Tecnológico Nacional de México

Instituto Tecnológico de Veracruz

EJ24 7J1A - Lenguajes y Autómatas 2

Docente: Ofelia Gutiérrez Giraldi

Proyecto: Compilador ELL (EasyLearningLenguage)

Equipo 1 - Integrantes:

Honorio Acosta Ruiz

Laura Espejo Alvarado

Rodrigo Ernesto Loyo García

H. Veracruz, Ver., 12 de marzo de 2024

Contenido

ELL (EasyLearningLenguage)

Propósito

EasyLearningLenguage (ELL) se propone a ser un lenguaje de programación tipado y de propósito general que será diseñado para permitir el desarrollo de aplicaciones básicas.

Se centrará en la lógica de programación básica, lo que lo hará adecuado para la enseñanza de conceptos fundamentales como variables, operaciones, condicionales y ciclos. Aunque ELL no tendrá una amplia gama de características avanzadas, será diseñado para ser fácilmente escalable a medida que los usuarios adquieren más habilidades en programación.

ELL será creado con el propósito de ser un lenguaje de programación en español de nivel básico.

Origen del nombre ELL: EasyLearningLenguage

Figura 1 - Icono del lenguaje

El nombre de nuestro lenguaje es ELL, debido a que es un lenguaje de programación orientado a facilitar el aprendizaje de la lógica básica de programación para programadores novatos.

El nombre ELL proviene de las siglas EasyLearningLanguage, lo que en español significa Lenguaje Fácil de Aprender.

Manual de Usuario

El respectivo manual busca proporcionar una guía clara y completa para nuestro lenguaje que permita a los programadores utilizar el lenguaje de manera efectiva, donde como objetivos tiene los siguientes:

- Facilitar el aprendizaje del lenguaje:

  El manual proporciona una introducción clara al lenguaje de programación, explicando los conceptos básicos, sintaxis y características únicas del lenguaje.

- Ayudar a los programadores a detectar y corregir errores:

  Incluye información sobre los errores que pueden ocurrir durante la programación con el lenguaje, incluyendo como corregir estos errores.

- Proporcionar información de referencia:

  Es una fuente completa de información de referencia para el lenguaje, que incluye una lista detallada de las palabras clave, operadores y tipos de datos; permitiendo a los programadores buscar rápidamente información específica cuando la necesita.

Conceptos Básicos

Tipo de datos

Cuando escribimos programas en el lenguaje ELL, necesitamos utilizar diferentes tipos de datos para almacenar información. Los que podemos usar en ELL son los siguientes:

Entero: Este tipo de dato representa un número entero.

Flotante: El tipo de dato "flotante" se utiliza para representar números con decimales.

Cadena: La cadena es un tipo de dato que se utiliza para representar texto. Podemos almacenar palabras, frases o cualquier combinación de caracteres en una cadena.

Carácter: El tipo de dato "carácter" se utiliza para representar símbolos individuales, como letras, números o símbolos especiales.

Booleano: El tipo de dato "booleano" se utiliza para representar valores de verdadero o falso. Es útil cuando necesitamos evaluar condiciones o tomar decisiones en nuestro programa.

Palabras Clave

En el lenguaje ELL, existen palabras clave especiales que tienen un significado especial y no pueden ser usadas como nombre de variables u otros identificadores en el código. A continuación, se muestra una lista de ellas con su respectiva descripción:

Inicio: Esta palabra clave se utiliza para indicar el comienzo del programa. Marca el punto de partida desde donde se ejecutarán las instrucciones.

Fin: Esta palabra clave indica el final del programa. Marca el punto en el que el programa termina su ejecución.

Establecer: Con la palabra clave "Establecer", podemos definir una variable en el programa. Una variable es como una caja donde podemos guardar diferentes tipos de información, como números o texto.

Interpretar: La palabra clave "Interpretar" se utiliza para leer el valor de una variable. Podemos obtener información del usuario o de otra parte del programa mediante esta instrucción.

Escribir: La palabra clave "Escribir" se utiliza para mostrar en la pantalla el valor de una variable o un texto específico. Con esta instrucción, podemos imprimir información para que el usuario la vea.

Falso y **Verdadero**: Estas palabras clave representan valores lógicos. "Falso" se utiliza para indicar algo que es incorrecto o no se cumple, mientras que "Verdadero" indica algo que es correcto o se cumple.

Para: La palabra clave "Para" marca el inicio de un ciclo repetitivo llamado bucle "Para". Nos permite ejecutar un bloque de código varias veces, siguiendo una condición y un incremento específicos.

Hasta que: La palabra clave "Hasta que" se utiliza en conjunto con el bucle "Para" y establece la condición para terminar el ciclo. Indica la condición que se evalúa antes de cada iteración.

Con incremento: La palabra clave "Con incremento" se utiliza en el bucle "Para" para indicar el incremento que se aplicará en cada iteración del ciclo. Por defecto, el incremento es 1, lo que significa que la variable de control del bucle aumentará en 1 después de cada iteración. Sin embargo, con la palabra clave "Con incremento", puedes especificar un valor diferente para el incremento, lo que te permite controlar cómo cambia la variable de control en cada repetición del bucle.

FinPara: Esta palabra clave marca el final del bucle "Para". Especifica el punto donde termina la repetición del bloque de código.

Mientras: La palabra clave "Mientras" indica el inicio de un bucle repetitivo llamado bucle "Mientras". Nos permite ejecutar un bloque de código siempre que se cumpla una condición específica.

FinMientras: La palabra clave "FinMientras" marca el final del bucle "Mientras". Indica el punto donde termina la repetición del bloque de código.

