Language Specification:  
 1 .Language Definition:  
 1.1 Alphabet:  
 1.1.

a. Upper (A-Z) and lower case letters (a-z) of the English alphabet  
 b. Underline character '\_';  
 c. Decimal digits (0-9);  
 1.2 Lexic:  
 a.Special symbols, representing:  
 - operators “+” “-” “\*” “/” “=” “<” “<=” “=” “>=” “>”  
 - separators “[“ “]” “{“ “}” “:” “;” space  
 - reserved words:  
 char if else int of main println readln while   
 b.identifiers  
 -a sequence of letters and digits, such that the first character is a letter; the rule is:  
 <**Identifier> ::**= <Letter> {<Letter> | <Digit>}  
 <**Letter> ::**= A | B | . ..| Z | a | … | z  
 <**Digit> ::**= 0 | 1 | ... | 9  
 c.constants  
 1.integer - rule:  
 <**s\_int>** ::= 0 | [+|-]<nz\_digit>{<digit>}

<**nz\_digit> ::**= 1|2|...|9

<**digit> ::**= 0 | <nz\_digit>  
 2.character  
 <**character> ::**= <letter>|<digit>  
 3.string  
 <**constchar> ::**= <string>  
 <**string> ::**= {<char>}  
 <**char> ::**= <letter>|<digit>  
a) Syntactical rules:  
 <**Type>**  ::= bool | char | int | real | string  
 <**Statement>** ::= <if> |<while> | <println> | <association> | <arraydecl> | <readln> | <simpledecl> | <simplearraydecl> |{<Statement>}

<**if>** ::= “if” +”(“ <Expression> ”)” <Statement> “else” ”:” <Statement>

<**while>** ::= “while” “(“ <Expression> “)” <Statement>

<**println>** ::= “println” “(“ <Expression> “)” “;”

<**association>** ::= <Identifier> “=” <Expression> “;”

<**arraydecl>** ::= <Identifier> “[“ <s\_int> ”]” “=” <Expression> “;”

<**readln>** ::= <Identifier> “=” “readln” “(“ <Expression> “) ”;”

<**simpledecl>** ::= <Type> <Identifier> “;”

<**simplearraydecl>** ::= <Type> <Identifier> ”[“ <d\_int> “]” “;”

<**Expression>** ::= <Expression> +(“&&” | ”||” | <Operation> | <Relation>) <Expression>

| <Expression> “[“ <Expression> “]”

| <s\_int>

| true

| false

| <Identifier>

| “this”

| “new” “int” “[“ <Expression> “]”

| “new” <Identifier> “(” “)”  
 | “!” <Expression>

| “(“ <Expression> “)“

| <length>

<**length>** ::= <Expression> ”.” “length”

**<Relation>** ::= < | <=| == | != | >= | >

**<Operation>** ::= + | - | \* | / | %

The tokens are codified according to the following table:  
- identifiers - code = the code in the ST  
- constants - code 1  
- reserved words: each word has its own code  
- operators: each operator has its own code  
- separators: each separator has its own code  
Codification:

|  |  |
| --- | --- |
| Token type | Code |
| identifier | the code in the ST |
| constant | 1 |
| { | 2 |
| } | 3 |
| [ | 4 |
| ] | 5 |
| ( | 6 |
| ) | 7 |
| ; | 8 |
| : | 9 |
| . | 10 |
| , | 11 |
| \* | 12 |
| + | 13 |
| ++ | 14 |
| % | 15 |
| - | 16 |
| = | 17 |
| > | 18 |
| >= | 19 |
| == | 20 |
| <= | 21 |
| < | 22 |
| || | 23 |
| && | 24 |
| public | 25 |
| static | 26 |
| private | 27 |
| int | 28 |
| real | 29 |
| string | 30 |
| bool | 31 |
| char | 32 |
| void | 33 |
| main | 34 |
| new | 35 |
| println | 36 |
| readln | 37 |
| if | 38 |
| else | 39 |
| while | 40 |
| != | 41 |
| ! | 42 |
| true | 43 |
| false | 44 |
| this | 45 |
| length | 46 |