Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 1
Primer Semestre 2025
Catedráticos:
Ing. Manuel Castillo
Ing. Kevin Lajpop
Ing. Mario Bautista
Tutores académicos:
Facundo Perez
Brandon Tejaxun
Xhunik Miguel

SimpliCode
Proyecto 2

Tabla de Contenido

1. Objetivo General

Aplicar los conocimientos sobre la fase de análisis léxico y sintáctico de un compilador para la realización de un intérprete sencillo, con las funcionalidades principales para que sea funcional.

2. Objetivos específicos

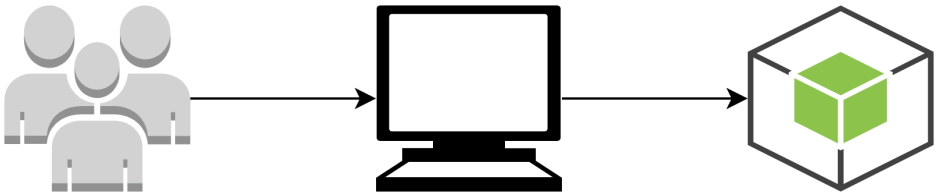
- Reforzar los conocimientos de análisis léxico y sintáctico para la creación de un lenguaje de programación.
- Aplicar los conceptos de compiladores para implementar un proceso de interpretación de código de alto nivel.
- Aplicar los conceptos de compiladores para analizar un lenguaje de programación y producir las salidas esperadas.
- Aplicar la teoría de compiladores para la creación de soluciones de software.
- Generar aplicaciones utilizando la arquitectura cliente-servidor.

3. Descripción General

El curso de Organización de Lenguajes y Compiladores 1, perteneciente a la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, necesita un intérprete capaz de reconocer un nuevo lenguaje de programación y generar reportes.

Por tanto, a usted, que es estudiante del curso de Compiladores 1, se le encomienda realizar el proyecto llamado SimpliCode. Dado sus altos conocimientos en temas de análisis léxico y sintáctico.

Arquitectura Cliente - Servidor



4. Entorno de Trabajo

4.1 Editor

El editor será parte del entorno de trabajo, cuya finalidad será proporcionar ciertas funcionalidades, características y herramientas que serán de utilidad para el usuario. La función principal del editor será el ingreso del código fuente que será analizado. En este se podrán abrir diferentes archivos al mismo tiempo y deberá mostrar la línea actual. El editor de texto se tendrá que mostrar en el navegador. Queda a discreción del estudiante el diseño.

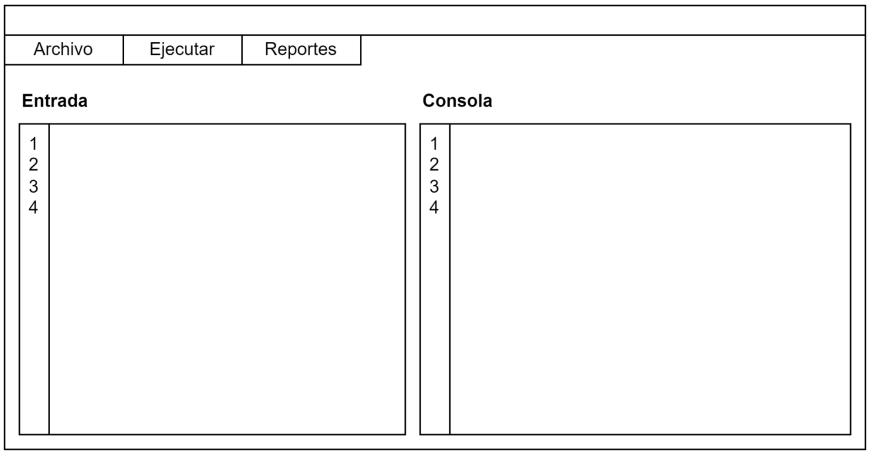


Figura 1: Propuesta Interfaz

4.2 Funcionalidades

- **Nuevo archivo:** El editor deberá de ser capaz de crear archivos en blanco.
- **Abrir archivos:** El editor deberá abrir archivos .ci y colocar su contenido en el área de entrada
- **Guardar:** El editor debe tener la capacidad de guardar el estado del archivo que se está trabajando.

4.4 Herramientas

- **Ejecutar:** hará el llamado al intérprete, el cual se hará cargo de realizar los análisis léxico y sintactico , además de ejecutar todas las sentencias.

4.5 Reportes

- **Reporte de Errores:** se mostrarán todos los errores encontrados al realizar el análisis léxico y sintactico .

- **Generar Árbol de Análisis Sintáctico:** se deberá generar una imagen del árbol de análisis sintáctico resultante del último archivo ejecutado.

