Tecnológico de Monterrey - Campus Monterrey

BabyDuck entrega 0

Desarrollo de aplicaciones avanzadas de ciencias computacionales

Profa. Elda Guadalupe Quiroga González

Daniela Ramos García A01174259

16 de abr. de 25

1.- Diseñar las Expresiones Regulares que representan a los diferentes elementos de léxico que ahí aparecen.

- ID [a-z]([a-z\-])*
- CTE_INT [0-9]+
- CTE_FLOAT [0-9]+\.[0-9]{1,3}
- CTE_STRING '[^'\n]*'
- =
- \+
- -
- *
- /
- !=
- <
- >
- ;
- :
- ,
- \(
- \)
- \{
- \}
- \[
- \]

2.- Listar todos los Tokens que serán 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`

3.- Diseñar las reglas gramaticales (Context Free Grammar) equivalentes a los diagramas.

Programa → program id ; <p_var> <p_func> main <Body> end

p_var → <VARS>

p_var → ε

p_func → <FUNCS>

p_func → ε

VARS → var <VARS'> <mas_VARS>

VARS' → id <V_id> : <TYPE> ;

mas_VARS → <VARS'> <mas_VARS>

mas_VARS → ε

V_id → , id <V_id>

V_id → ε

TYPE → int

TYPE → float

Body → { <b_STATEM> }

b_STATEM → <STATEMENT> <b_STATEM>

b_STATEM → ε

STATEMENT → <ASSIGN>

STATEMENT → <CONDITION>

STATEMENT → <CYCLE>

STATEMENT → <F_Call>

STATEMENT → <Print>

FUNCS → void id (<FUNCS'>) [<p_vars> <Body>] ;

FUNCS' → <f_funcs>

FUNCS' → ε

f_funcs → id : <TYPE> <mas_funcs>

mas_funcs → , <f_funcs>

mas_funcs → ε

ASSIGN → id = <EXPRESION> ;

CYCLE → while (EXPRESION) do CUERPO ;

CONDITION → if (EXPRESION) Body <c_else> ;

c_else → else <Body>

c_else → ε

Print → print (<Print'> <mas_Print>) ;

Print' → EXPRESION

Print' → cte.string

mas_Print → , <Print'><mas_Print>

mas_Print → ε

CTE → cte_int

CTE → cte_float

F_Call → id (<F_Call'>) ;

F_Call' → fc_exp

F_Call' → ε

fc_exp → <EXPRESION> <mas_exp>

mas_exp → , <fc_exp>

mas_exp → ε

EXPRESION → <EXP> <comp>

comp → <c_op><EXP>

comp → ε

c_op → >

c_op → <

c_op → !=

EXP → <TERMINO><EXP'>

EXP' → <e_op><EXP>

EXP' → ε

e_op → -

e_op → +

TERMINO → <FACTOR>TERMINO'>

TERMINO' → <t_op><TERMINO>

TERMINO' → ε

t_op → *

t_op → /

FACTOR → <fa_op> <fa_exp>

fa_exp → (<EXPRESION>)

fa_exp → ε

fa_op → <fa_op'> <fa_dec>

fa_op → ε

fa_op' → <e_op>

fa_op' → ε

fa_dec → id

fa_dec → CTE