BNF RESUMIDA DA LINGUAGEM PERL

Lael Santa Rosa; Murilo Urquiza; Thalia Barbosa; Ana Ferreira

{ Perl é uma linguagem de programação de uso geral originalmente desenvolvida para manipulação de texto e agora usada para uma ampla gama de tarefas, incluindo administração de sistemas, desenvolvimento web, programação de rede, desenvolvimento de GUI e muito mais.

A linguagem pretende ser prática (fácil de usar, eficiente, completa) ao invés de bonita (pequena, elegante, mínima). Seus principais recursos são que é fácil de usar, suporta programação procedural e orientada a objetos (OO), possui suporte integrado poderoso para processamento de texto e possui uma das coleções de módulos de terceiros mais impressionantes do mundo. }

```
tokens=[
        // generated tokens
        SUBROUTINE_IDENTIFIER =  'SUBROUTINE_IDENTIFIER'
        SCALAR_IDENTIFIER = [$][a-z | A-Z | 0-9]+
        VECTOR_IDENTIFIER = [@][a-z | A-Z | 0-9]+
        FLOAT_NUMBER = 'FLOAT_NUMBER'
        INTEGER_NUMBER = 'INTEGER_NUMBER'
        STRING = STRING'
        ARRAY_OF_NUMBERS = 'ARRAY_OF_NUMBERS'
        ARRAY_OF_STRINGS = 'ARRAY_OF_STRINGS'
        FUNC_IDENTIFIER = 'FUNC_IDENTIFIER '

        // operators
        OPERATOR_LT_NUMERIC = '<'
        OPERATOR_GT_NUMERIC = '>'
        OPERATOR_PLUS_PLUS = '++'
        OPERATOR_MINUS_MINUS = '--'
        OPERATOR_AND = '&&'
        OPERATOR_OR = '||'
        OPERATOR_NOT = '!'
        OPERATOR_DIV = '/'
        OPERATOR_MUL = '*'
        OPERATOR_MOD = '%'
        OPERATOR_PLUS = '+'
        OPERATOR_MINUS = '-'
        OPERATOR_GE_NUMERIC = '>='
        OPERATOR_LE_NUMERIC = '<='
        OPERATOR_EQ_NUMERIC = '=='
        OPERATOR_NE_NUMERIC = '!='
        OPERATOR_ASSIGN = '='

        LEFT_PAREN = '('
        RIGHT_PAREN = ')'
        LEFT_BRACE = '{'
        RIGHT_BRACE = '}'
        LEFT_BRACKETS = ‘[’
        RIGHT_BRACKETS = ‘]’
        SEMICOLON = ';'
        COMMENT_LINE = ‘#.*’

        // reserveds
        RESERVED_IF = 'if'
        RESERVED_ELSIF = 'elsif'
        RESERVED_ELSE = 'else'
        RESERVED_WHILE = 'while'
        RESERVED_DEFAULT = 'default'
        RESERVED_ASYNC = 'async'
        RESERVED_PACKAGE = 'package'
        RESERVED_USE = 'use'
        RESERVED_PRINT = 'print'
        RESERVED_SAY = 'say'
        RESERVED_TRY = 'try'
        RESERVED_CATCH = 'catch'
        RESERVED_SWITCH = 'switch'
        RESERVED_CASE = 'case'
        RESERVED_RETURN = 'return'
        RESERVED_EXIT = 'exit'
        RESERVED_FUNC = 'func'
        RESERVED_SUBROUTINE = ‘sub’’
]

<string_special_tab> ::= '\t'
<string_special_newline> ::= '\n'
<string_special_return> ::= '\r'
<string_special_space> ::= ' '
<string_special_null> ::= ''

<number> ::= [0-9]+
<letter> ::= [a-z | A-Z]+

// BNF INICIAL

<root> ::= <file_items>

<file_items> ::= <file_item>*

<file_item> ::= !<<eof>>
{
        <namespace_definition>
        | <user_vars_declarations> [statement_item]
        | <statement_item>
}

// package
```

<namespace_definition> ::= <namespace_definition_name> <block>

<namespace_definition_name> ::= 'package' <any_package>

<any_package> ::= 'package::name'

<block> ::= '{' <block_content> '}'

<block_content> ::= <file_item>*

// vars
<use_vars_declarations> ::= <scalar_declaration> | <vector_declaration>

<scalar_declaration> ::=
<scalar_identifier> <semicolon>
| <scalar_identifier> = STRING <semicolon>
| <scalar_identifier> = INTEGER_NUMBER <semicolon>
| <scalar_identifier> = FLOAT_NUMBER <semicolon>

<scalar_identifier> ::= <sigil_scalar><name_scalar>

<sigil_scalar> ::= '$'

<name_scalar> ::= (<number> | <letter>)$^+$

<vector_declaration> ::=
<vector_identifier> <semicolon>
| <vector_identifier> = ARRAY_OF_NUMBERS <semicolon>
| <vector_identifier> = ARRAY_OF_STRINGS <semicolon>

<vector_identifier> ::= <sigil_vector><name_vector>

<sigil_vector> ::= '@'

<name_vector> ::= (<number> | <letter>)$^+$

// print and say
<print_string_or_scalar> ::=
<reserved_print> “(<print_possibilities> <string_special_space>)*(<print_possibilities>)”
<print_possibilities> ::= <name_or_null> | <scalar_identifier> | <string_special_newline> | <string_special_tab> | <string_special_space>
<print_vector> ::= <reserved_print> <sigil_scalar><name_vector>'['<number>']'

<say_string_or_scalar> ::=
<reserved_say> “(<print_possibilities> <string_special_space>)*(<print_possibilities>)”
<say_vector> ::= <reserved_say> <sigil_scalar><name_vector>'['<number>']'

