Containers in EZ Language

A container is a data structure, whose instances are collections of other objects. A good example is the std::vector class in c++. Two types of containers can be distinguished according to the way the elements are being stored in memory, sequence containers and associative containers. Thus, we have defined the following containers according to the categories mentioned above:

- ➢ Sequence containers :
  - vector
  - list
  - array
- ➢ Associative containers :
  - set
  - map

The EZ Language developer will be able to create fixed size arrays (Array), dynamic size arrays (Vector), lists for speed and performance purposes, sets, or maps to create key-value pairs.

● Common methods :

In order to make developing using EZ Language easier, we have defined several frequently used methods that are common between containers.

Let c be a container created used EZ Language :

Name	Syntax	Description
size	`c.size()`	Returns the number of elements.
is_empty	`c.is_empty()`	Test whether the container is empty.
clear	`c.clear()`	Removes all elements from the container B.
print	`c.print()`	Prints the content of c.

- ## Sequence containers

  - ### Array :

A static array is an indexed collection of elements of the same type (called base type). As each element has its own unique index, arrays (unlike sets) may contain the same value several times. It has fixed size that cannot change through the program and may be declared in many ways :

```
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,...}
```

The first one declares an array with its bounds. The second one declares an array with its size. While the third one allows the EZ Language developer to declare an array and initialize it in the same statement using the **'='** operator.

EZ Language provides several functions for array's data manipulation, which are explained in the following table :

Let **a** be a container of type array declared in EZ Language :

Name	Syntax	Description
fill	`a.fill(value)`	Fills the array with the passed value as a parameter.
randomize	`a.randomize(min_value,max_value)`	Generates random values according to the type of the array. For an array of numeric values, it generates values ∈ [min_value,max_value]. For arrays of type string, it generates random strings whose length ∈ [Min_value, max_value].
max	`a.max()` `a.max(attribute)`	Returns the element with the largest value of the array, of numeric or object type, indicating the numeric attribute.
min	`a.min()` `a.min(attribute)`	Returns the element with the smallest value of the array of numeric type, or object type indicating the numeric attribute.
sort	`a.sort()` `a.sort(attribute)`	Sorts an array of numeric type, or sorts an array of objects according to the attribute passed value as a parameter.
sum	`a.sum()` `a.sum(attribute)`	Returns the sum of the array of numeric type, or the sum of the array of objects according to the attribute passed value as a parameter.

count	`a.count(value)`	Returns the count of the elements equal to the value passed as a parameter.
remove	`a.remove(value)`	Removes all elements equal to the value passed as parameter.
range	`a.range()`	Returns the bounds inf and sup of **a.**

- ## Vector :

A vector represents a dynamic array that can change in size. The memory of a dynamic array is reallocated in order to grow in size when new elements are inserted.

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

The developer may declare a vector without initialization using the first declaration (precising the size of the vector is optional), or initialize it in the same time with values v1,....vn of the vector base type using the second declaration.

EZ Language provides several functions for the vector's data manipulation, which are detailed in the following table :

Let **v** be a container of type vector declared in EZ Language :

Name	Syntax	Description
fill	`v.fill(value)`	Fills the vector, whose size is known, with the passed value as a parameter.
randomize	`v.randomize(min_value,max_value)`	Generates random values according to the type of the vector. For a vector of numeric values, it generates values ∈ [min_value,max_value]. For vectors of type string, it generates random strings whose length ∈ [Min_value, max_value].
max	`v.max()` `v.max(attribute)`	Returns the element with the largest value of the vector, of numeric or object type, indicating the numeric attribute.
min	`v.min()` `v.min(attribute)`	Returns the element with the smallest value of the array of numeric type, or object type indicating the numeric attribute.
sort	`v.sort()` `v.sort(attribute)`	Sorts a vector of numeric type, or sorts a vector of objects according to the attribute passed value as a parameter.

