REFERENCE GRAMMAR

G(L) = {*Vn, Vt, S, P*}, where

Vn – nonterminal symbols,

Vt – terminal symbols,

S – starting symbol,

P – finite set of production rules.

The "|" symbol is used to separate the alternatives.

The text between ? ? is the description of the terminal symbol

Symbols written in **bold** are terminal

Vn = {<start>, <functions>, <main>, <function>, <parameters>, <code>, <parameter>, <valid_identifier>, <letter>, <identifier>, <digit>, <statement>, <if_statement>, <assignment>, <variable_declaration>, <function_call>, <return>, <bool_condition>, <elif>, <else>, <bool_prefix>, <bool_member>, <bool_operator>, <calculation>, <data_type>, <calculation_prefix>, <calculation_members>, <calculation_member>, <operator>, <numerical>, <string>, <bool>, <int>, <float>, <text>, <char>, <call_parameters>, <call_parameter>, <valid_returns>}

Vt = {**fun, a…z, A…Z, 0…9, if, elif, else, (,), {, }, _, !, >, <, =, !, >=, <=, !=, ==, &&, ||, -, +, *, /, var, ., main, ~, ”**}

S = {<start> }

P = {
<start> → <functions><main>
<functions> → ε | <function> | <function><functions>
 <function> → **fun** <valid_identifier>**(**<parameters>**){**<code>**}**
<parameters> → ε | <parameter>
<parameter> → <valid_identifier> | <parameter>**,** <parameter>
<valid_identifier> → <letter><identifier> | **_**<identifier>
<identifier> → <letter>| **_** | <digit> | ε | <identifier><identifier>
<letter> → **a…z | A…Z**
<digit> → **0…9**
<code> → <statement> | <statement><code>
<statement>→ <if_statement>|<assignment>|<variable_declaration>|<function_call>|<return>
<if_statement> → **if(**<bool_condition>**){**<code>**}<**elif><else>
<elif> → ε | **elif(**<bool_condition>**){<**code**>} |** <elif><elif>
<else> → ε | **else(**<bool_condition>**){<**code**>}**
<bool_condition> → <bool_prefix><bool_member><bool_operator><bool_condition> | <bool_prefix><bool_member>

<bool_member> → <valid_identifier>| <bool> | <calculation>
<bool_prefix> → ε | **!**
<bool_operator> → > | < | == | != | >= | <= | **&&** | **||**
<assignment> → <valid_identifier> **=** <calculation> | <valid_identifier> = <bool_condition> |
<valid_identifier> = <data_type> | <valid_identifier> **=** <function_call>
<calculation> → <calculation_prefix><calculation_members>
<calculation_prefix> → ε | **-**
<calculation_members> → <calculation_member><operator><calculation_members> |
<calculation_member>
<operator> → **+ | - | * |** /
<calculation_member> → <numerical> | <valid_identifier> | <function_call>
<data_type> → <string> | <numerical> | <bool>
<numerical>→<int> | <float>
<int> → <digit> | <digit><int>
<float> → <int>**.**<int> | <int>
<string> → **”**<text>**”**
<text> → <char> | <char><text>
<char> → ε | ? any visible ASCII character ?
<variable_declaration> → **var** <variables>
<variables> → <valid_identifier> | <assignment> | <variables>**,** <variables>
<function_call> → <valid_identifier>**(<**call_parameters>**)**
<call_parameters>→ ε | <call_parameter>
<call_parameter> → <valid_identifier> | <data_type> | <function_call> | <call_parameter>**,**
<call_parameter>
<return> → **~ <**valid_returns>
<valid_returns> → <data_type> | <calculation> | <function_call> | <bool_condition>
<main> → **fun main(**<parameters>**){**<code>**}**
}