<name_or_null> ::= [a-z | A-Z | 0-9]*

```
// statement
<statement_item> ::=
<sub_definition>              // 1
| <compound_statement> // 2
| <statement>                 // 3

<sub_definition> ::= ['async'] 'sub' <sub_names_token>    // 1
<sub_names_token> ::= ['package::name::'] <subname>   // 1
<subname> ::= (<number> | <letter>)+                        // 1

<compound_statement> ::= // 2
<if_compound>             // 2.1
| <while_compound>      // 2.2
| <default_compound>   // 2.3
| <trycatch_compound> // 2.4
| <switch_compound>   // 2.5
| <cases_sequence>    // 2.6

<if_compound> ::= 'if' <conditional_block> <if_compound_elsif>* [<if_compound_else>] // 2.1
<if_compound_elsif> ::= 'elsif' <conditional_block>                                  // 2.1
<if_compound_else> ::= 'else' <unconditional_block>                                  // 2.1
<unconditional_block> ::= <block>                                                    // 2.1
<conditional_block> ::= <condition_expr> <block>                                    // 2.1
<condition_expr> ::= <parse_parenthesized_expression>                                // 2.1

<while_compound> ::= 'while' <parse_conditional_block>                          // 2.2
<parse_conditional_block> ::= <condition_expr> <block>                          // 2.2
<condition_expr> ::= <parse_parenthesized_expression>                           // 2.2
<parse_parenthesized_expression> ::= '(' <parenthesised_expr_content> ')'        // 2.2
<parenthesised_expr_content> ::= <expr> {recoverWhile=recover_parenthesised} // 2.2
<recover_parenthesised> ::= !(')' | '{' | '}' | <<checkSemicolon>>                // 2.2

<default_compound> ::= 'default' <block>     // 2.3

<trycatch_compound> ::= 'use TryCatch;' <try_expr> <block> [<catch_expr> <block>]* // 2.4
<try_expr> ::= 'try'                                                               // 2.4
<catch_expr> ::= 'catch' [<catch_condition>]                                       // 2.4
<catch_condition> ::= '(' <variable_declaration_element> ')'                       // 2.4
<variable_declaration_element> ::= <sigil_scalar><name_scalar>                     // 2.4

<switch_compound> ::= 'switch' <switch_condition> <block>             // 2.5
<switch_condition> ::= '(' <name_scalar> ')' '{' <cases_sequence> '}' // 2.5

<cases_sequence> ::=<case_compound>+ [<case_default>] // 2.6
<case_compound> ::= 'case' <case_condition> <block>        // 2.6
<case_condition> ::=
STRING
```

```
| FLOAT_NUMBER
| INTEGER_NUMBER
| SCALAR_IDENTIFIER                                                   // 2.6
<case_default> ::= <if_compound_else>                                 // 2.6

<statement> ::= <sub_declaration>                                                    // 3
<sub_declaration> ::= 'sub' <sub_names_token> '(' <sub_declaration_parameters> ')' // 3
<sub_declaration_parameters> ::= <attribute> (',' <attribute>)*                      // 3
<attribute> ::= SUBROUTINE_IDENTIFIER                                                // 3

// return and exit
<return_expr> ::= 'return' [<parse_list_expr>]
<parse_list_expr> ::=
STRING
| FLOAT_NUMBER
| INTEGER_NUMBER
| SCALAR_IDENTIFIER

<exit_expr> ::= 'exit' [<optional_scalar_expr_arguments>]
<optional_scalar_expr_arguments> ::= '(' <number> ')' | <number>

// use and sub
<parse_use_statement> ::= 'use' <any_package>

<sub_expr> ::=
['async'] 'sub' <sub_names_token> [ '(' <sub_definition_parameters> ')' ] <block>
<sub_definition_parameters> ::= <attribute> (',' <attribute>)*

// expression
<expr> ::=
| <equal_expr>              // 1
| <compare_expr>            // 2
| <add_expr>                // 3
| <mul_expr>                // 4
| <op_5_expr>               // 5
| <op_3_expr>               // 6
| <and_expr>                // 7
| <or_expr>                 // 8
| <assign_or_flow_expr>     // 9
| <deref_expr>              //10
| <atom_expr>               // 11

<equal_expr> ::= <expr> ({'=='|'!='} <expr>)+ // 1

<compare_expr> ::= <expr> ({'>='|'<='|'>'|'<'} <expr> )+ // 2

<mul_expr> ::= <expr> ({'*'|'/'|'%'} <expr>)+ // 3
```

<add_expr> ::= <expr> ({'+'|'-'} <expr>)+ // 4

<op_5_expr> ::= <prefix_unary_expr> // 5
<prefix_unary_expr> ::= {'!'} <expr>   // 5

<op_3_expr> ::= <pref_pp_expr> | <suff_pp_expr>   // 6
<pref_pp_expr> ::= ('++'|'--') <expr>   // 6
<suff_pp_expr> ::= <expr> ('++'|'--')   // 6

<and_expr> ::= <expr> ({'&&'} <expr>)+   // 7

<or_expr> ::= <expr> ({'||'} <expr>)+   // 8

<assign_or_flow_expr> ::= <assign_expr>   // 9
<assign_expr> ::= <expr> ({'='} <expr>)+   // 9

<deref_expr> ::= <array_index>   //10
<array_index> ::= <sigil_scalar><name_vector>'[' <number> ']'   //10

<atom_expr> ::=   // 11
INTEGER_NUMBER   // 11
| FLOAT_NUMBER   // 11
| <scalar_identifier>   // 11 (not string)
| <sub_names_token> // 11