Repetir: La palabra clave "Repetir" inicia un bucle repetitivo llamado bucle "Repetir". Nos permite ejecutar un bloque de código al menos una vez y luego repetirlo mientras se cumpla una condición específica.

Finaliza cuando: La palabra clave "Finaliza cuando" se utiliza en conjunto con el bucle "Repetir" para especificar la condición de salida del bucle. Indica la condición que se evalúa después de cada iteración.

Si: La palabra clave "Si" marca el inicio de una estructura condicional llamada "Si". Permite evaluar una condición y ejecutar un bloque de código si la condición es verdadera.

Entonces: La palabra clave "Entonces" se utiliza después de una condición en una estructura condicional "Si" para indicar el bloque de código que se ejecutará si la condición es verdadera.

Sino: La palabra clave "Sino" se utiliza en una estructura condicional "Si" para indicar un bloque de código alternativo que se ejecutará si la condición es falsa.

FinSi: La palabra clave "FinSi" marca el final de la estructura condicional "Si". Indica el punto donde termina el bloque de código condicional.

Conforme: La palabra clave "Conforme" marca el inicio de una estructura condicional llamada "Conforme". Permite evaluar múltiples casos y ejecutar un bloque de código según el caso correspondiente.

Hacer: La palabra clave "Hacer" se utiliza después de un caso en una estructura condicional "Conforme" para indicar el bloque de código que se ejecutará si se cumple ese caso.

Caso: La palabra clave "Caso" se utiliza en una estructura condicional "Conforme" para indicar un caso específico que se evalúa.

En otro caso: La palabra clave "En otro caso" se utiliza en una estructura condicional "Conforme" como un caso por defecto. Indica el bloque de código que se ejecutará si ninguno de los otros casos se cumple.

FinConforme: La palabra clave "FinConforme" marca el final de la estructura condicional "Conforme". Indica el punto donde termina el bloque de código condicional.

Funcion: La palabra clave “Funcion” se utiliza para declarar una función.

FinFuncion: La palabra clave “FinFuncion” indica el término de una función.

Retornar: La palabra clave “Retornar” es para que una función retorne un valor.

Operadores

Existen operadores que nos permiten realizar diferentes tipos de operaciones en el código. Los operadores son símbolos especiales que se utilizan para realizar cálculos, comparaciones y manipulaciones de datos.

Operadores de asignación: Estos operadores se utilizan para asignar valores a variables.

=	Operador que indica asignación.

Operadores aritméticos: Estos operadores se utilizan para realizar cálculos matemáticos.

+	Operador que indica una suma.
-	Operador que indica una resta.
*	Operador que indica una multiplicación.
/	Operador que indica una división.
%	Operador que indica al módulo.

Operadores de comparación: Estos operadores se utilizan para comparar valores y evaluar si una condición es verdadera o falsa.

==	Operador de comparación Igual que.
!=	Operador de comparación diferente.
>	Operador de comparación mayor que.
<	Operador de comparación menor que.
>=	Operador de comparación mayor o igual que.
<=	Operador de comparación menor o igual que.

Operadores lógicos: Estos operadores se utilizan para realizar operaciones lógicas y combinar condiciones.

&&	Operador lógico AND.
||	Operador lógico OR.
!	Operador lógico NOT.

Nombres de las variables en ELL

Cuando necesitemos asignar un nombre a una variable en ELL, debemos seguir ciertas normas. Estas normas nos ayudarán a escribir nombres de variables que sean claros, legibles y no entren en conflicto con las palabras reservadas del lenguaje. Aquí están las reglas para nombrar variables en ELL:

- El primer carácter del nombre de la variable debe ser una letra (mayúscula o minúscula) o un guion bajo (_).
- Después del primer carácter, se pueden utilizar números, letras o guiones bajos en el nombre de la variable.
- Es recomendable que los nombres de las variables sean legibles y descriptivos, de manera que podamos entender su significado al leerlos. Por ejemplo, en lugar de utilizar acrónimos o abreviaturas que no sean claros, es mejor utilizar nombres que se auto-documenten.
- Es importante tener en cuenta que los nombres de las variables no pueden coincidir con las palabras reservadas del lenguaje. Las palabras reservadas son palabras que tienen un significado especial en el lenguaje y se utilizan para funciones específicas. Algunos ejemplos de palabras reservadas en ELL son "Inicio", "Fin", "Para", "Si", entre otras.

Literales en ELL

Los valores literales son aquellos que podemos asignar a las variables. Dependiendo del tipo de variables podremos asignar valores u otros.

Literales de enteros

El entero que podemos utilizar será *Entero*. Los literales que le asignemos siempre será un numero entero.

```
Establecer Entero variableEnt = 10;
```

Literales de decimales

El tipo de dato decimal que podemos manejar son el *Flotante*. Para este caso la representación del literal de decimales será con separación de un punto entre la parte entera y la parte decimal.

```
Establecer Flotante variableFlo = 10.5;
```

Literales de caracteres y cadenas

El tipo de dato carácter que podemos manejar es el *Caracter* y el tipo de dato cadena que podemos manejar es el *Cadena*.

Para los caracteres utilizaremos comillas simples para delimitarlos, mientras que para las cadenas utilizaremos comillas dobles.

```
Establecer Caracter variableCar = ‘a’;

Establecer Cadena variableCad = “cadena”;
```

Literales de booleanos

Para los booleanos se maneja el tipo de dato *Booleano*. Las literales que se le asigna siempre será Verdadero o Falso.

```
Establecer Booleano variableBolV = Verdadero;
```

Sintaxis

Indicar el comienzo y fin del programa

Con la siguiente estructura indicamos el inicio y el final de nuestro código.

```
Inicio
      #Bloque de sentencias
Fin
```

Declaración de variables

Una variable es como una caja donde puedes guardar cosas, como números o palabras, y luego usarlas más tarde en el programa.

