# EZ Language

# I. Instructions conditionnelles

## A. If

La première forme d'exécution conditionnelle en EZ Language respecte la syntaxe suivante :

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

Le bloc d'instructions est exécutée uniquement si le bloc condition (évaluation booléenne) est vraie.

Par exemple, l'instruction conditionnelle suivante permet l'affichage d'une variable entière :

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

Il est également possible de préciser le comportement attendu du programme lorsque la condition n'est pas vérifiée via l'utilisation du mot-clé **else** :

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

```
local Variable is integer
Variable = 5
If Variable > 3 do
   print "variable supérieur à 3"
else
   print "variable inférieur à 3"
endif
```

Enfin on peut concaténer plusieurs conditions afin de tester plusieurs cas possibles :

```
local Variable is integer
Variable = 5
If Variable > 3 do
   print “variable supérieur à 3”
else
   If Variable < 0 do
      print “variable négative”
   else
      print “variable positive inférieur à 3”
   endif
endif
```

## B.when

L’instruction **when**, permet de tester la valeur d’une et une seule variable et d’exécuter un bloc d’instructions selon celle-ci. Il est toutefois nécessaire de rappeler que le mot-clé **endcase** est obligatoire après chaque bloc d’instruction, exception faite du case **default** qui doit toujours être le dernier cas possible et être suivis du mot-clé **endwhen**. Bien que pouvant toujours être représentée via plusieurs instructions **if** , le **when** permet d’obtenir un code plus clair et plus léger. Elle respecte la syntaxe suivante :

```
when expression is
        case constant1
                bloc1
        endcase
        case constant2
                bloc2
        endcase
        default
                blocdefault
        endcase
endwhen
```

Voici un petit exemple d'utilisation de l'instruction switch :

```
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. Instructions itératives

Les instructions itératives permettent de répéter une ou plusieurs instructions selon plusieurs critères. L'EZ Language offre trois instructions itératives : tant que(while), jusqu'à(repeat … until) et pour(for).

## A. while

L'instruction **while** répète simplement le ou les instruction(s) tant que la condition est vraie. De fait, elle effectue dans un premier temps le test de la condition puis effectue le bloc d'instructions si le test est un succès.
Elle a pour syntaxe :

**while** condition **do** instruction(s) **endwhile**

```
local x is integer
x = 0

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

Ici la variable x qui est initialisée à 0 avant la boucle, est incrémentée tant qu'elle est inférieure à 10.

## B.repeat .. until

L'instruction **repeat...until** est similaire à l'instruction while dans le sens où elle répète le bloc d'instruction(s) tant que la condition n'est pas vérifiée. La syntaxe est la suivant :

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

Il est **important** de noter ici, que le programme effectue un passage dans la boucle avant de vérifier la condition.

## C.for

L'instruction **for** permet de répéter un certain nombre de fois un bloc d'instructions. Les variables a et b représente les bornes minimale et maximale de l'intervalle de valeurs que prend x au cours de l'exécution de la boucle. Notons également le mot-clé **step**, permettant de définir les variations de la variable x entre deux itérations. Ce paramètre est facultatif et par défaut, la valeur de x est incrémentée de 1 par itération.La syntaxe est donc la suivante :

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

qui est **équivalent** à la syntaxe suivante :

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

Voici un exemple d'utilisation :

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

Il est également possible dans le cas d'un parcours de conteneur tel que le vecteur d'être simplifié :

```
for élément e in nom_vecteur do
    ...
endfor
```

Ici, on parcourt le vecteur, élément par élément.

# III. Fonctions / Procédures

Une fonction est un regroupement d'instructions dans un bloc identifié qui pourra être exécuté comme une instruction en appelant l'identifiant de la fonction dans le programme.

## A. Syntaxe

Ici, nous faisons la distinction entre fonction qui retourne un résultat et procédure qui n'en retourne pas.

| | |
|---|---|
| **Syntaxe d'une fonction** | **function** Name_Fonction **(**argument is type1**) return** type2<br>instruction(s)<br>return variable1;<br>**endfunction** |
| **Syntaxe d'une procédure** | **procedure** Name_Procedure **(**argument is type1**)**<br>instruction(s)<br>**endprocedure** |

L'utilisateur peut également fournir une valeur par défaut dans les arguments des fonctions et procédures.

| | |
|---|---|
| **Syntaxe d'une fonction avec argument par défaut** | **function** Name_Fonction **(**argument is type1 = valeur**) return** type2<br>instruction(s)<br>return variable1;<br>**endfunction** |
| **Syntaxe d'une procédure avec argument par défaut** | **procedure** Name_Procedure **(**argument is type1 = valeur**)**<br>instruction(s)<br>**endprocedure** |

Les appels des fonctions et procédures sont les suivants :

| | |
|---|---|
| **Appel d'une fonction** | local variable is type = Name_Fonction(argument); |
| **Appel d'une procédure** | Name_Fonction(argument); |

## B.Surcharge

La surcharge est le principe de définir une fonction ou procédure plusieurs fois dans le cas où elle prendrait des paramètres différents.

| | |
|---|---|
| **Déclarations** | procedure Name_Fonction (argument1 is type1)<br>  instruction(s)<br>endprocedure<br><br>procedure Name_Procedure (argument2 is type2)<br>  instruction(s)<br>endprocedure |
| **Appels** | local A is type1<br>local B is type2<br>Name_Fonction(A)<br>Name_Procedure(B) |

A l'appel de la fonction, la version utilisée sera en fonction de l'argument passé en paramètre par l'utilisateur.

# IV. Flux d'entrée / sortie

Pour utiliser la sortie standard et afficher à l'écran, on utilise l'instruction **print**. Dans le but de conserver une simplicité d'utilisation, on écrira d'abord l'instruction d'affichage puis on écrira à la suite les variables à afficher en utilisant le séparateur **+**. L'instruction affichera les valeurs des différentes variables sur la sortie correspondante.

**print** "valeur de la variable" **+** variable

On définit une entrée standard qui est le clavier de l'utilisateur. Pour récupérer les saisies de l'utilisateur, on utilise l'instruction **read**. Celle-ci est bloquante, le programme est donc interrompu tant que l'utilisateur n'a pas appuyé sur entrée pour valider la saisie. Le résultat de la saisie est affecté à une variable passée en paramètre.

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