The LANG'24 Language Specification

1 Lexical structure

Programs in the LANG'24 programming language are written in ASCII character set, e.g., no additional characters denoting post-alveolar consonants are allowed.

Programs in the LANG'24 programming language consist of the following lexical elements:

- *Literals*:
  - *numerical literals*:
    A nonempty finite string of decimal digits (`0...9`) optionally preceded by a sign (`+` or `-`).
  - *character literals*:
    A character enclosed in single quotes (`'`). A character in a string literal can be specified by (a) any printable ASCII character, i.e., with ASCII code in range $\{32...126\}$ but the single quote and the backslash must be preceded by the backslash (\), (b) a control sequence `\n` denoting the end of line or (c) an ASCII code represented as `\XX` where `X` stands for any uppercase hexadecimal digit (`0...9` or `A...F`).
  - *string literals*:
    A possibly empty string of characters enclosed in double quotes (`"`). A character in a string literal can be specified by (a) any printable ASCII character, i.e., with ASCII code in range $\{32...126\}$ but the double quote and the backslash must be preceded by the backslash (\), (b) a control sequence `\n` denoting the end of line or (c) an ASCII code represented as `\XX` where `X` stands for any uppercase hexadecimal digit (`0...9` or `A...F`).
- *Symbols*:
  `( ) { } [] . , : ; == != < > <= >= * / % + - ^ =`
- *Identifiers*:
  A nonempty finite string of letters (`A...Z` and `a...z`), decimal digits (`0...9`), and underscores (`_`) that (a) starts with either a letter or an underscore and (b) is not a keyword or a constant.
- *Built-in data types:*
- *Keywords*:
  - literals: `true` `false` `nil` `none`
  - built-in data types: `bool` `char` `int` `void`
  - operators: `and` `not` `or` `sizeof`
  - statements: `return` `if` `then` `else` `while`
- *Comments*:
  A string of characters starting with a hash (`#`) and extending to the end of line.
- *White space*:
  Space, horizontal tab (HT), line feed (LF) and carriage return (CR). Line feed alone denotes the end of line within a source file. Horizontal tabs are considered to be 8 spaces wide.

Lexical elements are recognized from left to right using the longest match approach.

2 Syntax structure

The concrete syntax of the LANG'24 programming language is defined by a context free grammar with the start symbol *definitions* and the following productions:

definitions
⟶ (*type-definition* | *variable-definition* | *function-definition*)$^{+}$

type-definition
⟶ identifier = *type*

variable-definition
⟶ identifier : *type*

function-definition
⟶ identifier ((*parameters*)$^{?}$) : *type* (= *statement* ({ *definitions* })$^{?}$)$^{?}$

parameters
⟶ (^)$^{?}$ identifier : *type* (, (^)$^{?}$ identifier : *type*)*

statement
⟶ *expression* ;
⟶ *expression* = *expression* ;
⟶ `if` *expression* `then` *statement* (`else` *ststement*)$^{?}$
⟶ `while` *expression* : *statement*
⟶ `return` *expression* ;
⟶ { (*statement*)$^{+}$ }

type
⟶ `void` | `bool` | `char` | `int`
⟶ [intconst] *type*
⟶ ^ *type*
⟶ (*components*)
⟶ { *components* }
⟶ identifier

components
⟶ identifier : *type* (, identifier : *type*)*

expression
⟶ voidconst | boolconst | charconst | intconst | strconst | ptrconst
⟶ identifier (((*expression* (, *expression*)*)$^{?}$))$^{?}$
⟶ prefix-operator *expression*
⟶ *expression* binary-operator *expression*
⟶ < *type* > *expression*
⟶ *expression* [*expression*]
⟶ *expression* . identifier
⟶ *expression* ^
⟶ `sizeof` (*type*)
⟶ (*expression*)

Symbols voidconst, boolconst, charconst, intconst, strconst, and ptrconst denote void constant `none`, boolean constants `true` and `false`, character literals, integer literals, string literals, and pointer constant `nil`, respectively.

The precedence of the operators is as follows:

postfix operators	`[·] ^ .`	THE HIGHEST PRECEDENCE
prefix operators	`not + - ^ <·>`	
multiplicative operators	`* / %`	
additive operators	`+ -`	
relational operators	`== != < > <= >=`	
conjunctive operator	`and`	
disjunctive operator	`or`	THE LOWEST PRECEDENCE

Binary operators are left associative.

The `else` part of the conditional statement binds to the nearest preceding `if` part.

3 Semantic structure

3.1 Name binding

Namespaces. There are two kinds of a namespaces:

1. Names of types, functions, variables and parameters belong to one single global namespace.
2. Names of components belong to structure- or union-specific namespaces, i.e., each structure or union defines its own namespace containing names of its components.

Scopes. Two new scopes are created in every function definition

identifier `(` *parameters* `)` `:` *type* `=` *statement* `{` *definitions* `}`

as follows:

1. The name, the parameter types and the result type belong to the scope in which the function is defined.
2. The parameter names belong to the scope nested within the scope in which the function is defined.
3. Statements and definitions belong to the scope nested within the scope in which parameter names are defined.

If there are no parameters, statements or definitions, the scopes are created nevertheless.

All names declared within a given scope are visible in the entire scope unless hidden by a definition in the nested inner scope. A name can be declared within the same scope at most once.