LFPC Lab №3

Syntax definition:

```
// var decs
int i = 10;
string s = "abc";
double d = 1.2;

// function dec
fun add(int i1, int i2) ret int {
        return i1 + i2;
}

// recursive functio
fun fib(int n) ret int {
        if (n == 0) return 0;
        if (n == 1) return 1;
        else fib(n - 1) + fib(n - 2);
}

int i = add(2, 1);

// arrays
array<string> a = {"a", "b", "c"};
string s = a[0];
```

Code example:

```
// code example
fun addValues(array<int> a) ret int {
        int sum = 0;
        for (int i = 0 to length(a); i++) {
                sum += a[i];
        }
}

fun main() ret void {
array<int> a = {1, 2, 3, 5};

int sum = addValues(a);
}
```

Grammar of the language:

Table 1: EBNF meta notation

<x>	means x is non-terminal
x or ‘x’	means x is a terminal
[x]	means x is optional (0 or 1 occurences of x)
x*	means 0 or more occurences of x
x+	means 1 or more occurences of x
“|”	separates alternatives
“{” and “}”	are used for grouping alternatives

The grammar of the DSL is G = (V_N, V_T, S, P) where:

V_N = {<program>, <statements>, <statement>, <nosemicolon_statement>, <semicolon_statement>, <ctrlflow_statement>, <block>, <comment>, <return_statement>, <expression>, <assignment>, <for_statement>, <if_statement>, <declaration>, <function_dec>, <parameter>, <variable_dec>, <variable_init>, <type>, <scalar_type>, <multid_type>, <function_call>, <prefix_expression>, <infix_expression>, <postfix_expression>, <bracket_expression>, <paranthesis_expression>, <identifier>, <number>, <integer>, <double>, <character>, <string>, <digit>, <nonzero_digit>, <operators>, <infix_op>, <postfix_op>, <prefix_op>},

V_T = {‘;’, //, [,], {, }, (,), ‘,’, +, -, ++, --, +=, -=, !, ==, <=, >=, >, <, %, *, /, array, int, double, break, for, if, else, void, ret, fun, ‘_’, a, b, c, ... z, A, B, ... Z, 0, 1, ... 9},

S = <program>,

P = {

STATEMENTS:

<program> ::= <statements>*

<statements> ::= <statement>*

<statement> ::= <nosemicolon_statement> | {<semicolon_statement> ‘;’ }

<nosemicolon_statement> ::= <ctrlflow_statement> | <block> | <comment>

<semicolon_statement> ::= <declaration> | <return_statement> | <expression> | <assignment> | **break**

<return_statement> ::= **return** <expression>

<ctrlflow_statement> ::= <for_statement> | <if_statement>

<for_statement> ::= **for (** <variable_dec> **to** <expression> **;** <expression> **)** <statement>

<if_statement> ::= **if (**<expression>**)** <statement> [<else_statement>]

<else_statement> ::= **else** {<if_statement> | <statement>}

<comment> ::= ‘//’ <character>*

<block> ::= **{** <statements>* **}**

<return_type> ::= <type> | **void**

<assignment> ::= <identifier> **=** <expression>

DECLARATIONS:

<declaration> ::= <function_dec> | <variable_dec>

<function_dec> ::= **fun** <identifier> **(**<parameter>*) **ret** <return_type> <block>

<parameter> ::= <type> <identifier> [‘,’]

<variable_dec> ::= <type> <identifier> [<variable_init>]

<variable_init> ::= ‘=’ <expression>

DATA TYPES:

<type> ::= <scalar_type> | <multidim_type>

<scalar_type> ::= **int** | **double** | **string**

<multidim_type> ::= **array<**<scalar_type>**>** <identifier>

EXPRESSIONS:

<expression> ::= <identifier> | <number> | <prefix_expression> | <infix_expression> | <postfix_expression> | <bracket_expression> | <paranthesis_expression> | <cbracket_expression> | <function_call>

<function_call> ::= <identifier> **(** {<expression> ‘,’}*)

<prefix_expression> ::= <prefix_op> <expression>

<postfix_expression> ::= <expression> <postfix_op>

<infix_expression> ::= <expression> <infix_op> <expression>

<paranthesis_expression> ::= **(**<expression>**)**

<cbracket_expression> ::= **{**<expression>**}**

<bracket_expression> ::= [<expression>] ‘[’ <expression> ‘]’

IDENTIFIERS, NUMBERS, OPERATORS:

<identifier> ::= <character> {<character> | <digit>}*

<number> ::= <integer> | <double>

<integer> ::= <nonzero_digit><digit>

<double> ::= <integer> ‘.’ <digit>*

<character> ::= **a** | **b** | **c** . . . **z** | **A** | **B** . . . **Z** | _

<digit> ::= **0** | **1** | . . . | **9**

<nonzero_digit> ::= **1** | . . . | **9**

<operators> ::= <infix_op> | <prefix_op> | <postfix_op>

<infix_op> ::= + | **-** | && | || | % | == | / | * | < | <= | > | >= | , | ;

<prefix_op> ::= ++ | **--** | **!**

<postfix_op> ::= ++ | − | , | ;

}

Note: <string> is any combination of UTF-8 characters surrounded by double quotes.