Tecnológico de Monterrey - Campus Monterrey

BabyDuck - entrega #1 (Léxico y Sintaxis)

Desarrollo de aplicaciones avanzadas de ciencias computacionales

Profa. Elda Guadalupe Quiroga González

Daniela Ramos García A01174259

29 de abr. de 25

1. Herramienta de generación de automática de compiladores.

El compilador se desarrollará usando la herramienta de gocc, el cuál es un generador de compiladores ligero para Go que, a partir de un archivo BNF, produce de forma automática un *scanner* (DFA) y un *parser* (PDA LR-1). Se corre como una herramienta de línea de comandos (gocc <archivo>.bnf) sobre cualquier plataforma compatible con Go.

Gocc basa su *lexer* en autómatas finitos deterministas que reconocen expresiones regulares definidas en la sección léxica del BNF, e implementa un *parser* de tipo LR-1 (pushdown automaton) capaz de resolver automáticamente conflictos *shift/reduce* y *reduce/reduce* y el orden de declaración.

El formato de entrada exige un único archivo que combine la parte léxica y la parte sintáctica con posibilidad de incrustar *action expressions* en << >>. El BNF extendido permite incluir un encabezado con << import "paquete" >> para añadir código Go propio que se invoca en cada producción reconocida .

Gocc se distribuye bajo Apache License 2.0, liberando su uso y modificación en proyectos tanto libres como comerciales. Gocc ofrece un CLI con opciones para habilitar resolución automática de conflictos (-a), logging de depuración (-debug_lexer, -debug_parser), generar solo *parser* (-no_lexer), y personalizar directorio y paquete de salida (-o, -p) .

Para inyectar lógica propia, basta con insertar action expressions al final de cada alternativa de producción, devolviendo un (interface{}, error). El texto entre << y >> se evalúa en el momento del *reduce*, permitiendo construir AST o realizar traducciones directas sin escribir código fuera de la gramática . Gocc combina teoría (DFAs para el léxico, PDA LR-1 para la sintaxis) con una interfaz simple de BNF+Go, ofreciendo un flujo ágil para prototipar lenguajes y DSLs en el entorno Go.

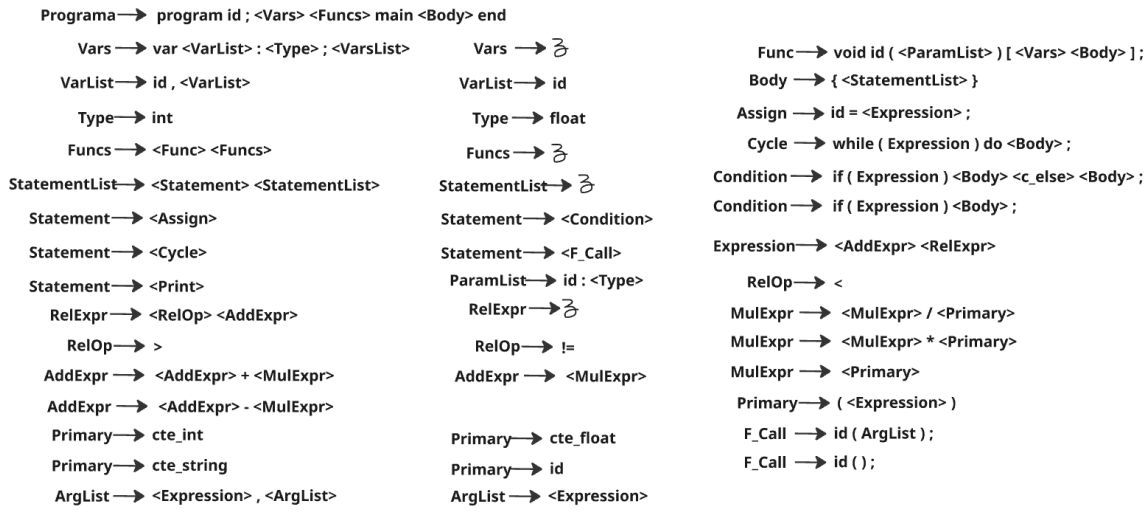
2. Expresiones Regulares

id : [a-z][a-z-]*
cte_int : [0-9]+
cte_float : [0-9]+.[0-9]+
cte_string : '[^']'
whitespace : [\t\r\n]+
comment : **(?:.|\r|\n)?**

