**Universidad Nacional del Nordeste**

**Facultad de Ciencias Exactas y Naturales y Agrimensura**

**Carrera: Licenciatura en Sistemas de Información**

**Cátedra: Teoría de la Computación**

**Año: 2025**

# Trabajo Integrador: ñe'ẽasaha - Compilador Guaraní


**Docentes:**

Dr. Bernal, Ruben Alfredo

Dr. Irrazábal, Emanuel Agustín

**Integrantes del Grupo**:

Lezcano Airaldi, Melisa Agostina – DNI: 45.248.126

Acosta, Juan Agustín – DNI: 43532954

Marder, Manuela – DNI: 45.374.423

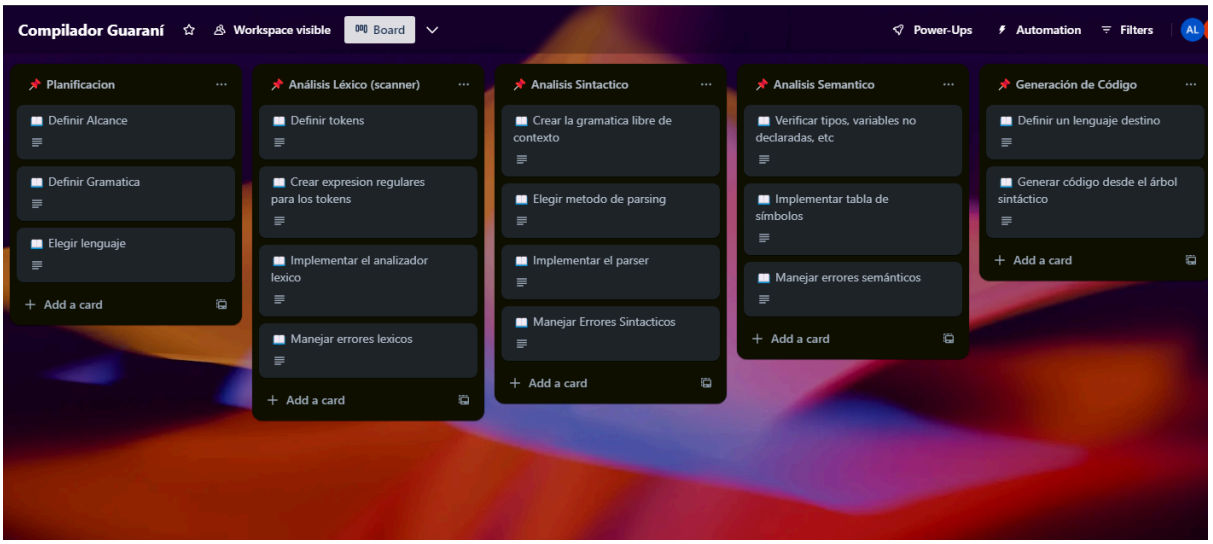
Lago, Osvaldo Agustín - DNI: 44.826.719

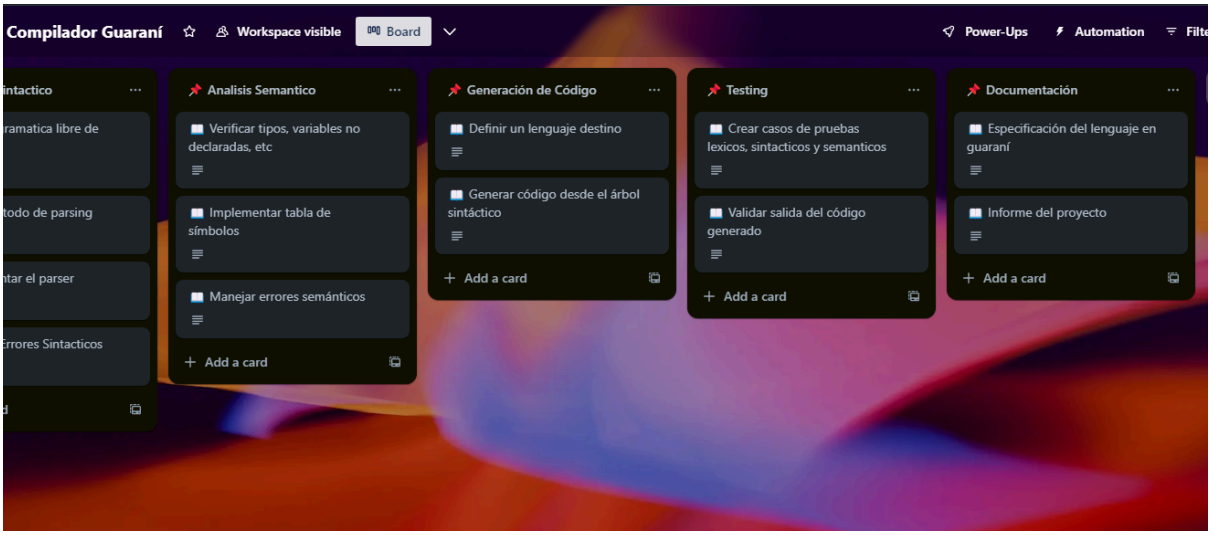
**Tema seleccionado:** Diseño de un analizador sintáctico para un lenguaje educativo en guaraní

