2016/2017

Tutoriel Français

Easy laguage

sara zalarhe

Table des matières

1. Easy language EZ

1.1 Description du langage

EZ langage est un langage inspiré du c++, facile d'utilisation, et destiner à tout public, grâce à ses fonctions déjà prédéfinies ; cela facilite l'implémentation des programmes tout en gardant la rigueur et la performance.

1.2 Outils nécessaires

Pour se lancer dans la programmation en EZ langage vous allez avoir besoins de certains outils :

1. Un éditeur de texte : pour écrire un code en langage EZ, il vaut faut un éditeur de texte intégrant la coloration syntaxique du langage pour vous simplifier l'implémentation. Vous avez le choix entre gedite et Kate, deux éditeurs de textes disponible sous linux.
2. Un Compilateur
3. Un debugger

1.3 Créer un projet EZ

Un projet EZ est constitué d'un seul fichier contenant votre code source. Pour valider votre fichier et pour que celui-là soit reconnu par le compilateur il doit avoir l'extension « .ez » ou « .ezl »

2. Premier Pas avec EZ

La meilleure façon d'apprendre un langage est d'écrire un programme minimal qui va simplement afficher « Hello Word », cela va ne permettre de découvrir la structure et la syntaxe du langage.

Voici un simple affichage de « hello word » en EZ :

```
1 // Mon premier programme en EZ
2 program premier_prog
3 procedure premier_prog()
4     print " Hello Word "
5 end procedure
```

L'affichage correspondant à notre « premier programme »

```
Hello Word
```

Examinons le programme ligne par ligne :

Ligne1 : // Mon premier programme en EZ

Le symbole « // » signifie que la ligne est un commentaire, celui-ci n'intervient pas sur le comportement du programme.

Nous pouvons également utiliser les caractères « /* ici plusieurs ligne */ » qui permet de mettre en commentaire une section de code (plusieurs lignes). Le « /* » indique le débute des commentaires, tandis que « */ » indique la fin des commentaires.

Ligne2 : program premier_prog

Le mot clé « programme » indique le début d'un programme, il sera intégré au début de tout programme réalisé. « premier_prog » est le nom du programme

Ligne3 : procedure premier_prog

Une procédure est une fonction qui ne retourne aucune valeur. Ici le nom de notre procédure (premier_prog) est le même que le programme, cela indique que c'est la fonction principale (le main).

Ligne4 : print " Hello Word "

Celle ligne permet l'affichage de la chaine " Hello Word ", grâce à la fonction « print » nous pouvons afficher (écran ou console) une chaine de caractère ainsi que des valeurs retournée d'un programme.

```
1 // Mon premier programme en EZ
2 program premier_prog
3 procedure premier_prog()
4     print ″ Hello Word ″
5     print ″ c' est mon premier programme ″
6 end procedure
```

On rajoutant la nouvelle ligne nous obtiendrons :

```
Hello Word C'est mon premier programme
```

Nous remarquons que le programme nous affiche les deux chaines l'une après l'autre sur la même ligne, pour pallier à ce bémol, nous pouvons indiquer un retour à la ligne grâce au caractère « \n »

```
1 // Mon premier programme en EZ
2 program premier_prog
3 procedure premier_prog()
4     print ″ Hello Word \n ″
5     print ″ c' est mon premier programme ″
6 end procedure
```

Nous obtiendrons l'affichage suivant :

```
Hello Word

C'est mon premier programme
```

Ligne5: end procedure

Le mot « end » indique la fin d'une section (procedure, boucle, condition…), ici en l'occurrence la fin de notre procédure, le mot « procedure » est facultatif, nous pouvons nous contenter uniquement du mot « end »

3. Les variables

La programmation va au-delà que l'affichage d'une simple phrase comme nous avons vu précédemment. Un programme permet de réaliser des traitements plus compliquer, notamment des calculs ou autres. Pour la réalisation d'une simple addition, nous allons donner à l'ordinateur des variables, qui correspondent aux chiffres concernés par l'addition.

Voici une simple addition en EZ :

```
1 // Ma premiére fonction d'addition
2 program prog
3 procedure addition()
4     a is integer = 5
5     b is integer = 2
6     resultat is integer = a+b
7     print " resultat = ". resultat
8 end
```

```
resultat = 7
```

Examinons les lignes de 4 à 7 :

Ligne 4 : ici nous déclarons une variable que nous appelons « a » à laquelle nous donnant la valeur 5. « is integer » indique le type de la variable. (Voir section type).

