EBNF Ruby

prog : expression_list;

expression_list : (expression terminator) {expression terminator};

expression : function_definition | require_block | if_statement | unless_statement | rvalue | return_statement | while_statement | for_statement ;

require_block : REQUIRE literal_t;

function_definition : function_definition_header function_definition_body END;

function_definition_body : expression_list;

function_definition_header : DEF function_name [function_definition_params] crlf ;

function_name : id_function | id ;

function_definition_params: [LEFT_RBRACKET] function_definition_params_list [RIGHT_RBRACKET]

function_definition_params_list : (id) {COMMA id};

return_statement : RETURN rvalue;

function_call : (function_name) (LEFT_RBRACKET [function_call_param_list] RIGHT_RBRACKET | function_call_param_list);

function_call_param_list : function_call_params;

function_call_params : (rvalue) {COMMA rvalue};

if_elsif_statement : ELSIF rvalue crlf expression_list [ELSE crlf expression_list] {ELSIF rvalue crlf expression_list [ELSE crlf expression_list]};

if_statement : IF rvalue (crlf | THEN) expression_list (END | ELSE | if_elsif_statement)
[[crlf] expression_list] END;

unless_statement : UNLESS rvalue crlf expression_list END;

while_statement : WHILE rvalue crlf while_expression_list END;

while_expression_list : (expression | RETRY | BREAK) terminator {(expression | RETRY | BREAK) terminator};

for_statement : FOR [LEFT_RBRACKET] expression SEMICOLON expression SEMICOLON expression [RIGHT_RBRACKET] crlf for_expression_list END;

for_expression_list : (expression | RETRY | BREAK) terminator {(expression | RETRY | BREAK) terminator};

assignment : lvalue (PLUS_ASSIGN | MINUS_ASSIGN | MUL_ASSIGN | DIV_ASSIGN | MOD_ASSIGN | EXP_ASSIGN | ASSIGN) rvalue;

array_assignment : (lvalue) (array_definition ASSIGN rvalue | ASSIGN array_definition);

array_definition : LEFT_SBRACKET [array_definition_elements] RIGHT_SBRACKET;

array_definition_elements : (rvalue) {COMMA rvalue};

array_selector : (id | id_global | function_call) LEFT_SBRACKET rvalue RIGHT_SBRACKET;

int_result : (int_t) (MUL | DIV | MOD | PLUS | MINUS) int_t {(MUL | DIV | MOD | PLUS | MINUS) int_t};

float_result : (float_t | int_result) (MUL | DIV | MOD | PLUS | MINUS) (float_t | int_result) ({(MUL | DIV | MOD | PLUS | MINUS) (float_t | int_result)};

string_result : [int_result MUL] literal_t {MUL int_result};

lvalue : id | id_global ;

rvalue : [LEFT_RBRACKET] [NOT | BIT_NOT]
((lvalue | array_assignment | int_result | float_result | string_result | assignment | function_call | literal_t | bool_t | float_t | int_t | nil_t)
(EXP | MUL | DIV | MOD | PLUS | MINUS | BIT_SHL | BIT_SHR | BIT_AND | BIT_OR | BIT_XOR | LESS | GREATER | LESS_EQUAL | GREATER_EQUAL | EQUAL | NOT_EQUAL | OR | AND)
(lvalue | array_assignment | int_result | float_result | string_result | assignment | function_call | literal_t | bool_t | float_t | int_t | nil_t))
[RIGHT_RBRACKET]
{[LEFT_RBRACKET] [NOT | BIT_NOT]
(lvalue | array_assignment | int_result | float_result | string_result | assignment | function_call | literal_t | bool_t | float_t | int_t | nil_t)
(EXP | MUL | DIV | MOD |PLUS | MINUS | BIT_SHL | BIT_SHR | BIT_AND | BIT_OR | BIT_XOR | LESS | GREATER | LESS_EQUAL | GREATER_EQUAL | EQUAL | NOT_EQUAL | OR | AND)
(lvalue | array_assignment | int_result | float_result | string_result | assignment | function_call | literal_t | bool_t | float_t | int_t | nil_t)) [RIGHT_RBRACKET]};

literal_t : LITERAL;

float_t : FLOAT;

int_t : INT;

bool_t : TRUE | FALSE ;

nil_t : NIL;

id : ID;

id_global : ID_GLOBAL;

id_function : ID_FUNCTION;

terminator : (SEMICOLON | crlf) {SEMICOLON | crlf};

crlf : CRLF ;

ESCAPED_QUOTE : '\\"';