El equipo eligió diseñar un analizador sintáctico para un lenguaje ficticio basado en palabras simples del idioma guaraní, con estructuras sintácticas de tipo sujeto-verbo-objeto. La elección se fundamenta en el interés por integrar contenidos culturales y lingüísticos locales con herramientas computacionales, y en la posibilidad de aplicar de manera didáctica los conceptos de **gramáticas libres de contexto** y **parsing descendente.**

**Reflexión sobre el avance del trabajo**

Durante esta primera etapa, el equipo logró establecer la idea central del proyecto, definir los objetivos generales y comenzar con el diseño de la gramática en notación **BNF**. También se tomaron referencias de diferentes proyectos visuales para inspirar la salida gráfica del árbol sintáctico. El diseño y el desarrollo del proyecto fue posible gracias a la utilización de herramientas colaborativas para coordinar tareas. Estas incluyen Trello, para la organización del desarrollo de la app, Discord para las reuniones y GitHub para el código compartido.






Se planteó la idea de un diseño sencillo y atractivo que facilite el uso de la herramienta para aquellos que no se encuentran en el campo de la informática.

### Implementación del analizador sintáctico

Para la implementación del analizador sintáctico se utilizará **Node.js** como entorno de ejecución, debido a su flexibilidad, velocidad y amplia comunidad de soporte. La elección de esta tecnología permite desarrollar una aplicación ligera y multiplataforma, utilizando JavaScript como lenguaje principal. Además, se contempla la integración de una interfaz gráfica simple para visualizar el árbol sintáctico generado a partir del análisis, como herramienta didáctica complementaria. El código será organizado de manera modular para facilitar su mantenimiento y futuras extensiones.

### Normas de funcionamiento del equipo

Las normas acordadas incluyen:

- Participación equitativa y activa en todas las etapas
- Asistencia y colaboración en las reuniones a través de Discord.
- Chequeo de tareas en Trello.
- Respeto por los tiempos de entrega intermedios
- Registro de avances en un documento compartido

### Resolución de conflictos

Hasta el momento no se presentaron conflictos graves. Las diferencias de criterio fueron resueltas mediante votación en reuniones. Se prioriza el respeto y la escucha activa.

**Cumplimiento de las tareas asignadas**

Cada integrante cumplió con las tareas pactadas en tiempo y forma. Se utilizó una tabla de seguimiento en Google Docs para avanzar con el desarrollo del paper. A la fecha, se completaron la definición del tema, la estructura del lenguaje, la gramática base y el diseño inicial del parser.

**Criterios para la distribución de roles. Esquema de rotación de roles:**

A pesar de que no todos los roles pudieron poner en práctica su función, fuimos flexibles y descentralizados con este formato hasta el día de la fecha; así todos aportando ideas sin un rol específico.

- Líder inicial: Lago  Agustin, encargado de organizar tareas y moderar reuniones.
- Encargado de documentación: Lezcano Melisa, responsable del resumen, BNF y redacción del informe.
- Programación: Manuela Marder, encargada del diseño del parser y pruebas.
- Revisión y pruebas: Agustin Acosta, responsable de validar salidas y errores sintácticos.

