Lexic.txt

Alphabet:

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

b. decimal digits (0-9)

c. special characters (+, -, \*, /, (, ), =, {, }, [, ], <, >, \_, ', ", ;, ,)

Lexic:

a. special symbols, representing:, >, <=, >=, =, ==

- separators: [, ], {, }, (, ), ', ", space, newline, tab

- reserved words: int, char, string, if, else, for, print, input

b. identifiers:

-> a sequence of letters and digits

- operators: +, -, \*, /, < such that the first character is a letter

-> rule: identifier = letter | {letter | digit}.

letter = "a" | "b" | "c" | ... | "z" | "A" | "B" | "C" | ... | "Z".

digit = "0" | "1" | "2" | ... | "9".

c. constants:

1. integer

-> rule: const\_int = "0" | ["+" | "-"] non\_zero\_digit {digit}.

digit = "0" | non\_zero\_digit.

non\_zero\_digit = "1" | "2" | "3" | ... | "9".

2. character:

-> rule: const\_char = "'" char "'".

char = letter | digit

3. string:

-> rule: string\_const = """ {char} """.

Syntax.in

Syntax:

program ::= type "main" "(" ")" "{" compound\_stmt "return " identifier | const\_int | const\_string "}"

compound\_stmt ::= stmt | stmt compound\_stmt

stmt ::= simple\_stmt | struct\_stmt

simple\_stmt ::= declaration\_stmt | assignment\_stmt | io\_stmt

declaration\_stmt ::= type identifier

type ::= type\_ | array\_declaration

type\_ ::= "int" | "char" | "string"

array\_declaration ::= type\_ "[" const\_int "]"

assignment\_stmt ::= identifier "=" expression

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

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

factor ::= "(" expression ")" | identifier | const\_int | const\_string

io\_stmt ::= read\_stmt | write\_stmt

read\_stmt ::= identifier "= input" "(" const\_string ")"

write\_stmt ::= "print" "(" identifier ")" | "print" "(" const\_int | const\_string ")"

struct\_stmt ::= compound\_stmt | if\_stmt | for\_stmt

if\_stmt ::= "if" condition stmt | "if" "(" condition ")" stmt "else" stmt

condition ::= expression relation expression

relation ::= "<" | "<=" | "=" | "<>" | ">=" | ">"

for\_stmt ::= "for" "(" assignment\_stmt ";" condition ";" expression ")" "{" stmt "}"

token.in

Tokens:

int

char

string

if

else

for

print

input

return

>

<

=

==

>=

<=

{

}

(

)

[

]

+

-

\*

/

\_

'

"

,

;