir("El resultado es: ");
70     imprimir(resultado);
71 } #fin
```

*Ejemplo 5: Calculadora Simple*

## Explicación de Reglas Sintácticas

### 1. bloque\_elemento

```
bloque_elemento
: sentencia
| funcion_definicion
;
```

Define elementos que pueden existir dentro del bloque principal del programa: una sentencia o una función.

### 2. Programa

```
programa
: (declaracion | funcion_definicion | clase_definicion | declaracion_variable_sentencia)*
| INICIO LLAVEIZQ bloque_elemento* LLAVEDER FIN
;
```

Es la regla inicial. Define la estructura general del programa:

- ✓ Permite múltiples declaraciones, definiciones de funciones o clases antes del bloque principal.
- ✓ Luego exige que el programa comience con inicio { ... } fin, y dentro de ese bloque puede haber más sentencias o funciones.

### 3. declaracion y lista\_declaraciones

```
declaracion
: tipo_dato lista_declaraciones PUNTOYCOMA
;

lista_declaraciones
: declaracion_variable (COMA declaracion_variable)*
;
```

Describe una declaración de variables:

- ✓ declaracion es un tipo de dato seguido por una o más declaraciones.
- ✓ lista\_declaraciones permite múltiples variables separadas por comas.

#### 4. funcion\_definicion

```
funcion_definicion
: FUNCION tipo_dato ID PARIZQ parametros? PARDER bloque
;
```

Declara una función usando la palabra clave `funcion`, seguida del tipo de retorno, nombre, parámetros y bloque de código.

#### 5. clase\_definicion

```
clase_definicion
: CLASE ID (IGUAL ID)? LLAVEIZQ clase_cuerpo* LLAVEDER
;
```

Define una clase con la palabra clave `clase`. Puede extender de otra clase (= ID) y contiene atributos o métodos.

#### 6. clase\_cuerpo, atributo, método

```
clase_cuerpo
: atributo
| metodo
;

atributo
: tipo_dato ID PUNTOYCOMA
| tipo_dato ID CORCHETEIZQ LITENTERO CORCHETEDER PUNTOYCOMA
;

metodo
: METODO tipo_dato? ID PARIZQ parametros? PARDER bloque
;
```

- ✓ `clase_cuerpo` puede ser un atributo o un método.
- ✓ `atributo` define una variable de clase (normal o arreglo).
- ✓ `metodo` define un método, con o sin tipo de retorno.

## 7. parametros y parámetro

```
parametros
: parametro (COMA parametro)*
;

parametro
: tipo_dato ID
;
```

Define los parámetros de una función o método, separados por comas.

## 8. tipo\_dato

```
tipo_dato
: ENTERO | FLOTANTE | BOOLEANO | CADENA | ID
;
```

Tipos válidos: entero, flotante, booleano, cadena o un identificador personalizado.

En nuestras primeras versiones del Lenguaje solo se podía definir los tipos de Datos Entero, Flotante e ID, esta actualización ya se incluye el tipo de dato Cadena y Booleano.

## 9. Sentencia

```
sentencia
: asignacion
| estructura_condicional
| estructura_seleccion
| estructura_repetitiva
| estructura_para
| sentencia_imprimir
| sentencia_lectura
| sentencia_retorno
| bloque
| llamada_metodo
| declaracion_variable_sentencia
;
```

Agrupar todos los tipos de sentencias válidas en el lenguaje: asignaciones, estructuras de control, impresiones, declaraciones, etc.

## 10. declaracion\_variable\_sentencia

```
declaracion_variable_sentencia
: tipo_dato lista_declaraciones PUNTOYCOMA
;
```

Versión de declaracion usada dentro de una sentencia (por ejemplo, en el bloque principal).

## 11. declaracion\_variable y declaracion\_variable\_simple

```
declaracion_variable
: ID
| ID IGUAL expresion
| ID CORCHETEIQ LITENTERO CORCHETEDER
| ID CORCHETEIQ LITENTERO CORCHETEDER IGUAL LLAVEIQ lista_valores LLAVEDER
| ID (['' expresion ''])? (':' expresion)?
;

declaracion_variable_simple
: tipo_dato lista_declaraciones
;
```

Formas válidas de declarar una variable, incluyendo inicialización, arreglos simples o con valores.

## 12. lista\_valores

```
lista_valores
: expresion (COMA expresion)*
;
```

Lista de valores usada para inicializar arreglos o colecciones.

### 13. asignacion y asignacion\_simple

```
asignacion
: ID IGUAL expresion PUNTOYCOMA
| ID CORCHETEIZQ expresion CORCHETEDER IGUAL expresion PUNTOYCOMA
| ID IGUAL LLAVEIZQ lista_valores LLAVEDER PUNTOYCOMA
;

asignacion_simple
: ID IGUAL expresion
| ID CORCHETEIZQ expresion CORCHETEDER IGUAL expresion
| ID IGUAL LLAVEIZQ lista_valores LLAVEDER
;
```

Asignación de valores a variables o arreglos. La versión simple no requiere punto y coma (útil en ciclos para).

### 14. Estructuras de control

```
estructura_condicional
: SI PARIZQ expresion PARDER bloque (SINO bloque)?
;

estructura_seleccion
: SELECCIONAR PARIZQ expresion PARDER LLAVEIZQ caso+ caso_defecto? LLAVEDER
;

caso
: CASO expresion IGUAL_CASO sentencia* ROMPER PUNTOYCOMA
;

caso_defecto
: DEFECTO IGUAL_CASO sentencia*
;

estructura_repetitiva
: MIENTRAS PARIZQ expresion PARDER bloque
;


estructura_para
: 'para' '(' (asignacion_simple | declaracion_variable_simple) ';' expresion ';' asignacion_simple ')' bloq
;
```

- ✓ estructura\_condicional: si (...) { ... } sino { ... }.
- ✓ estructura\_seleccion: tipo switch con seleccionar (...) { caso ... romper; ... }.
- ✓ estructura\_repetitiva: ciclo mientras.
- ✓ estructura\_para: ciclo para, con inicialización, condición y actualización.

### 15. bloque