**Avances del Proyecto:**

**1.Gramática del lenguaje**

<programa>  :: = <declaraciones>

<declaraciones> ::= <declaracion>*

<declaracion> :: = <asignacion>

| <impresión>

| <condicional>

| <bucle>

| <declaracion_variable>

<asignacion> :: = IDENTIFICADOR "ha'e" <expresion>

<impresion> :: = "ñe’e" <expresion>

<condicional> :: = "ramo" <expresion> ":" <bloque> ["ambue:" <bloque>]

<bucle> :: = "guara" IDENTIFICADOR "ha'e" <expresion> ":" <bloque>

<funcion> ::= "hembiapo" IDENTIFICADOR "(" [ <parametros> ] ")" ":" <bloque>

<parámetros> ::= IDENTIFICADOR ("," IDENTIFICADOR)*

<llamada_funcion> ::= IDENTIFICADOR "(" [ <argumentos> ] ")"

<argumentos> ::= <expresion> ("," <expresion>)*

<instrucción> ::= ... | <retorno>

<retorno> ::= "jujey" <expresión>

<bloque> :: = <instrucción> | "{" <declaraciones> "}"

<declaracion_variable> ::= <tipo> <IDENTIFICADOR> [ "ha'e" <expresión> ]

<tipo> ::= "entero" | "decimal" | "booleano" | "string"

<expresion> ::= <expresion> "+" <expresion>

| <expresion> "-" <expresion>

| <expresión> "*" <expresión>

| <expresion> "/" <expresion>

| <expresion> "==" <expresion>

| <expresion> "<" <expresion>

| <expresion> "<=" <expresion>

| <expresión> "ha" <expresión>  (operador AND lógico)

| <expresión> "tera" <expresión>  (operador OR lógico)

| LITERAL_ENTERO

| LITERAL_STRING

| LITERAL_BOOLEANO

| IDENTIFICADOR

| "(" <expresión> ")"

<IDENTIFICADOR>  ::= [a-zA-Z_][a-zA-Z0-9_]*

<LITERAL_ENTERO>  ::= [0-9]+

<LITERAL_DECIMAL>  ::= [0-9]+ "." [0-9]+

<LITERAL_STRING>  ::= "\"" [^\"]* "\""

<LITERAL_BOOLEANO> ::= "true" | "false"

**2. Reglas de Producción Simplificadas**

programa  ::= <declaraciones>

declaraciones  ::= <declaración>*

declaración  ::= <asignación>

| <impresión>

| <condicional>

| <bucle>

| <declaracion_variable>

| <expresión>

**3. Ejemplos de Código**

**-Asignación**

asignación → identificador '=' expresión

Ejemplo: x ha'e 5

**-Expresiones Aritméticas**

expresion ::= termino (("+" | "-") termino)*

termino ::= factor (("*" | "/") factor)*

factor ::= número | identificador | "(" expresión ")"

Ejemplo: x ha'e 3 + 4 * 2

**-Impresión**

print → ñe’e expresión

Ejemplo: ñe’e x

**-Condicionales**

condicional → ramo '(' expresión ')' '{' declaraciones '}' [ambue '{' declaraciones '}']

ejemplo:

```
ramo (x  < 10) {
  ñe’e "es chico"
} ambue {
  ñe’e "es grande"
```

}

**-Bucle For**

bucle → guara '(' asignacion ';' expresión ';' asignacion ')' '{' declaraciones '}'

```
guara (i ha'e 0; i < 10; i ha'e i + 1) {
  ñe’e i
}
```

**-Funciones**

```
hembiapo saludar(nombre) {
    ñe’e "Hola " + nombre
}
saludar("Agustín")
hembiapo sumar(a, b) {
    egueru a + b
}
ñe’e sumar(3, 4)   // Imprime 7
```

**4. Precedencia de Operadores**

| Nivel | Operadores | Descripcion |
|---|---|---|
| 1 | () | Parentesis |
| 2 | *,/ | Multiplicacion,Division |
| 3 | +,- | Suma,Resta |
| 4 | ==, <, <= | Comparacion |
| 5 | Ha,terra | Logicos(and, or) |