Ligne 5 : nous déclarons une deuxième variable avec laquelle nous allons réaliser notre addition, elle est du même type que la variable précédente, car vous doutez bien que nous ne pouvons pas additionner deux tables avec 5 chaises, cela n'a aucun sens. Pour les variables c'est la même chose, il faut que les variables concernées par le traitement soient du même type, ici Integrer (entier).

Ligne 6 : cette ligne indique le résultat de l'addition que nous plaçons dans une variable « resultat », évidemment du même type que les variables.

Ligne 7 : affichage du résultat de l'addition. Entre guillemets la chaine de caractère que nous souhaitons afficher, suivi de notre variable « resultats » contenant le résultat de l'addition.

Le symbole « . » signifie la concaténation de notre chaine avec la valeur du résultat obtenu(ici un chiffre), afin d'obtenir le tout en une phrase (sur la même ligne).

Récapitulons :

- un programme permet de réaliser plusieurs traitements possibles
- pour déclarer une variable il faut indiquer son type (sa nature) : *nom_variable* **is** *son_type*
- dans une fonction (traitement) nous manipulons des variables des mêmes types

3.1 Variable global

Une variable global est une variable qui est accessible à toutes les procédures du programme (et les fonctions), elle est déclarée en dehors de toutes les fonctions. Nous pouvons l'utiliser à tout moment du programme. Si elle est initialisée à une valeur lors de la déclaration, celle-ci change en fonction du traitement réalisé dessus. (L'initialisation d'une variable globale n'est pas obligatoire)

```
1 program prog
2 global resultat is integer = 0
3 procedure addition()
4           a is integer =5
5           b is integer =2
6     resultat is integer = a+b
7     print " resultat = ". resultat
8 end
```

```
resultat = 7
```

La ligne 2 : Indique la déclaration d'une variable global. Elle est toujours déclarée au début du programme et précédée du mot clé « global ».

3.2 Variable local

Une variable local est une variable qui est accessible que par la procédure en question (la fonction contenant la variable), elle est déclarée à l'intérieur de la procédure et ait la même durée de vie que la procédure. L'initialisation de la variable n'est pas obligatoire.

```
1 program prog
2 procedure addition()
3           resultat is integer = 0
4           a is integer =5
5           b is integer =2
6     resultat is integer = a+b
7     print " resultat = ". resultat
8 end
```

```
resultat = 7
```

La ligne 3 : déclaration d'une variable local (variable classique).

4. Les types de base

4.1 Types primitifs :

Les valeurs correspondant aux variables sont stockées dans la mémoire sous forme de 0 et 1, le programme se réfère à la variable par son nom, en revanche, pour la stocker il lui faut indiquer le type de cette variable. Car en terme de mémoire cela diffère d'un type à l'autre (Chaine de caractère ou un grand nombre).

Les types de bases sont les types implémentés directement par le langage :

Integer (Entier) : représente les valeurs entières positives ou négatives allant de -2147483648 à 2147483647

Déclaration d'un entier :

```
a,b are integer
```

Nous pouvons déclarer plusieurs variables du même type sur la même ligne en les séparant par une virgule.

Réel : les réels représentent les nombres à virgules.

```
a is reel=1.5
```

Chaine de caractères : ce type permet de stocker l'ensemble des caractères existants en se basant sur la classe string du C++.

```
chaine is string =" coucou"
```

Ici nous avons initialisé notre variable « chaine » à la valeur « coucou ». Lors de la déclaration d'un string nous ne sommes pas obligé d'initialiser la variable. (" " Représente une chaine vide)

Booléen : est un type de variable à deux états :

1 vrai (true =1)
2 faux (false=0)

```
resultat is boolean
```

4.2 Les constantes :

Une variable constante est une variable à valeur fixe non editable.

```
1 program prog
2 procedure addition()
3           constant c is integer = 5
4           b is integer =2
5           resultat is integer = 0
6           resultat  = a+b
7     print " resultat = ". resultat
8 end
```

La lige 3 représente la déclaration d'une constante « c », celle-ci conservera toujours sa valeur (ici 5) durant la durée de vie de la procédure.

Si toute fois nous déclarons une autre variable « c » avec une autre valeur dans la même procédure, le programme affichera une erreur.