```
bloque
: LLAVEIZQ sentencia* LLAVEDER
;
```





Un bloque { ... } de sentencias.

## 16. Sentencias especiales

```
sentencia_imprimir
: IMPRIMIR PARIZQ expresion PARDER PUNTOYCOMA
;

sentencia_lectura
: LEER PARIZQ ID PARDER PUNTOYCOMA
;

sentencia_retorno
: RETORNAR expresion? PUNTOYCOMA
;
```

- ✓ `sentencia_imprimir`: imprime una expresión.
- ✓ `sentencia_lectura`: lee una variable desde entrada.
- ✓ `sentencia_retorno`: devuelve un valor desde una función.

## 17. llamada\_metodo

```
llamada_metodo
: (ID | SUPER) PUNTO ID PARIZQ argumentos? PARDER PUNTOYCOMA
;
```

Invoca un método desde un objeto (`obj.metodo(...)`), o desde `super`.

## 18. argumentos

```
argumentos
: expresion (COMA expresion)*
;
```

Lista de expresiones pasadas a una función o método.



## 19. Expresiones

```
expresion
: expresion_logica
| expresion_lista
;

expresion_lista
: LLAVEIZQ lista_valores LLAVEDER
;

expresion_logica
: expresion_relacional
| expresion_logica Y expresion_relacional
| expresion_logica O expresion_relacional
;

expresion_relacional
: expresion_aritmetica
| expresion_aritmetica operador_relacional expresion_aritmetica
;
```

```
expresion_aritmetica
: termino
| expresion_aritmetica SUMA termino
| expresion_aritmetica RESTA termino
;

termino
: factor
| termino MULTIPLICACION factor
| termino DIVISION factor
;

factor
: LITENTERO
| LITFLOTANTE
| VERDADERO
| FALSO
| LITERALCADENA
| ID
| ID CORCHETEIZQ expresion CORCHETEDER
| llamada_funcion
| PARIZQ expresion PARDER
| RESTA factor
;
```

- ✓ expresion: puede ser lógica o una lista de valores.
- ✓ expresion\_logica: incluye operadores &, ??.
- ✓ expresion\_relacional: comparación de expresiones aritméticas.
- ✓ expresion\_aritmetica: suma y resta.
- ✓ termino: multiplicación y división.
- ✓ factor: elementos básicos (literales, variables, llamadas, paréntesis).



## 20. llamada\_funcion

```
llamada_funcion  
: ID PARIZQ argumentos? PARDER  
;
```

Invoca una función directamente por nombre: f(...

## VII. Proceso de Ejecución de Código

Para la ejecución de código Easy se ha realizado de forma sencilla mediante un script de ejecución

```
run_easy.bat
1 @echo off
2 REM ==== Configuración ====
3 set ANTLR_JAR=antlr-4.13.1-complete.jar
4 set SRC_DIR=.
5 set OUTPUT_CPP=programa.cpp
6 set OUTPUT_EXE=programa.exe
7
8 REM ==== Verificar argumentos ====
9 if "%~1"==" " (
10     echo Uso: run_easy archivo.ec
11     exit /b 1
12 )
13
14 REM ==== Ejecutar traductor ====
15 echo [1/3] Generando C++ desde %~1 ...
16 java -cp "%ANTLR_JAR;%SRC_DIR%" Main %~1
17
18 REM ==== Compilar C++ ====
19 echo [2/3] Compilando %OUTPUT_CPP% ...
20 g++ "%OUTPUT_CPP%" -o "%OUTPUT_EXE%"
21 if errorlevel 1 (
22     echo Error de compilación.
23     exit /b 1
24 )
25
26 REM ==== Ejecutar programa ====
27 echo [3/3] Ejecutando %OUTPUT_EXE% ...
28 "%OUTPUT_EXE%"
29
```

1 Script de ejecución para Windows

```
run_easy.sh
$ run-easy.sh
1 #!/bin/bash
2
3 # ==== Configuración ====
4 ANTLR_JAR="antlr-4.13.1-complete.jar"
5 SRC_DIR="."
6 OUTPUT_CPP="programa.cpp"
7 OUTPUT_EXE="programa.out"
8
9 # ==== Verificar argumentos ====
10 if [ -z "$1" ]; then
11     echo "Uso: run_easy archivo.ec"
12     exit 1
13 fi
14
15 # ==== Ejecutar traductor ====
16 echo "[1/3] Generando C++ desde $1 ..."
17 java -cp "$ANTLR_JAR:$SRC_DIR" Main "$1"
18
19 # ==== Compilar C++ ====
20 echo "[2/3] Compilando $OUTPUT_CPP ..."
21 g++ "$OUTPUT_CPP" -o "$OUTPUT_EXE"
22 if [ $? -ne 0 ]; then
23     echo "Error de compilación."
24     exit 1
25 fi
26
27 # ==== Ejecutar programa ====
28 echo "[3/3] Ejecutando $OUTPUT_EXE ..."
29 ./"$OUTPUT_EXE"
30
```

2- Script de ejecución en Linux / macOS



## VIII. ANEXOS



*Ilustración 3: Imagen Compilador EasyC*



*Ilustración 4: Imagen Lenguaje Easy*