Informe:
Lenguaje de Programación
"EVOLA"

Curso: Compiladores
Semestre: V - 2025-1
Docente: Vicente Enrique Machaca Arceda

Integrantes: Davis Yovanny Arapa Chua
Sebastian Adriano Castro Mamani
Piero Adrian Delgado Chipana
Miguel Andres Flavio Ocharan Coaquira

1 de julio de 2025

Resumen

Este informe presenta el diseño e implementación del lenguaje de programación EVOLA, desarrollada para ilustrar los conceptos fundamentales en la construcción de compiladores. El lenguaje incluye tipos de datos básicos, estructuras de control y funciones, compilando a código ensamblador SPIM MIPS. Se detallan las especificaciones léxicas, la gramática formal, y la implementación completa del compilador incluyendo análisis léxico, sintáctico, semántico y generación de código.

Índice

1 Introducción

El lenguaje de programación EVOLA ha sido diseñado con el objetivo principal de ilustrar los conceptos fundamentales involucrados en la construcción de un compilador. La motivación detrás de su creación es ofrecer un lenguaje lo suficientemente simple como para ser comprendido y compilado en un plazo razonable, pero lo bastante completo como para demostrar las fases clave de la compilación:

1.1 Características Principales

Toma inspiración de lenguajes como C y Python en su sintaxis y semántica básica. Sus características principales incluyen:

Tipos de Datos Básicos
Soporta tipos de datos fundamentales como enteros (`int`), números de punto flotante (`float`), booleanos (`bool`) y cadenas de caracteres (`string`). También incluye el tipo `void` para funciones que no retornan valor.

Declaraciones
Permite la declaración de variables globales y locales, con la opción de inicialización en el momento de la declaración.

Estructuras de Control
- Condicionales: `if-else`
- Bucles: `while` y `for`

Funciones
Soporta la definición de funciones, incluyendo parámetros, tipos de retorno y una función `main` como punto de entrada obligatorio del programa.

Expresiones
Permite la construcción de expresiones aritméticas, relacionales y lógicas.

Entrada/Salida
Proporciona una función `print` para la salida de datos.

Compilación a SPIM MIPS
El objetivo final del compilador es producir código ensamblador ejecutable en simuladores SPIM.

Análisis Estático
Realiza análisis léxico, sintáctico y semántico, incluyendo la verificación de tipos y la gestión de ámbitos mediante una tabla de símbolos.

2 Especificación Léxica

Los tokens que conforman el lenguaje EVOLA se detallan en la siguiente tabla. Cada token se define mediante una expresión regular que describe el patrón de caracteres que lo forma.

Token	Expresión Regular	Descripción
Palabras Clave		
INT	int	Palabra reservada para el tipo de dato entero.
FLOAT	float	Palabra reservada para el tipo de dato de punto flotante.
BOOL	bool	Palabra reservada para el tipo de dato booleano.
STRING	string	Palabra reservada para el tipo de dato cadena de caracteres.
VOID	void	Palabra reservada para funciones que no retornan valor.
IF	if	Palabra reservada para la estructura de control condicional.
ELSE	else	Palabra reservada para la rama alternativa de la estructura `if`.
WHILE	while	Palabra reservada para la estructura de control de bucle `while`.
FOR	for	Palabra reservada para la estructura de control de bucle `for`.
MAIN	main	Palabra reservada para la función principal, punto de entrada del programa.
RETURN	return	Palabra reservada para retornar un valor desde una función.
PRINT	print	Palabra reservada para la función de salida estándar.
TRUE	true	Palabra reservada para el valor booleano verdadero.
FALSE	false	Palabra reservada para el valor booleano falso.
Identificadores		
ID	[a-zA-Z_][a-zA-Z0-9_]*	Nombres para variables y funciones.
Literales		
INT_NUM	\d+	Números enteros.
FLOAT_NUM	FLOAT-exp-regular	Números de punto flotante (incluye notación científica básica).

