Universidad Católica del norte

ESCUELA DE INGENIERÍA UCN COQUIMBO

# Compilador “Flaite”

## -Documentación-

Integrantes: Adrián Elgueta Navarro y Paula Nuñez Bravo.

Docente: José Luis Veas Muñoz.

Asignatura: Fundamentos de Ciencias de la Computación.

Fecha de entrega: 17 de junio de 2025.

# Índice

## Análisis Léxico.

El análisis léxico es la etapa donde el compilador convierte una cadena de caracteres en una secuencia de tokens. Se implementó usando Flex, y los principales tokens reconocidos incluyen:

| Expresión reconocida | Token devuelto | Descripción |
|---|---|---|
| "numeta" | NUMETA | Declaración de una variable entera |
| "tiene" | ASSIGN | Operador de asignación (=) |
| "suelta_la_voz" | SUELTA_LA_VOZ | Función de impresión (print) |
| "la_ura" | LA_URA | Palabra reservada para condición if |
| "en_vola" | EN_VOLA | Palabra reservada para else |
| "al_toke" | AL_TOKE | Palabra reservada para bucle while |
| "sapeate" | SAPEATE | Booleano true |
| "palabrita" | PALABRITA | Tipo de dato string |
| "funcion" | FUNCION | Declaración de funciones |
| "return" | RETURN | Retorno de funciones |
| "," | COMA | Separador de parámetros o argumentos |
| "==" | EQ | Operador de igualdad |
| "(" | LPAREN | Paréntesis izquierdo |
| ")" | RPAREN | Paréntesis derecho |

| Expresión reconocida | Token devuelto | Descripción |
|---|---|---|
| "{" | LBRACE | Llave izquierda |
| "}" | RBRACE | Llave derecha |
| ";" | SEMICOLON | Fin de instrucción |
| "!=" | NEQ | Operador de desigualdad |
| "<=" | LEQ | Menor o igual |
| ">=" | GEQ | Mayor o igual |
| "<" | LT | Menor que |
| ">" | GT | Mayor que |
| "+" | PLUS | Suma |
| "-" | MINUS | Resta |
| "*" | MULT | Multiplicación |
| "/" | DIV | División |
| \"[^\"]*\" | STRING_LITERAL | Literales de texto (strings) |
| [0-9]+ | NUM_LITERAL | Literales numéricos |
| [a-zA-Z_][a-zA-Z0-9_]* | ID | Identificadores de variables o funciones |
| [ \t\r\n]+ | — | Espacios en blanco (ignorados) |
| . | ERROR | Caracteres no reconocidos |

## Diseño del AST (Árbol de Sintaxis Abstracta)

El Árbol de Sintaxis Abstracta (AST) es una representación jerárquica de la estructura lógica del código fuente. A diferencia del árbol de derivación completo, el AST omite detalles sintácticos innecesarios como paréntesis o puntos y coma, y conserva solamente los elementos esenciales para la evaluación semántica del programa.

En este compilador, el AST se construye mediante una jerarquía de clases en C++, donde cada nodo representa una estructura del lenguaje creado: declaraciones, asignaciones, operaciones, etc.

Algunas de las principales clases que componen el AST:

- Clase Base (ASTNode): Es la clase raíz para todos los nodos del AST. Define la función virtual evaluate() que debe ser implementada por las subclases.

```
class ASTNode {
public:
    virtual ~ASTNode() {}
    virtual void evaluate();
};
```

- ExpressionNode (y derivados): Subclase base para todas las expresiones que devuelven un valor (como operaciones matemáticas, variables o llamadas a funciones).
  - NumberNode: representa valores numéricos.
  - StringNode: representa literales de texto.
  - VariableNode: accede a variables (numéricas o de tipo string)
  - BinaryOpNode: representa operaciones binarias como suma, resta, comparaciones, etc.
- Instrucciones y sentencias: Estas clases heredan de ASTNode y no retornan valores, sino que modifican el estado del programa.
  - DeclarationNode: declara una nueva variable.
  - AssignmentNode: asigna un valor a una variable.
  - PrintNode: imprime en pantalla.
  - IfElseNode: condicional.
  - WhileNode: bucle.
  - InputNode: solicita entrada del usuario.
  - DeclareStringNode: manejo de strings.

- AssignStringNode: manejo de strings.

