Especificación del Lenguaje Atlas de Alto Nivel

Este documento describe el vocabulario, la gramática y las características del lenguaje de alto nivel del Simulador Atlas, indicando dónde se define cada elemento en el código fuente.

1. Vocabulario (Léxico)

El análisis léxico y la definición de tokens se encuentran en **`src/compiler/Lex_analizer.py`**.

Palabras Reservadas (Keywords)

Definidas en el diccionario `reserved` (Líneas 7-32 de `Lex_analizer.py`).

Categoría	Palabras Clave	Descripción
Control de Flujo	`si`, `si_no`, `mientras`, `para`, `romper`, `continuar`, `retornar`	Estructuras de control estándar.
Tipos Enteros	`entero2` (16b), `entero4` (32b), `entero8` (64b)	Tipos numéricos de tamaño fijo.
Otros Tipos	`flotante`, `doble`, `caracter`, `cadena`, `booleano`, `vacio`	Tipos de datos básicos.
Modificadores	`con_signo`, `sin_signo`, `constante`	Modificadores de tipo.
Estructura	`funcion`, `estructura`, `externo`	Definición de bloques principales.
Memoria	`nuevo`, `eliminar`	Gestión dinámica de memoria (malloc/free).

Operadores y Delimitadores

Definidos en la lista `tokens` y reglas regex (Líneas 37-123 de `Lex_analizer.py`).

Tipo	Operadores	Variable en Código
Asignación Compuesta	`+=`, `-=`, `*=`, `/=`, `%=`	`t_PLUSEQ`, `t_MINUSEQ`, etc.
Incremento	`++`, `--`	`t_PLUSPLUS`, `t_MINUSMINUS`
Comparación	`==`, `!=`, `<=`, `>=`	`t_IGUAL`, `t_DISTINTO`, etc.
Lógicos	`&&`, `||`	`t_ANDLOG`, `t_ORLOG`
Acceso	`->`, `.`	`t_FLECHA`, `t_PUNTO`
Aritméticos	`+`, `-`, `*`, `/`, `%`	`t_MAS`, `t_MENOS`, etc.
Bitwise	`&`, `|`, `^`, `!` (NOT)	`t_AND`, `t_OR`, etc.
Delimitadores	`{`, `}`, `(`, `)`, `[`, `]`, `;`, `,`	`t_LLAVEIZQ`, `t_PUNTOCOMA`, etc.

Literales (Reglas Complejas)

Definidos mediante funciones en `Lex_analizer.py`.

- **Identificadores (`ID`)**: `[a-zA-Z][a-zA-Z0-9_]*` (Línea 129).
- **Flotantes (`FLOT`)**: Soporta notación científica `1.23e-10` (Línea 136).
- **Enteros (`ENTERO`)**: Soporta Hexadecimal `0xFF` y Decimal `123` (Línea 146).
- **Caracteres (`CARACTER`)**: `'c'` con soporte de escapes (Línea 156).
- **Cadenas (`CADENA`)**: `"texto"` con soporte de escapes (Línea 165).

---

2. Gramática (Sintaxis)

La gramática formal EBNF se encuentra en **`Documentacion/Taller2/gramatica/gramatica.ebnf`**.

Estructura General

Un programa es una secuencia de declaraciones. * **Regla:** `program ::= declaration_list` (Línea 7). * **Declaraciones:** Funciones, Estructuras o Variables Globales (Línea 11).

Funciones

- **Definición:** `funcion tipo nombre(params) { ... }`
- **Regla:** `function_decl` (Línea 17).
- **Externas:** `externo funcion ...` (Línea 21) para funciones definidas en otros módulos o ensamblador.

Estructuras (Structs)

- **Definición:** `estructura Nombre { tipo miembro; ... };`
- **Regla:** `struct_decl` (Línea 31).

Sentencias (Statements)

Definidas en la regla `statement` (Línea 69).

Sentencia	Sintaxis	Regla EBNF (Línea)
Variable	`tipo id = expr;`	`var_decl_stmt` (41)
If-Else	`si (cond) stmt [si_no stmt]`	`if_stmt` (89)
While	`mientras (cond) stmt`	`while_stmt` (91)
For	`para (init; cond; inc) stmt`	`for_stmt` (93)
Return	`retornar expr;`	`return_stmt` (99)
Bloque	`{ stmt... }`	`block` (79)

Expresiones

Jerarquía de precedencia definida desde la línea 109.

