

Lenguaje A

Alberto Ruiz Andrés
Antonio Sanjuán de la Mano
Mario Villacorta García





INTRODUCCIÓN

- Descripción del Proyecto
- Metodología de trabajo
- Esquema funcional
- Analizador léxico
- Analizador sintáctico
- Árbol de sintaxis abstracta
- Tabla de símbolos

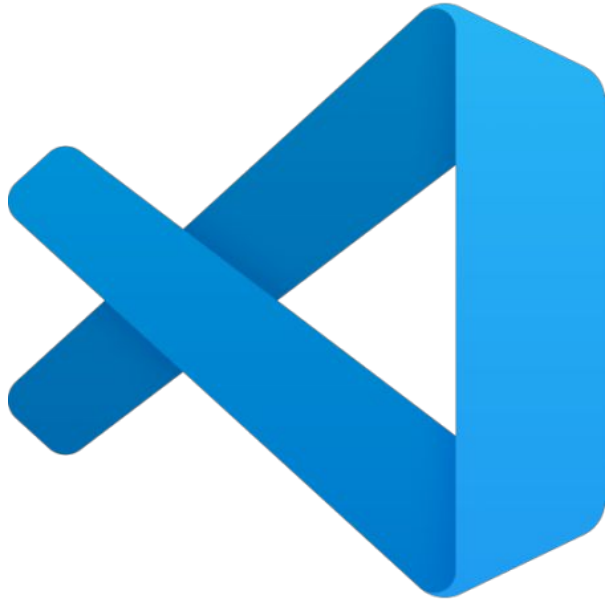


Descripción del proyecto

- **Variables de tipado débil:** Numéricas
 - Aritmética real: $+$ $-$ $*$ $/$ $^$
 - Aritmética entera: $\%$ mcd mcm
- Carácter
 - Aritmética entera: $+$ $-$ $*$ $/$ $^$ $\%$ mcd mcm
- Booleanos
 - Álgebra booleana: $!$ $\&\&$ $\|$ $==$ $!=$ $>$ $>=$ $<$ $<=$ $?:$
- **Literales cadena:** imprimibles
- **Sentencias**
 - Asignación (y as. condicional)
 - var <nombreVariable> = <expresión>;
 - var <nombreVariable> = <expresión lógica> ? <expresión T> : <expresión F>;
 - Condicional: simple (if), doble (if else)
 - Bucle: condicional (while), condicional de condición posterior (do while).
- **Funciones del sistema**
 - I/O: print, scan
 - Trigonómicas: seno, coseno, tangente, arcoseno, arcocoseno, arcotangente.
 - Aritméticas: potenciación (operador), máximo común divisor, mínimo común múltiplo, logaritmo.

