Les conteneurs en EZ Language

Les conteneurs sont des objets permettant de stocker d'autres objets, par exemple on peut citer le vecteur en C++. On peut distinguer entre deux catégories de conteneurs selon le classement des éléments en mémoire, des conteneurs de données en séquences et d'autres conteneurs associatifs. Ainsi, nous avons défini les conteneurs suivants selon les catégories citées :

- Conteneurs pour des données en séquences dans la mémoire :
  - **vector**
  - **list**
  - **array**

- Conteneurs associatifs :
  - **set**
  - **map**

Le développeur en EZ Language aura ainsi le choix de créer des tableaux de taille fixe, des tableaux dynamiques (vector), des listes pour des raisons de rapidité et performance, des sets pour représenter les ensembles et des maps pour la création des paires clé-valeur.

1. Méthodes communes:

Nous définissons plusieurs méthodes communes entre les conteneurs qu'on utilise souvent lors de la manipulation des données afin de faciliter le développement pour l'utilisateur :

Soit **c** un conteneur créé en EZ Language :

Nom	Syntaxe	Description
size	c.size()	Retourne le nombre d'éléments.
is_empty	c.is_empty()	Vérifie si le conteneur est vide.
clear	c.clear()	Efface le contenu du conteneur c.
print	c.print()	Affiche le contenu de c.
range	c.range()	Retourne les bornes inf et sup de c.
first	c.first()	Retourne le premier élément du conteneur c.
last	c.last()	Retourne le dernier élément du conteneur c.

2. Conteneurs de séquences de données

1. Array

Un tableau statique (array) représente une collection indexée d'éléments de même type (appelé le type de base). Comme chaque élément a un indice unique, les tableaux (à la différence des ensembles) peuvent, sans ambiguïtés, contenir plusieurs fois la même valeur. Sa déclaration nécessite d'avoir à priori une taille fixe et peut se faire de plusieurs façons :

```
Scope array_name is array[lower_bound..upper_bound] of type
Scope array_name is array[size] of type
Scope array_name is array[size] of type = [val1,val2,...]
```

La première déclaration sert à déclarer le tableau et ses bornes en même temps. Ensuite, dans la deuxième déclaration, on fournir la taille du tableau. Enfin, on donne la possibilité aussi aux développeurs EZ de déclarer un tableau et l'initialiser au même temps avec l'opérateur =.

EZ Language fournit plusieurs fonctions pour la manipulation des données d'un array, qui sont expliquées dans le tableau suivan t:

Soit **a** un conteneur de type array déclaré en EZ Language.

Nom	Syntaxe	Description
fill	a.fill(value)	Remplit le tableau avec la valeur passée en paramètre.
randomize	a.randomize(min_value,max_value)	Génère des valeurs aléatoires selon le type du tableau. Pour un tableau de valeurs numériques, génère des valeurs ∈ [min_value,max_value]. Pour des tableaux de type string, génère des chaînes aléatoires dont la longueur ∈ [min_value,max_value].
max	a.max() a.max(attribut)	Retourne l'élément le plus grand de **a,** de type numérique ou de type objet en indiquant l'attribut numérique en paramètre.
min	a.min() a.min(attribut)	Retourne l'élément le plus petit de **a** de type numérique ou de type objet en indiquant l'attribut numérique.
sort	a.sort() asort(attribut)	Trie le tableau de valeurs numérique, ou trie le tableau d'objets selon l'attribut passé en paramètre.
sum	a.sum() a.sum(attribut)	Retourne la somme d'un tableau de valeurs numériques, ou la somme de tableau d'objets selon l'attribut passé en paramètre.
count	a.count(value)	Retourne le nombre d'éléments égaux à la valeur passée

		en paramètre.
remove	a.remove(value)	Supprime tous les éléments de **a** égaux à la valeur passée en paramètre.

2. Vector

Un vecteur est un tableau dynamique n'ayant pas de taille ou de longueur fixe. La mémoire d'un tableau dynamique est ré-allouée quand on affecte une valeur au tableau. La structure de ce type est désignée par des constructions de la forme :

```
Scope vector_name is vector of type
Scope vector_name is vector of type size
Scope vector_name is vector of type = { v1,v2,... }
```

Le développeur peut déclarer un vector sans l'initialiser avec la première déclaration fournie, préciser la taille du vector en utilisant la deuxième déclaration ou bien déclarer et initialiser en même temps avec des valeurs v1, ….,vn de même type que le vector en utilisant la troisième déclaration.

EZ Language fournit plusieurs fonctions pour la manipulation des données d'un vector, qui sont détaillées dans le tableau suivant :

Soit v un conteneur de type vector déclaré en EZ Language.

