Functions, procedures and conditions

I. Conditional statements :

A. If :

The first conditional form in EZ Language must respect the following syntax:

If condition **do** instruction(s) **endif**

The statements block is executed only if the condition block (Boolean evaluation) is true.

For example, the following conditional statement allows the display of an integer variable:

```
local Variable is integer
Variable = 5
if Variable > 3 do
   Print Variable
endif
```

It is also possible to specify a block to be executed when the condition is not verified by using the keyword **else** :

If condition **do** instruction(s) **else** instruction(s) **endif**

```
local Variable is integer
Variable = 5
If Variable > 3 do
   print “variable greater than 3”
else
   print “variable less than 3”
endif
```

Finally, several conditions can be concatenated in order to test several possible cases :

```
local Variable is integer
Variable = 5
If Variable > 3 do
   print "variable greater than 3"
else
   If Variable < 0 do
      print "negative variable"
   else
      print "positive variable less than 3"
endif
```

A. When :

The **when** statement is used to test the value of one and only one variable and to execute a block of statements according to it. However, it is necessary to remember that the **endcase** keyword is mandatory after each instructions block, except for the **default** case which must always be the last possible case and be followed by the keyword **endwhen**. Although it can always be represented via several **if** statements, the **when** provides a clearer and a lighter cod. It follows the following syntax :

```
When expression is
    Case constant1
        block1
    endcase

    case constant2
        block2
    endcase

    default
        default_block
    endcase
Endwhen
```

Here is a small example of using the **when** statement :

```
local x, y are integer
x = 0

when y is
   case 1
      x = x +1
   endcase
   case 2
      x = x +2
   endcase
   default
      x = x
   endcase
endwhen
```

II. Iterative Instructions :

The iterative instructions makes possible to repeat one or more instructions according to several criteria. The EZ Language offers three iterative statements: while, repeat ... until and for.

A. While :

The **while** statement simply repeats the instruction(s) as long as the condition is true. In fact, it first performs the test of the condition and then executes the instruction block if the test is true.

Its syntax is :

```
While condition do instruction(s) endwhile
```

```
local x is integer
x = 0

while x < 10 do
   x = x +1
endwhile
```

Here the variable x which is initialized to 0 before the loop, is incremented as long as it is less than 10.

B. repeat .. until :

The repeat ... until statement is similar to the while statement in that it repeats the statement block(s) until the condition is true. The syntax is as follows:

```
repeat  instruction(s)  until  condition  endrepeat
```

C. for :

The for statement is used to repeat a block of statements a number of times. The variables a and b represent the minimum and maximum limits of the interval of values assigned to x during the execution of the loop. Note also the keyword **step** allows to define the variations of the variable x between two iterations. This parameter is optional and by default, the value of x is incremented by 1 per iteration. The syntax is therefore the following :

```
local a, b are integer
local var, k are integer
for var in a..b step k do
     instruction(s)
Endfor
```

Which is equivalent to the following syntax:

```
local a, b are integer
for var is integer in a..b step k is integer do
     instruction(s)
Endfor
```

Here is an example of use:

```
local x,k,y are integer
x = k = 0
y = 10
for x is integer in x..y do
   k = k + 1
endfor
```

It is also possible to write a simplified version in case of iterating over a container such as the vector :

```
for element e in vector_name do
   ...
endfor
```

III. Functions / Procedures :

A function is a grouping of instructions in an identified block that can be executed as an instruction by calling the identifier of the function in the program.

A. Syntax :

Here, we distinguish between function that returns a result and procedure that does not.

Function syntax	**function** Name_Fonction (argument is type1) **return** type2
	instruction(s)
	return variable1;
	endfunction

Procedure syntax	**procedure** Name_Procedure (argument is type1)
	instruction(s)
	endprocedure

It is also possible to provide a default value in the functions and procedures arguments.

Function syntax with default argument	**function** Name_Fonction (argument is type1 = value) **return** type2
	instruction(s)
	return variable1;
	endfunction

Procedure syntax with default argument	**procedure** Name_Procedure (argument is type1 = value)
	instruction(s)
	endprocedure

Calls for functions and procedures are as follows :

Function call	**local** variable **is** type = Name_Fonction(argument);
Procedure call	Name_Fonction(argument);

B. Overloading :

Overloading is the ability of defining a function or a procedure several times with different parameters.

Declarations	**procedure** Name_Procedure (argument1 is type1)
	instruction(s)
	Endprocedure
	
	procedure Name_Procedure (argument2 is type2)
	instruction(s)
	endprocedure

Calls	**local** A is type1
	local B is type2
	Name_Fonction(A)
	Name_Procedure(B)

When calling the function, the version used will depend on the argument passed as parameter by the developer.

IV. Input/Output Streams :

To use the standard output and display on the screen, the **print** statement is used. In order to maintain a simplicity of use, one will first write the display instruction and then write the variables to be displayed using the separator **+**. The instruction will display the values of the different variables on the corresponding output.

```
print  “valeur de la variable”  +   variable
```

A standard input is defined which is the keypad of the user. To retrieve user input, use the **read** statement. This statement is blocking, so the program is interrupted until the user has pressed enter to confirm the entry. The result of the input is assigned to a variable passed as a parameter.

```
local saisie is string
read  saisie
```