Para declarar una variable seguimos la siguiente estructura, toma en cuenta los tipos de datos (*Tipo de datos*) que tiene el lenguaje ELL y el cómo deben nombrarse (*Nombres de las variables en ELL*). Es importante no olvidar nuestro delimitador al final de la sentencia (;).

```
Establecer [Tipo_Dato] [Nombre_Variable];
```

Asignación de variables

Una vez que hemos declarado una variable, podemos asignarle un valor dependiendo del tipo de dato que sea.

```
[Variable] [Operador Asignación] [ Valor ];
```

Al igual lo podemos hacer directamente al declarar nuestra variable.

```
Establecer [ Tipo_Dato ] [ Nombre_Variable ] [Operador Asignación] [ Valor ];
```

Lectura de variables

Se usa para leer el valor de una variable ingresada por el usuario.

```
Interpretar [ Variable ];
```

Imprimir

Imprimir se refiere a mostrar información; es una forma fácil de visualizar resultados, mensajes o datos. Para imprimir definimos la siguiente estructura:

```
Escribir “Cadena”;

Escribir [ Variable ];
```

Solo se puede hacer una acción a la vez, es decir, si vas a imprimir una cadena solamente será la cadena, pero no puedes imprimir una cadena y una variable juntas. Las cadenas de texto van entre comillas dobles.

Matrices en ELL

Una matriz es como una cuadricula donde puedes organizar datos en filas y columnas. Para declarar una matriz se sigue la siguiente estructura, donde la longitud es un numero entero y puedes tener 2 o más longitudes:

```
Establecer [ Tipo_Dato ] [ LONGITUD ] [ LONGITUD2 ] [ Variable ];
```

Arreglos en ELL

Un arreglo es como una lista ordenada de elementos del mismo tipo, donde cada elemento tiene una posición única. Lo podemos ver como una fila de casillas donde puede guardar información y cada casilla tiene su propio número para identificarla. Para declarar un arreglo se sigue la siguiente estructura:

```
Establecer [ Tipo_Dato ] [ LONGITUD ] [ Variable ];
```

Funciones en ELL

Una función es un bloque de código reutilizable que realiza una acción especifica cuando se llama desde otra parte del programa, al igual podemos conocerlo como un método.

Es importante aclarar que las funciones solo pueden ser declaras antes del Inicio. Para declarar una función se sigue la siguiente estructura:

```
Funcion [ Tipo_Dato ] [ Nombre_Método ]  ( [ Argumentos ] )

      #Bloque de sentencias

      Retornar [ Expresión ];

FinFuncion [ Nombre_Método ]
```

Un método vacío, es una función que no devuelve ningún valor cuando es invocado, comúnmente usados para imprimir mensajes en consola o cualquier tipo de acción sin

necesidad de devolver un valor. Para declarar un método vacío solo no hay que agregar el tipo de dato del método y excluir el retornar.

Para llamar a una función se usa la siguiente estructura, donde si no se tiene argumentos se dejaría solo los paréntesis.

```
[ Nombre_Método ] ( ARGUMENTOS );
```

Sentencias decisión en ELL

Las sentencias de decisión son sentencias que nos permiten tomar una decisión para poder ejecutar un bloque de sentencias u otro. Las sentencias de decisión que contiene el lenguaje son:

Si – Entonces – Sino

La estructura de la sentencia Si-Entonces-Sino es:

```
Si [ condición ] Entonces
      #Bloque de sentencias
Sino
      #Bloque de sentencias
FinSi
```

La parte del Sino no tiene por qué existir. En este caso tendríamos una sentencia Si-Entonces.

```
Si [condición] Entonces
      #Bloque de sentencias
FinSi
```

La sentencia Si-Entonces-Sino pueden estar anidadas y así nos encontraríamos con una sentencia Si-Entonces-SinoSi, la cual tendría la siguiente estructura:

```
Si [condición] Entonces
      #Bloque de sentencias
Sino Si [condición] Entonces
```

```
            #Bloque de sentencias
      Sino
            #Bloque de sentencias
      FinSi
FinSi
```

Conforme

Para los casos en los que se tienen muchas ramas o caminos de ejecución en una sentencia Si tenemos la sentencia Conforme. La sentencia Conforme evalúa una expresión y ejecutara el bloque de sentencias que coincida con el valor de la expresión.

El valor de la expresión puede ser numérico o al igual se pueden utilizar expresiones cuya evaluación sean cadenas.

La estructura de la sentencia Conforme es:

```
Conforme [condición] Hacer
      Caso [valor1]:
            #Bloque de sentencias
      Caso [valor2]:
            #Bloque de sentencias
      En Otro Caso:
            #Bloque de sentencias
FinConforme
```

Sentencias Bucle en ELL

Las sentencias de bucle nos van a permitir ejecutar un bloque de sentencias tantas veces como queramos, o tantas veces como se cumpla una condición. Las sentencias de bucle que contiene el lenguaje son:

Para

La estructura del bucle Para es:

```
Para [sentencia_inicio] Hasta que [condición] Con incremento [valor_entero_ó_decimal]
      #Bloque de sentencias
FinPara
```

Las funcionalidades en las que podemos utilizar la sentencia Para puede ser como un contador.

Mientras

La estructura repetitiva Mientras realiza una primera evaluación antes de ejecutar el bloque. Si la expresión es verdadera pasa a ejecutar de forma repetitiva el bloque de sentencias.