5. opérateurs

5.1 Opérateurs d'affectation

Un opérateur d'affectation permet d'affecter une valeur à une variable.

```
A = 5
```

Ici nous affectons la valeur 5 à la variable « A ». Nous pouvons également affecter le contenu d'une variable à une autre :

Si A contenais déjà une autre valeur, celle-ci va être écrasé et remplacer pas la valeur de B.

```
A = B
```

Voici des exemples :

```
1 program prog
2 procedure prog()
3     a,b is integer
4     a=2
5     b=10
6     a=b
7     b=7
8     print " a = ". a . " b = ". b
9 end
```

Voici le résultat retourné :

```
a = 10  b= 7
```

Ce programme affiche les dernières valeurs de a et b, nous remarquons qu'à chaque affectation, la valeur précédente est écrasée par la nouvelle.

5.2 Opérateurs arithmétique

Les opérateurs arithmétiques supportés par EZ sont :

Operateur	Description
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo

Les opérations (+,-,*, /) correspondent à leurs opération mathématique, tandis que le modulo (%), correspond au reste d'une division entière entre deux valeurs.

EZ propose également deux autres fonctions de calcule (Puissance et valeur absolu d'un nombre) :

- ✓ a pow b : retourne le résultat de a puissance b
- ✓ abs a : retourne la valeur absolue de a

```
1 program prog
2 procedure prog()
3     a,b is integer
4     a=2
5     b=10
6     a=a+b
7 print " a = ". a
8 end
```

Il existe aussi des opérateurs d'affectation réalisons une opération arithmétique au même

```
a = 12
```

1. += : qui réalise la somme de la valeur affectée et l'ancienne valeur de la variable ensuite il réalise l'affectation de celle-ci à la variable.
2. -= : réalise la soustraction de la valeur affectée et l'ancienne valeur de la variable ensuite il réalise l'affectation de celle-ci à la variable.
3. *= : réalise la multiplication de la valeur affectée et l'ancienne valeur de la variable ensuite il réalise l'affectation de celle-ci à la variable.
4. /= : réalise la division de la valeur affectée et l'ancienne valeur de la variable ensuite il réalise l'affectation de celle-ci à la variable.

5.3 Operateurs d'incrémentation et décrémentation

L'opérateur d'incrémentation (++) et de décrémentation (--), augmente ou réduise d'une valeur de 1 une variable.

La particularité de ces opérateurs, c'est que ils peuvent être utilisé en préfixe « a++ » ou en suffixe « ++a ».Bien que leurs fonctionnalité na change pas (incrémenter ou décrémenter) ils peuvent tout de même avoir une différence importante.

- Dans le cas où l'opérateur est utilisé comme préfixe (a++), le programme réalise l'affectation ensuite l'incrémentation.

```
1 program prog
2 procedure prog()
3     a,b is integer
4     a=2
5     b=10
6     a= a++
7     b= b--
8 print " a = ". a . " b = ". b
9 end
```

```
a = 3 b=9
```

- Dans le cas où l'opérateur est utilisé comme suffixe (++a), le programme réalise l'incrémentation ensuite l'affectation.

5.4 : Les opérateurs de comparaison

Les opérateurs de comparaison permettent de comparer deux expressions ou deux valeurs (variables), le résultat retourné est d'une nature booléenne (vrai ou faux).

Les opérateurs supportés par EZ :

Operateur	Description
<	Inférieur à
>	Supérieur à
<=	Inférieur ou égal
>=	Supérieur ou égal
!=	diffèrent
==	Egal à

Voici quelques exemples :

```
1  program prog
2  procedure prog()
3      a,b is integer
4      a=2
5      b=10
6      a==b      // retourne false
7      a!=b      // retourne true
8      a < b     // retourne true
9      a > b     // retourne false
10     a >= b    // retourne false
11     a <= b    // retourne true
```

5.5 : Les opérateurs logiques

Il existe des opérateurs que nous appelons logique, permet de vérifier si plusieurs conditions sont vraies :

Operateur	Dénomination	Effet
&&	AND	Vérifie que tous les conditions sont vraies
||	OU	Vérifie qu'au moins une condition est vraie
^	XOR	Retourne la valeur faut si les deux conditions sont vrai au même temps
!	NOT	Inverse l'état d'une variable booléenne (retourne la valeur 1 si la variable vaut 0, 0 si elle vaut 1)

Exemple d'utilisation :

```
1 program prog
2 procedure prog()
3     a,b is boolean
4     a=true
5     b=false
7     a && b    // retourne false
8     a || b    // retourne true
9     ! b       // inverse la valeur de b (b = true)
10    a ^ b     // return false car la valeur de « a » été inverse (ligne 9)
```