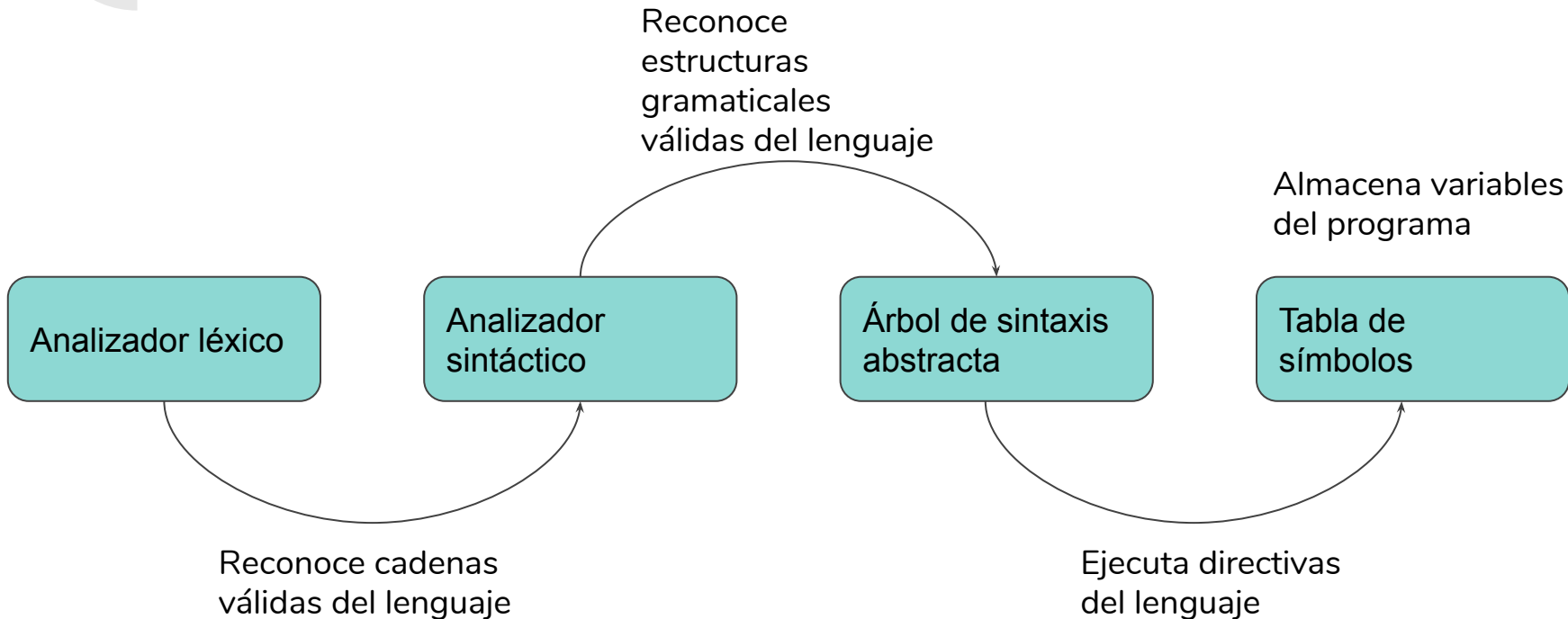


Metodología de trabajo





Esquema funcional





Analizador léxico

- *LETTER* [a-zA-Z]
- *DIGIT* [0-9]
- *DIGITS* DIGIT+
- *EXP* [eE][+-]?[DIGIT]
- *FLOAT1* DIGITS"."DIGITS
- *FLOAT2* DIGITSEXP
- *FLOAT3* DIGITS"."DIGITSEXP
- *IDENT* (_|LETTER)(_|LETTER|DIGIT)*
- *NEWLN* \n
- *COMSEP* \;
- *WSPC* [\t\f\r]
- *WSPCS* {WSPC}+
- *FLOAT* {FLOAT1}{FLOAT2}{FLOAT3}
- *OP1* [-+/*=<>?:()!^{\},%]
- *OP2* ("==""!=""<="">=""&""|""|""++""--""+=""-=")
- *STRSTART* ["]
- *CHAR* \.?\'
- *COMSIMP* \. \.
- *COMMULT* /*"([^*]| *+ [^*]) * *+ /"

Analizador sintáctico

- Un solo tipo para todos los no terminales.

- `struct sStackType {`
 - `unsigned char flag;`
 - `union {`
 - `double vFloat;`
 - `char *vStr;`
 - `struct ast_s *ast;`
 - `char vChar;`
 - `} u;`
 - `} s;`

- Evitar problemas de compatibilidad de producciones.

- Producciones principales

- `prog` -> `elemprog` | `prog elemprog`
- `elemprog` -> `sentencia` | `“,”`
 - Programa es un número indefinido de sentencias o sentencia vacía
- `sentencia` -> `asignacion “,”` | `funcionSistema “,”` | `condicional` | `mientras`
 - Modificación del estado del programa
- `condicional` -> `“if” “(” ternario “)” cuerpo` (simple o múltiple) | `“if” “(” ternario “)” cuerpo` (simple o múltiple) `“else” cuerpo` (simple o múltiple)
- `mientras` -> `“while” “(” ternario “)” cuerpo` (simple o múltiple) | `“do” cuerpo` (simple o múltiple) `“while” “(” ternario “)”`
- `cuerpo` -> `sentencia` | `cuerpo sentencia` | `“,”`
 - Reconocedor sentencias múltiples
- `ternario` -> `ternario “?” ternario “:” ternario` | `expresión`
 - Reconocedor de la aritmética del lenguaje.

