TEC | Tecnológico de Costa Rica

IC 5701 - Compiladores e intérpretes

Gramática del lenguaje *TerraCode*

Estudiante:

Jozafath Pérez Fernández - 2023107460

Alana Calvo Bolaños - 2022040915

Mary Paz Álvarez Navarrete - 2023138604

José Gabriel Jiménez Chacón - 2021128841

Docente:

SANABRIA RODRIGUEZ AURELIO

Año:

2025

Tabla de contenido

Motivación

Decidimos crear ***TerraCode*** porque nos gusta mucho todo lo relacionado con la jardinería y el ambiente natural. Pensamos que sería interesante mezclar ese gusto con la programación, así que quisimos desarrollar un lenguaje que usará términos del mundo de las plantas, pero que también tuviera sentido en lo que hace un lenguaje de programación.

Una de nuestras ideas principales fue hacerlo lo más **interactivo** posible, especialmente para personas que estén familiarizadas con el cuidado de jardines. Por eso, usamos palabras como **‘’Sembrar’’** para empezar funciones, **‘’Fotosíntesis’’** para los ciclos, o **‘’Podar’’** para imprimir mensajes. Todo está pensado para que las estructuras tengan un doble sentido: funcionan como parte del código, pero también suenan naturales para alguien que conoce sobre plantas.

Nos motiva mucho terminar este lenguaje completo, no solo para pasar el curso, sino porque creemos que podría tener usos reales en el futuro. Por ejemplo, imaginamos que ***TerraCode*** pueda ser una herramienta útil para construir sistemas automatizados de cultivo, simular el crecimiento de plantas o incluso gestionar jardines inteligentes..

En resumen, elegimos esta temática porque nos representa y porque creemos que puede ser una forma divertida y diferente de aprender y aplicar la lógica de programación.

Análisis

El proyecto consiste en la creación de un compilador que satisfaga las necesidades del curso de Compilador e Intérpretes, al menos en la completación de la primera fase del proyecto el presente análisis consta de plasmar las fortalezas, debilidades, puntos chistosos y ejemplificación en la comparación de nuestro lenguajes contra lenguajes ya consolidados.

Al analizar el lenguaje podemos identificar ciertas **fortalezas** como lo son:

- Las reglas establecidas se implementan pensando en el uso de un **parser LL(1)**, el que permite trabajar sin ambigüedades debido a su definición como parser.
- La notación utilizada para definir las reglas gramaticales es formal, se utilizan las gramáticas libres de contexto **EBNF** como estándar del curso.
- La modularidad que existe entre las reglas gramaticales y la semántica de los nombres de las palabras reservadas, ya que permite una mejor comprensión de ambas partes.
- La gramática tiene definidas las principales reglas para poder trabajar, como lo son funciones, definición de variables, comparaciones, condicionales, bucles o operaciones aritméticas, además se hace uso de una estructura de control personalizada que puede aportar agilidad en el código dependiendo del problema que se plantee.
- Al ser algo en desarrollo, tiene facilidad de darle una dirección al lenguaje que se quiera llegar a producir.

Además también podemos identificar **puntos débiles** del lenguaje:

- Principalmente su parcial implementación de las distintas estructuras disponibles en otros lenguajes, como los arreglos unidimensionales, arreglos bidimensionales, tablas **‘’hash”**, **‘’struct´s”**, funciones lambda, tipos de datos como el char, manejo de bits, integración de bibliotecas.
- La existencia de poca o nula documentación del uso del lenguaje, al menos en esta primera fase, ya que al ser algo nuevo en desarrollo se queda corto de esta característica.
- La falta en la implementación de manejo de potenciales **errores**.
- Posibles mejoras en la implementación de las reglas, llámense **ambigüedades**, o posibles mejoras en eficiencia si se implementaran de otra forma las reglas.

Fue difícil pero los **puntos chistosos** de ***TerraCode*** son:

- **Programación a lo jardinero:**
  En ***TerraCode***, la jardinería no solo se queda en las macetas, Programar es como plantar un árbol: eliges una semilla (variable), la riegas (asignación) y ves cómo crece (función o estructura de control). No hay nada más satisfactorio que escribir **“Fotosintesis”** en lugar de un aburrido **“while”**.
- **Un Podar por un print:**
  Entonces, en lugar de **"print"**, decimos **‘’Podar”**. Sí, cómo podar las plantas... pero, en este caso, ¡podamos esos mensajes de manera sangrienta! Imprimir mensajes ya no suena tan bien cuando puedes decir que estás podando la salida.
- **¡Cuidado con los Ácaros del código!**
  En lugar de un simple **‘’not”** para negaciones, ¡aquí lo llamamos Ácaro! ¿Quién no querría eliminar un ácaro en su jardín? Al igual que en la vida real, un ácaro puede arruinar todo el flujo o quieres que siga creciendo esta plaga maldita.

