Language Manual- MiniC Compiler

MiniC is minimalistic compiler designed as a part of the Compilers course project. This is simple and is inspired by the programming languages, Python and C.

Lexical Considerations

MiniC program has keywords, identifiers, constants, operators, list-separators. Everything in MiniC is separated by white spaces. MiniC is case-sensitive.

Keywords: The program uses the following keywords which are used as the syntax of the programming language:

if, else, for, while, bool, uint, int, char, float, void, break, true, false, read, open, file, return, continue, out

Identifiers: Among identifiers, we use lower case for all the variables and upper case for the first character when identifying functions. For example, *foo* is a variable and *Foo* is a function. Identifiers can include alphabets and digits.

digit := "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |"8" | "9" |
alphabet := "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"

Constants: We use constants which are int, float, char. We use lower case for constants.

Operators: MiniC defines the following operands:

Arithmetic	+, -, *, /, %
Boolean	!
Relational	>, <, >=, <=, ==, !=
Conditional	&, |
Assign	=

Separators: We use space (“ ”), comma (,)
Parenthesis: “(“, “)”

Comments: Commented lines are identified by "#" at the start and the end (e.g. #commented part#).
Data types: MiniC include void, uint, int, float, char, bool, file.

- **Array:** type arr_name [size1] [size2]
  An array is declared by specifying the array type, name, size. The size of each dimension is denoted with additional [size] units.

Syntax and Semantics

- **Input:** to read the input from standard input, we use read(). This reads a single character.
  - read()

- **Output:** to print the output to standard output, we use out(). These expressions are separated by comma(,). The strings are given inside single quotes (e.g. 'xy') and variables, symbols are written regularly similar to python.
  - out('exp',+,foo)

- **if / if -else:** evaluates stmtA if expA is true, otherwise stmtB
  - if (expA): stmtA
  - if (expA): stmtA
    else: stmtB

- **for:** evaluates stmtA in the loop with initialization expA, condition expB, updation expC
  - for (expA;expB,expC): stmtA

- **While**: evaluate stmtA if the expA is satisfied
  - while(expA):stmtA

- **Return:** this statement returns the value of a function
  - return expA
  - return

- **Break:** this statement breaks an ongoing loop
  - break

- **Continue:** this statement skips a loop and continues next round
  - continue

- **Functions:** we define a function with a return type, name, zero or more parameters and return statement. Similar to C.
  - type name (param){ return_stmt}

  Referenced functions should be declared always before their reference.

- **File:** we use the following syntax for reading and opening a file.
    - Open : f = open("file.txt","r")
    - Read : read(f)

Meta Notations

<xyz>	non-terminal
xyz	terminal
[x]	Zero or one occurrence of x
x*	Zero or more occurrences of x
x+	One or more occurrences of x
{x}	grouping
|	Alternative (or)

Micro Syntax

<digit> := "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |"8" | "9" |
<alphabet> := "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
<list_sep> := “,”
<sign> := “+” | “-”
<uint> := <digit>+
<int> := <sign> ?<uint>
<float> := <sign> ?<uint>.<uint>
<char> := <alphabet>+
<bool> := “true” | “false”
<value> := <uint> | <int> | <float> | <char> | <id> | STRING | ‘EOF’
<type> := “uint” | “int” | “float” | “char” | “void” | “bool” | “file”
<array_decl> := <type> (<uint>) <id>
<id> := [a-zA-Z][a-zA-Z0-9]*
STRING := "" .*? ""
FUNC : [A-Z][a-zA-Z0-9_]*

Macro Syntax

<program> := program “{“ <start>“}”
<start> := (decl | method)*

<decl> := type { <id> | <id> '[' <exp> ']' }
<method> := type <func> <args> block
<args> := ((decl)* | (value)*)
block := "{" (<decl>|<stmt>)* "}"
<stmt> := <exp> | <decl> | <init> | <if-block> | <for-block> | <while-block> | <return_block> | <break_block> | <continue_block> | <init_stmt> | <cond_block>| <out_block> | <func_call> | <read>
<cond_block> := <exp> ? <exp> : <exp>
<read_block> := read () | read(value)
<open_block> := open(STRING,STRING)
<func_call> := FUNC args
<if-block> := if (<exp>) : <block> [else : <block>]
<for-block> := for (type <exp>; <exp>;<exp>) : <block>
<while-block> := while (<exp>) : <block>
<return_block> := return | return <exp>
<break_block> := break
<continue_block> := continue
<init> := <exp> <assgn_op> <exp>
<loc> := <id> {[<exp>]}*
<assgn_op> := "="
<cond_op> := '>' | '>=' | '<' | '<=' | '==' | '!='
<bool_op> := '&' | '|'
<exp> := (<exp>) | <exp> ('/'| '*' | '%') <exp> | <exp> ('+' | '-') <exp> | !<exp> | <value> | <exp> <asgn_op> <exp> | <exp> <cond_op> <exp> | <exp> <bool_op> <exp> | <read_block> | <open_block> | <func_call> | <arr_identifier>
<arr_identifier> | ID [<exp>]*
<out_block> := out(<exp>)

Rules:

- The program should have a main function
- Multiple declarations are not allowed
- No usage of variables before the declaration
- Every program contains a method ‘main’
- Mismatch in type while assigning
- Array size <uint> should be greater than zero
- Return type should match with the type of <exp> being returned
- <exp> in *if* and *while* should be bool type