The Programming Language *V* Cem Bozşahin

1 Syntax

program → *declaration_list function_list*
function_list → *function* | *function_list function*
function → *basic_type* **func id** (*parameter_list*) *function_body* **endfunc**
function_body → *declaration_list statement_list*
declaration_list → *declaration_list declaration* **;** | ε
declaration → **var** *variable_list*
parameter_list → *variable_list* | ε
variable_list → **id :** *type* | *variable_list* **, id :** *type*
type → *basic_type vector_extension*
basic_type → **int** | **real**
vector_extension → **[num]** | **[ ]** | ε
statement_list → *statement* **;** | *statement_list statement* **;**
statement → *assignment_statement*
| *return_statement* | *print_statement*
| *read_statement* | *for_statement*
| *if_statement* | *while_statement*

assignment_statement → *variable* **:=** *expression*
variable → **id** | **id [** *expression* **]**

lexpression →*expression* | *expression* **relop** *expression*
| *lexpression* **logop** *lexpression* | **logop** *lexpression*

expression → *term* | *expression* **addop** *term*
term → *factor* | *term* **mulop** *factor*

factor → *variable* | **id** (*argument_list*)
| **num** | (*expression*) | *unary_operator expression*

unary_operator → −
argument_list → *expression_list* | ε
expression_list → *expression* | *expression_list* **,** *expression*
print_statement → **print** <comma-separated list of expressions or literals strings>
read_statement → **read** <comma-separated list of vector cells or simple variables>
return_statement → **return** *expression*

for_statement → **for** *variable* **:=** *expression* **to** *expression* **by** *expression*
statement_list **endfor**

if_statement → **if** *lexpression* **then** *statement_list* **endif**
| **if** *lexpression* **then** *statement_list* **else** *statement_list* **endif**

while_statement → **while** *lexpression* **do** *statement_list* **endwhile**

2 Lexical Conventions

1. Comments start with '%'. The rest of the line is ignored.
2. **mulop** stands for the operators: $*$, $/$, **mod** and **div**.
3. **addop** stands for the operators: $+$ and $-$.
4. **relop** stands for the operators: $=, <>, <, <=, >=$ and $>$.
5. **logop** stands for the operators: **and, or, not**.
6. The language is case sensitive and the keywords are reserved. They appear boldface in the grammar. They are:

func	**endfunc**	**int**	**real**	**return**	**to**	**by**	**and**
mod	**div**	**if**	**then**	**else**	**endif**	**endfor**	**or**
do	**print**	**read**	**while**	**endwhile**	**for**		**not**

7. **id** stands for identifiers. An identifier is a sequence of letters or digits that should start with a letter. There is no language specific restriction on the length.
8. **num** stands for unsigned numbers. A number is composed of three parts. First part is a sequence of digits, the integral part of the number. It may be optionally followed by a decimal point and a sequence of digits i.e. the fraction part. Lastly the exponent part comes, starting with an `e` or `E` followed by optional `-` or `+` and a sequence of digits, i.e. the value of the exponent. This follows the C style decimal number notation. A preceding 0 does not mean an octal number as it does in C; there is no support for different number bases.
9. Literal strings are double-quoted, and may contain double quotes as data. In this case, the quotes are escaped, e.g. `"this is \" a quote"`. Backslash can be written as $\backslash\backslash$.

3 Semantics

1. A program written in *V* commences by executing the function named **main**. Functions can be defined in any order. Global variables are declared at the top, before any function definition takes place. A variable has the scope over the block over which it is defined. The statements of the function are executed in turn until a return statement or the end of the function body is encountered. Upon reaching the end of the function body with no return statement, the return value is integer or real zero, depending on the type of the function.
2. Boolean type is not supported. When an expression is used as a condition for **if** or **while** statements, a value of zero means false and non-zero means true.
3. The **by**-phrase of **for** is optional, and defaults to 1 or 1.0 depending on the variable. The variable is simple.
4. Vector indices start with 1.
5. The expression in [*expression*] must be integer-valued (of **int** type).

6. Vector extension [ ] can only be used in a parameter list.
7. A vector parameter with no bound specification ([ ]) can accept a vector argument of any size.
8. The special character **\n** means newline.
9. Parameters to functions are always passed by value; return is also by value.
10. *V* allows mixed mode arithmetic where **int** values are promoted to **real** if mixed. Forcing a real value to integer, i.e. type narrowing, is not allowed.
11. **div** applies to **int**-valued operands only.

4 History

The name *V* stands for 'vector'. The language is essentially RISC-V's "G" base extended to "V" in a high-level language. It should be relatively easy to compile into RISC-V, or to MIPS which has array capabilities.

Happy compiling.