  Nota: se recomienda el uso de Graphviz

- **Reporte de Tabla de Símbolos:** se mostrarán todas las variables, métodos y funciones que han sido declarados dentro del flujo del programa.

4.6 Área de Consola

En esta área se mostrarán los resultados, mensajes y todo lo que sea indicado dentro del lenguaje. Tiene como restricción el no ser editable por el usuario y únicamente puede mostrar información.

5. Descripción del Lenguaje

5.1 Case Sensitive

El lenguaje es case sensitive por lo que hace distinción entre mayúsculas y minúsculas.

```
// Ejemplo
Entero a = 0
ENTero A = 0

// “A” y “a” son variables distintas
// Entero no es lo mismo que ENTero
```

5.2 Comentarios

Los comentarios son una forma elegante de indicar que función tiene cierta sección del código que se ha escrito, simplemente para dejar un mensaje en específico. El lenguaje deberá soportar dos tipos de comentarios que son los siguientes:

5.2.1 Comentarios de una línea

Este comentario comenzará con // y deberá terminar con un salto de línea.

5.2.2 Comentarios multilínea

Este comentario comenzará con /* y terminará con */.

```
// Este es un comentario de una línea
/*
Este es un comentario
Multilínea
Para este lenguaje
*/
```

5.3 Tipos de Datos

Los tipos de dato que soportará el lenguaje en concepto de un tipo de variable se definen a continuación:

Tipo	Descripción	Ejemplo	Observaciones	Valor por defecto
Entero	Admite únicamente valores numéricos enteros	1, -120, etc	Se maneja cualquier cantidad de dígitos.	0
Decimal	Admite valores numéricos con decimales	1.2, -5.3, etc	Se maneja cualquier cantidad de decimales	0.0
Booleano	Admite valores que indican verdadero o falso	Verdadero, Falso	Si se asigna un valor booleano a un entero se tomará como 1 o 0 respectivamente	Verdadero
Caracter	Tipo de dato que únicamente aceptará un único carácter, y estará delimitado por comillas simples	'a', 'b', 'c', etc	Se permitirá cualquier carácter entre las comillas simples, incluyendo las secuencias de escape	'\u0000' (carácter 0)
Cadena	Es un grupo o conjunto de caracteres que pueden tener cualquier carácter, y este se encontrará delimitado por comillas dobles. " "	"cadena"	Se permitirá cualquier carácter entre las comillas dobles, incluyendo las secuencias de escape	“” (string vacío)

5.4 Secuencias de escape

Las secuencias de escape se utilizan para definir ciertos caracteres especiales dentro de cadenas de texto. Las secuencias de escape disponibles son las siguientes:

Secuencia	Descripción	Ejemplo
\n	Salto de línea	"Hola\nMundo"
\\	Barra invertida	"C:\\miCarpeta"
\"	Comilla doble	"\"esto es una cadena\""
\t	Tabulación	"\tEsto es una tabulación"
\'	Comilla simple	"\'Estas son comillas simples\'"

5.5 Operadores Aritméticos

5.5.1 Suma

Es la operación aritmética que consiste en realizar la suma entre dos o más valores. Para esta se utiliza el signo más **(+)**.

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

+	Entero	Decimal	Boolean	Carácter	Cadena
Entero	Entero	Decimal	Entero	Entero	Cadena
Decimal	Decimal	Decimal	Decimal	Decimal	Cadena
Boolean	Entero	Decimal			Cadena
Carácter	Entero	Decimal		Cadena	Cadena
Cadena	Cadena	Cadena	Cadena	Cadena	Cadena

Nota: Cualquier otra combinación no especificada en esta tabla es invalida ya que es error de tipo semántico (no es necesario reportarlo).

5.5.2 Resta

Es la operación aritmética que consiste en realizar la resta entre dos o más valores. Para esta se utiliza el signo menos **(-)**.

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

-	Entero	Decimal	Boolean	Carácter
Entero	Entero	Decimal	Entero	Entero
Decimal	Decimal	Decimal	Decimal	Decimal
Boolean	Entero	Decimal		
Carácter	Entero	Decimal		

Nota: Cualquier otra combinación no especificada en esta tabla es invalida ya que es error de tipo semántico (no es necesario reportarlo).

5.5.3 Multiplicación

Es la operación aritmética que consiste en sumar un número (multiplicando) tantas veces como indica otro número (multiplicador). El signo para representar la operación es el asterisco **(*)**.

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

*	Entero	Decimal	Carácter
Entero	Entero	Decimal	Entero
Decimal	Decimal	Decimal	Decimal
Carácter	Entero	Decimal	

Nota: Cualquier otra combinación no especificada en esta tabla es invalida ya que es error de tipo semántico (no es necesario reportarlo).

5.5.4 División

Es la operación aritmética que consiste en partir un todo en varias partes, al todo se le conoce como dividendo, al total de partes se le llama divisor y el resultado recibe el nombre de cociente. El operador de la división es la diagonal **(/)**.