3. Tokens reconocidos por el lenguaje.

	Nombre del Token	Lexema	Descripción
1	PROGRAM	program	Palabra reservada para iniciar un programa
2	ID	[a-z]([a-z\-])*	Identificador de variables o funciones
3	SEMICOLON	;	Fin de instrucción
4	MAIN	main	Función principal del programa
5	END	end	Palabra reservada para terminar bloques
6	LBRACE	{	Llave izquierda (inicio de bloque)
7	RBRACE	}	Llave derecha (fin de bloque)
8	ASSIGN	=	Operador de asignación
9	LT	<	Menor que (comparación)
10	GT	>	Mayor que (comparación)
11	NEQ	!=	Diferente que (comparación)
12	CTE_INT	[0-9]+	Constante entera
13	CTE_FLOAT	[0-9]+\.[0-9]{1,3}	Constante flotante
14	PLUS	+	Operador de suma
15	MINUS	-	Operador de resta
16	VOID	void	Tipo de retorno vacío
17	LPAREN	(	Paréntesis izquierdo
18	COLON	:	Dos puntos, común en declaraciones
19	COMMA	,	Separador de elementos
20	RPAREN	)	Paréntesis derecho
21	LBRACKET	[	Corchete izquierdo (arreglos)
22	RBRACKET	]	Corchete derecho (arreglos)
23	MULT	*	Operador de multiplicación
24	DIV	/	Operador de división
25	VAR	var	Palabra clave para declarar variables
26	PRINT	print	Instrucción para imprimir en pantalla
27	CTE_STRING	'[^'\n]*'	Constante de cadena
28	WHILE	while	Palabra clave para ciclo `while`
29	DO	do	Palabra clave que acompaña al `while`
30	IF	if	Palabra clave para condicional
31	ELSE	else	Alternativa al condicional `if`

4. Reglas gramaticales (CFG).



5. Declaración de sintax en BNF.

```
Start : Programa ;
Programa
   : "program" id ";" Vars Funcs "main" Body "end" ;
Vars
   : empty
   | "var" VarList ":" Type ";" Vars ;
VarList
   : id
   | id "," VarList ;
Type
   : "int"
   | "float" ;
Funcs
   : empty
   | Func Funcs ;
Func
   : "void" id "(" ")" "{" Vars Body "}" ";"
   | "void" id "(" ParamList ")" "{" Vars Body "}" ";"
;
ParamList
   : id ":" Type
   | id ":" Type "," ParamList ;
Body
   : "{" StatementList "}" ;
StatementList
   : empty
   | Statement StatementList ;
Statement
   : Assign
   | Condition
   | Cycle
   | F_Call
   | Print ;
Assign
   : id "=" Expression ";" ;
Expression
   : AddExpr RelExpr ;
RelExpr
   : empty
   | RelOp AddExpr ;
RelOp
   : "<"
   | ">"
   | "!=" ;
AddExpr
   : AddExpr "+" MulExpr
   | AddExpr "-" MulExpr
   | MulExpr ;
MulExpr
   : MulExpr "*" Primary
   | MulExpr "/" Primary
   | Primary ;
```

```
Primary
   : "(" Expression ")"
   | id
   | cte_int
   | cte_float
   | cte_string ;
Print
   : "print" "(" ")" ";"
   | "print" "(" ArgList ")" ";" ;
ArgList
   : Expression
   | Expression "," ArgList ;
Cycle
   : "while" "(" Expression ")" "do" Body ";" ;
Condition
   : "if" "(" Expression ")" Body "else" Body ";"
   | "if" "(" Expression ")" Body ";" ;
F_Call
   : id "(" ")" ";"
   | id "(" ArgList ")" ";"
```

6. Test-Plan

Alcance: Incluye pruebas de unidad al parser y al lexer, declaraciones, asignaciones, bloques, funciones, comentarios y errores sintácticos. No incluye generación de AST ni acciones semánticas.

