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
    >
    <
    =
    ==
    >=
    <=
    {
    }
    (
    )
    [
    ]
    +
    -
    *
    /
    _
    '
```

"

,

;