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

/	Entero	Decimal	Carácter
Entero	Decimal	Decimal	Decimal
Decimal	Decimal	Decimal	Decimal
Carácter	Decimal	Decimal	

Nota: Cualquier otra combinación no especificada en esta tabla es invalida ya que es error de tipo semántico (no es necesario reportarlo).

5.5.5 Potencia

Es una operación aritmética de la forma **a^b**, donde a es el valor de la base y b es el valor del exponente que nos indicará cuantas veces queremos multiplicar el mismo número. Por ejemplo 5 ^ 3, a=5 y b=3 tendríamos que multiplicar 3 veces 5 para obtener el resultado final; 5x5x5 que da como resultado 125.

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

^	Entero	Decimal
Entero	Entero	Decimal
Decimal	Decimal	Decimal

Nota: Cualquier otra combinación no especificada en esta tabla es invalida ya que es error de tipo semántico (no es necesario reportarlo).

5.5.6 Módulo

Es una operación aritmética que obtiene el resto de la división de un número entre otro. Para realizar la operación se utilizará el signo **(%)**.

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

%	Entero	Decimal
Entero	Decimal	Decimal
Decimal	Decimal	Decimal

Nota: Cualquier otra combinación no especificada en esta tabla es invalida ya que es error de tipo semántico (no es necesario reportarlo).

5.5.7 Negación Unaria

Es una operación que niega el valor de un número, es decir que devuelve el contrario del valor original. Se utiliza el símbolo menos **(-)**.

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

- exp	Resultado
Entero	Entero
Decimal	Decimal

Nota: Cualquier otra combinación no especificada en esta tabla es invalida ya que es error de tipo semántico (no es necesario reportarlo).

5.6 Operadores Relacionales

Son los símbolos que tienen como finalidad comparar expresiones, dando como resultado valores booleanos.

Operador	Descripción	Ejemplo
==	Igualación: compara ambos valores y verifica si son iguales - Iguales → Verdadero - No iguales → Falso	1==1 "hola" == "hola" 25.654 == 54.34
!=	Diferenciación: compara ambos lados y verifica si son distintos - Iguales → Falso - No iguales → Verdadero	1 != 2 var1 != var2 50 != 30
<	Menor que: compara ambos lados y verifica si el derecho es mayor que el izquierdo. - Derecho mayor → Verdadero - Izquierdo mayor → Falso	25.5 < 30 54 < 25 50 < 'F'
<=	Menor o igual que: Compara ambos lados y verifica si el derecho es mayor o igual que el izquierdo. - Derecho mayor o igual → Verdadero - Izquierdo mayor → Falso	25.5 <= 30 54 <= 25 50 <= 'F'
>	Mayor que: compara ambos lados y verifica si el izquierdo es mayor que el derecho. - Derecho mayor → Falso - Izquierdo mayor → Verdadero	25.5 > 30 54 > 25 50 > 'F'
>=	Mayor o igual que: Compara ambos lados y verifica si el izquierdo es mayor o igual que el derecho. - Derecho mayor → Falso - Izquierdo mayor o igual → Verdadero	25.5 >= 30 54 >= 25 50 >= 'F'

Especificaciones de los operadores relacionales

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

	Entero	Decimal	Boolean	Carácter	Cadena
Entero					
Decimal					
Boolean					
Carácter					
Cadena					

Nota: Cualquier otra combinación no especificada en esta tabla es invalida ya que es error de tipo semántico (no es necesario reportarlo).

5.8 Operadores Lógicos

Son los símbolos que tienen como finalidad comparar expresiones a nivel lógico (verdadero o falso). A continuación, se definen los símbolos que serán aceptados dentro del lenguaje:

Operador	Descripción	Ejemplo	Observaciones
||	**OR**: compara expresiones lógicas y si al menos una es verdadera, entonces devuelve verdadero y en otro caso retorna falso.	bandera || 5<2 Devuelve Verdadero	bandera es Verdadero
&&	**AND**: compara expresiones lógicas y si ambas son verdaderas, entonces devuelve verdadero y en otro caso retorna falso.	flag && flag Devuelve Verdadero	flag es Verdadero
!	**NOT**: devuelve el valor inverso de una expresión lógica si esta es verdadera entonces devolverá false, de lo contrario retorna verdadero	!var Devuelve Falso	var es Verdadero

5.9 Signos de Agrupación

Los signos de agrupación serán utilizados para agrupar operaciones aritméticas, lógicas o racionales. Los símbolos de agrupación están dados por **(** y **)**.