1. **Asignación:** `=`, `+=`, etc. (`assignment`, L111).
2. **Lógico OR:** `||` (`logical_or`, L117).
3. **Lógico AND:** `&&` (`logical_and`, L119).
4. **Bitwise:** `|`, `^`, `&` (L121-125).
5. **Igualdad:** `==`, `!=` (`equality`, L127).
6. **Relacional:** `<`, `<=`, `>`, `>=` (`relational`, L131).
7. **Aditiva:** `+`, `-` (`additive`, L135).
8. **Multiplicativa:** `*`, `/`, `%` (`multiplicative`, L139).
9. **Unaria:** `!`, `-`, `++`, `--`, `*` (ptr), `&` (addr) (`unary`, L143).
10. **Postfix:** `++`, `--`, `.`, `->`, `[]`, `()` (`postfix`, L147).
11. **Primaria:** Literales, Paréntesis, `nuevo`, `eliminar` (`primary`, L158).

Sistema de Tipos

Definido en la regla `type` (Línea 50). * Soporta tipos base (`entero8`, `flotante`, etc.) y punteros (`*`). * Ejemplo: `entero8**` es un puntero a puntero de entero de 64 bits.

---

3. Diagramas de Sintaxis (Railroad Diagrams)

Los diagramas de sintaxis completos se encuentran generados como imágenes en la carpeta `Documentacion/Taller2/gramatica/diagram/`.

Estos diagramas permiten visualizar la gramática de forma gráfica (Railroad Diagrams):

- **Bloques Rectangulares**: Representan **No Terminales** (reglas gramaticales complejas, ej. `expression`, `statement`).
- **Bloques Ovalados/Redondeados**: Representan **Terminales** (tokens literales como `si`, `{`, `+`, `ID`).
- **Flujo**: Se leen de izquierda a derecha siguiendo las líneas.

Ejemplos Clave

- **`if_stmt.png`**: Muestra visualmente la bifurcación opcional del `si_no`.
- **`while_stmt.png`**: Ilustra la estructura del bucle `mientras`.
- **`expression.png`**: Visualiza la jerarquía de precedencia llamando a reglas de menor nivel (ej. `assignment` -> `logical` -> … -> `primary`).

Estos diagramas corresponden exactamente a las reglas definidas en el archivo EBNF y son útiles para validar visualmente la sintaxis.

---

4. Conceptos Clave

A continuación, se describen los conceptos fundamentales implementados en el proyecto y su ubicación en el código fuente.

Concepto	Descripción	Implementación Principal
Análisis Léxico (Scanner)	Convierte el código fuente en una secuencia de tokens. Identifica palabras clave, operadores y literales.	`src/compiler/Lex_analizer.py`
Análisis Sintáctico (Parser)	Verifica que la secuencia de tokens cumpla con la gramática definida (EBNF). Construye el AST.	`src/compiler/syntax_analizer.py`
AST (Abstract Syntax Tree)	Representación jerárquica y abstracta de la estructura del código fuente.	`src/compiler/ast_nodes.py`
Análisis Semántico	Valida reglas de contexto, tipos de datos y declaraciones de variables/funciones.	`src/compiler/semantic_analyzer.py`
Tabla de Símbolos	Estructura de datos que almacena información sobre identificadores (variables, funciones) y sus atributos.	`src/compiler/symbol_table.py`
Generación de Código	Traduce el AST validado a código intermedio o ensamblador del simulador.	`src/compiler/code_generator.py`
Preprocesador	Procesa directivas preliminares antes de la compilación (ej. inclusión de archivos, macros).	`src/compiler/Preprocessor.py`
Ensamblador	Convierte el código ensamblador (mnemónicos) a código máquina binario ejecutable.	`src/compiler/ensamblador.py`

Concepto	Descripción	Implementación Principal
Linker (Enlazador)	Resuelve referencias entre módulos y combina archivos objeto en un ejecutable final.	`src/compiler/Linker.py`
Simulador (Máquina Virtual)	Emula el hardware (CPU, Memoria, E/S) para ejecutar el código máquina generado.	`src/machine/`