Lab 2_13 oct 2023

Lexic.txt (EBNF)

Alphabet:
a. Upper (A-Z) and lower case letters (a-z) of the English alphabet
b. Special characters `"_", "+", "-", "*", "/", "<", "-", "=", ">", ".", "[", "]", "{", "}", ":", ";", " "`
c. Decimal digits (0-9)

1. Lexic:
   a. Special symbols, representing:

- operators + - * / = < <= == >= : != & |
- separators ; space {} [] ()
- reserved words:
  - char string int list if then else read write float function end_function do
  b. Identifiers:
- a sequence of letters; the rule is:
  - identifier := letter | letter{letter}
  - letter := "a" | ... | "z" | "A" | "B" | ... | "Z"
  c. Constants:
  1. integer:
  - non_null_digit := "1" | "2" | ... | "9"
  - unsigned_number := "0" | non_null_digit
  - digit := "0" | non_null_digit
  - number := "0" | ["-" | "+"] non_null_digit {digit}
  - number2 := "0" | {digit}
  2. charachter:
  - char := 'letter' | 'digit'
  3. string:
  - constchar := "string"
  - string := char{string}
  4. float:
  - constfloat := number "." number2

token.in

```
```int
float
char
string
list
if
```
```

```
then
else
while
read
write
function
end_function
do
+ the rest of the reserved words
```

Syntax.in (EBNF)

The words - predefined tokens are specifiec between " and "

```
Sintactical rules:
decllist := declaration | declaration ";" decllist
declaration := identifier ":" type
type1 := "char" | "float" | "string" | "int"
listdecl := type1 "list" "[" number "]"
type := type1 | listdecl
cmpdstmt := "function" stmtlist "end_function"
stmtlist := simplstmt | structstmt
simplstmt := assignstmt | iostmt
assignstmt := identifier "=" expression
expression := expression "+" term | expression "-" term | term
term := term "*" factor | term "/" factor | factor
factor := "(" expression ")" | identifier | number
iostmt := "read" "(" identifier ")" | "write" "(" identifier ")" | "write" "(" integer
")" | "write" "(" char ")" | "write" "(" string ")"| "write" "(" identifier ")"
structstmt := cmpdstmt | ifstmt | whilestmt
ifstmt := "if" "(" condition ")" "then" "{" stmt "}" ["else" "{" stmt "}" ]
whilestmt := "while" "(" condition ")" "do" stmt
condition := expression relation expression
relation := "<" | "<=" | "=" | "!=" | ">=" | ">"
```