Token	Expresión Regular	Descripción
STRING_LITERAL	STR-LITERAL-exp-regular	Secuencias de caracteres encerradas entre comillas dobles.
Operadores		
PLUS	\+	Operador de suma.
MINUS	-	Operador de resta.
TIMES	*	Operador de multiplicación.
DIVIDE	/	Operador de división.
MOD	%	Operador de módulo.
EQ	==	Operador de igualdad.
NE	!=	Operador de desigualdad.
LT	<	Operador "menor que".
GT	>	Operador "mayor que".
LE	<=	Operador "menor o igual que".
GE	>=	Operador "mayor o igual que".
AND	&&	Operador lógico AND.
OR	||	Operador lógico OR.
EQUALS	=	Operador de asignación.
Puntuación		
LPAREN	(	Paréntesis izquierdo.
RPAREN	)	Paréntesis derecho.
LBRACE	{	Llave izquierda (inicio de bloque).
RBRACE	}	Llave derecha (fin de bloque).
COMMA	,	Coma (separador de parámetros/argumentos).
SEMI	;	Punto y coma (fin de sentencia).
Ignorados		
COMMENT	//.* | /*...*/	Comentarios de una línea o multilínea. Se ignoran.
WHITESPACE	[\t\n]+	Espacios, tabuladores, saltos de línea. Se ignoran.

3 Gramática

La gramática formal del lenguaje EVOLA, que define su estructura sintáctica, se presenta a continuación. Esta gramática está diseñada para ser LL(1) y es la base para el analizador sintáctico.

```
programa -> funciones

funciones -> funcion funciones
funciones ->

funcion -> tipo ID funcion_rest
funcion -> MAIN LPAREN RPAREN LBRACE bloque RBRACE
```

```
8
9  funcion_rest -> inicializacion SEMI
10 funcion_rest -> LPAREN parametros RPAREN LBRACE bloque RBRACE
11
12 parametros -> parametro parametros_rest
13 parametros ->
14
15 parametros_rest -> COMMA parametro parametros_rest
16 parametros_rest ->
17
18 parametro -> tipo ID
19
20 bloque -> instrucciones
21 bloque ->
22
23 instrucciones -> instruccion instrucciones
24 instrucciones ->
25
26 instruccion -> declaracion SEMI
27 instruccion -> For
28 instruccion -> If
29 instruccion -> Print
30 instruccion -> Return
31 instruccion -> While
32 instruccion -> ID id_rhs_instruccion
33
34 declaracion -> tipo ID inicializacion
35 inicializacion -> EQUALS exp
36 inicializacion ->
37
38 id_rhs_instruccion -> EQUALS exp SEMI
39 id_rhs_instruccion -> llamada_func SEMI
40
41 If -> IF LPAREN exp RPAREN LBRACE bloque RBRACE  Else
42 Print -> PRINT LPAREN exp_opt RPAREN SEMI
43 Else -> ELSE LBRACE bloque RBRACE
44 Else ->
45
46 While -> WHILE LPAREN exp RPAREN LBRACE bloque RBRACE
47
48 For -> FOR LPAREN for_assignment SEMI exp SEMI for_assignment RPAREN
49        LBRACE bloque RBRACE
50 for_assignment -> ID EQUALS exp
51
52 Return -> RETURN exp_opt SEMI
53 exp_opt -> exp
54 exp_opt ->
55
56 exp -> E
57 E -> C E_rest
58
59 E_rest -> OR C E_rest
60 E_rest ->
61
```

```
C -> R C_rest

C_rest -> AND R C_rest
C_rest ->

R -> T R_rest

R_rest -> EQ T R_rest
R_rest -> GE T R_rest
R_rest -> GT T R_rest
R_rest -> LE T R_rest
R_rest -> LT T R_rest
R_rest -> NE T R_rest
R_rest ->

T -> F T_rest

T_rest -> PLUS F T_rest
T_rest -> MINUS F T_rest
T_rest ->

F -> A F_rest

F_rest -> TIMES A F_rest
F_rest -> DIVIDE A F_rest
F_rest -> MOD A F_rest
F_rest ->

A -> ID llamada_func
A -> INT_NUM
A -> LPAREN exp RPAREN
A -> FLOAT_NUM
A -> STRING_LITERAL
A -> TRUE
A -> FALSE

lista_args -> exp lista_args_rest
lista_args ->

lista_args_rest -> COMMA exp lista_args_rest
lista_args_rest ->

llamada_func -> LPAREN lista_args RPAREN
llamada_func ->

tipo -> BOOL
tipo -> FLOAT
tipo -> INT
tipo -> STRING
tipo -> VOID
```

Listing 1: Gramática formal de LISC

4 Implementación

La implementación de los analizadores léxico y sintáctico, así como las fases subsiguientes del compilador para EVOLA, se encuentra disponible en el repositorio que contiene este informe.

