Gordon Daniel gdie2324 prelungire

Alphabet:

a. Upper (A-Z) and lower (a-z) case letters of the English alphabet

b. Underline character ("_")

c. Decimal digits (0-9)

d. Symbols

Lexic:

a. Special symbols:

- operators: +, -, *, /,^ , =, <, <=, >, >=, !=, ==, &&, ||
- separators: (), [], {}, :, ;, space
- reserved words: string number if else elif fi while elihw in read print for rof in and or not

b. Identifiers:

- sequence of letters and digits such that the first character is a letter
- identifier ::= letter | letter {letter}

  letter ::= "A" | "B" | ... | "Z" | "a" | ... | "z"

c. Constants:

1. int ::= sign non_zero_positive_number | non_zero_positive_number | digit

   sign ::= "+" | "-"

   non_zero_positive_number ::= non_zero_digit {digit}

   non_zero_digit ::= "1" | "2" | ... | "9"

   digit ::= "0" | "1" |...| "9"

2. char ::= "letter" | "digit" | specialchar

   specialchar ::= "_" | "$" | "#" | "@" | "!" | ...

3. string ::= char {string}

Syntax

The words - predefined tokens are specified between " and ":

Sintactical rules:

program ::= [decllist ";"] stmtlist ";"

decllist ::= declaration | declaration ";" decllist

declaration ::= type1 IDENTIFIER ";" | type1 IDENTIFIER "=" expression | arraydecl

type1 ::= "STRING" | "NUMBER"

arraydecl ::= "ARRAY" type1 "[" nr "]" IDENTIFIER ";"

type ::= type1 | arraydecl

stmtlist ::= stmt | stmt ";" stmtlist

stmt ::= simplstmt | structstmt | declaration

simplstmt ::= assignstmt | iostmt

assignstmt ::= IDENTIFIER "=" expression

expression ::= expression "+" term | term | expression "-" term

term ::= term "*" factor | factor | term "/" factor

factor ::= "(" expression ")" | IDENTIFIER | number

iostmt ::= "READ" IDENTIFIER | "PRINT" IDENTIFIER

structstmt ::= ifstmt | whilestmt | forstmt

ifstmt ::= "IF" condition ":" stmtlist ";" {["ELSIF" condition ":" stmt ";"]} ["ELSE" stmt ";"] "FI" ";"

whilestmt ::= "WHILE" condition ":" stmtlist "ELIHW"

forstmt ::= "FOR" IDENTIFIER "IN" "(" expression "," expression ")" ":" stmtlist ";" "ROF" ";"

condition ::= expression RELATION expression {[and condition]} {[or condition]}

RELATION ::= "<" | "<=" | "==" | "<>" | ">=" | ">" | "!="

Tokens

:

;

+

-

*

/

=

(

)

[

]

{

}

&&

||

and

or

not

if

else

elsif

fi

while

elihw

for

in

rof

identifier 0

constant 1

print

read

<=

<

>

=>

==

!=

#