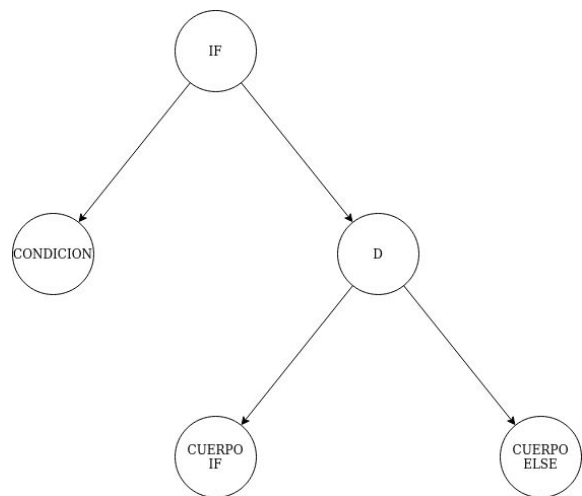
Árbol de sintaxis abstracta

```
typedef struct ast_s
{
    unsigned tag;
    unsigned lineno;
    union {
        struct
        {
            struct ast_s *lft, *rgt;
        } ptr;
        char *sVal;
        double dVal;
        char cVal;
    } u;
} ast_t;
```

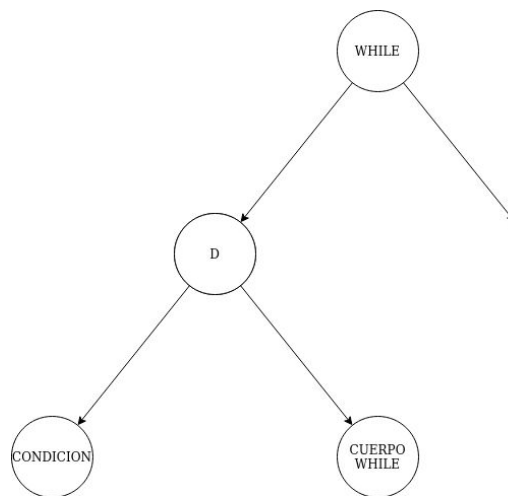
tag: se emplea para indicar la instrucción del nodo

tipo:

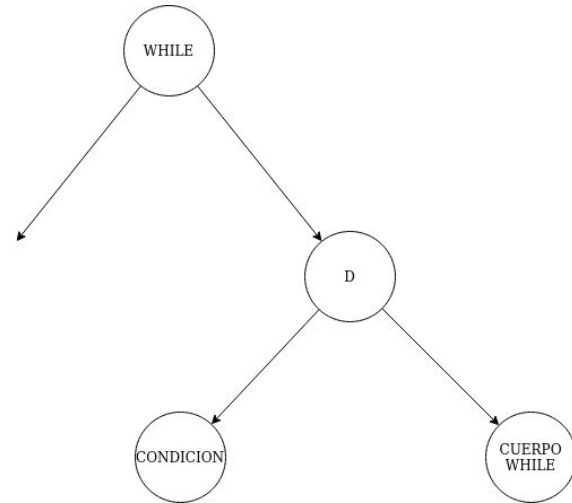
- otro árbol binario, en el caso de tratarse de un **no terminal**,
- un nodo cadena, tipo flotante o carácter, en el caso de tratarse de un **terminal**



AST CONDICIONAL DOBLE



AST BUCLE WHILE



AST BUCLE DO WHILE



Tabla de símbolos

- **symtab:** estructura tipo lista doblemente enlazada y ordenada que almacena pares clave-valor. Se almacena el nombre de la variable su valor asociado.
- **inmod:** método que busca en la lista una variable que coincida con una cadena pasada por parámetro.
- **insertModify:** se añade una entrada en la lista o se modifica el valor de una celda si alguna clave coincide con la dada.
- **get:** método que devuelve el valor del nodo si está declarado, en el caso de que sea anterior o posterior en el árbol se desplaza
- **read:** es un método que se usa para leer la variable y buscarla mediante el método get anterior.