6. Flux d'entrée / sortie

Dans les exemples précédents, le programme fourni peu d'interaction avec l'utilisateur, nous avons utilisé la fonction « print » qui permet l'affichage des résultats à l'écran. EZ dispose également d'une fonction « read » qui permet d'interagir avec l'utilisateur et lire les données saisies par celui-ci.

```
1 program prog
2 procedure prog()
3     age is string
4     print " quel age avez vous ?"
5     read age
6    print " Vous avez :" . age
```

```
>> quel age avez vous ?

20ans

>> Vous avez 20ans
```

Ici, nous avons un simple programme qui demande à l'utilisateur son âge. La fonction « print » permet l'affichage de la chaine de caractère ainsi que la valeur saisie par la suite.

La fonction « read » lit la valeur saisie par l'utilisateur et la stocke dans la variable « age » qui sera affichée par la suite.

Ici la variable stockée est une chaine de caractère. Le type de la variable contenant le saisi de l'utilisateur dépend du programme.

7. les conditions

7.1 Condition if

Une condition permet à un programme de prendre une décision. Le principe est simple, nous utilisons les conditions pour faire réagir notre programme différemment en fonction des circonstances.

Pour la mise en place d'une condition, nous allons utiliser les opérateurs de comparaison vue précédemment.

Nous allons maintenant écrire un programme qui affiche la majorité d'une personne.

```
1  program prog
2  procedure prog()
3      age is integer
4      print " quel age avez vous ?"
5      read age
6      print " Vous avez :" . Age
7      if(age > 18)then
8            print" vous êtes majeur"
9      else
10           print" vous êtes mineur"
11     end if
```

Le résultat affiché :

```
>> quel age avez vous ?

20

>> Vous avez 20ans vous êtes majeur
```

Ligne 8 : cette ligne vérifie si l'âge saisi par l'utilisateur est bien supérieur à 18 sinon l'ordinateur exécutera le code indiqué dans le « else ».

7.2 Condition avec When

Il existe d'autre façon de réaliser les tests, bien que le « if » nous permette de réaliser tous les tests possibles, il existe un moyen pour simplifier l'écriture du code.

Le « when » permet de vérifier une condition sur une variable et exécuter le bloc d'instruction correspondant selon l'état de celle-ci.

Exemple :

```
1  program prog
2  procedure prog()
3      age is integer
4      when age<18 is
5            case 1
6                  print" Bonjour mineur!"
7            case 2
8                  print" vous êtes majeur"
9
10
11
```

8. les boucles

8.1 Qu'est-ce qu'une boucle :

Une boucle est une répétition d'instructions tant qu'une condition est respectée.

EZ Langage offre trois boucles (instructions itératives) :

1. tant que(while)
2. jusqu'à(repeat … until)
3. pour(for).

8.2 While

La boucle while signifie (tant que), nous demandons à l'ordinateur d'exécuter un bloc d'instruction tant que notre condition est vrai.

Exemple :

```
1  program prog
2  procedure prog()
3      a is integer
4      a=0;
5      while a<=5 do
6            a=a+1
7      end while
```

Ici nous réalisons une simple boucle qui tant que la variable « a » est inférieur à 5 il incrémente de 1 la valeur de « a ».

Voici les instructions réalisées par la boucle :

```
>> a=0          // premier test de la boucle a<5 → a=1
>> a=1          // a<5      → a=2
>> a=2          // a<5      → a=3
>> a=3          // a<5      → a=4
>> a=4          // a<5      → a=5
>> a=5          // a=5 donc fin de boucle
```

8.3 Reapet..until

Cette boucle est très similaire à la boucle « while ». Elle permet de répéter un bloc d'instructions jusqu'à ce que la condition soit vérifiée.

La différence entre la boucle « while » et la boucle « reapet..until », est l'emplacement de la condition, la boucle « while » peut ne jamais être exécuté si la condition n'est jamais vérifié, en revanche avec la boucle « reapet..until » le bloc d'instruction est exécuté au moins une fois.

Exemple : Nous allons réaliser le même programme que précédemment avec la boucle reapet..until :

```
1 program prog
2 procedure prog()
3     a is integer
4     a=0;
5     repeat
6           a=a+1
7     until a=3
```

Voici les instructions réalisées par la boucle :

```
>> a=0          // initialisation
>> a=1          // premier passage dans la boucle
>> a=2          // deuxiéme passage dans la boucle
>> a=3          // troixéme passage dans la boucle
                // a=3 donc fin de boucle
```

Ici le programme incrémente la valeur de « a » avant de vérifier la condition.