5.10 Precedencia de Operaciones

La precedencia de operadores nos indica la importancia de que una operación debe realizarse por encima del resto. A continuación, se define la misma:

Nivel	Operador	Asociatividad
0	-	No Asociativa
1	^	Derecha
2	*, /, %	Izquierda
3	+, -	Izquierda
4	==, !=, <, <=, >, >=	Izquierda
5	!	Derecha
6	&&	Izquierda
7	||	Izquierda

Nota: el nivel 0 es el nivel de mayor importancia

5.12 Declaración de Variables y Asignación de Variables

Una variable deberá de ser declarada antes de poder ser utilizada. Todas las variables tendrán un tipo de dato y un nombre de identificador. Las variables podrán ser declaradas global y localmente.

Es posible realizar declaraciones múltiples, para ello es necesario ingresar una lista de nombres, separadas por comas.

Las variables no pueden cambiar de tipo de dato, se deben mantener con el tipo declarado inicialmente, por lo que se debe de validar que el tipo de la variable y el valor sean compatibles.

```
// Declaración
```

// Declaración sencilla
ingresar **<nombre variable>** como **<tipo de dato>** con valor **<expresión>**
ingresar **<nombre variable>** como **<tipo de dato>**

// Declaración múltiple
ingresar **<lista de variables>** como **<tipo de dato>** con valor **<lista de expresiones>**
ingresar **<lista de variables>** como **<tipo de dato>**

// Asignación
<nombre variable> -> **<expresión>**
<lista de variables> -> **<lista de expresiones>**

// Ejemplo
ingresar var1 como entero con valor 12
ingresar var2 como entero
var2 -> 10

ingresar var3, var4, var5 como entero con valor 10, 15, 20
ingresar var6, var7, var8 como entero
var6, var7, var8 -> 10, 20, 30

5.13 Casteos

Los casteos son una forma de indicar al lenguaje que convierta un tipo de dato en otro, por lo que, si queremos cambiar un valor a otro tipo, es la forma adecuada de hacerlo. Para hacer esto, se colocará la palabra reservada del tipo de dato destino entre paréntesis seguido de una expresión.

```
// Casteo
(<tipo de dato>) <expresión>

// Ejemplo
ingresar edad como entero con valor (Entero) 15.6 // Toma el valor entero 15
ingresar letra como caracter con valor (caracter) 70 // Toma el valor 'F' ya
que el 70 en ascii es F

ingresar precio como decimal
precio -> (decimal) 16 // Toma el valor 16.0
```

El lenguaje aceptará los siguientes casteos:

- Entero -> Decimal
- Decimal -> Entero
- Entero -> Cadena
- Entero -> Caracter
- Decimal -> Cadena
- Caracter -> Entero
- Caracter -> Decimal

5.14 Incremento y Decremento

Los incrementos y decrementos nos ayudan a realizar la suma o resta continua de un valor de uno en uno, es decir si incrementamos una variable, se incrementará de uno en uno, mientras que, si realizamos un decremento, hará la operación contraria.

```
// Incremento y Decremento
inc(<expresión>)
dec(<expresión>)

// Ejemplo
```

```
ingresar edad como entero con valor 18
inc(edad) // Toma el valor 19
dec(edad) // Toma el valor 18
```

5.15 Listas

Los vectores son una estructura de datos que puede almacenar valores de forma limitada, y los valores que pueden almacenar son de un único tipo; Entero, Decimal, Booleano, Caracter o Cadena. **El lenguaje permitirá únicamente el uso de arreglos de una a tres dimensiones**

```
ingresar Lista (<dimensiones>, <tipo>) <identificador> -> (<Lista de valores>)

// Ejemplo
ingresar Lista (1, Entero) listaEnteros -> (1,2,3,4,5)
ingresar Lista (2, Caracter) listaCaracteres -> (('1','@'),('b','5'))
ingresar Lista (3, Cadena) listaCadenas -> (((“hola”,”mundo”),(“a”’,”:)”)), ((“si”,”sale”),(“Compi”,”1”)))
```

5.15.1. Acceso a Listas

```
<identificador> [<indice>]
<identificador> [<indice>][<indice>]
<identificador> [<indice>][<indice>][<indice>]

// Ejemplo
imprimir listaEnteros[0] // resultado a mostrar es “1”
imprimir listaCaracteres[1][0] // resultado a mostrar es “b”
imprimir listaCadenas[1][1][0] // resultado a mostrar es “Compi”
```

5.15.2 Modificación de Listas

```
<identificador> [<indice>] = <expresion>
<identificador> [<indice>][<indice>] = <expresion>
<identificador> [<indice>][<indice>][<indice>] = <expresion>

// Ejemplo
listaEnteros[0] = 15 // el valor almacenado ahora es “15”
listaCaracteres[1][0] = ‘C’ // el valor almacenado ahora es “C”
listaCadenas[1][1][0] = “compiladores” // el valor almacenado ahora es “compiladores”
```

5.16 Condicionales

5.16.1 Condicional Si

Esta es una instrucción que permite ejecutar un bloque de instrucciones cuando una condición es válida. La instrucción necesita una condición para ejecutar un bloque asignado, cuando la condición es falsa se ejecuta otro bloque de instrucciones. Es posible que el bloque de instrucciones que ejecuta una condición falsa, sea opcional. Es obligatorio que ingrese una condición en esta instrucción.

