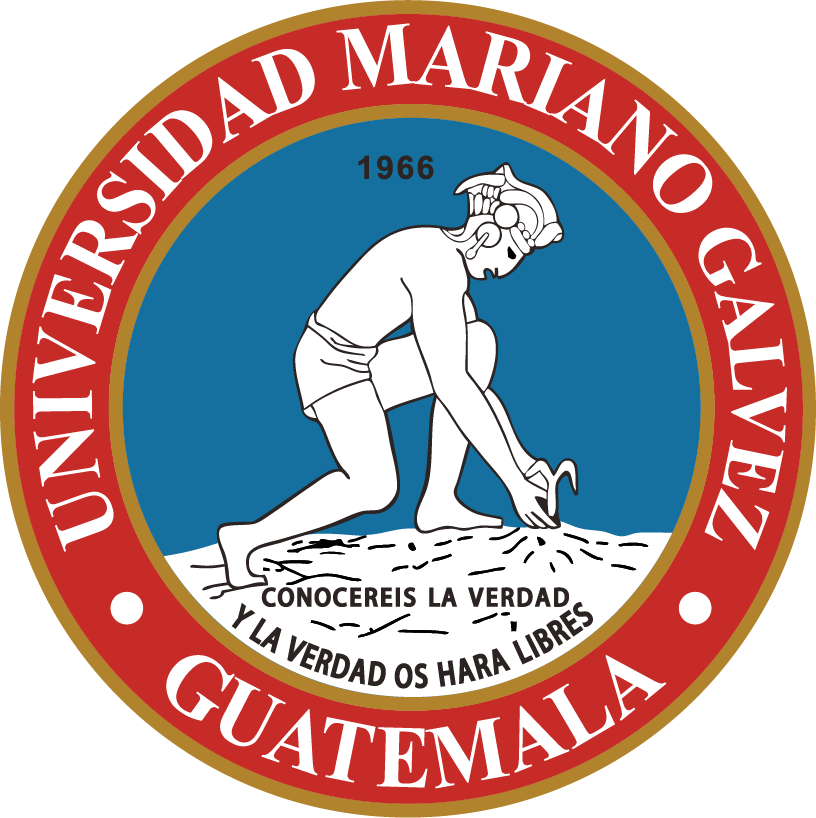
Universidad Mariano Gálvez de Guatemala

San Juan Sacatepéquez

Facultad de Ingenería en Sistemas de Información

Análisis de Sistemas 1

Ing. Catalan



Jonathan Elias Sor Monroy

75-20-1988

Guatemala 22 de Febrero del 2024

Contenido

[Introducción 3](#_Toc198582159)

[Analizador Léxico 3](#_Toc198582160)

[Análisis Sintáctico (CUP) 3](#_Toc198582161)

[Definición formal de la gramática de SimpleScript 3](#_Toc198582162)

[Patrones utilizados para reconocer los tokens 3](#_Toc198582163)

# Introducción

El presente documento técnico describe el desarrollo de un analizador léxico y sintáctico para el lenguaje de programación SimpleScript, diseñado como parte del proyecto de semestre del curso de compiladores. Este lenguaje ha sido definido con una sintaxis sencilla y estructurada, orientada a la enseñanza de conceptos fundamentales de análisis léxico, gramáticas formales y parsers.

El objetivo principal de este trabajo fue construir un traductor que valide la sintaxis de programas escritos en SimpleScript utilizando las herramientas JFlex para el análisis léxico y CUP (Construction of Useful Parsers) para el análisis sintáctico. Como parte de esta fase del proyecto, no se aborda aún la generación de código intermedio, enfocándonos exclusivamente en el reconocimiento y validación estructural del lenguaje fuente.

# Analizador Léxico

Se utilizó **JFlex** para definir los patrones que identifican los tokens del lenguaje. Entre ellos:

* **Palabras clave:** detectadas por coincidencia exacta ("IF", "DEFINE", etc.)
* **Identificadores:** letras seguidas de letras, números o guiones bajos ([a-zA-Z][a-zA-Z0-9\_]\*)
* **Números enteros y flotantes:** mediante expresiones regulares ([0-9]+ y [0-9]+\\.[0-9]+)
* **Cadenas:** comillas dobles con cualquier carácter que no sea salto de línea
* **Comentarios:** estilo // hasta el final de la línea
* **Ignorados:** espacios, tabulaciones y saltos de línea

Además, se agregó una regla para detectar cualquier carácter ilegal y registrar errores léxicos personalizados.