8.4 For

La boucle « for » est très utilisé en programmation, c'est l'équivalent de la boucle « while » écrite d'une autre manière. La particularité de la boucle « for » est de pouvoir réaliser un bloc d'instruction selon un intervalle.

Voici un exemple qui va bien vous illustrer l'utilisation de la boucle « for » :

```
1 program prog
2 procedure prog()
3     variable x, y, z are integer
4     x = z = 0
5     y = 5
6     for i is integer in x..y do
7           z = z + 1
8     end for
```

```
>> z=0          // initialisation
>> z=1          // i=0
>> z=2          // i=1
>> z=3          // i=3
>> z=4          // i=4
>> z=5          // i=5
```

Ici nous remarquons que nous réalisons les instructions de la boucle en variant une variable « i » allant de 0 à 5 .

En écrivant une boucle « for » nous avons déjà une idée du nombre de passage dans la boucle contrairement à une boucle « while ». L'ordinateur exécute le bloc d'instructions dans la boucle tant que « i » n'a pas atteint la limite de l'intervalle.

9. fonctions et procédures

9.1 Procédure

Une procédure est tout simplement une fonction qui ne retourne aucune valeur, c'est un bloc d'instruction indépendant qui sera éventuellement appelé ailleurs dans le programme comme une seule instruction.

Exemple :

```
1     program prog            // début du programme
2     procedure afficher(chaine is string)     // declaration d' une procedure
3           print chaine
4     end procedure
5
6     procedure prog()              // le main
7     message is string
8     message =  "Bonjour débutant!"
9     afficher(message)
10    end procedure
```

Ici nous avons écrit une procédure que nous appelons « afficher » qui prend une chaine de caractère comme argument d'entrée et réalise un simple affichage de cette chaine. (ligne 1 à 4)

Ensuite dans le « main » nous réalisons un simple appelle à cette procédure (ligne 9) avec une chaine de caractère que nous souhaitons afficher (ici la variable message).

Voici le résultat affiché par le programme :

```
>> Bonjour débutant !
```

9.2 Fonction

Une fonction est l'équivalent d'une procédure, un bloc d'instruction exécuté séparément et appelé dans d'autres fonctions ou procédures. À la différence d'une procédure, une fonction à un type de retour, c'est-à-dire elle retourne forcement une valeur.

Nous allons écrire une simple fonction qui effectue la somme de deux éléments (passé en paramètre) et nous retourne la valeur calculé.

```
1     program prog
2     procedure somme(a is integer, b is integer)return integer
3           sum is integer
4           sum= a + b
5           return sum
6     end procedure
7
8     procedure prog()
9     resultat is integer
10    resultat = somme(5,9)
11    Print  "la sommes =  ".resultat
12    end procedure
```

Analysons maintenant ce programme :

Ligne 2 : la déclaration de la fonction somme, elle prend deux arguments de type entier et nous retourne un type entier (grâce au mot clé return suivi du type de retour) qui correspond à la somme dans notre exemple.

Ligne 5 : Le mot clé « return » indique que la fonction retourne forcement une valeur (ici la somme de a et b) le type de retour a été précisé dans la déclaration de la fonction.

Ligne 10 : nous faisons appelle à la fonction somme avec deux valeurs (5 et 9), nous stockons le résultat retourné par la fonction dans la variable « resultat » que nous afficherons par la suite.

L'affichage du résultat :

```
>> la somme = 14
```

9.3 Surcharge des procédures/fonctions

EZ nous permet de définir plusieurs fois la même fonction/procédure dans le cas où elle pourrait prendre des paramètres différents, c'est ce qu'on appelle la surcharge des fonctions.

Nous allons redéfinir la fonction somme comme précédemment, mais cette fois ci avec des arguments de type réel.

```
1     program prog
2     procedure somme(a is integer, b is integer)return integer
3           sum is integer
4           sum= a + b
5           return sum
6     end procedure
7
2     procedure somme(a is reel, b is reel)return reel
3           sum is reel=
4           sum= a + b
5           return sum
6     end procedure
6
8     procedure prog()
9           resultat1 is integer
9           resultat2 is reel
10          resultat1 = somme(5,9)
10          resultat2 = somme(1.23,10.2)
11          print  "la sommes des entiers =  ".resultat1
11          print "la sommes des réels =  ".resultat2
12    end procedure
```

Voici les résultats affichés :

```
>> la somme des entiers = 14

>> la somme des réels = 11.43
```

Ici nous avons défini deux fois la même fonction « somme » avec un type des paramètres différents, c'est l'ordinateur qui différencie les deux fonctions en fonction de type des arguments passés dans le programme principale.

