2025

Nicolas Lobos Gahona

UCN ITI

17/6/25

COMPILADOR FLEX Y BISON



**Introducción**

Este proyecto consiste en el diseño e implementación de un lenguaje de programación propio, junto con su compilador e intérprete. El trabajo abarca la definición de la gramática del lenguaje, la construcción de un Árbol de Sintaxis Abstracta (AST), y la ejecución de los programas escritos en dicho lenguaje a través de la interpretación del AST.

**Objetivo del lenguaje inventado y del compilador**

El objetivo principal del lenguaje inventado es proporcionar una herramienta sencilla para aprender y experimentar con conceptos fundamentales de programación, como variables, operaciones aritméticas, estructuras de control (condicionales y bucles), y manejo de cadenas/strings. Este compilador permite analizar, interpretar y ejecutar programas escritos en este lenguaje, facilitando así la comprensión del proceso de análisis sintáctico y semántico, así como la ejecución de instrucciones a partir de un AST.

**Definición formal de la gramática (BNF)**

*<programa>* ::= <instrucciones>

*<instrucciones>* ::= <instruccion>

| <instrucciones> <instruccion>

*<instruccion>* ::= <declaracion>

| <asignacion>

| <impresion>

| <resolucion>

| <condicional>

| <bucle>

*<declaracion>* ::= "int" <id> "=" <expresion> ";"

| "float" <id> "=" <expresion> ";"

| "str" <id> "=" <cadena> ";"

*<asignacion>* ::= <id> "=" <expresion> ";"

*<impresion>* ::= "imprimir" "(" <expresion> ")" ";"

*<resolucion>* ::= "resolver" "(" <expresion> ")" ";"

*<condicional>* ::= "if" "(" <expresion> ")" "{" <instrucciones> "}"

| "if" "(" <expresion> ")" "{" <instrucciones> "}" "else" "{" <instrucciones> "}"

*<bucle>* ::= "while" "(" <expresion> ")" "{" <instrucciones> "}"

*<expresion>* ::= <expresion> <op\_arit> <expresion>

| <expresion> <op\_comp> <expresion>

| "(" <expresion> ")"

| <numero>

| <decimal>

| <cadena>

| <id>

*<op\_arit>* ::= "+" | "-" | "\*" | "/"

*<op\_comp>* ::= "==" | "!=" | "<" | "<=" | ">" | ">="

*<numero>* ::= [0-9]+

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

*<cadena>* ::= "\"" .\*? "\""

*<id>*  ::= [a-zA-Z\_][a-zA-Z0-9\_]\*

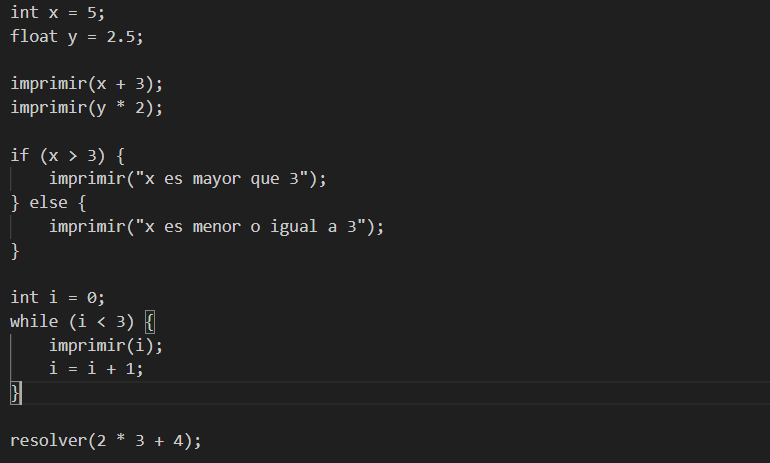
**Descripción de las reglas principales**

* *Declaración de variables:* Permite definir variables de tipo entero (int), flotante (float) o cadena (string), asignándoles un valor inicial.
* *Asignación:* Permite modificar el valor de una variable existente o al crear una.
* *Impresión (imprimir):* Muestra en pantalla el resultado de una expresión o el contenido de una variable o cadena.
* *Resolución (resolver):* Evalúa y muestra el resultado de una expresión aritmética.
* *Condicional (if/else):* Permite ejecutar instrucciones dependiendo de una condición, son bloques que se ejecutan solo si se cumple dicha condición.
* *Bucle (while):* Permite repetir un bloque de instrucciones mientras se cumple una condición.
* *Expresiones:* Soporta operaciones aritméticas y comparaciones, así como el uso de paréntesis para agrupar

***Elementos léxicos (tokens)***

* *Palabras reservadas:* int, float, str, if, else, while, imprimir, resolver
* *Identificadores:* Nombres de variables, formados por letras, dígitos y guiones bajos, comenzando por una letra o guion bajo.
* *Números enteros:* Secuencias de dígitos (Ej: 123)
* *Números decimales:* Dígitos con punto decimal (Ej: 3.14)
* *Cadenas:* Texto entre comillas dobles (Ej: "Hola mundo")
* *Operadores aritméticos:* +, -, \*, /
* *Operadores de comparación:* ==, !=, <, <=, >, >=
* *Símbolos especiales:* =, ;, (, ), {, }

***Ejemplo de sintaxis Válida***

******

***Diseño del Árbol de Sintaxis Abstracta (AST)***

**Descripción de la estructura del AST**

El Árbol de Sintaxis Abstracta (AST) es una estructura jerárquica que representa la organización lógica de un programa escrito en el lenguaje propio. Cada nodo del árbol corresponde a una construcción sintáctica, como una operación, una instrucción o un valor. El AST se construye durante el análisis sintáctico y es utilizado para la interpretación o ejecución del programa. En la implementación, cada nodo del AST es una instancia de la clase AST (definida en su header ast.hpp), que puede contener información sobre el tipo de nodo, sus hijos, valores asociados y operadores.