Se usará la palabra reservada "o si" cuando una condición es falsa y se trata de volver hacer otra validación con otra condición. Este tipo de ejecución puede ser opcional o repetirse una o muchas veces

```
// Condicional si
// instrucción con bloque de instrucciones con condición verdadera
si <condición> entonces
   <instrucciones>
fin si

// instrucción con bloque de instrucciones con condición verdadera y falsa
si <condición> entonces
   <instrucciones>
de lo contrario
   <instrucciones>
fin si
// instrucción con múltiples condiciones
si <condición> entonces
   <instrucciones>
o si <condición> entonces
   <instrucciones>
de lo contrario
   <instrucciones>
fin si

// Ejemplo
```

```
si x < 50 entonces
    imprimir "x es menor a 50"
    // más sentencias
o si x > 50 entonces
    imprimir "x es mayor a 50"
    // más sentencias
de lo contrario
    imprimir "x es igual a 50"
    // más sentencias
fin si
```

5.16.2 Selección múltiple

Esta condición permite ejecutar una lista de instrucciones en base a un valor ingresado, existen varias opciones posibles y cuando el valor coincida con una de las opciones se ejecuta un conjunto de instrucciones. Existe una palabra reservada "de lo contrario" para ejecutar automáticamente cuando la lista de opciones no se cumple, pero es de forma opcional.

```
// Selección múltiple
segun <expresión> hacer
    en caso de ser <expresión> entonces
        <instrucciones>
        detener
    en caso de ser <expresión> entonces
        <instrucciones>
        detener
    ...
    ...
    de lo contrario entonces
        <instrucciones>
        detener
fin segun

// Ejemplo
ingresar edad como entero con valor 18
```

```
segun edad hacer
    en caso de ser 10 entonces
        imprimir "tengo 10 años"
        // más sentencias
        detener
    en caso de ser 18 entonces
        imprimir "tengo 18 años"
        // más sentencias
        detener
    de lo contrario entonces
        imprimir "no conozco mi edad"
        // más sentencias
        detener
fin segun
```

5.17 Ciclos

5.17.1 Ciclo Para

Ejecuta un conjunto de instrucciones, con un límite de repeticiones. Es necesario ingresar un valor nicial y también un valor final, el valor final es el que le indica al ciclo, en momento para terminar de ralizar las repeticiones. Otro de los elementos necesarios en este ciclo es el número de pasos que realizará entre cada repetición.

```
// Ciclo para
// Ciclo "para" con salto definido
para <variable> -> <valor inicial> hasta <valor final> con incremento <valor del salto> hacer
    <instrucciones>
fin para


// Ejemplo
para i -> 0 hasta 5 con incremento i++ hacer
    imprimir "i = " + i
fin para

para i -> 5 hasta 0 con decremento i-- hacer
    imprimir "i = " + i
fin para
```

5.17.2 Ciclo Mientras

Este ciclo ejecuta un conjunto de instrucciones sin límite definido. Para poder realizar una repetición es necesario que exista una condición.

```
// Ciclo mientras
mientras <condición> hacer
    <instrucciones>
fin mientras

// Ejemplo
mientras x < 100 hacer
    si x > 50 entonces
        imprimir "x es mayor a 50"
        // más sentencias
```

```
    de lo contrario
        imprimir "x es menor que 100"
        // más sentencias
    fin si
    x++
fin mientras
```

5.17.3 Ciclo Repetir hasta

Ejecuta un conjunto de instrucciones sin límite definido. A diferencia del ciclo anterior, este ciclo realiza una única repetición sin restricción. Para poder realizar las demás repeticiones es necesario que exista una condición.

```
// Ciclo Repetir hasta
repetir
    <instrucciones>
hasta que <condición>

// Ejemplo
ingresar var1 como entero con valor 5

repetir
    si var1 >= 1 && var < 3 entonces
        imprimir verdadero
    de lo contrario
        imprimir falso
    fin si
    var1--
hasta que var1 > 0
```

5.18 Sentencias de Transferencia

Las sentencias de transferencia nos permiten manipular el comportamiento de los bucles, ya sea para detenerlo o para saltarse algunas iteraciones. El lenguaje soporta las siguientes sentencias:

5.18.1 Detener

Hace que se salga del ciclo inmediatamente, es decir que el código que se encuentre después del "detener" en la misma iteración no se ejecutara y este se saldrá del ciclo.

```
detener

// Ejemplo

para i -> 0 hasta 9 con incremento i++ hacer
    si i == 5 entonces
        imprimir "Se sale del ciclo en el No." + i
        detener
    fin si
    imprimir i
fin para
```

5.18.2 Continuar

Puede detener la ejecución de la iteración actual y saltar a la siguiente. La sentencia continuar siempre debe de estar dentro de un ciclo, de lo contrario será un error.

```
continuar

// Ejemplo
para i -> 0 hasta 9 con incremento i++ hacer
    si i == 5 entonces
        imprimir "Se salta el No." + i
        continuar
    fin si
    imprimir i
fin para
```

5.18.3 Retorno

La sentencia retorno finaliza la ejecución de un método o función y puede devolver un valor específicamente.

```
// Retorno
retornar
retornar <expresión>

// Ejemplo
// Finaliza la ejecución
para i -> 0 hasta 9 con incremento i++ hacer
    si i == 5 entonces
        retornar
    fin si
    imprimir i
fin para

// Devuelve un valor
funcion suma entero con parametros(valor1 entero, valor2 entero)
    retornar valor1 + valor2
fin funcion
```

5.19 Funciones

Una función es una subrutina de código que se identifica con un nombre, tipo y un conjunto de parámetros. En el lenguaje definido las funciones serán declaradas con la estructura palabra reservada "funcion", luego un id para identificar la función y su tipo, seguido de una lista de parámetros dentro de paréntesis (la lista de parámetros es opcional) y finalmente la palabra reservada "fin funcion".

Cada parámetro debe estar compuesto por un identificador seguido de su tipo, para el caso de que sean varios parámetros se debe utilizar comas para separar cada parámetro y en el caso de que no se usen parámetros no se deberá incluir nada dentro de los paréntesis.

Para las funciones vendrá un valor de retorno que coincida con el tipo con el que se declaró la función.

```
// Funciones
// Función sin parámetros
funcion <nombre> <tipo dato>
    <instrucciones>
fin funcion

// Función con parámetros
funcion <nombre> <tipo dato> con parametros (<lista de parametros>)
    <instrucciones>
fin funcion

// Ejemplo
funcion suma entero con parametros(valor1 entero, valor2 entero)
    retornar valor1 + valor2
fin funcion
```

Cabe destacar que **no habrá sobrecarga de funciones y métodos** dentro de este lenguaje por lo que solo puede existir una función o método con el id declarado por lo que no se creará otra función o método con un id previamente utilizado ya que esto debe generará un error de tipo semántico (no es necesario reportarlo).

NOTA: Las funciones pueden traer funciones y/o parámetros dentro de ellas como recursividad.

5.20 Procedimientos

Un procedimiento también es una subrutina de código que se identifica con un nombre y un conjunto de parámetros, aunque a diferencia de las funciones estas subrutinas no deben de retornar un valor. En el lenguaje definido los métodos serán declaradas con la estructura palabra reservada "procedimiento", luego un id para identificar la función, seguido de una lista de parámetros dentro de paréntesis (la lista de parámetros es opcional) y finalmente la palabra reservada "fin procedimiento".

Cada parámetro debe estar compuesto por un identificador seguido de su tipo, para el caso de que sean varios parámetros se debe utilizar comas para separar cada parámetro y en el caso de que no se usen parámetros no se deberá incluir nada dentro de los paréntesis.

```
// Procedimientos
// Procedimiento sin parámetros
procedimiento <nombre>
    <instrucciones>
fin procedimiento

// Procedimiento con parámetros
procedimiento <nombre> con parametros (<lista de parametros>)
    <instrucciones>
fin procedimiento

// Ejemplo
procedimiento holaMundo
    imprimir "Hola Mundo"
fin procedimiento
```

Cabe destacar que **no habrá sobrecarga de funciones y procedimientos,** dentro de este lenguaje por lo que solo puede existir una función o procedimiento con el id declarado si se crea otra función o procedimiento con un id previamente utilizado es error de tipo semántico (no es necesario reportarlo).

NOTA: Los procedimeitnos pueden traer funciones y/o parámetros dentro de ellas como recursividad.

5.21 Llamada de Procedimientos y Funciones

Este tipo de instrucción realiza la ejecución de un procedimiento o función, para poder realizarlo es necesario ingresar el identificador de la función o procedimiento y la lista de parámetros necesarios.

```
// Llamada de procedimientos y funciones
ejecutar <identificador>()
ejecutar <identificador>(<lista de parametros>)

// Ejemplo
imprimir "Llamada de procedimiento"
ejecutar holaMundo()
```

```
imprimir "Llamada de función"
ingresar var2 como entero
var2 -> suma(12, 15)
imprimir "El total es: " + var2
```

5.22 Objetos

Es posible crear tipos de datos abstractos utilizando la palabra reservada “objeto”. La sintaxis se muestra a continuación.

```
// Objeto
objeto <identificador> (
        // Atributos
        <identificador> <tipo>
        <identificador> <tipo>
        …
)

// Ejemplo
objeto Carro (
        color Cadena
        modelo Cadena
        convertible Booleano
        velocidad Entero
)
```

5.22.1 Métodos en Objetos

Es posible definir procedimientos propios de un objeto.