put_first	`v.put_first(element)`	Adds the value « element » passed as a parameter at the beginning of the vector.
put_last	`v.put_last(element)`	Adds the value « element » passed as a parameter at the end of the vector.
remove_last	`v.remove_last()`	Removes the last element of the vector.
remove_first	`v.remove_first()`	Removes the first element of the vector.
sum	`v.sum()` `v.sum(attribute)`	Returns the sum of the vector of numeric type, or the sum of the vector of objects according to the attribute passed value as a parameter.
average	`v.average()` `v.average(attribute)`	Returns the average of the vector of numeric type, or the sum of the vector of objects according to the attribute passed value as a parameter.
count	`v.count(value)`	Returns the count of the elements equal to the value passed as a parameter.
remove	`v.remove(value)`	Removes all elements equal to the value passed as parameter.
find	`v.find(value)` `v.find(attribute, value)`	Finds the value passed as parameter in a vector of a simple type or a vector of objects according to the attribute passed value as a parameter.
store	`v.store(file_name)`	Stores the data of the vector in a file, whose name is passed as a parameter, in CSV format.
restore	`v.restore(file_name)`	Restores the data of the vector in a file, whose name is passed as a parameter. (previously stored used the store function).
range	`v.range()`	Returns the bounds inf and sup of **v.**
first	`v.first()`	Returns the first element of the vector.
last	`v.last()`	Returns the last element of the vector.
remove_at	`v.remove_at(pos)`	Removes the element at the position passed as a parameter.
put_at	`v.put_at(pos, element)`	Inserts the element at the position passed as a parameter.

- **List :**

It's a container allowing quick insertion and removal of items from anywhere in the container.

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

Similarly, it is possible to declare the list container without or with Initialization and size. Let **l** be a container of type list declared in EZ Language:

Name	Syntax	Description
fill	`l.fill(value)`	Fills the list, whose size is known, with the passed value as a parameter.
randomize	`l.randomize(min_value,max_value)`	Generates random values according to the type of the list. For a list of numeric values, it generates values ∈ [min_value,max_value]. For lists of type string, it generates random strings whose length ∈ [Min_value, max_value].
max	`l.max()` `l.max(attribute)`	Returns the element with the largest value of the list, of numeric or object type, indicating the numeric attribute.
min	`l.min()` `l.min(attribute)`	Returns the element with the smallest value of the list of numeric type, or object type indicating the numeric attribute.
sort	`l.sort()` `l.sort(attribute)`	Sorts a list of numeric type, or sorts a list of objects according to the attribute passed value as a parameter.
sum	`l.sum()` `l.sum(attribute)`	Returns the sum of the list of numeric type, or the sum of the list of objects according to the attribute passed value as a parameter.
count	`l.count(value)`	Returns the count of the elements equal to the value passed as a parameter.
remove	`l.remove(value)`	Removes all elements equal to the value passed as parameter.
range	`l.range()`	Returns the bounds inf and sup of **l.**
first	`l.first()`	Returns the first element of the list.
last	`l.last()`	Returns the last element of the list.

- ## Associative containers

  - ### Set :

A set is a collection of elements having the same scalar type, the value of an element also identifies, so each value must be unique. The construction of a set is done in the same way as the other containers :

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

Let **s** be a set declared in EZ Language. Several functions are provided In order to make sets' data manipulation easier :

Name	Syntax	Description
insert	`s.insert(element)`	Adds an element to the set.
max	`l.max()` `l.max(attribute)`	Returns the element with the largest value of the set, of numeric or object type, indicating the numeric attribute.
min	`l.min()` `l.min(attribute)`	Returns the element with the smallest value of the set of numeric type, or object type indicating the numeric attribute.
sum	`l.sum()` `l.sum(attribute)`	Returns the sum of the set of numeric type, or the sum of the set of objects according to the attribute passed value as a parameter.
remove	`l.remove(value)`	Removes the value passed as parameter.