10. Manipuler les chaines de caractères

EZ langage propose plusieurs fonctions prédéfinit pour faciliter la manipulation des chaines de caractères, nous allons voir dans les exemples qui viennent quelques fonctions fondamentales et les plus utilisées. Pour plus d'informations et de précisions veuillez consulter la documentation du langage.

10.1 Découper une chaine :

Nous allons écrire un petit programme qui aura pour fonctionnalité de découper une chaine en une sous chaine que nous souhaitons.

```
1    program prog
2    procedure prog()
3          chaine, sousChaine are string
4          longueur is integer
5          chaine = "Bonjour je suis un robot"
6          sousChaine=chaine.substring(0,6)
7          Longueur= sousChaine.length()
8          Print sousChaine." de longueur = ".longueur
9    end procedure
```

Observez bien la ligne 5, ici grâce la fonction « substring » nous allons découper notre chaine en une sous chaine commençant à partir du caractère 0 et qui a pour longueur 6 (6 caractères).

dans la ligne 7, nous faisons appelle à la fonction « lenght() », cette fonction retourne un entier correspondant à la langueur d'une chaine (ici notre sous chaine)

Voici le résultat affiché par le programme :

```
>> Bonjour de longueur = 6
```

La fonction « substring » ici prend deux paramètres, le premier correspond à la position du caractère à partir duquel nous souhaitons commencer le découpage, le deuxième correspond à la longueur de la chaine à couper(le nombre de caractère voulus), l'indice de caractère commence à 0 pour une chaine.

La fonction « substring » peut également prendre qu'un seule paramètre qui correspondra au caractère du début, si nous ne précisons pas la longueur le programme prendra la longueur de la chaine par défaut.

10.2 Remplacer une chaine

Maintenant nous souhaitons remplacer tous les occurrences d'un caractère par un autre ou une chaine de caractère par une autres, pour cela nous allons faire appel à la fonction « replace ».

```
1    program prog
2    procedure prog()
3          chaine, sousChaine are string
4          chaine =  "Bonjour le monde"
5          sousChaine=chaine.replace("o", "#")
6          Print sousChaine
7    end procedure
```

Ici nous allons tout simplement remplacer tous les « o » de la chaine par le caractére « # »

Résultat obtenu :

```
>> B#nj#ur le m#nde
```

10.3 Trouver une chaine

Pour trouver si un caractère ou une chaine existe dans une autre chaine, nous allons utiliser la fonction « find » qui permet de retourner la position de la première occurrence de la chaîne/ caractère recherché dans la chaîne principal.

Si un seul caractère correspond cela ne suffit pas, il faut que la chaîne entière corresponde.

```
1    program prog
2    procedure prog()
3          chaine is string
4          occu is integer
5          chaine =  "Bonjour le monde"
6          occu=chaine.find("le")
7          Print " occurence trouvé à la position :".occu
8    end procedure
```

Le programme parcourt la chaine et noue retourne la position de la première occurrence.

Résultat affiché :

```
>>occurrence trouvé à la position : 8
```

Il existe également la fonction « findFirstoF » qui retourne la position de la première occurrence, et la fonction « findLastOf » qui retourne la position de la dernière occurrence trouvé.

Plusieurs d'autres fonctions sont proposées par le language EZ, vous les retrouverais en détail dans notre documentation

11. conteneurs

En définition un conteneur est un objet qui contient des objets du même type. Imaginons nous voulons stocker plusieurs valeurs dans le même endroit et les afficher dans l'ordre ensuite, pour cela EZ langage nous offre la possibilité de les stocker dans plusieurs conteneurs selon notre besoin, nous allons réaliser des exemples avec tous les conteneurs proposé pour vous illustrer de plus près leurs utilisation et leurs intérêt.

11.1 Tableau (Array)

Un tableau est un ensemble de cases numéroter de 0 à la taille souhaité. Chaque case contient un élément du tableau.