4.1 Estructura del Proyecto

El proyecto está estructurado en varios módulos Python, cada uno encargado de una parte específica del proceso de compilación:

`AnalizadorLexico.py`
: Implementa el analizador léxico utilizando la biblioteca PLY. Define los tokens y las reglas para identificarlos en el código fuente.

`ArbolSintactico.py`
: Contiene la lógica para el análisis sintáctico predictivo LL(1) basado en una tabla de parseo (`tabla_sintactica.csv`). Construye el Árbol de Sintaxis Abstracta (AST) si el código es sintácticamente correcto. También incluye la funcionalidad para visualizar el AST con Graphviz.

`AnalizadorSintactico.py`
: Realiza el análisis semántico. Recorre el AST, gestiona la tabla de símbolos (`TablaSimbolos.py`) para verificar declaraciones, tipos, ámbitos y reportar errores semánticos.

`TablaSimbolos.py`
: Define la estructura y la lógica para la tabla de símbolos, crucial para el análisis semántico.

`GeneradorSPIM.py`
: Encargado de la fase de generación de código. Traduce el AST (validado semánticamente) a código ensamblador SPIM MIPS.

`main.py` Es el script principal que orquesta todas las fases del compilador, desde la lectura del código fuente (`codigo.txt`) hasta la generación del código ensamblado (`salida/codigo_ensamblado.asm`).

4.2 Archivos Auxiliares

`gramatica.txt`
: Archivo de texto con la gramática formal del lenguaje.

`tabla_sintactica.csv`
: Tabla de parseo LL(1) utilizada por el analizador sintáctico.

`codigo.txt` Archivo de ejemplo con código fuente en LISC.

`salida/` Directorio donde se guardan los artefactos de la compilación, como el código ensamblado.

`arbol_sintactico/`
Directorio para las visualizaciones del AST.

5 Conclusiones

El desarrollo del compilador para EVOLA ha culminado con éxito en la creación de una herramienta funcional capaz de traducir programas escritos en un lenguaje imperativo simple a código ensamblador SPIM MIPS. Los principales logros incluyen:

- Un **analizador léxico** robusto capaz de tokenizar correctamente el código fuente.
- Un **analizador sintáctico predictivo LL(1)** que valida la estructura del código y construye un Árbol de Sintaxis Abstracta (AST) representativo.
- Un **analizador semántico** que gestiona una tabla de símbolos, verifica tipos, ámbitos y detecta una variedad de errores semánticos comunes.
- Un **generador de código** que traduce eficazmente las construcciones del AST a instrucciones SPIM, manejando variables, expresiones, estructuras de control y llamadas a funciones.
- La integración de todas las fases en un flujo de compilación coherente orquestado por `main.py`.

5.1 Dificultades Encontradas

Durante el diseño e implementación del compilador EVOLA, se presentaron varios desafíos:

- **Diseño de la Gramática LL(1):** Asegurar que la gramática del lenguaje fuera libre de ambigüedades y adecuada para el análisis predictivo LL(1) requirió iteraciones y ajustes cuidadosos.
- **Integración de Fases:** Coordinar la salida de una fase como entrada para la siguiente (e.g., los tokens del léxico al sintáctico, el AST del sintáctico al semántico) y asegurar la coherencia de los datos fue un aspecto crucial.
- **Análisis Semántico:** La implementación de la tabla de símbolos, la gestión de ámbitos anidados y la correcta verificación de tipos (especialmente con promoción de tipos o conversiones implícitas) fue compleja.
- **Generación de Código SPIM:** Traducir las construcciones de alto nivel (como bucles `for` o llamadas a funciones con paso de parámetros y gestión del stack frame) a instrucciones MIPS detalladas requirió una planificación meticulosa.

En resumen, EVOLA, aunque modesto en su alcance, cumple su propósito como un lenguaje diseñado para la enseñanza y la experimentación en el campo de los compiladores, proporcionando una visión clara del viaje desde el código fuente hasta el código ejecutable.