La estructura de la sentencia Mientras es la siguiente:

```
Mientras [condición] Hacer
      #Bloque de sentencias
FinMientras
```

Repetir

La estructura de la sentencia Repetir es la siguiente:

```
Repetir
      #Bloque de sentencias
Finalizar cuando [condición];
```

Tabla de errores

Número de error	Tipo	Ubicación	Token	Descripción	Solución.
1	Error léxico	Línea 3, columna 6	,	Símbolo ‘,’ desconocido	Remover símbolo ‘,’
2	Error léxico	Línea 5, columna 6	17.16.3	Se encontró un numero invalido	Remover o revisar numero
2	Error sintáctico	Línea 3, Columna 24	;	Se encontró símbolo “;”	Se esperaba uno de los siguientes: - <CADENA_TEXTO> - <CARÁCTER_TEXTO> - <NUMERO_ENTERO> - <NUMERO_DECIMAL> - “Falso” - “Verdadero” - “!” - “(“ - <VARIABLE>
3	Error sintáctico	Línea 13, columna 32	Mientras	Se encontró el símbolo “Mientras”	Se esperaba uno de los siguientes: - “.” - “+” - “-” - “*” - “/” - “%” - “| |” - “==” - “!=” - “>” - “<” - “>=” - “<=”
4	Error sintáctico	Línea 19, columna 9	<VARIABLE>	Se encontró el símbolo <VARIABLE>	Se esperaba uno de los siguientes: - <CADENA_TEXTO> - <CARÁCTER_TEXTO> - <NUMERO_ENTERO> - <NUMERO_DECIMAL> - “Falso” - “Verdadero”
5	Error sintáctico	Línea 1, columna 1	Fin	Se encontró el símbolo “Fin”	Se esperaba uno de los siguientes: - <COMENTARIO> - “Interpretar” - “Escribir” - ''Establecer'' - ''Para''

					- "Mientras" - "Repetir" - "Si" - "Conforme" - <VARIABLE>
6	Error semántico	Línea 15, columna 4	b	El símbolo “b” ya está declarado en este ámbito	Quitar declaración doble
7	Error semántico	Línea 12, columna 3	mostrar	La función “mostrar” no puede retornar ningún valor	Quitar retornar
8	Error semántico	Línea 4, columna 9	salario	La función “salario” no puede retornar un tipo de dato “Entero”	Retornar tipo de dato “Flotante”
9	Error semántico	Línea 8, columna 4		La función debe terminar con el mismo nombre	Colocar el nombre la función después del “FinFunsion”
10	Error semántico	Línea 4, columna 2	cal1	El símbolo “cal1” no está declarado en este ámbito	Declarar el símbolo
11	Error semántico	Lina 1, columna 7		El símbolo no es una variable	No se le puede asignar un valor a una llamada de función

Manual del Sistema

El respectivo manual busca proporcionar una guía clara y completa para nuestro lenguaje que permita a los programadores utilizar el lenguaje de manera efectiva y dar información clara del lenguaje, donde como objetivos tiene los siguientes:

- Describir el sistema: El manual de sistema tiene como objetivo principal proporcionar una descripción completa y detallada del sistema.
- Orientar en la solución de problemas: El manual debe proporcionar información sobre la solución de problemas comunes que los usuarios pueden enfrentar al utilizar el sistema.
- Servir como referencia: El manual de sistema debe ser una referencia útil para los usuarios, brindando información detallada sobre todas las funciones, configuraciones y aspectos técnicos del sistema.

Requisitos del sistema

- Tener Java instalado.
- Tener JavaCC instalado.

Descripción del sistema

EC (EasyCompiler) es un compilador de código abierto que admite el lenguaje de programación ELL (EasyLearningLenguage). El compilador se ejecuta en cualquier plataforma que tenga instalada Java.

Análisis Léxico

El análisis léxico es la primera etapa del proceso de compilación donde se analiza la entrada de texto del programa fuente y se divide en unidades léxicas significativas llamadas “tokens”. Estos tokens incluyen palabras clave, identificadores, operadores, contantes y otros elementos del lenguaje.

Tabla de valores

Valor	Tipo
Inicio	PR_Arranque_Programa
Fin	PR_Cierre_Programa
#	Simbolo_Comentario

edad	Variable
Establecer	PR_Definicion_Variable
Entero	PR_Tipo_De_Dato
Flotante	PR_Tipo_De_Dato
Cadena	PR_Tipo_De_Dato
Caracter	PR_Tipo_De_Dato
Booleano	PR_Tipo_De_Dato
;	Delimitador
Interpretar	PR_Lectura
Escribir	PR_Escritura
123	Numero_Entero
123.123	Numero_Flotante
"Cadena"	Cadena_Texto
'C'	Caracter_Texto
=	Operador_Asignacion
+	Operador_Suma
-	Operador_Resta
*	Operador_Mult
/	Operador_Div
%	Operador_Mod
Falso	PR_Booleano_Falso
Verdadero	PR_Booleano_Verdadero
&&	Operador_Logico_AND
||	Operador_Logico_OR
!	Operador_Logico_NOT
==	Operador_igualQue
!=	Operador_diferente
>	Operador_mayorQue
<	Operador_menorQue
>=	Operador_mayorIgualQue
<=	Operador_menorIgualQue
Para	PR_Ciclo_Para
Hasta que	PR_Condicion_Ciclo_Para