# Análisis Sintáctico (CUP)

Se construyó un analizador sintáctico con **CUP**, siguiendo la estructura del lenguaje SimpleScript. Se definieron:

* **Terminales y no terminales**
* **Producción inicial:** programa ::= declaraciones;
* **Reglas de precedencia** para operadores
* **Producciones para:**
  + funciones
  + expresiones aritméticas y lógicas
  + asignaciones
  + condicionales (IF, ELSE, ELSEIF)
  + ciclos WHILE y LOOP
  + impresión múltiple (PRINT arg1, arg2, ...)
* **Manejo de errores:** personalizado con mensajes indicando línea, columna y símbolo inesperado

# Definición formal de la gramática de SimpleScript

programa ::= declaraciones;  
  
declaraciones ::= declaraciones declaracion  
 | declaracion;  
  
declaracion ::= funcion  
 | definicion  
 | print  
 | asignacion  
 | condicional  
 | ciclo\_loop;  
  
funcion ::= FUNCTION ID PAR\_IZQ argumentos PAR\_DER instrucciones END;  
  
argumentos ::= argumentos COMA ID  
 | ID  
 | /\* vacío \*/;  
  
instrucciones ::= instrucciones instruccion  
 | instruccion;  
  
instruccion ::= retorno  
 | definicion  
 | print  
 | asignacion  
 | ciclo\_while  
 | ciclo\_loop  
 | condicional;  
  
ciclo\_loop ::= LOOP PAR\_IZQ asignacion\_simple PUNTOYCOMA expresion PUNTOYCOMA asignacion\_simple PAR\_DER DO instrucciones END;  
  
asignacion\_simple ::= ID ASIGNACION expresion;  
  
condicional ::= IF expresion THEN instrucciones elseif\_chain END;  
  
elseif\_chain ::= ELSE instrucciones  
 | ELSEIF expresion THEN instrucciones elseif\_chain  
 | /\* vacío \*/;  
  
ciclo\_while ::= WHILE expresion DO instrucciones END;  
  
retorno ::= RETURN expresion PUNTOYCOMA;  
  
definicion ::= DEFINE ID ASIGNACION expresion PUNTOYCOMA;  
  
asignacion ::= ID ASIGNACION expresion PUNTOYCOMA;  
  
print ::= PRINT print\_args PUNTOYCOMA;  
  
print\_args ::= print\_args COMA print\_arg  
 | print\_arg;  
  
print\_arg ::= expresion;  
  
expresion ::= expresion operador\_aritmetico expresion  
 | llamadaFuncion  
 | ID  
 | ENTERO  
 | FLOTANTE  
 | CADENA  
 | TRUE  
 | FALSE  
 | NOT expresion  
 | PAR\_IZQ expresion PAR\_DER;  
  
operador\_aritmetico ::= SUMA  
 | RESTA  
 | MULTIPLICACION  
 | DIVISION  
 | IGUALDAD  
 | DIFERENTE  
 | MENOR  
 | MAYOR  
 | MENOR\_IGUAL  
 | MAYOR\_IGUAL  
 | AND  
 | OR;  
  
llamadaFuncion ::= ID PAR\_IZQ argumentosLlamada PAR\_DER;  
  
argumentosLlamada ::= argumentosLlamada COMA expresion  
 | expresion  
 | /\* vacío \*/;

# Patrones utilizados para reconocer los tokens

| **Tipo de Token** | **Patrón en JFlex** | **Token devuelto** |
| --- | --- | --- |
| **Palabras clave** | "DEFINE" | DEFINE |
|  | "PRINT" | PRINT |
|  | "IF" | IF |
|  | "THEN" | THEN |
|  | "ELSE" | ELSE |
|  | "ELSEIF" | ELSEIF |
|  | "WHILE" | WHILE |
|  | "LOOP" | LOOP |
|  | "FUNCTION" | FUNCTION |
|  | "RETURN" | RETURN |
|  | "END" | END |
|  | "DO" | DO |
| **Booleanos** | "true" | TRUE |
|  | "false" | FALSE |
| **Operadores lógicos** | "AND" | AND |
|  | "OR" | OR |
|  | "NOT" | NOT |
| **Operadores aritméticos** | "+", "-", "\*", "/" | SUMA, RESTA, etc. |
| **Operadores relacionales** | "<=", ">=", "==", "!=", "<", ">" | MENOR\_IGUAL, etc. |
| **Símbolos** | "=", "(", ")", ",", ";" | ASIGNACION, PAR\_IZQ, etc. |
| **Entero** | [0-9]+ | ENTERO |
| **Flotante** | [0-9]+\.[0-9]+ | FLOTANTE |
| **Cadena** | \"([^\"\n]\*?)\" | CADENA |
| **Identificador** | [a-zA-Z]([a-zA-Z0-9\_])\* | ID |
| **Comentario** | "//".\* | *(ignorado)* |
| **Espacio** | [ \t\r\n]+ | *(ignorado)* |
| **Carácter ilegal** | . (cualquier otro carácter) | ERROR + mensaje a consola |