Syntax and semantics of Mini-PL (27.01.2020)

Mini-PL is a simple programming language designed for pedagogic purposes. The language is purposely small and is not actually meant for any real programming. Mini-PL contains few statements, arithmetic expressions, and some IO primitives. The language uses static typing and has three built-in types representing primitive values: **int**, **string**, and **bool**. The BNF-style syntax of Mini-PL is given below, and the following paragraphs informally describe the semantics of the language.

Mini-PL uses a single global scope for all different kinds of names. All variables must be declared before use, and each identifier may be declared once only. If not explicitly initialized, variables are assigned an appropriate default value.

The Mini-PL **read** statement can read either an integer value or a single *word* (string) from the input stream. Both types of items are whitespace-limited (by blanks, newlines, etc). Likewise, the **print** statement can write out either integers or string values. A Mini-PL program uses default input and output channels defined by its environment. Additionally, Mini-PL includes an **assert** statement that can be used to verify assertions (assumptions) about the state of the program. An **assert** statement takes a **bool** argument. If an assertion fails (the argument is *false*) the system prints out a diagnostic message.

The arithmetic operator symbols '+', '-', '*','/' represent the following functions:

```
"+" : (int, int) -> int           // integer addition
"-" : (int, int) -> int           // integer subtraction
"*" : (int, int) -> int           // integer multiplication
"/" : (int, int) -> int           // integer division
```

The operator '+' *also* represents string concatenation (i.e., this one operator symbol is *overloaded*):

```
"+" : (string, string) -> string  // string concatenation
```

The operators '&' and '!' represent logical operations:

```
"&" : (bool, bool) -> bool        // logical and
"!" : (bool) -> bool              // logical not
```

The operators '=' and b '<' are overloaded to represent the comparisons between two values of the same type T (**int**, **string**, or **bool**):

```
"=" : (T, T) -> bool              // equality comparison
"<" : (T, T) -> bool              // less-than comparison
```

A **for** statement iterates over the consequent values from a specified integer range. The expressions specifying the beginning and end of the range are evaluated once only, at the beginning of the **for** statement. The **for** control variable behaves like a constant inside the loop: it cannot be assigned another value (before exiting the **for** statement). A control variable needs to be declared before its use in the **for** statement (in the global scope). Note that loop control variables are *not* declared inside **for** statements.

Context-free grammar for Mini-PL

The syntax definition is given in so-called *Extended Backus-Naur* form (EBNF). In the following Mini-PL grammar, the notation X* means 0, 1, or more repetitions of the item X. The '|' operator is used to define alternative constructs. Parentheses may be used to group together a sequence of related symbols. Brackets ("[" "]") may be used to enclose optional parts (i.e., zero or one occurrence). Reserved keywords are marked bold (as "**var**"). Operators, separators, and other single or multiple character tokens are enclosed within quotes (as: ". ."). Note that nested expressions are always fully parenthesized to specify the execution order of operations.

```
<prog>   ::=  <stmts>
<stmts>  ::=  <stmt> ";" ( <stmt> ";" )*
<stmt>   ::=  "var" <var_ident> ":" <type> [ ":=" <expr> ]
           |  <var_ident> ":=" <expr>
           |  "for" <var_ident> "in" <expr> ".." <expr> "do"
                 <stmts> "end" "for"
           |  "read" <var_ident>
           |  "print" <expr>
           |  "assert" "(" <expr> ")"

<expr>   ::=  <opnd> <op> <opnd>
           |  [ <unary_op> ] <opnd>

<opnd>   ::=  <int>
           |  <string>
           |  <var_ident>
           |  "(" expr ")"

<type>   ::=  "int" | "string" | "bool"
<var_ident> ::= <ident>

<reserved keyword> ::=
              "var" | "for" | "end" | "in" | "do" | "read" |
              "print" | "int" | "string" | "bool" | "assert"
```

Lexical elements

In the syntax definition the symbol <*ident*> stands for an identifier (name). An identifier is a sequence of letters, digits, and underscores, starting with a letter. Uppercase letters are distinguished from lowercase.

In the syntax definition the symbol <*int*> stands for an integer constant (literal). An integer constant is a sequence of decimal digits. The symbol <*string*> stands for a string literal. String literals follow the C-style convention: any special characters, such as the quote character (") or backslash (\), are represented using escape characters (e.g.: \").

A limited set of operators include (only!) the ones listed below.

'+' | '-' | '*' | '/' | '<' | '=' | '&' | '!'

In the syntax definition the symbol <*op*> stands for a binary operator symbol. There is one unary

operator symbol (<*unary_op*>): '!', meaning the logical *not* operation. The operator symbol '&' stands for the logical *and* operation. Note that in Mini-PL, '=' is the *equal* operator - not assignment.

The predefined type names (e.g.,"**int**") are reserved keywords, so they cannot be used as (arbitrary) identifiers. In a Mini-PL program, a comment may appear between any two tokens. There are two forms of comments: one starts with "`/*`", ends with "`*/`", can extend over multiple lines, and may be nested. The other comment alternative begins with "`//`" and goes only to the end of the line.

Sample programs

```
var X : int := 4 + (6 * 2);
print X;
```

```
var nTimes : int := 0;
print "How many times?";
read nTimes;
var x : int;
for x in 0..nTimes-1 do
    print x;
    print " : Hello, World!\n";
end for;
assert (x = nTimes);
```

```
print "Give a number";
var n : int;
read n;
var v : int := 1;
var i : int;
for i in 1..n do
    v := v * i;
end for;
print "The result is: ";
print v;
```