Con incremento	PR_Incremento_Ciclo_Para
FinPara	PR_Fin_Ciclo_Para
Mientras	PR_Ciclo_Mientras
FinMientras	PR_Fin_Ciclo_Mientras
Repetir	PR_Ciclo_Repetir
Finaliza cuando	PR_Condicion_Ciclo_Repetir
Si	PR_Condicional_Si
Entonces	PR_Entonces
Sino	PR_Conficional_Sino
FinSi	PR_Fin_Condicional_Si
Conforme	PR_Condicional_Conforme
Hacer	PR_Hacer
Caso	PR_Conforme_Caso
:	Operador_Dos_Puntos
En otro caso	PR_Conforme_Caso_Predeterminado
FinConforme	PR_Fin_Condicional_Conforme
(	Parentesis_Abierto
)	Parentesis_Cerrado
[	Indice_Abierto
]	Indice_Cerrado
,	Separador_Parametros
Funcion	PR_Declarar_Funcion
FinFuncion	PR_Fin_Funcion
Retornar	PR_Retornar

Tabla 3 – Tabla de valores

Expresiones regulares

Número entero:$(\text{número})^+$

Número flotante: $(número)^+.(número)^+$

Comentario: $\#(\text{letra}|\ \text{símbolo}\ |\ \text{número})^*$

Variable: $(\text{letra}\ |\ \_)(\text{número}\ |\ \text{letra}\ |\ \_)^*$

Una cadena: $"(\text{letra}\ |\ \text{número}\ |\ \text{símbolo})^*\ "$

Un carácter: $'(\text{letra}\ |\ \text{número}\ |\text{símbolo}\ |\ \epsilon)\ '$

Análisis Sintáctico

El análisis sintáctico es la segunda etapa del proceso de compilación donde se verifica la estructura gramatical del programa fuente utilizando una gramática formal. Durante esta etapa se aplican reglas gramaticales para reconocer las construcciones sintácticas del lenguaje, como declaraciones, expresiones y sentencias.

Sintaxis

Programa principal en ELL

Un programa en ELL siempre tiene que iniciar con la palabra reservada “Inicio” y terminar con la palabra reservada “Fin”, además entre estas dos palabras debe de ir al menos una sentencia.

$$\text{PROGRAMA} \rightarrow \text{Inicio} \ (\ \text{SENTENCIA} \ )^{+} \ \text{Fin}$$

Declaración de variables en ELL

La declaración de una variable se considera una sentencia, la cual siempre tiene que iniciar con la palabra reservada “Establecer” seguida de un tipo de dato y luego un nombre para la variable, opcionalmente puede estar el operador de asignación y el valor el cual tomara la variable, y finalmente un punto y coma.

$$\text{DECLARACION\_VARIABLE} \rightarrow \text{Establecer TIPO\_DATO VARIABLE} \ (= \ \text{VALOR})? \ ;$$

Asignación de variables en ELL

La asignación de variables es también una sentencia, la cual tiene que iniciar con un nombre de variable, seguido de un operador de asignación, luego un valor y finalmente un punto y coma.

$$\text{ASIGNACION\_VARIABLE} \rightarrow \text{VARIABLE OPERADOR\_ASIGNACION VALOR} \ ;$$

Leer del teclado en ELL

La sentencia de leer un dato empieza por la palabra reservada “Interpretar” seguida de un nombre de variable y finalmente un punto y coma.

$$\text{LEER\_DATO} \rightarrow \text{Interpretar VARIABLE} \ ;$$

Imprimir en pantalla en ELL

La sentencia de imprimir en pantalla debe iniciar con la palabra reservada “Escribir” seguida por algún valor y luego un punto y coma.

IMPRIMIR_DATO → Escribir VALOR ;

Matrices en ELL

Para declarar una matriz, debe iniciar con la palabra reservada “Establecer” seguida del tipo de dato y la longitud entera por cada dimensión de la matriz, al final ira el nombre que se le asignara a la matriz.

MATRIZ → Establecer TIPO_DATO [ENTERO] ([ENTERO])$^{+}$ VARIABLE ;

Arreglos en ELL

Para declarar un arreglo, debe iniciar con la palabra reservada “Establecer” seguida del tipo de dato y la longitud entera del arreglo, al final ira el nombre que se le asignara al arreglo.

ARREGLO → Establecer TIPO_DATO [ENTERO] VARIABLE ;

Funciones en ELL

Para declarar una función, debe iniciar con la palabra reservada “Funcion”, seguida del tipo de dato que devolverá la función, es importante aclarar que las funciones solo pueden ser declaras antes del Inicio. Si no se coloca el tipo de dato se tomará como una función vacía. Seguido del tipo de dato ira el nombre del método y luego, entre paréntesis, los parámetros. Después, puede o no seguir una sentencia y, finalmente la palabra reservada “FinFuncion” junto al nombre del método. Si la función no es vacía, entonces al final de la sentencia debe estar la palabra reservada “Retornar” para devolver.

DECLARAR_FUNCION → Funcion (TIPO_DATO)? IDENT. ARGUMENTOS

(SENTENCIA)* (Retornar EXPRESION ;)? FinFuncion IDENT.

Para llamar a una función se debe escribir el nombre de la función seguido de sus argumentos entre paréntesis, de no ser así, solo colocar los paréntesis para su identificación.

