M+- Language Description

The syntax of M+- is defined as follows:

```
============================================================
prog -> block

block -> declarations program_body.

declarations -> declaration SEMICOLON declarations |.

declaration -> var_declaration
             | fun_declaration.

var_declaration -> VAR ID array_dimensions COLON type.

type -> INT
      | REAL
      | BOOL.

array_dimensions -> SLPAR expr SRPAR array_dimensions |.

fun_declaration -> FUN ID param_list COLON type CLPAR fun_block CRPAR.

fun_block -> declarations fun_body.

param_list -> LPAR parameters RPAR.

parameters -> basic_declaration more_parameters |.

more_parameters -> COMMA  basic_declaration more_parameters |.

basic_declaration -> ID basic_array_dimensions COLON type.

basic_array_dimensions -> SLPAR SRPAR basic_array_dimensions |.

program_body -> BEGIN prog_stmts END.

fun_body -> BEGIN prog_stmts RETURN expr SEMICOLON END.

prog_stmts -> prog_stmt SEMICOLON prog_stmts |.

prog_stmt -> IF expr THEN prog_stmt ELSE prog_stmt
           | WHILE expr DO prog_stmt
           | READ identifier
           | identifier ASSIGN expr
           | PRINT expr
           | CLPAR block CRPAR.

identifier -> ID array_dimensions.

expr ->  expr OR bint_term | bint_term.

bint_term -> bint_term AND bint_factor | bint_factor.

bint_factor -> NOT bint_factor
             | int_expr compare_op int_expr
             | int_expr.
```

```
compare_op -> EQUAL | LT | GT | LE |GE.

int_expr -> int_expr addop int_term | int_term.

addop -> ADD | SUB.

int_term -> int_term mulop int_factor | int_factor.

mulop -> MUL | DIV.

int_factor -> LPAR expr RPAR
            | SIZE LPAR ID basic_array_dimensions RPAR
            | FLOAT LPAR expr RPAR
            | FLOOR LPAR expr RPAR
            | CEIL LPAR expr RPAR
            | ID modifier_list
            | IVAL
            | RVAL
            | BVAL
            | SUB int_factor.

modifier_list -> LPAR arguments RPAR | array_dimensions.

arguments -> expr more_arguments |.

more_arguments -> COMMA expr more_arguments |.

============================================================
```

Terminals of M+-

```
"+"  => ADD
"-"  => SUB
"*"  => MUL
"/"  => DIV

"&&" => AND
"||" => OR
"not" => NOT

"="  => EQUAL
"<"  => LT
">"  => GT
"=<"  => LE
">="  => GE

":=" => ASSIGN

"("   => LPAR
")"   => RPAR
"{"   => CLPAR
"}"   => CRPAR
"["   => SLPAR
"]"   => SRPAR

":"  => COLON
";"  => SEMICLON
","  => COMMA
```

```
"if" => IF
"then" => THEN
"while" => WHILE
"do" => DO
"read" => READ
"else" => ELSE
"begin" => BEGIN
"end" => END
"print" => PRINT
"int"   => INT
"bool"  => BOOL
"real"  => REAL
"var"   => VAR
"size" => SIZE
"float" => FLOAT
"floor" => FLOOR
"ceil"  => CEIL
"fun"   => FUN
"return" => RETURN

{alpha}[_{digit}{alpha}]* => ID          (identifier)
{digit}+ => IVAL                         (integer)
{digit}*.{digit}+ => RVAL                (real)
"false" => BVAL                          (booleans)
"true" => BVAL

where:

alpha = [a-zA-Z]
digit = [0-9]
```

Program comments:

M+- has two types of comments: multi-line comments

```
    /* comment */
```

and one line comments

```
    % comment
```

The multi-line comments allow nesting of comments ...

Commentary on the M+- grammar

An M+- program is a block that is a list of declarations followed by a program body

```
============================================================
prog -> block.

block -> declarations program_body.
============================================================
```

The declarations can either be function declarations or variable declarations each declaration is terminated by a semi-colon.

```
============================================================
declarations -> declaration SEMICOLON declarations |.

declaration -> var_declaration | fun_declaration.
============================================================
```

A variable declaration is preceded by the reserved word "var" and declares an identifier or an array whose type is attached by a colon followed by the basic type. Arrays sizes may be given as expressions in terms of variables in whose scope the declaration lies. This allows one to declare a local array of a size dependent on some input (such as an array which is an argument to the function).

M+- only has only three basic types: reals, integers, booleans.

A function declaration is preceded by the reserved word "fun" and consists of an identifier followed by an argument list with a type followed by the function block. This consist of a declaration list followed by the function body enclosed in curly parentheses. The argument list consist of a (possibly empty) list of variable declarations separated by commas. Arrays are declared in argument lists without their size indicated but with the number of dimensions indicated. Arrays are passed by reference, thus they are passed as a pointer to the location at which they are stored.

A function can call any function which has already been declared or is declared in the same block. Thus, (mutually) recursive functions are permissible. Functions are also allowed to use variables defined in the same block.