- Ejecución: Durante la ejecución (evaluate()), los nodos realizan sus respectivas operaciones:
  - Las asignaciones actualizan el mapa “variables” o “string_vars”.
  - Las operaciones aritméticas y comparativas devuelven valores enteros.
- Variables Globales: En el AST se utilizan estructuras globales para representar el estado.

```
extern std::map<std::string, int> variables;
extern std::map<std::string, std::string> string_vars;
```

- Ejemplo:

```
numeta x;
x tiene 5;
suelta_la_voz(x);
```

  Se representaría en el AST como:

  - StatemenListNode
    - DeclarationNode(x)
    - AssignmentNode(x, NumberNode(5))
    - PrintNode(VariableNode(x))

## Análisis Sintáctico

El análisis sintáctico es una etapa fundamental del compilador que se encarga de verificar que la secuencia de tokens entregada por el analizador léxico tenga una estructura gramatical válida según las reglas del lenguaje definido.

- Objetivo del Análisis Sintáctico: Transformar una secuencia de tokens en una estructura jerárquica (el AST), validando que las construcciones del programa siguen las reglas gramaticales (ej: que una asignación se haga de la forma *x tiene 5*; y no *5 x tiene*;)

Se usó Bison como generador de analizadores sintácticos. En el archivo *parser.y*, se definieron:

  - Las reglas gramaticales
  - Las acciones semánticas que construyen los nodos del AST
  - La precedencia y asociación de los operadores (ej: "*" tiene mayor precedencia que "+")
- Ejemplo de producción sintáctica:

```
| ID ASSIGN expression SEMICOLON {
    $$ = new AssignmentNode($1, $3);
  }
```

  Esta regla permite que el compilador reconozca expresiones como *"x tiene 5;"* y construya un nodo AssignmentNode con el identificador *"x"* y el valor *"5"*.
- Manejo del árbol (AST): Cada producción del parser construye y retorna nodos del árbol de sintaxis abstracta (AST). Estas estructuras permiten luego interpretar y ejecutar el código. Por ejemplo, al analizar esta porción de código:

```
numeta a;
a tiene 10;
```

  El parser construirá un *StatementListNode* que incluye un *DeclarationNode*("a") y un *AssignmentNode*("a", *NumberNode*(10)).

## Manual de Usuario

El lenguaje *flaite* es un lenguaje de programación personalizado, inspirado en el español informal chileno. Fue diseñado para enseñar conceptos básicos de programación como variables, estructuras de control, funciones y entrada/salida, en un entorno lúdico y cercano.

- Tipos de datos:
  - **numeta**: número entero.
    ```
    numeta edad;
    ```
  - **palabrita**: cadena de texto (string).
    ```
    palabrita mensaje;
    ```

- Declaración y asignación de variables:
  - Se declara con la palabra reservada correspondiente, y se le asigna un valor con ***tiene***:
    ```
    numeta x;
    x tiene 10;

    palabrita saludo;
    saludo tiene "hola mundo";
    ```

- Entrada y salida:
  - **Entrada** (leer desde teclado):
    ```
    sapeate(edad);
    ```
  - **Salida** (imprimir por pantalla):
    ```
    suelta_la_voz("Hola!");
    suelta_la_voz(edad);
    ```

- Condicionales:
  - Se escriben con ***la_ura*** (…) {…} ***en_vola*** {…}
    ```
    la_ura (edad >= 18) {
        suelta_la_voz("Eres mayor de edad");
    } en_vola {
        suelta_la_voz("Eres menor");
    }
    ```

- Ciclos:
  - Ciclo while con la palabra clave ***al_toke***:

```
al_toke (contador < 5) {
    suelta_la_voz(contador);
    contador tiene contador + 1;
}
```

## Ejemplo completo

```
numeta base;
numeta exponente;
numeta resultado;
numeta contador;

palabrita texto;

texto tiene "Ingresa la base:";
suelta_la_voz(texto);
sapeate(base);

texto tiene "Ingresa el exponente:";
suelta_la_voz(texto);
sapeate(exponente);

resultado tiene 1;
contador tiene 0;

texto tiene "Calculando:";
suelta_la_voz(texto);
suelta_la_voz(base);
suelta_la_voz("Elevado a: ");
suelta_la_voz(exponente);

al_toke (contador < exponente) {
    resultado tiene resultado * base;

    texto tiene "Valor parcial:";
    suelta_la_voz(texto);
    suelta_la_voz(resultado);

    contador tiene contador + 1;
}

texto tiene "Resultado final:";
suelta_la_voz(texto);
suelta_la_voz(resultado);
```