LLAMAR_FUNCION → IDENT. PAREN_ABIERTO (ARGS.)* PAREN_CERRADO ;

Sentencia Si en ELL

Una sentencia “Si” debe de iniciar con la palabra reservada “Si” seguida de una condición, luego de la palabra reservada “Entonces”, luego debe de haber por lo menos una sentencia, opcionalmente puede estar la palabra reservada “Sino” seguida de al menos una sentencia, y finalmente debe terminar con la palabra reservada “FinSi”.

$$\text{SENTENCIA\_SI} \rightarrow \text{Si CONDICION Entonces } (\text{ SENTENCIA })^{+}$$

$$(\text{ Sino } (\text{ SENTENCIA })^{+})? \text{ FinSi}$$

Sentencia Conforme en ELL

Una sentencia “Conforme” debe de iniciar con la palabra reservada “Conforme” seguida de un nombre de variable, luego de la palabra reservada “Hacer”, después debe de por lo menos una palabra reservada “Caso” seguida de una constante, luego dos puntos y al menos una sentencia, opcionalmente puede estar la palabra reservada “En otro caso” seguida de dos puntos y por lo menos una sentencia, finalmente la palabra reservada “FinConforme”.

$$\text{SENTENCIA\_CONFORME} \rightarrow \text{Conforme VARIABLE Hacer}$$

$$(\text{ Caso CONSTANTES : } (\text{ SENTENCIA })^{+})^{+}$$

$$(\text{ En otro caso : } (\text{ SENTENCIA })^{+})? \text{ FinConforme}$$

Sentencia Mientras en ELL

Una sentencia “Mientras” debe iniciar con la palabra reservada “Mientras” seguida de una condición y luego la palabra reservada “Hacer”, debe seguir por lo menos una sentencia y finalmente la palabra reservada “FinMientras”.

$$\text{SENTENCIA\_MIENTRAS} \rightarrow \text{Mientras CONDICION Hacer } (\text{ SENTENCIA })^{+} \text{ FinMientras}$$

Sentencia Repetir en ELL

Una sentencia “Repetir” inicia con la palabra reservada “Repetir” seguida de por lo menos una sentencia, luego debe de ir la palabra reservada “Finaliza cunado” seguida de una condición y finalmente un punto y coma.

$$\text{SENTENCIA\_REPETIR} \rightarrow \text{Repetir } (\text{ SENTENCIA })^{+} \text{ Finaliza cuando CONDICION ;}$$

Sentencia Para en ELL

Una sentencia “Para” inicia por la palabra reservada “Para” seguida de una asignación de variable o de una declaración de variable, luego de la palabra reservada “Hasta que” seguida de una condición, después la palabra reservada “Con incremento” seguida de un número entero o decimal, luego debe haber al menos una sentencia y finalmente la palabra reservada “FinPara”.

SENTENCIA_PARA → Para (ASIGNACION_VARIABLE | DECLARACION_VARIABLE)
Hasta que CONDICION Con incremento (NUMERO_ENTERO | NUMERO_DECIMAL)
(SENTENCIA)$^{+}$ FinPara

Gramáticas del Lenguaje

⟨PROGRAMA⟩ → (⟨DECLARACION_FUNCION⟩)* Inicio (⟨SENTENCIAS⟩)$^{+}$ Fin

⟨CONSTANTES⟩ → NUMERO_ENTERO | NUMERO_DECIMAL | CADENA_TEXTO |
CARACTER_TEXTO | BOOLEANO_FALSO | BOOLEANO_VERDADERO
⟨TIPO_DATO⟩ → ENTERO | FLOTANTE | CADENA | CARACTER | BOOLEANO

⟨EXPRESION⟩ → ⟨LOGICO_OR⟩
⟨LOGICO_OR⟩ → ⟨LOGICO_AND⟩ (LOGICO_OR ⟨LOGICO_AND⟩)*
⟨LOGICO_AND⟩ → ⟨COMPARACION⟩ (LOGICO_AND ⟨COMPARACION⟩)*
⟨COMPARACION⟩ → ⟨OPERANDO⟩ ((OPERADOR_IGUAL | OPERADOR_DIFERENTE |
OPERADOR_MENOR_IGUAL | OPERADOR_MENOR | OPERADOR_MAYOR) ⟨OPERANDO⟩)?
⟨OPERANDO⟩ → ⟨TERMINO⟩ ((SUMA | RESTA) ⟨TERMINO⟩)*
⟨TERMINO⟩ → ⟨FACTOR⟩ ((MULTIPLICACION| DIVISION | MODULO) ⟨FACTOR⟩)*
⟨LLAMADAS⟩ → VARIABLE (⟨INDICE_ARREGLO⟩ | PAREN_ABIERTO
(⟨ARGUMENTOS⟩)? PAREN_CERRADO)?
⟨FACTOR⟩ → ⟨LLAMADAS⟩
⟨FACTOR⟩ → ⟨CONSTANTES⟩
⟨FACTOR⟩ → PAREN_ABIERTO ⟨EXPRESION⟩ PAREN_CERRADO
⟨FACTOR⟩ → LOGICO_NOT ⟨FACTOR⟩

⟨FACTOR⟩ → (SUMA | RESTA) ⟨FACTOR⟩

⟨INIDICE_ARREGLO⟩ → (INDICE_ABIERTO ⟨EXPRESION⟩ INDICE_CERRADO)$^{+}$

⟨DECLARACION⟩ → ESTABLECER ⟨TIPO_DATO⟩ (⟨INIDICE_ARREGLO⟩)? VARIABLE (⟨ASIGNACION⟩)?