NOTA: Cualquier referencia a variables que no estén declaradas se refieren a atributos del objeto.

```
// Métodos en objetos
// Métodos sin parámetros
<nombreobjeto> -> metodo <nombre>
   <instrucciones>
fin metodo

// Métodos con parámetros
<nombreobjeto> -> metodo  <nombre> con parametros (<lista de parametros>)
```

```
   <instrucciones>
fin metodo

// Ejemplo
Carro -> metodo mostrarModelo
        imprimir “Modelo: ” + modelo
fin metodo
```

5.22.2 Instanciación de Objetos

Es necesario enviar valores a todos los atributos del objeto durante la creación de la instancia en el orden que son declarados.

```
// Instancia
ingresar objeto <nombre de objeto> <identificador> -> <nombre de objeto>(<lista de valores>)

// Ejemplo
ingresar objeto Carro nuevoCarro -> Carro(
        “rojo”,
        “mazda 2009”,
        Falso,
        60
)
```

5.22.3 Acceso a Atributos y Métodos de Objetos

Para acceder a atributos o métodos de una instancia de un objeto específico se hará mediante el caracter de punto “**.**”.

```
// Acceso a atributos
<identificador>.<atributo de objeto>

// Acceso a procedimientos
ejecutar <identificador>.<método de objeto>()
ejecutar <identificador>.<método de objeto>(<lista de valores>)

// Ejemplo acceso atributo
imprimir nuevoCarro.color
imprimir nuevoCarro.velocidad

// Ejemplo acceso procedimiento
ejecutar nuevoCarro.acelerar(120)
```

5.23 Función imprimir

Esta función nos permite imprimir expresiones agregando un salto de línea al final del contenido.

```
// Imprimir
imprimir <expresión> // Imprime sin salto de linea
imprimir nl <expresión> // Imprime con un salto de linea

// Ejemplo
imprimir "Hola Mundo"
imrpimir nl "Sale Compi"
imprimir "Resultado de la suma: \n" + valor
```

5.24 Funciones Nativas

5.24.1 Minúscula

Esta función recibe como parámetro una expresión de tipo cadena y retorna una nueva cadena con todas las letras minúsculas.

```
// Minuscula
minuscula (<expresión>)

// Ejemplo
var1 -> minuscula ("hOlA mUnDo")
```

5.24.2 Mayúscula

Esta función recibe como parámetro una expresión de tipo cadena y retorna una nueva cadena con todas las letras mayúsculas.

```
// Mayuscula
mayuscula (<expresión>)

// Ejemplo
var1 -> mayuscula ("hOlA mUnDo")
```

5.24.3 Longitud

Esta función recibe como parámetro una cadena y devuelve el tamaño de este.

```
// Longitud
longitud (<expresión>)

// Ejemplo
nombre -> "Brunno"
var1 -> longitud(nombre)
```

5.24.4 Truncar

Esta función recibe como parámetro un valor numérico. Permite eliminar los decimales de un número, retornando un entero.

```
// Truncar
truncar (<expresión>)

// Ejemplo
var1 -> truncar(15.4654849) // Entero 15
```

5.24.5 Redondear

Esta función recibe como parámetro un valor numérico. Permite redondear los números decimales según las siguientes reglas:

- Si el decimal es mayor o igual que 0.5, se aproxima al número superior
- Si el decimal es menor que 0.5, se aproxima al número inferior

```
// Redondeo
redondear (<expresión>)
```

```
// Ejemplo
var1 -> redondear(5.8) // Valor 6
var2 -> redondear(5.4) // Valor 5
```

5.24.6 Tipo

Esta función retorna una cadena con el nombre del tipo de dato evaluado.

```
// Tipo
Tipo(<expresión>)

// Ejemplo
imprimir "El tipo de dato de var1 es " + tipo(var1)
```

6. Reportes

Los reportes son parte fundamental de SimpliCode, ya que muestra de forma visual las herramientas utilizadas para realizar la ejecución del código.

A continuación, se muestran ejemplos de estos reportes. Queda a discreción del estudiante el diseño de estos.

6.1 Tabla de errores

El reporte de errores debe contener la información suficiente para detectar y corregir errores en el código fuente.

