Lexic.txt:
Alphabet:

a. upper and lower case letters of English alphabet: A-Z and a-z
b. decimal digits: 0-9
c. underline character: '_'

Data types:

a. simple: 'int' and 'char'
b. user-defined: 'list'

Lexic:

a. special symbols:

-operators: '+', '-', '/', '*', '=', 'and', 'or', '<', '>', '<=', '>=', '==', '<>'
-separators: '()', ':', ';', 'space', 'new line'
-reserved words: list, char, int, go from   to, check   otherwise, return, write, read

b. identifiers:

-sequence of letter and digits starting with a letter and no longer than 256 characters:

```
identifier = letter | letter{ letter | digit | '_' }
letter = 'A'|'B'|...|'Z'|'a'|'b'|...|'z'
zerodigit = '1'|...|'9'
digit = '0'|zerodigit
```

c. constants:

-int:

```
const_int = '0' | ['+'|'-']zerodigit{digit}
```

-char:

```
const_char = 'letter' | 'digit'
```

-string:

```
const_string = "char{string}"
```

-boolean:

```
const_bool = 'true'|'false'
```

-list:

```
const_list_int = "["const_int|const_int","{const_int}"]"
const_list_char = "["const_char|const_char","{const_char}"]"
...
```

token.in:
Reserved words:

```
	+
	-
	/
	*
	=
	and
	or
	<
	>
	<=
	>=
	==
	<>
	()
	:
	;
	space
	new line
	list
	char
	int
	go from   to
	check   otherwise
	return
	write
	read
```

Syntax.in:
Syntactic rules:

```
	type = "int"|"char"|"string"|"boolean"
	relation = "<"|">"|"<="|">="|"=="|"<>"

	declaration = TYPE identifier ";"
	list_declaration = "list[" TYPE "]" identifier ";"

	input = "read" identifier";"
	output = "write" identifier ";"
	return = "return" (identifier | const_int | const_char | const_string | const_list) ";"
```

```
        assignment = identifier "=" (identifier | const_int | const_char | const_string | const_list)
";"

        ifstmt = "check" condition ":" {stmt} ";otherwise:" {stmt} ";"
        condition = identifier RELATION identifier;
        stmt = assignment | input | output | return;

        loop = "go from" (TYPE assignment | identifier) "to" (identifier | const_int) ":" {stmt} ";"

        function = "function" identifier "("params"):"
{declaration|list_declaration|input|output|assignment|ifstmt|loop|return}
        params = (declaration|list_declaration) | (declaration|list_declaration) "," params
        program = {function}
```