Nom	Syntaxe	Description
fill	v.fill(value)	Remplit tout le vector dont la taille est déjà connue, avec la valeur passée en paramètre.
randomize	v.randomize(min_value,max_value)	Génère des valeurs aléatoires selon le type du tableau. Pour un tableau de valeurs numériques, génère des valeurs ∈ [min_value,max_value]. Pour des tableaux de type string, génère des chaînes aléatoires dont la longueur ∈ [min_value,max_value].
max	v.max() v.max(attribut)	Retourne l'élément le plus grand de **v** de type numérique ou de type objet en indiquant l'attribut numérique en paramètre.
min	v.min() v.min(attribut)	Retourne l'élément le plus petit de **v** de type numérique ou de type objet en indiquant l'attribut numérique en paramètre.
sort	v.sort()	Trie le vector de valeurs numériques, ou trie le tableau

	v.sort(attribut)	d'objets selon l'attribut de classe passé en paramètre.
put_first	v.put_first(élément)	Ajoute la valeur « élément » passée en paramètre à la fin du vector.
put_last	v.put_last(élément)	Ajoute la valeur « élément » passée en paramètre au début du vector.
remove_last	v.remove_last()	Supprime le dernier élément du vector.
remove_first	v.remove_first()	Supprime le premier élément du vector.
sum	v.sum() v.sum(attribut)	Retourne la somme d'un vector de valeurs numériques, ou la somme d'un vector d'objets selon l'attribut numérique de classe en question passé en paramètre.
average	v.average() v.average (attribut)	Retourne la moyenne d'un vector de type numérique ou la moyenne d'un objet selon l'attribut de classe en question passé en paramètre.
count	v.count(value) v.count(attribut, value)	Retourne le nombre d'éléments égaux à la valeur passée en paramètre en cas de vector de type simple, pour un vector de type objet on fournit l'attribut de classe.
remove	v.remove(value)	Supprime la valeur passée en paramètre.
find	v.find(value) v.find(attribut, value)	Cherche une valeur passée en paramètre d'un vector de type simple ou vector d'objets selon le nom d'attribut passé en paramètre.
store	v.store(file_name)	Stocke les données du vector dans un fichier passé en paramètre en format CSV.
restore	v.restore(file_name)	Restaure les données à l'aide du fichier passé en paramètre.

3. List

Un conteneur list permet l'insertion et la suppression rapide d'élément depuis n'importe quel endroit du conteneur.

```
Scope list_name is list of type
Scope list_name is list of type size
Scope list_name is list of type = { v1,v2 ,... }
```

De même, on fournit la possibilité de déclarer le conteneur **list** sans ou avec l'initialisation et la taille. Soit `l` un conteneur de type list en EZ Language :

Nom	Syntaxe	Description
fill	l.fill(value)	Remplis la liste dont la taille est déjà connue, avec la valeur passée en paramètre.

randomize	l.randomize(min_value,max_value)	Génère des valeurs aléatoires selon le type de list. Pour un tableau de valeurs numériques, génère des valeurs ∈ [min_value,max_value]. Pour des vectors de type string, génère des chaînes aléatoires de longueur ∈ [min_value,max_value].
max	l.max() l.max(attribut)	Retourne l' élément le plus grand de `l` de type numérique ou de type objet en indiquant l'attribut numérique en paramètre.
min	l.min() l.min(attribut)	Retourne l' élément le plus petit de `l` de type numérique ou de type objet en indiquant l'attribut numérique en paramètre.
sort	l.sort() l.sort(attribut)	Trie le tableau de valeurs numériques, ou trie le conteneur l d'objets selon l'attribut passé en paramètre.
sum	l.sum() l.sum(attribut)	Retourne la somme d'une list de valeurs numériques, ou la somme d'une list d'objets selon l'attribut passé en paramètre.
count	l.count(value)	Retourne le nombre d'éléments égaux à la valeur passée en paramètre.
remove	l.remove(value)	Supprime la valeur passée en paramètre.

3. Conteneurs associatifs

1. Set

Un ensemble est une collection de valeurs ayant le même type scalaire. Les valeurs n'ont pas d'ordre intrinsèque, une même valeur ne peut donc pas apparaître deux fois dans un ensemble. La construction d'un ensemble se fait de la même manière que les autres conteneurs:

```
Scope set_name is set of type
Scope set_name is set of type size
Scope set_name is set of type = {v1,v2,... }
```

Soit **`s`** un conteneur de type set déclaré en EZ Language, plusieurs fonctions sur ce conteneur sont fournies aux développeurs pour faciliter la manipulation des données :

Nom	Syntaxe	Description
insert	s.insert()	Ajoute un élément dans le conteneur.
max	s.max() s.max(attribut)	Retourne l' élément le plus grand de **s** de type numérique ou de type objet en indiquant l'attribut numérique en paramètre.
min	s.min() s.min(attribut)	Retourne l' élément le plus petit de **s** de type numérique ou de type objet en indiquant l'attribut numérique en paramètre.
sum	s.sum() s.sum(attribut)	Retourne la somme d'un set de valeurs numériques, ou la somme d'un set d'objets selon l'attribut passé en paramètre.
remove	s.remove(value)	Supprime la valeur passée en paramètre.