Comparación con otros lenguajes:

- **Con Python:**
  A diferencia de Python, que es conocido por su simplicidad y flexibilidad, ***TerraCode*** le da un giro creativo a las cosas al incorporar conceptos botánicos. Mientras que Python usa print para mostrar mensajes, nosotros preferimos un relajado Podar para imprimir, porque en ***TerraCode*** las cosas crecen de manera orgánica.
- **Con JavaScript:**
  Mientras que JavaScript es conocido por su complejidad y flexibilidad, ***TerraCode*** se enfoca más en un código fluido y fácil de entender, al igual que un jardín bien diseñado. En lugar de la sintaxis críptica de JavaScript, nuestras estructuras de control Exterior y Interior hacen que el flujo de código sea tan natural como ver crecer una planta.
- **Con C++:**
  Si bien C++ es un lenguaje poderoso y eficiente, ***TerraCode*** se aleja de la rigidez de la sintaxis y nos permite trabajar con estructuras más "orgánicas" y personalizadas. En lugar de preocuparse por la memoria y los punteros como en C++, aquí estamos más enfocados en hacer que las funciones florezcan y las variables crezcan de forma sana.

Gramática EBNF

Programa ::= Instrucciones

Instrucciones ::= Instruccion (Instruccion+es)*

Instruccion ::= Comentario | Declaracion | Asignacion | Funcion | Repeticion | Condicional | Imprimir | CicloCasos | Retorno

Comentario ::= "*" Texto "*"

Declaracion ::= TipoVar Identificador
TipoVar ::= "Maceta" | "Semilla"

Retorno ::= "Cosechar" (Expresion)?

Asignacion ::= Identificador "=" Expresion

Funcion ::= "Sembrar" Identificador "(" ")" "{" Instrucciones "}" "Florecer"

Repeticion ::= "Fotosintesis" "(" Expresion ")" "{" Instrucciones "}"

Condicional ::= If+ Else?
If ::= "Exterior" "(" Expresion ")" "{" Instrucciones "}"
Else ::= "Interior" "{" Instrucciones "}"

Expresion ::= ExpresionLogica | ExpresionAritmetica | ExpresionComparacion | ExpresionIgualdad | ExpresionNegacion | Concatenacion | ValorEstandar | LlamadaFuncion | "(" Expresion ")"

ValorEstandar::= Identificador | Entero | Flotante | String | ValorBooleano

LlamadaFuncion ::= Largo | Entrado | StringEntero | StringFlotante | (Identificador "(" (Expresion ("," Expresion)*)? ")")

ExpresionLogica ::= ExpresionLogicaTerm (OpLogico ExpresionLogicaTerm)*
ExpresionLogicaTerm ::= ExpresionComparacion | ExpresionIgualdad | "(" ExpresionLogica ")" | ValorBooleano | ExpresionNegacion

ExpresionComparacion ::= ExpresionAritmetica OpComparacion ExpresionAritmetica

ExpresionIgualdad ::= (ExpresionAritmetica | String | ValorBooleano) "CACTUS" (ExpresionAritmetica | String | ValorBooleano)

ExpresionAritmetica ::= ExpresionAritmeticaTerm (OpAritmetico ExpresionAritmeticaTerm)*
ExpresionAritmeticaTerm ::= Entero | Flotante | Identificador | "(" ExpresionAritmetica ")"

ExpresionNegacion::= "ACARO" ExpresionLogica

Concatenacion ::= (String | Identificador) "SUMATRA" (String | Identificador)

OpLogico ::= "INJERTO" | "DIOICA"
OpAritmetico ::= "SUMATRA" | "RESEDA" | "BAMBÚ" | "DIOSMA" | "DALIA" | "MAGNOLIA"
OpComparacion ::= "SECUOYA" | "BONSÁI" | "ROSA" | "LIRIO"

ValorBooleano ::= "Viva" | "Muerta"

Imprimir ::= "Podar" "(" String ")"
Largo ::= "Medir" "(" Identificador | String ")"
Entrada ::= "Recolectar" "(" String ")"
StringEntero ::= "Madurar" "(" String | Identificador ")"
StringFlotante ::= "Germinar" "(" String | Identificador ")"