A variable, array, or function in a M+- program can only be legally used if it has been declared in an enclosing block or function.

```
============================================================
var_declaration -> VAR ID array_dimensions COLON type.

type -> INT
      | REAL
      | BOOL.

array_dimensions -> SLPAR expr SRPAR array_dimensions |.

fun_declaration -> FUN ID param_list COLON type CLPAR fun_block CRPAR.

fun_block -> declarations fun_body.

param_list -> LPAR parameters RPAR.

parameters -> basic_declaration more_parameters |.

more_parameters -> COMMA  basic_declaration more_parameters |.

basic_declaration -> ID basic_array_dimensions COLON type.

basic_array_dimensions -> SLPAR SRPAR basic_array_dimensions |.
============================================================
```

The difference between a program body and a function body is that the function body MUST end with a return statement. Otherwise both consist of a series of program statements separated by semi-colons. Program statements include conditional ("if ... then ... else ...") statements, while loops, read statements, assignments, print statements, and blocks. Notice that a block permits the declaration of local variables and functions and is delimited by curly braces.

```
============================================================
program_body -> BEGIN prog_stmts END.

fun_body -> BEGIN prog_stmts RETURN expr SEMICOLON END.

prog_stmts -> prog_stmt SEMICOLON prog_stmts |.

prog_stmt -> IF expr THEN prog_stmt ELSE prog_stmt
           | WHILE expr DO prog_stmt
           | READ identifier
           | identifier ASSIGN expr
           | PRINT expr
           | CLPAR block CRPAR.

identifier -> ID array_dimensions.

============================================================
```

There are three kinds of expression in M+-: integer, real, and boolean expressions. The syntax cannot distinguish these expressions and thus some type checking is necessary (and some coercions).

Boolean expressions are used in conditional and while statements. Boolean expressions include the ability to compare integer and real expressions.

```
============================================================
expr ->  expr OR bint_term | bint_term.

bint_term -> bint_term AND bint_factor | bint_factor.

bint_factor -> NOT bint_factor
             | int_expr compare_op int_expr
             | int_expr.

compare_op -> EQUAL | LT | GT | LE |GE.

int_expr -> int_expr addop int_term | int_term.

addop -> ADD | SUB.

int_term -> int_term mulop int_factor | int_factor.

mulop -> MUL | DIV.

int_factor -> LPAR expr RPAR
            | DIM LPAR ID basic_array_dimensions RPAR
            | SIZE LPAR ID basic_array_dimensions RPAR
            | FLOAT LPAR expr RPAR
            | FLOOR LPAR expr RPAR
            | CEIL LPAR expr RPAR
            | ID modifier_list
            | IVAL
            | RVAL
            | BVAL
            | SUB int_factor.

============================================================
```

A modifier list is either the arguments of a function or the address list of an array. Clearly these must be correctly typed.

```
============================================================

modifier_list -> LPAR arguments RPAR | array_dimensions.

arguments -> expr more_arguments |.

more_arguments -> COMMA expr more_arguments |.

============================================================
```

Some M+- program examples

```
/* program matrix.m */

fun mult_matrix(a[][]:real,b[][]:real,c[][]:real):bool
 {  var ab[size(a)][size(b[])]:real;
    var n:int; var m:int; var p:int; var i:int; var j:int; var k:int;
    var ret: bool;

     begin
       %  check the dimensions agree
       n:= size(a); m:= size(b); p:= size(c[]);
       if n=size(c) && m=size(a[]) && p=size(b[]) then
          { begin
               % form ab = a*b
               i := 0;
               while i<n do { begin
               j := 0;
                 while j<p do { begin
                   ab[i][j]:= 0.0;
               k:=0;
                   while k<m do { begin
                     ab[i][j]:= ab[i][j] + a[i][k]*b[k][j];
                     k := k+1;
                   end };
                   j:= j+1;
                 end };
                 i:= i+1;
               end };
               % write out ab into c
             i:=0;
               while i<n do { begin
               j:=0;
                 while j<p do { begin
                   c[i][j]:= ab[i][j];
                   j:=j+1;
                 end };
                 i:= i+1;
               end };
               ret:= true;
          end }
       else ret:= false;
       return ret;
     end
  };

fun read_matrix(a[][]:real):bool
  { var i:int; var j:int;
    begin
       i:= 0;
       while i < size(a[]) do {begin
         j := 0;
         while j < size(a[]) do {begin
             j := j+1;
             read a[i][j];
         end };
         i := i+1;
       end };
       return true;
    end};

fun write_matrix(a[][]:real):bool
  { var i:int; var j:int;
    begin
       i:= 0;
       while i < size(a[]) do {begin
         j:=0;
         while j < size(a[]) do {begin
             j := j+1;
             print a[i][j];
         end};
         i := i+1;
       end};
       return true;
    end};

var X[2][2]:real;

begin
  if read_matrix(X)
  then if mult_matrix(X,X,X)
  then if write_matrix(X)
  then {begin end}
  else {begin end}
  else {begin end}
  else {begin end};
end


============================================================


/*
      program sumarray.m

    The program reads in a list of N reals, where N is specified by
    the user, sums them and then writes out the answer.  It
    illustrates the use of dynamic array sizing and local function
    definitions.
```

```
 */

%  A function for summing a list of N numbers:
fun sum_list(N:int):real
    { var X[N]:real;
      fun read_list(X[]:real):bool
          { var i:int;
            begin
              i:= 0;
              while i<size(X) do
                 { begin read X[i]; i:=i+1; end };
              return true;
            end };
      fun sum_list(X[]:real):real
          {var i:int;
           var sum:real;
            begin
              i:= 0; sum:= 0.0;
              while i<size(X) do
                 { begin sum:= sum + X[i]; i:=i+1; end };
              return sum;
            end};
      var x:real;
      begin
        print read_list(X);
        return sum_list(X);
      end };
var M:int;
begin
  read M;
  print sum_list(M);
end
```