Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación

Fase de Compilación: Análisis Semántico

Lucía Alejandra Guzmán Domínguez 20262
Bianca Calderón 22272
Daniel Eduardo Dubón Ortiz 22233

I. Introducción general

El compilador está desarrollado en Python y utiliza **ANTLR4** para la generación del analizador léxico y sintáctico.

El presente soporta declaraciones de variables, funciones, clases, estructuras de control y operaciones aritméticas y lógicas. El compilador incluye múltiples fases: análisis léxico, sintáctico, construcción del árbol de sintaxis abstracta (AST), análisis semántico, y una interfaz web para compilar código a través de una API.

II. Requisitos del sistema

Para ejecutar y desarrollar Compiscript se requieren los siguientes componentes:

- Python 3.8+
- ANTLR 4.13.1
- Librerías de Python:
    - antlr4-python3-runtime
    - flask
    - flask-cors

III. Estructura del proyecto

El proyecto se compone de los siguientes archivos principales:

Archivo	Descripción
Compiscript.g4	Gramática del lenguaje en ANTLR
CompiscriptLexer.py	Analizador léxico generado automáticamente
CompiscriptParser.py	Analizador sintáctico generado automáticamente
CompiscriptListener.py	Listener para eventos del parser
CompiscriptVisitor.py	Visitor base para construir el AST
ast_builder.py	Construcción del árbol de sintaxis abstracta
ast_nodes.py	Definiciones de nodos del AST
SemanticListener.py	Análisis semántico, manejo de tipos y ámbitos

symbol_table.py	Tabla de símbolos (variables, funciones, clases)
symbols.py	Tipos y herramientas semánticas
PrettyErrorListener.py	Manejo y visualización de errores sintácticos
Driver.py	Programa principal que compila archivos `.cps`
server.py	Servidor Flask para compilar desde una API REST
treeutils.py	Utilidades para visualizar árboles de análisis

IV. Funcionamiento del compilador

4.1. Análisis léxico

La gramática del lenguaje definida en `Compiscript.g4` permite generar el analizador léxico usando ANTLR. Este componente convierte el código fuente en una secuencia de tokens reconocibles, como palabras clave, identificadores, literales y operadores.

4.2. Análisis sintáctico

El parser generado interpreta la estructura de los tokens según las reglas gramaticales. El resultado es un árbol de análisis sintáctico (parse tree), que puede recorrerse usando `CompiscriptListener` o `CompiscriptVisitor`.

4.3. Árbol de sintaxis abstracta (AST)

El AST se construye a partir del árbol de análisis utilizando la clase `AstBuilder`, que hereda de `CompiscriptVisitor`. Los nodos del AST están definidos en `ast_nodes.py`, utilizando la notación `@dataclass` para representar programas, declaraciones, instrucciones y expresiones de forma estructurada.

4.4. Análisis semántico

La clase `SemanticListener` realiza validaciones semánticas durante el recorrido del árbol. Verifica tipos, ámbitos, declaraciones, asignaciones y el uso correcto de estructuras del lenguaje. Utiliza una pila de ámbitos (`ScopeStack`) y una tabla de símbolos (`SymbolTable`).

V. Representación de tipos

El sistema de tipos se define en `symbols.py`. Se soportan tipos primitivos como:

- `integer`
- `boolean`

- `string`
- `float`
- `void`

También se soportan tipos compuestos como `arreglos`, y se incluyen funciones auxiliares para validar compatibilidad de tipos, operaciones válidas y asignaciones permitidas.