CicloCasos ::= "Planta" "(" Expresion ")" "{" Instrucciones “Flor” Identificador Casos CasoDefault "}"
Casos ::= ("Hoja" "(" (String | Entero | Flotante | ValorBooleano) ")" "{" Instrucciones "}")+
CasoDefault ::= "Hoja" "(" ")" "{" Instrucciones "}"

String ::= “#” Texto “#”
Texto ::= [A-Z]?[a-z]?[0-9]?[ÁÉÍÓÚáéíóúÑñÜü]?[¡!¿?,.'"@#$%+-/*]*
Identificador ::= [a-zA-Z][a-zA-Z0-9]*
Entero ::= ("+"|"-")?[0-9]+
Flotante ::= ("+"|"-")?[0-9]+"."[0-9]+

Ejercicios de números

Ejercicio 01 - Sumatoria

```
Sembrar sumatoria(numeros) {
    *Iniciar la suma con cero*
    Maceta suma = 0

    *Verificar si el número es válido*
    Exterior(numeros BONSÁI 1) {
        Podar(#El numero debe ser mayor que cero.#)
    }

    *Realizar la sumatoria desde 1 hasta 'n'*
    Fotosintesis(numeros SECUOYA 1) {
        suma = suma SUMATRA numeros
        numeros = numeros RESEDA 1
    }

    *Retornar la suma final*
    Cosechar suma
} Florecer
```

Ejercicio 02 - Factorial

```
Sembrar factorial(plantas) {
    *Calcular el número de formas en las que se pueden organizar 'plantas'
en un jardín*
    Exterior(plantas BONSÁI 0) {
      Podar(#No se pueden organizar un número negativo de plantas#)
    }
    Maceta combinaciones = 1

    Fotosíntesis(plantas SECUOYA 1) {
        combinaciones = combinaciones BAMBÚ plantas
        plantas = plantas RESEDA 1
        Cosechar combinaciones
    }
} Florecer
```

Ejercicio 03 - Tablas de multiplicar

```
* Función que retorna la tabla de multiplicar de un número *
Sembrar tablaMultiplicar(numero, tamannoTabla) {
    **
    *Entradas: El número a multiplicar y el tamaño de la tabla.*
    *Salidas: Impresión en pantalla de la tabla de multiplicar.*
    **

    Maceta contador = 1
    Podar(#Tabla de multiplicar del número #, numero)

    Fotosintesis(contador LIRIO 10) {
         Podar(contador, # x #, numero, # = #, contador  BAMBU
numero)
        contador = contador SUMATRA 1
    }
} Florecer
tablaMultiplicar(5, 10)
```

Ejercicios de texto

Ejercicio 01 - Concatenación de Textos

```
Sembrar concatenar_textos() {
    *Definir las variables para almacenar los textos*
    Maceta texto1
    Maceta texto2
    Maceta mensaje_completo

    *Solicitar el primer texto al usuario*
    texto1 = Recolectar(#Ingrese el primer fragmento de texto: #)

    *Solicitar el segundo texto al usuario*
    texto2 = Recolectar(#Ingrese el segundo fragmento de texto: #)

    *Concatenar los textos*
    mensaje_completo = texto1 SUMATRA texto2

    *Mostrar el mensaje completo*
    Podar(#El mensaje completo es: #, mensaje_completo, #.#)
} Florecer
```

Ejercicio 02 - Menú - Estructura única

```
Sembrar menu() {
    *Asignar la condición*
    Maceta condicion = Viva

    Planta(condicion CACTUS Viva) {
        *Imprime las opciones del menú*
        Podar(#       Menú Principal   #)
        Podar(#1. Inventario#)
        Podar(#2. Ventas#)
        Podar(#3.  Salir#)
        *Preguntar al usuario por la opción que desea*
        Flor opcion = Recolectar(#Ingrese la opción a realizar: #)

        *Casos*
        *Caso 1*
        Hoja(#1#) {
            inventario()
            condicion = Muerta
        }

        *Caso 2*
        Hoja(#2#) {
            ventas()
            condicion = Muerta
        }

        *Caso 3*
        Hoja(#3#) {
            Podar(#Está saliendo del programa....#)
            condicion = Muerta
       }

       *Caso Predeterminado*
       Hoja() {
           Podar(#Opción Inválida. Vuelva a intentarlo#)
           Podar(# #)
```

```
        }
    }
} Florecer
```