***Los principales tipos de nodos en el AST son:***

* **Nodos de datos:**
* *NODO\_NUMERO:* Representa un número entero.
* *NODO\_DECIMAL:* Representa un número decimal.
* *NODO\_CADENA:* Representa una cadena de texto.
* *NODO\_IDENTIFICADOR:* Representa una variable.
* **Nodos de operaciones:**
* *NODO\_BINOP:* Representa una operación binaria (suma, resta, multiplicación, división, comparación).
* *NODO\_ASIGNACION:* Representa la asignación de un valor a una variable.
* **Nodos de instrucciones:**
* *NODO\_DECLARACION:* Declaración de una variable con su tipo y valor inicial.
* *NODO\_IMPRIMIR:* Instrucción para imprimir una expresión o cadena.
* *NODO\_RESOLVER:* Instrucción para evaluar y mostrar una expresión.
* *NODO\_IF, NODO\_IF\_ELSE:* Estructuras condicionales.
* *NODO\_WHILE:* Bucle de repetición.
* *NODO\_INSTRUCCIONES:* Lista de instrucciones (bloque).

***Justificación del Diseño***

El diseño del AST que se usó tiene como objetivo ser claro y fácil de modificar. La idea fue organizar cada parte del lenguaje en tipos de nodos separados. De esta forma, cada nodo representa una parte del código redactado, lo que hace más sencillo interpretarlo y ejecutarlo después. Por ejemplo, si más adelante quisiéramos añadir funciones o nuevos operadores, se podría hacerlo sin tener que rehacer todo el sistema.

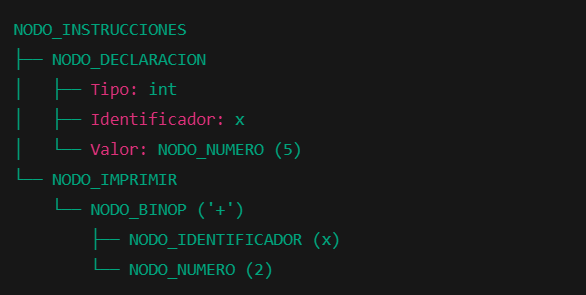
Además, separar la lógica la creación del árbol y la ejecución del programa permitió tener un código más limpio y fácil de entender. En resumen, este diseño nos pareció una buena forma de mantener el proyecto organizado, flexible y escalable para las modificaciones futuras.

***Diagrama ATS***

**Ejemplo de código (básico):**

int x = 5;

imprimir(x + 2);



Este diagrama muestra cómo el AST agrupa las instrucciones principales y cómo cada instrucción se descompone en sus componentes básicos, facilitando la interpretación y ejecución del programa.

**Ejemplo de programa con while**

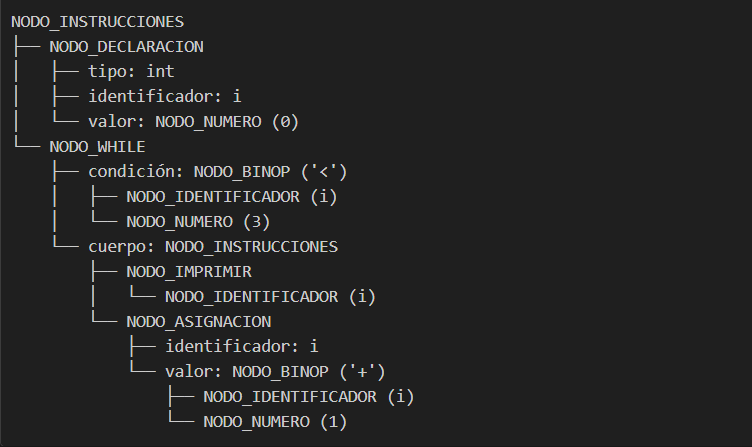
int i = 0;

while (i < 3) {

imprimir(i);

i = i + 1;

}

***ATS generado:***

***Manual***

**Características principales**

***Tipos de datos soportados:***

* int (entero)
* float (decimal)
* str (cadena de texto)

**Instrucciones básicas:**

* Declaración de variables
* Asignaciones
* Impresión (imprimir)
* Resolución de expresiones (resolver)
* Condicionales (if / else)
* Bucles (while)

***Reglas de sintaxis***

***Indicaciones de compilación y ejecución***

1. Escribe tu programa en un archivo de texto con código dentro (por ejemplo, programa.txt).
2. Abrir la terminal y compilar con los siguientes comandos:
   * + - bison -d compilador.y
       - flex compiladorflex.l
       - g++ -o compilador compilador.tab.cpp lex.yy.cpp AST.cpp -std=c++11
3. Ejecutar el programa:
   * + - Si quieres escribir directamente en la consola: ./compilador.exe
       - Si es con un archivo con código: ./compilador.exe < nom\_archivo.txt