11.1.1 Afficher un tableau

Dans l'exemple suivant nous allons simplement déclarer un tableau avec des valeurs aléatoire et nous allons les afficher, ensuite les trier et afficher le tableau final trié:

```
1    program prog
2    procedure prog()
3          tab is array[5] of integer = {5,9,23,4,17}
4          taille is integer
5          tab.print()
6          taille = Tab.size()
7          Print" la taille du tableau est de : ". taille
8    end procedure
```

Analysons maintenant les lignes de ce programme :

Ligne 1 : correspond à la déclaration d'un tableau d'entier de taille 5 (le premier élément du tableau commence à l'indice 0).

Ligne 5 : ici nous affichons simplement les éléments du tableau, EZ propose plusieurs fonctions déjà prédéfinit pour la manipulation des tableaux dont la fonction « print() », qui est d'ailleurs commune aux autres types de conteneurs(list, vector…), elle permet d'afficher tous les éléments du tableau.

Ligne 6 : la fonction « size(«) retourne un entier correspondant à la taille du tableau.

```
>> 5 9 23 4 17

>>la taille du tableau est de :5
```

11.1.2 Trier un tableau :

Pour trier un tableau, il existe une fonction prédéfinit permettant de trier les éléments du tableau dans ordre croissant.

```
1    program prog
2    procedure prog()
3          tab is array[5] of integer = {5,9,23,4,17}
4          tab.sort()
5          tab.print()
6    end procedure
```

Résultats affichés

```
>> 4 5 9 17 23
```

Dans la ligne 4 nous faisons appelle à la fonction « sort() » sur notre tableau, ainsi nous obtenons un tableau trié.

Si nous souhaitons un tri dans un ordre décroissant, nous n'avons pas d'autres choix que de parcourir le tableau via une boucle et comparer les éléments entre eux.

11.1.3 Supprimer un ou tous les éléments du tableau :

pour supprimer un élément en particulier du tableau, la fonction « remove() » nous permet de le réaliser en toute simplicité. En revanche la fonction « clear() » permet de supprimer tous les éléments du tableau.

```
1    program prog
2    procedure prog()
3          tab is array[5] of integer = {5,9,23,4,17}
4          tab.remove(23)
5          tab.print()
6    end procedure
```

Résultat affiché

```
>> 5 9 4 17
```

11.1.4 Accéder à un élément du tableau :

Chaque case d'un tableau peut être utilisée comme n'importe quelle autre variable, il n'y a aucune différence. Il faut juste y accéder d'une manière un peu spéciale. On doit indiquer le nom du tableau et le numéro de la case entre [].

Exemple :

```
1    program prog
2    procedure prog()
3          tab is array[5] of integer = {5,9,23,4,17}
4          tab.print()
5          print tab[2]
6    end procedure
```

A la ligne 5 nous demandons uniquement l'affichage de la valeur stocker dans la case 2.

Résultats affichés

```
>> 5 9 23 4 17

>> 23
```

EZ langage propose d'autres fonctions facilitant le traitement des tableaux (consulter la documentation du langage pour plus d'informations).

11.2 Vecteur « Vector »

Un vecteur un est conteneur d'objet, exactement comme un tableau, sauf que la taille d'un vecteur peut changer dynamiquement, c'est-à-dire que nous ne fixons pas de taille à la déclaration comme pour les tableaux, de plus La mémoire du vecteur est ré-allouée à chaque affectation de valeurs.

11.2.1 Déclaration d'un vector

```
1    program prog
2    procedure prog()
3          v is vector of integer
4          v.put_last(5)
5          v.put_last(10)
6          v.put_last(1)
7          v.put_last(9)
8          v.print()
9    end procedure
```

Ici nous réalisons exactement la même chose que l'exemple précédant.

Ligne 3 : correspond à la déclaration d'un vecteur d'entier.

Ligne 4-7 : nous insérons des valeurs dans notre vecteur v, grâce à la fonction « put_last() » qui insert à chaque fois l'élément passé en paramètre au début du vecteur.

Ligne 8 : comme précédemment la fonction « print() », affiche tous les éléments du vecteur

Résultat affiché :

```
>> 9 1 10 5
```

11.2.2 Accéder à un élément du vector

Pour faciliter l'accès aux attributs d'un vecteur et leurs manipulations, le language EZ a prévu plusieurs fonctions pour cela. Dans l'exemple suivant nous allons aborder les plus importantes et les plus utilisées, n'hésitez pas à consulter la documentation pour découvrir les autres fonctions sur les vecteurs.

```
1     program prog
2     procedure prog()
3           v is vector of integer
4           a,b are integer
5           c is boolean
6           v.put_last(5)
7           v.put_last(10)
8           v.put_last(1)
9           v.put_last(9)
10          a=v.first()
11          b=v.last()
12          c= v.find(1)
13          print "a =".a. "b =".b. "c =".c
14    end procedure
```

Dans ce programme, nous affectons quelques valeurs numériques à notre vecteur v (lignes 6–9).

