Programación Avanzada I
Tarea 16

Andrés César Magaña Martínez

Problema 2.

Map

Maps are a kind of associative containers that stores elements formed by the combination of a key value and a mapped value.

In a map, the key value is generally used to uniquely identify the element, while the mapped value is some sort of value associated to this key. Types of key and mapped value may differ. For example, a typical example of a map is a telephone guide where the name is the key and the telephone number is the mapped value.

Internally, the elements in the map are sorted from lower to higher key value following a specific strict weak ordering criterion set on construction.

As associative containers, they are especially designed to be efficient accessing its elements by their key (unlike sequence containers, which are more efficient accessing elements by their relative or absolute position).

Therefore, the main characteristics of a map as an associative container are:

* Unique key values: no two elements in the map have keys that compare equal to each other. For a similar associative container allowing for multiple elements with equivalent keys, see multimap.

* Each element is composed of a key and a mapped value. For a simpler associative container where the element value itself is its key, see set.

* Elements follow a strict weak ordering at all times. Unordered associative arrays, like unordered_map, are available in implementations following TR1.

Maps are also unique among associative containers in that the implement the direct access operator (operator[]) which allows for direct access of the mapped value.

Programación Avanzada I
Tarea 16

Andrés César Magaña Martínez

In their implementation in the C++ Standard Template Library, map containers take four template parameters:

```
template < class Key, class T, class Compare = less<Key>,
   class Allocator = allocator<pair<const Key,T> > > class map;
```

Where the template parameters have the following meanings:

* Key: Type of the key values. Each element in a map is uniquely identified by its key value.

* T: Type of the mapped value. Each element in a map is used to store some data as its mapped value.

* Compare: Comparison class: A class that takes two arguments of the key type and returns a bool. The expression comp(a,b), where comp is an object of this comparison class and a and b are key values, shall return true if a is to be placed at an earlier position than b in a strict weak ordering operation. This can either be a class implementing a function call operator or a pointer to a function (see constructor for an example). This defaults to less¡Key¿, which returns the same as applying the less-than operator (a¡b). The map object uses this expression to determine the position of the elements in the container. All elements in a map container are ordered following this rule at all times.

* Allocator: Type of the allocator object used to define the storage allocation model. By default, the allocator class template is used, which defines the simplest memory allocation model and is value-independent.

In the reference for the map member functions, these same names (Key, T, Compare and Allocator) are assumed for the template parameters.

This container class supports bidirectional iterators. Iterators to elements of map containers access to both the key and the mapped value. For this, the class defines what is called its value_type, which is a pair class with its first value corresponding to the const version of the key type (template parameter Key) and its second value corresponding to the mapped value (template parameter T):

Programación Avanzada I
Tarea 16

Andrés César Magaña Martínez

```
typedef pair<const Key, T> value_type;
```

Iterators of a map container point to elements of this value_type. Thus, for an iterator called it that points to an element of a map, its key and mapped value can be accessed respectively with:

```
map<Key,T>::iterator it;
(*it).first;              // the key value (of type Key)
(*it).second;             // the mapped value (of type T)
(*it);                    // the "element value" (of type pair<const Key,T>)
```

Naturally, any other direct access operator, such as $->$ or [] can be used, for example:

```
it->first;                // same as (*it).first   (the key value)
it->second;               // same as (*it).second  (the mapped value)
```

Fuente: http://www.cplusplus.com/reference/stl/map/