  - ### Map :

Maps are sorted containers that store elements formed by a combination of a key-value, with unique keys. A map can be declared as follows :

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

Let **m** be a map. Several functions are provided by EZ Language to manipulate maps :

Name	Syntax	Description
insert	`s.insert(key,value)`	Adds a pair to the map according to the map base type.
exist	`m.exist(key)`	Returns a boolean false/true indicating whether a key exists in the map.
find	`m.find(key)`	Looks for the provided key in the map and returns the corresponding value. **find** is always preceded by **exist** otherwise an exception is raised by **find.**
remove	`m.remove(key)`	Removes the pair key-value whose key is passed as parameter.

- ## Examples :
  - ### Array :

C++	EZ

`#include <iostream>`
`#include <numeric>`
`#include <algorithm>`
 	
`using namespace std;`
 	
`template<class T>`
`class Array {`
`public:`
`  class iterator {`
`  private:`
`  T *ptr;`
`  public:`
`  iterator(T *ptr) : ptr(ptr) {}`
 	
`  iterator operator++() {`
`  iterator i(ptr);`
`  ++ptr;`
`  return i;`
`  }`
 	
`  bool operator!=(const iterator &other) { return ptr != other.ptr; }`
 	
`  const T &operator*() const { return *ptr; }`
`  };`
 	
`private:`
`  int size;`
`  T *m_array;`
`public:`	

<table>
<tr><td>   Array(int size) {
        this->size = size;
        m_array = new T[size];
    }

    void fill(T v) {
        for (int i = 0; i < size; ++i)
            m_array[i] = v;
    }

    iterator begin() const {
        return
                iterator(m_array);
    }

    iterator end() const {
        return
                iterator(m_array + size);
    }

    T &operator[](const int index) {
        return
                m_array[index];
    }
};

/*Usage (in main function)*/
Array<int> tab(10);
tab.fill(1);
int sum = accumulate(tab.begin(), tab.end(), 0);
int max = *std::max_element(&tab[0], &tab[10]);</td>
<td>tab is array[1..10] of integer
tab.fill(1)
sum, max are integer
sum = tab.sum()
max = tab.max()</td></tr>
</table>

- **Vector :**

<table>
<tr><th>C++</th><th>EZ</th></tr>
<tr><td>#include <algorithm>
#include <vector>

using namespace std;
/*declare vector of int*/
vector<int> v(10);</td><td>v is vector of 10 integer</td></tr>
<tr><td>/*fill in the vector with random values in 1..100*/
vector<int> v(10);
srand(time(NULL));
generate(v.begin(), v.end(), []() {
    return rand() % (100 - 1) + 1;
});</td><td>v.randomize(1,100)</td></tr>
<tr><td>/*Erase first and last value*/
v.erase(v.begin());
v.erase(v.end() - 1);</td><td>v.remove_first()
v.remove_last()</td></tr>
</table>

`/*insert element in front and back*/`
`v[0] = 5;`
`v[v.size() - 1] = 2;`
	`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*/` `for (auto i : v) cout << i << "";`	`v.print()`
`/*clear the vector*/` `v.clear();`	`v.clear()`

- **List :**

C++	EZ
`#include <iostream>` `#include <list>`	

`using namespace std;`
`list<int> l;`
`std::list<int>::iterator it;`
`srand(time(NULL));`
	`l is list of 10 integer`

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

`for (auto it:l) {`
`  cout <<" " <<it;`
`}`
	`l.print()`

- **Set :**

C++	EZ

`#include <iostream>`
`#include <set>`
 	
`using namespace std;`
 	
`set<int> s = {10, 5, 2};`
 	
`s.insert(20);`
 	
`int min = *(max_element(s.begin(), s.end()));`
 	
`int max = *(min_element(s.begin(), s.end()));`
 	
`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)`

- **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));`
 	
`map<string, int>::iterator it;`
 	
`for (it = m.begin(); it != m.end(); ++it)`
`cout << it->first << "=>" << it->second << '\n';`
	`m is map of <string, integer>`
	
	`m.insert("p1",20)`
	
	`m.insert("p2",25)`
	
	`m.print()`

Note :

The functions presented can be improved, this is only a first version. One can, for example, add functions with constraints and whose nomenclature will be as follows: fonctioname_if. One can think as well of count_if, remove_if, sum_if ... etc.