Ligne 10 : la fonction « first() » permet de retourner le premier élément dans le vecteur « v »

Ligne 11 :la fonction « last() » retourne le dernier élément du vecteur « v »

Ligne 12 : la fonction « find(1) » cherche la valeur passé en argument dans le vecteur, si celle se trouve bien dans le vecteur la fonction retourne true, false sinon.

Résultats affichés :

```
>> a =9 b = 5 c = true
```

11.2.3 Supprimer un ou tous les éléments du vecteur :

pour supprimer un élément en particulier du vecteur, la fonction « remove() » nous permet de le réaliser en toute simplicité. En revanche comme pour les tableaux, la fonction « clear() » permet de supprimer tous les éléments du vecteur.

Il existe également la fonction « remove_at(x) », qui supprime l'élément se trouvant à la position x.

```
1     program prog
2     procedure prog()
3           v is vector of integer
4           a,b are integer
5           c is boolean
6           v.put_last(5)
7           v.put_last(10)
8           v.put_last(1)
9           v.put_last(9)
10          v.print()
11          tab.remove(9)
12          tab.remove_at(3)
13          v.print()
14    end procedure
```

Résultats affichés

```
>> 9 1 10 5

>> 1 10
```

Ici nous réalisons un affichage avant et après suppressions, attentions l'indice d'un vecteur commence à la valeur 0.

11.3 Les Listes (list)

Une liste est un conteneur tout comme les tableaux et les vecteurs, l'avantage des listes, c'est leur capacité pour insérer, extraire et déplacer les éléments dans n'importe quelle position dans le conteneur.

11.3.1 Déclarations et fonctions d'une liste :

```
1     program prog
2     procedure prog()
3           ma_list is list of integer = {5,10,1,9}
4           a,b are integer
5           ma_list.print()
6           ma_list.sort()
7           ma_list.print()
8           a= ma_list.first()
9           b=ma_list.last()
10          ma_list.print()
11          print "a =". a . "b =".b
12    end procedure
```

Dans ce programme nous déclarons une liste d'entier que nous initialisons avec des valeurs (ligne 3).

Ligne 5 : affichage de la liste avant tout traitement dessus

Ligne 6 : la fonction « sort() » permet de trier la liste dans un ordre croissant

Ligne 7 : dans la variable « a » nous stockons le premier élément de la liste retourné par la fonction « first() »

Ligne 8 : dans la variable « b » nous stockons le dernier élément de la liste retourné par la fonction « last() »

Il existe avidement d'autres fonctions pour manipuler les listes que vous pouvez retrouver dans la documentation.

Résultats affichés :

```
>> 5 10 1 9
>> 1 5 9 10
>> a = 1 b= 10
```

Ici le premier élément et le dernier sont (1 et 10) car le programme exécute les instructions dans l'ordre.(ici nous trions la liste avant d'afficher le premier et le dernier élément).

Remarque : Il existe un autre type de conteneur « Set », qui a exactement le même comportement que les listes. La seule différence est la gestion des doublons par le conteneur (consulter la documentation du langage).

11.4 Map

Une Map est un conteneur associatif qui stocke des éléments formés par une combinaison de clé, valeur.

Dans une Map, le traitement se fait généralement avec la valeur de la clé (trie, identification,..)Car celle-ci est unique. Les valeurs de la Map stockent le contenue associé à la clé.

Les types de clé et la valeur peuvent être différents.

Imaginons nous souhaitons stocker l'indice langue et le payes associé à celui-là (ex : FR pour France).

Voici un programme réalisant ce traitement :

```
1    program prog
2    procedure prog()
3          ma_map is map of <string,string>
4          a,b are integer
5          ma_map.insert("FR","France")
6          ma_map.insert("EN","Angleterre")
7          ma_map.insert("US","USA")
8          ma_map.print()
9    end procedure
```

Analysons ce programme :

Ligne 3 : déclaration d'une « Map » de type chaine de caractère pour la clé et la valeur.

Ligne 5, 6, 7 : insertion de couple dans la « Map » grâce à la fonction « insert() »

Ligne 8 : affichage de la « Map »

Résultats affichés :

```
>> FR France
   EN Angleterre
   US USA
```