⟨ASIGNACION⟩ → ASIGNACION ⟨EXPRESION⟩

⟨DECLARACION_ARGUMENTOS⟩ → ⟨TIPO_DATO⟩ VARIABLE (COMA ⟨TIPO_DATO⟩ VARIABLE)*

⟨SENTENCIAS⟩ → ⟨SENTENCIA_ASIGNACION⟩

⟨SENTENCIAS⟩ → ⟨SENTENCIA_DECLARACION ⟩

⟨SENTENCIAS⟩ → ⟨LEER_DATO⟩

⟨SENTENCIAS⟩ → ⟨IMPRIMIR_DATO⟩

⟨SENTENCIAS⟩ → ⟨SENTENCIA_SI⟩

⟨SENTENCIAS⟩ → ⟨SENTENCIA_CONFORME⟩

⟨SENTENCIAS⟩ → ⟨SENTENCIA_PARA⟩

⟨SENTENCIAS⟩ → ⟨SENTENCIA_REPETIR⟩

⟨SENTENCIAS⟩ → ⟨SENTENCIA_MIENTRAS⟩

⟨SENTENCIA_DECLARACION⟩ → ⟨DECLARACION⟩ DELIMITADOR

⟨SENTENCIA_ASIGNACION⟩ → VARIABLE (⟨ASIGNACION⟩)? DELIMITADOR

⟨LEER_DATO⟩ → INTERPRETAR VARIABLE DELIMITADOR

⟨IMPRIMIR_DATO⟩ → ESCRIBIR ⟨EXPRESION⟩ DELIMITADOR

⟨SENTENCIA_SI⟩ → INICIO_CONDICIONAL_SI ⟨EXPRESION⟩ ENTONCES (⟨SENTENCIAS⟩)* (CONDICIONAL_SINO (⟨SENTENCIAS⟩)*)? FIN_CONDICIONAL_SI

⟨SENTENCIA_CONFORME⟩ → INICIO_CONDICIONAL_CONFORME VARIABLE HACER
(CASO ⟨CONSTANTES⟩ OPERADOR_DOS_PUNTOS (⟨SENTENCIAS⟩)*)+
(CASO_PREDETERMINADO OPERADOR_DOS_PUNTOS (⟨SENTENCIAS⟩)*)?
FIN_CONFORME

⟨SENTENCIA_PARA⟩ → INICIO_CICLO_PARA (VARIABLE ⟨ASIGNACION⟩ |
⟨DECLARACION⟩) CONDICION_CICLO_PARA ⟨EXPRESION⟩ INCREMENTO_CICLO_PARA
(NUMERO_ENTERO | NUMERO_DECIMAL) (⟨SENTENCIAS⟩)* FIN_CICLO_PARA

⟨SENTENCIA_REPETIR⟩ → INICIO_CICLO_REPETIR (⟨SENTENCIAS⟩)*
CONDICION_CICLO_REPETIR ⟨EXPRESION⟩ DELIMITADOR

⟨SENTENCIA_MIENTRAS⟩ → INICIO_CICLO_MIENTRAS ⟨EXPRESION⟩ HACER
(⟨SENTENCIAS⟩)* FIN_CICLO_MIENTRAS

⟨DECLARACION_FUNCION⟩ → FUNCION (⟨TIPO_DATO⟩)? VARIABLE PAREN_ABIERTO
(⟨DECLARACION_ARGUMENTOS⟩)? PAREN_CERRADO (⟨SENTENCIAS⟩)* (RETORNAR
⟨EXPRESION⟩ DELIMITADOR)? FIN_FUNCION

⟨ARGUMENTOS⟩ → ⟨EXPRESION⟩ (COMA ⟨EXPRESION⟩)*

Análisis Semántico

El análisis semántico es la tercera fase del proceso de compilación, donde se examina el significado del programa fuente más allá de su estructura gramatical.

Tareas principales:

- Verificación de tipos: se comprueba que las operaciones y expresiones sean compatibles con los tipos de datos que se utilizan en el programa.

- Resolución de identificadores: se verifica la existencia de las variables y funciones declaradas en el programa. Se verifica que se utilizan identificadores válidos y que no existan conflictos de nombres.

Implementación

Este análisis es esencial para garantizar que un programa sea coherente y correcto en cuanto a su significado y uso de variables. Al realizar una verificación exhaustiva de tipos y resolución de identificadores, se reducen los errores y se mejora la calidad del código generado.

Para llevar a cabo este análisis se incluyen comprobaciones adicionales, estas se realizan para que el código sea coherente y se pueda compilar correctamente.

Para realizar estas comprobaciones hemos integrado nuevas gramáticas en nuestro compilador. Estas gramáticas están diseñadas para analizar exhaustivamente la correcta manipulación de los símbolos en el código fuente. Estas gramáticas agregadas son las siguientes:

⟨EXPRESION⟩ → ⟨LOGICO_OR⟩

⟨LOGICO_OR⟩ → ⟨LOGICO_AND⟩ (LOGICO_OR ⟨LOGICO_AND⟩)*

⟨LOGICO_AND⟩ → ⟨COMPARACION⟩ (LOGICO_AND ⟨COMPARACION⟩)*

⟨COMPARACION⟩ → ⟨OPERANDO⟩ ((OPERADOR_IGUAL | OPERADOR_DIFERENTE | OPERADOR_MENOR_IGUAL | OPERADOR_MENOR | OPERADOR_MAYOR) ⟨OPERANDO⟩)?

⟨OPERANDO⟩ → ⟨TERMINO⟩ ((SUMA | RESTA) ⟨TERMINO⟩)*

⟨TERMINO⟩ → ⟨FACTOR⟩ ((MULTIPLICACION| DIVISION | MODULO) ⟨FACTOR⟩)*

⟨LLAMADAS⟩ → VARIABLE (⟨INDICE_ARREGLO⟩ | PAREN_ABIERTO

(⟨ARGUMENTOS⟩)? PAREN_CERRADO)?