LITERAL : '"' (ESCAPED_QUOTE | ~('\n'|'\r'))*? '"' | '\'' (ESCAPED_QUOTE | ~('\n'|'\r'))*? '\'';

COMMA : ',';

SEMICOLON : ';';

CRLF : '\n';

REQUIRE : 'require';

END : 'end';

DEF : 'def';

RETURN : 'return';

IF: 'if';

THEN : 'then';

ELSE : 'else';

ELSIF : 'elsif';

UNLESS : 'unless';

WHILE : 'while';

RETRY : 'retry';

BREAK : 'break';

FOR : 'for';

TRUE : 'true';

FALSE : 'false';

PLUS : '+';

MINUS : '-';

MUL : '*';

DIV : '/';

MOD : '%';

EXP : '**';

EQUAL : '==';

NOT_EQUAL : '!=';

GREATER : '>';

LESS : '<';

LESS_EQUAL : '<=';

GREATER_EQUAL : '>=';

ASSIGN : '=';

PLUS_ASSIGN : '+=';

MINUS_ASSIGN : '-=';

MUL_ASSIGN : '*=';

DIV_ASSIGN : '/=';

MOD_ASSIGN : '%=';

EXP_ASSIGN : '**=';

BIT_AND : '&';

BIT_OR : '|';

BIT_XOR : '^';

BIT_NOT : '~';

BIT_SHL : '<<';

BIT_SHR : '>>';

AND : 'and' | '&&';

OR : 'or' | '||';

NOT : 'not' | '!';

LEFT_RBRACKET : '(';

RIGHT_RBRACKET : ')';

LEFT_SBRACKET : '[';

RIGHT_SBRACKET : ']';

NIL : 'nil';

SL_COMMENT : ('#' ~('\r' | '\n')* '\n') -> skip;

ML_COMMENT : ('=begin' .*? '=end\n') -> skip;

WS : (' '|'\t')+ -> skip;

INT : [0-9]+;

FLOAT : [0-9]+'.'[0-9]+;

ID : [a-zA-Z_][a-zA-Z0-9_]*;

ID_GLOBAL : DOLLAR ID;

ID_FUNCTION : ID [!?];

DOLLAR : ‘$’;

Tabla de Tokens de Ruby

TOKEN	PATRÓN	LEXEMA
require	require	require
nil	nil	nil
when	when	when
def	def	def
false	false	false
not	not	not
while	while	while
for	for	for
then	then	then
do	do	do
if	if	if
true	true	true
begin	begin	begin
else	else	else
break	break	break
elsif	elsif	elsif
retry	retry	retry
unless	unless	unless
case	case	case
end	end	end
return	return	return
until	until	until
comma	,	,
semicolon	;	;
escaped_quote	\\”	\\”

crlf	\n	\n
plus	+	+
minus	-	-
div	/	/
mul	*	*
mod	%	%
exp	**	**
equal	==	==
not_equal	!=	!=
greater	>	>
less	<	<
less_equal	<=	<=
greater_equal	>=	>=
assign	=	=
pluss_assign	+=	+=
minus_assign	-=	-=
mul_assign	*=	*=
div_assign	/=	/=
mod_assign	%=	%=
exp_assign	**=	**=
bit_and	&	&
bit_or	|	|
bit_xor	^	^
bit_not	~	~
bit_shl	<<	<<
bit_shr	>>	>>
and	and | &&	and

or	or | ||	or
not	not | !	not
left_rbracket	(	(
right_rbracket	)	)
rigth_sbracket	[	[
left_sbracket	]	]
INT	[0,9]+	1
FLOAT	[0-9]+'.'[0-9]+;	1.1
DOLLAR	$	$
IDFUNCTION	[a-zA-Z_][a-zA-Z0-9_]*[! | ?]	empty!
ID	[a-zA-Z_][a-zA-Z0-9_]*	hola

*****El analizador léxico tendrá que tener en cuenta además los comentarios, espacios y tabulaciones.

- **SL_COMMENT** cuyo patrón es **('#' ~('\r' | '\n)*' '\n') -> skip;**
- **ML-COMMENT** cuyo patrón es **('=begin' .*? '=end\n') -> skip;**
- **WS cuyo** patrón es **(‘ ‘|’\t’)+ -> skip;**

Reparto del trabajo Entregable 1

NOMBRE	REPARTO DE TAREA
Raúl García – Hidalgo Tajuelo	10
Eimard Sobrino Zurera	10
Juan Alfredo García García	10
Eusebio Guijarro Collado	10