#	Tipo	Descripción	Línea	Columna
1	Léxico	El carácter "$" no pertenece al lenguaje	5	3
2	Sintáctico	Se encontró Identificador y se esperaba Expresión.	6	3

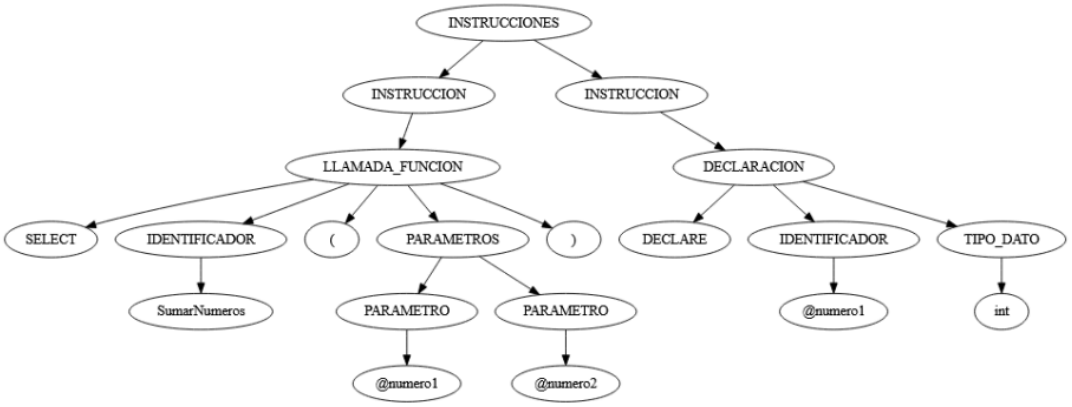
6.2 Tabla de Símbolos

Este reporte mostrará la tabla de símbolos después de la ejecución. Deberá mostrar las variables y arreglos declarados, así como su tipo, valor y toda la información que considere necesaria.

#	ID	Tipo	Tipo	Entorno	Valor	Línea	Columna
1	var	Variable	Entero	Funcion 1	10	15	20
2	var2	Variable	Cadena	Global	“Hola”	20	13
3	var3	Variable	Bool	Funcion 2	true	25	10

6.3 AST

Este reporte muestra el árbol abstracto de sintaxis producido al analizar los archivos de entrada. Este debe de representarse como un grafo. Se deben mostrar los nodos que el estudiante considere necesarios para describir el flujo realizado para analizar e interpretar sus archivos de entrada.



Nota: Se sugiere a los estudiantes utilizar la herramienta Graphviz para graficar su AST

7. Entregables

- **Código fuente del proyecto**
- **Manual de Usuario en un archivo Markdown**
  - Capturas de pantalla detallando cómo funciona su entorno de trabajo y los reportes que se generan.

- **Manual Técnico en un archivo Markdown**

    - Información importante del proyecto para que se pueda realizar el mantenimiento en el futuro. Especificar el lenguaje, herramientas utilizadas, métodos y funciones más importantes.

- **Archivo de Gramática en un archivo Markdown**
    - El archivo debe contener su gramática y debe de ser limpio, entendible y no debe ser una copia del archivo de Jison.
    - La gramática debe estar escrita en formato **BNF(Backus-Naur form).**
    - Solamente se debe escribir la gramática independiente del contexto.

8. Restricciones

- La entrega debe ser realizada mediante UEDI enviando el enlace del **repositorio privado** de GitLab en donde se encuentra su proyecto.
- El nombre del repositorio de Gitlab debe ser **OLC1_Proyecto2_#Carnet**
- Se debe agregar al auxiliar encargado como colaborador al repositorio de Gitlab, con permisos de desarrollador (Developer).
    - Sección B (Diego Facundo Pérez): **DFaxx**
    - Sección C (Brandon Tejaxun): **brandonT2002**
    - Sección N (Xhunik Miguel): **xhuniktzi**
- Lenguaje de Programación a utilizar: **Javascript/TypeScript**
- Herramientas para el análisis léxico y sintáctico: **Jison**
- **El proyecto debe ser realizado de forma individual.**
- Para graficar se puede utilizar cualquier librería (Se recomienda graphviz)
- Puede utilizar un framework como Angular, React, Vuejs, etc, para generar su entorno gráfico. Queda a discreción del estudiante cuál utilizar.
- **Copias completas/parciales** de: código, gramática, etc. serán merecedoras de una **nota de 0 puntos**, los responsables serán reportados al catedrático de la sección y a la Escuela de Ciencias y Sistemas.
- La calificación tendrá una duración de 30 minutos, acorde al programa del laboratorio.
- Se debe visualizar todo desde el navegador

9. Fecha de Entrega

Domingo, 04 de mayo de 2025 a las 23:59. La entrega será por medio de la plataforma UEDI. Entregas fuera de la fecha indicada, no se calificarán.

SE LE CALIFICARA DEL ÚLTIMO COMMIT REALIZADO ANTERIOR A ESTA FECHA.