2. Map

Il s'agit d'un conteneur trié, contenant les paires clé-valeur, avec des clés uniques. On peut déclarer une map de la manière suivante :

```
Scope map_name is map of <type,type>

Scope map_name is map of <type,type> = {<key1,value1>, <key2,value2>, ..}
```

Soit m un conteneur de type map. On peut utiliser plusieurs fonctions sur EZ Language à savoir :

Nom	Syntaxe	Description
insert	m.insert(key,value)	Ajoute un paire dans le conteneur map selon le type de map.
find	m.find(key)	Retourne la valeur de la clé passé en paramètre si elle existe sinon retourne -1.
remove	m.remove(key)	Supprime la paire clé-valeur dont la clé est passée en paramètre.

3. Exemples

1. Array

C++	EZ

#include <algorithm>
#include <numeric>
using namespace std;
 	
int tab[10]={1,1,1,1,1,1,1,1,1,1};
sort(&tab[0], &tab[10]);
int sum = accumulate(&tab[0], &tab[10], 0);
int count = std::count(tab, tab+10, 1);
int max = *std::max_element(&tab[0],&tab[10]);
 	
	tab is array[1..10] of integer
	
	tab.fill(1)
	tab.sort()
	sum, count, max are integer
	sum = tab.sum()
	count=tab.count(1)
	max=tab.max()

2. Vector:

C++	EZ

#include <algorithm>
#include <vector>
using namespace std;
 	
/*declare vector of int*/
vector<int> v(10);
 	
/*fill in the vector with random values in 1..100*/
srand(time(NULL));
generate(v.begin(), v.end(),[]()
{return rand()%(100-1)+1;});
 	
/*Erase first and last value*/
v.erase(v.begin());
v.erase(v.end()-1);
 	
/*insert element in front and back*/
v[0]=5;
v[v.size()-1]=2;
	
	v is vector of integer 10
	
	v.randomize(1,100)
	
	v.remove_first()
	v.remove_last()
	
	v.put_first(5)
	v.put_last(2)

/*sort vector elements*/ sort(v.begin(), v.end(), [](int a,int b){return a<b;});	v.sort()

/*remove*/
std::vector<int>::iterator it;
it = find (v.begin(), v.end(), 5);
if (it != v.end())
v.erase (it);
	v.remove(5)

/*count how often 2 occurs*/ int occ= count(v.begin(), v.end(), 2);	v.count(2)
/*compute average*/ float avg=accumulate(v.begin(), v.end(),0)/v.size();	avg is real = v.average()
/*print vector*/ cout<<"v = ["; for(auto i : v) cout<< i << " "; cout<<"]"<<endl;	v.print()
/*clear the vector*/ v.clear();	v.clear()

3. List

C++	EZ
#include <iostream> #include <list> using namespace std ;	
list<int> l; std::list<int>::iterator it;	l is list of integer 10

srand(time(NULL));
for(int i=0; i<10;++i){
  l.push_back((rand()%(100-1)+1));
}
	l.randomize(1,100)

cout << "l= [";
for (auto it:l){
  cout << ' ' << it;
}
cout << "]\n";
	l.print()

4. Set

C++	EZ

#include <iostream>
#include <set>
using namespace std;
 	
set<int> s={10,5,2};
s.insert(20);
 	
int min=*(max_element(begin(s),
end(s)));
int max= *(min_element(begin(s),
end(s)));
int sum= accumulate(s.begin(),
s.end(), 0);
s.erase(2);
	
	s is set of integer = {10,5,2}
	s.insert(20)
	
	min is integer = s.min()
	max is integer = s.max()
	sum is integer = s.sum()
	
	s.remove(2)

5. Map

C++	EZ
#include <iostream> #include <map> using namespace std; map<string,int> m; m.insert (pair<string,int>("p1",20)); m.insert (pair<string,int>("p2",25)); cout << "m contains:\n"; map<string,int>::iterator it; for(it=m.begin(); it!=m.end(); + +it) cout << it->first << " => " << it- >second << '\n';	 m is map of <char, integer> m.insert(“p1”,20) m.insert(“p2”,25) m.print()

Remarque

Les fonctions présentées peuvent être améliorées, ce n'est qu'une première version. On peut, par exemple, ajouter des fonctions avec des contraintes et dont la nomenclature sera comme suit : fonctionname_if. On peut penser ainsi à count_if, remove_if, sum_if ...etc.