Criterios de Aprobación

- Éxito: ≥ 99 % de los TC pasan sin errores; los errores esperados (TC4, TC5) se detectan correctamente.
- Rechazo: fallos inesperados en TC válidos o aceptación de programas sintácticamente incorrectos.

Riesgos y Mitigación

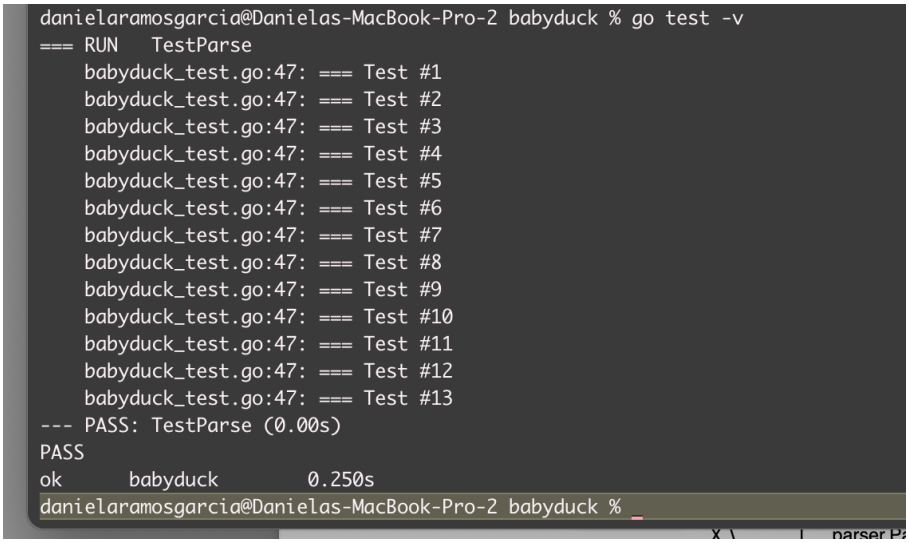
- Cambio en BNF: obliga a reescribir casos → mantener versión congelada durante pruebas.
- Conflictos LR(1): usar gocc -a para resolver automáticamente.
- Cobertura insuficiente: planear revisión de casos adicionales tras primera pasada.

Casos de Prueba.

ID	Descripción	Entrada	Resultado Esperado
TC1	Empty program	program p; main { } end	Sin errores de parseo
TC2	Variable declaration and assignment	program p; var x: int; main { x = 5; } end	Sin errores de parseo
TC3	Float constant with comment	** sample ** program p; var x: float; main { x = 3.14; } end	Sin errores de parseo

TC4	If statement	program p; var x: int; main { if (x < 10) { x = x + 1; }; } end	Sin errores de parseo
TC5	Comment handling	** Testing comments ** program p; var x: int; main { x = 5; } end	Sin errores de parseo
TC6	Print statement	program p; var x: int; main { x = 5; print(x); } end	Sin errores de parseo
TC7	Whitespace & newlines	"** Testing new line whitespace ** program p;\n\tmain { }\n\tend"	Sin errores de parseo
TC8	While loop across lines	"program p; var x: int; main\n\t{ while (x < 10) do { x = x + 1; }; }\n\tend"	Sin errores de parseo
TC9	Missing semicolon after identifier	program p main { } end	Error de parseo (falta ; tras p)
TC10	Missing end keyword	program p; main { }	Error de parseo (falta palabra end)
TC11	Missing semicolon after assignment	program p; var x: int; main { x = 5 } end	Error de parseo (falta ; tras 5)
TC12	Extra text after end	program p; var x: int; main { x = 5; } end extra	Error de parseo (texto inesperado)
TC13	Declaración de función	program p; var x: int; void f(a: int) [{ b = a + 2; }]; main { } end	Sin errores de parseo

Screenshot de ejecución de pruebas.



Liga de Github de la entrega:
https://github.com/danielaramosgarcia/Aplicaciones_avanzadas_TTC3002B/tree/main/Compiladores/babyduck