**5. Árbol de Sintaxis Abstracta(AST)**

| **Nodo** | **Contenido** |
|---|---|
| BinaryExpression | Operador, lado izquierdo, lado derecho |
| Literal | Valor (numero,string,booleano) |
| Identifier | Nombre de la variable |
| Assignment | Variable y valor a asignar |
| Print | Expresión a imprimir |
| IfStatement | Condición, bloque verdadero, bloque falso |
| ForStatement | Variable, inicio, condición,bloque |
| FuncitonDef | nombre,parametros,cuerpo(lista de instr) |
| FunctionCall | nombre,argumentos |
| Return | expresión |

**Orden de Operaciones**

- Paréntesis: ()
- Multiplicación / División: * , /
- Suma / Resta: +, -
- Comparaciones: == , < , <=
- Lógicos: ha, tera

**6. Tabla de Palabras Reservadas y Tokens**

| **Palabra en Guaraní** | **Significado** | **Símbolo Tradicional** | **Tipo de Token** |
|---|---|---|---|
| ha'e | asignación | = | operador / keyword |

| | | | |
|---|---|---|---|
| guara | bucle for | for | palabra clave |
| ramo | condicional | if | palabra clave |
| ambue | sino | else | palabra clave |
| ñe’e | print | print | palabra clave |
| ha | and | & | operador logico |
| tera | or | \|\| | operador logico |
| jujey | return | return | palabra clave |
| hembiapo | declaracion de funcion | function | palabra clave |
| == | igual | == | operador logico |
| < | menor | < | operador logico |
| <= | menor igual | <= | operador logico |
| () | paréntesis | () | delimitadores |
| {} | llaves | {} | delimitadores |

| | | | |
|---|---|---|---|
| ; | punto y coma | ; | delimitador |

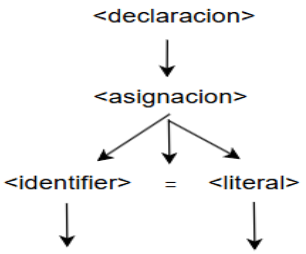
**7. Observaciones**

No se soportan comentarios.

La sintaxis está inspirada en una mezcla de estructuras de lenguajes clásicos y palabras clave en guaraní.

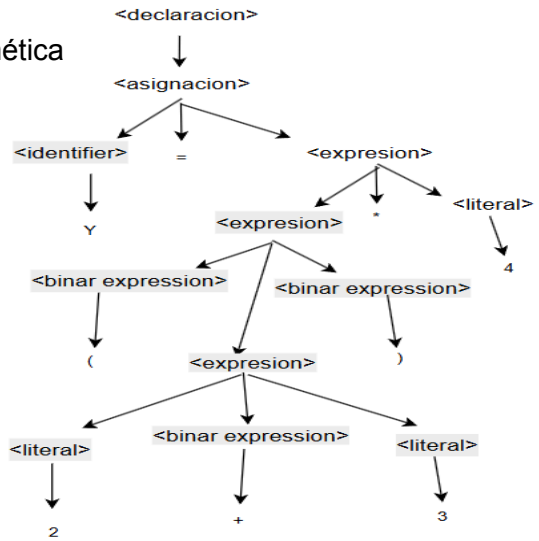
Ejemplo 1: Asignación simple

```
x ha'e 10
```



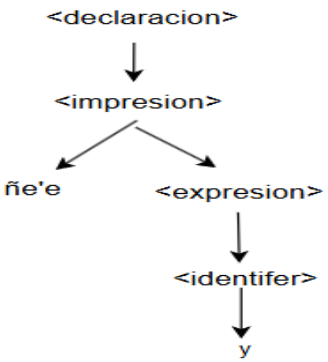

Ejemplo 2: Asignación con expresión aritmética

```
y ha'e (2 + 3) * 4
```




Ejemplo 3: Impresión de variable

```
ñe'e y
```



Ejemplo 4: Condicional con bloque

```
ramo (x < 5) {
  ñe'e "es menor"
} ambue {
  ñe'e "es mayor"
}
```