⟨FACTOR⟩ → ⟨LLAMADAS⟩

⟨FACTOR⟩ → ⟨CONSTANTES⟩

⟨FACTOR⟩ → PAREN_ABIERTO ⟨EXPRESION⟩ PAREN_CERRADO

⟨FACTOR⟩ → LOGICO_NOT ⟨FACTOR⟩

⟨FACTOR⟩ → (SUMA | RESTA) ⟨FACTOR⟩

⟨INIDICE_ARREGLO⟩ → (INDICE_ABIERTO ⟨EXPRESION⟩ INDICE_CERRADO)$^{+}$

⟨DECLARACION⟩ → ESTABLECER ⟨TIPO_DATO⟩ (⟨INIDICE_ARREGLO⟩)? VARIABLE

(⟨ASIGNACION⟩)?

⟨ASIGNACION⟩ → ASIGNACION ⟨EXPRESION⟩

⟨DECLARACION_ARGUMENTOS⟩ → ⟨TIPO_DATO⟩ VARIABLE

(COMA ⟨TIPO_DATO⟩ VARIABLE)*

⟨ARGUMENTOS⟩ → ⟨EXPRESION⟩ (COMA ⟨EXPRESION⟩)*

Estas gramáticas vienen implementadas en conjunto con las existentes para realizar las distintas comprobaciones. La implementación de nuevas gramáticas en nuestro compilador fortalece esta etapa, asegurando que el código cumpla con las reglas semánticas establecidas por el lenguaje ELL.

```
Se encontraron 2 errores durante la prueba semántica:

- Error semántico, Línea 12, columna 36.
    No se puede aplicar el operador '+' entre tipos de dato "Entero" y "Cadena".
- Error semántico, Línea 14, columna 23.
    El símbolo "edad" ya está declarado en este ámbito.
    Quitar declaración doble.
```

Ilustración 1.- Demostración de algunos errores semánticos

Generación de código intermedio

El código intermedio es una representación intermedia del programa generada durante el proceso de compilación. Su objetivo principal es facilitar la portabilidad del compilador y proporcionar una forma simple del código fuente. El código intermedio es generado de manera en que se exprese en una forma mas cercana al lenguaje máquina.

Implementación

En el caso del compilador ELL, se implementó el código de tres direcciones representado en cuádruplos utilizando notación prefija. Esta representación consiste en una serie de instrucciones donde cada cuádruplo contiene un operador en la posición inicial, seguido por los operandos necesarios para llevar a cabo la operación y terminando con una variable terminal.

Durante el proceso de compilación, el compilador ELL genera un archivo llamado “intermedio.txt”, que es un archivo de texto que contiene el código de tres direcciones. Cada línea de este archivo representa un cuádruplo o etiqueta y contiene la información necesaria para llevar a cabo una operación especifica del lenguaje. Este archivo sirve como una representación intermedia del programa compilado y puede ser utilizado para realizar las fases siguientes.

```
Inicio

    Establecer Entero a = 2 + 3 * 4;

Fin
```

Ilustración 2.- Código fuente

```
Inicio:
    = 2 t1
    = 3 t2
    = 4 t3
    * t2 t3 t4
    + t1 t4 t5
    = t5 a
    = a t6
```

Ilustración 3.- Código intermedio generado

Tabla de errores

Número de error	Tipo	Ubicación	Token	Descripción	Solución.
1	Error léxico	Línea 3, columna 6	,	Símbolo ‘,’ desconocido	Remover símbolo ‘,’
2	Error léxico	Línea 5, columna 6	17.16.3	Se encontró un numero invalido	Remover o revisar numero
2	Error sintáctico	Línea 3, Columna 24	;	Se encontró símbolo “;”	Se esperaba uno de los siguientes: - <CADENA_TEXTO> - <CARÁCTER_TEXTO> - <NUMERO_ENTERO> - <NUMERO_DECIMAL> - “Falso” - “Verdadero” - “!” - “(“ - <VARIABLE>
3	Error sintáctico	Línea 13, columna 32	Mientras	Se encontró el símbolo “Mientras”	Se esperaba uno de los siguientes: - “.” - “+” - “-” - “*” - “/” - “%” - “| |” - “==” - “!=” - “>” - “<” - “>=” - “<=”
4	Error sintáctico	Línea 19, columna 9	<VARIABLE>	Se encontró el símbolo <VARIABLE>	Se esperaba uno de los siguientes: - <CADENA_TEXTO> - <CARÁCTER_TEXTO> - <NUMERO_ENTERO> - <NUMERO_DECIMAL> - “Falso” - “Verdadero”
5	Error sintáctico	Línea 1, columna 1	Fin	Se encontró el símbolo “Fin”	Se esperaba uno de los siguientes: - <COMENTARIO> - “Interpretar” - “Escribir” - ''Establecer'' - ''Para''

					- "Mientras" - "Repetir" - "Si" - "Conforme" - <VARIABLE>
6	Error semántico	Línea 15, columna 4	b	El símbolo “b” ya está declarado en este ámbito	Quitar declaración doble
7	Error semántico	Línea 12, columna 3	mostrar	La función “mostrar” no puede retornar ningún valor	Quitar retornar
8	Error semántico	Línea 4, columna 9	salario	La función “salario” no puede retornar un tipo de dato “Entero”	Retornar tipo de dato “Flotante”
9	Error semántico	Línea 8, columna 4		La función debe terminar con el mismo nombre	Colocar el nombre la función después del “FinFunsion”
10	Error semántico	Línea 4, columna 2	cal1	El símbolo “cal1” no está declarado en este ámbito	Declarar el símbolo
11	Error semántico	Lina 1, columna 7		El símbolo no es una variable	No se le puede asignar un valor a una llamada de función