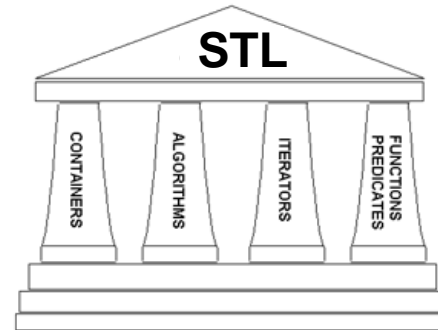




Práctica 3

La plantilla *map* de la STL

Grado en
Ingeniería
Informática



Estructuras
de Datos II
2020/21

Departamento de Tecnologías de la Información
Universidad de Huelva

La plantilla *map* de la STL

Objetivos

- ❑ Introducir la STL como recurso fundamental en la programación en C++
- ❑ Conocer las plantillas *map*, *pair* e *iterator* de la STL

Contenidos

1. La Biblioteca Estándar de Plantillas (*Standard Template Library*, STL)
2. El tipo *map*
3. El tipo *pair*
4. El tipo *iterator*



+

1. La STL

+ La STL

- ❑ **STL** es el acrónimo de *Standard Template Library* (Biblioteca Estándar de Plantillas)
- ❑ Proporciona 4 componentes denominados:
 - **Contenedores**, tanto secuenciales (p.e. listas, vectores, colas) como asociativos (p.e. conjuntos, diccionarios)
 - **Iteradores**, abstracciones que permiten recorrer y acceder a los elementos que forman parte de cualquier contenedor.
 - **Algoritmos**, implementación de un gran número de algoritmos para realizar tareas habituales, como p.e. ordenación y búsqueda.
 - **Objetos función**, objetos que pueden ser llamados como si fueran funciones. La STL incluye operaciones aritméticas, comparaciones y operaciones lógicas.
- ❑ Los tipos que define son **genéricos** gracias al uso de plantillas

+

2. El tipo *map*

+ El tipo *map*

❑ **Map** es un contenedor asociativo (un **diccionario**) con las siguientes características:

- Almacena elementos formados por una combinación de **clave-valor** (de tipos genéricos **key_type** y **mapped_type**, respectivamente)

clave
valor
- Los elementos almacenados siguen un orden especificado por la clave
- No se permite elementos repetidos ---> Con la misma clave

❑ La implementación habitual es mediante un árbol rojo-negro (ARN)

❑ El **par** clave-valor almacenado en el **map** es de tipo **value_type**:

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

clave valor
(Key_type) (mapped_type)
map (STL)

❑ La plantilla *map*:

```
template < class Key,                                // map::key_type
          class T,                                    // map::mapped_type
          class Compare = less<Key>,                 // map::key_compare
          class Alloc = allocator<pair<const Key,T> > // map::allocator_type
        > class map;
```

❑ Ejemplo de declaración:

```
map<int, string> miDiccionario;
```

clave valor

❑ Métodos más importantes:

Método	Descripción
<code>map (const key_compare& comp = key_compare(), const allocator_type& alloc = allocator_type())</code>	Constructor de diccionario vacío
<code>pair<iterator,bool> insert (const value_type& val) par <c, v></code>	Añade un nuevo elemento al diccionario. Devuelve un par formado por un iterador al nuevo elemento y cierto. Si la clave ya existe, no se inserta el elemento y devuelve un par formado por un iterador al elemento existente y falso.
<code>size_type erase (const key_type& k) clave</code>	Elimina del diccionario el elemento que se corresponde con la clave que se pasa como parámetro. Devuelve el número de elementos eliminados (0 ó 1).
<code>iterator find (const key_type& k) clave</code>	Devuelve el iterador correspondiente al par de clave <i>k</i> . Si esa clave no está en el diccionario, devuelve un iterador en la posición <i>end()</i> .

El tipo *map*

❑ Métodos más importantes:

Método	Descripción
<code>bool empty() const</code>	Devuelve cierto si el diccionario está vacío y falso en caso contrario
<code>iterator begin()</code>	Devuelve un iterador al primer elemento del diccionario. Como el tipo <i>map</i> mantiene los elementos ordenados según la clave, <i>begin</i> apunta al primer par en orden de clave.
<code>iterator end()</code>	Devuelve un iterador apuntando a la posición siguiente a la que ocupa el último elemento del diccionario.
<code>mapped_type& operator[]</code> <code>(const key_type& k)</code> <div>clave</div>	<p>Si la clave <i>k</i> está en el diccionario devuelve una referencia al valor asociado a <i>k</i>. Si la clave no está, se añade al diccionario. El operador [] se puede utilizar para modificar el valor asociado a una clave.</p> <p>Ejemplos:</p> <pre> map<string, int> meses; meses["enero"] = 31; //inserta el par <enero, 31> meses["febrero"] = 28; //inserta el par <febrero, 28> meses["marzo"] = meses["enero"]; //inserta el par <marzo, 31> cout << meses["abril"]; // inserta el par <abril, > meses["febrero"] = 29; // modifica el valor asociado a febrero y queda <febrero, 29> </pre> <p>Uso del operador []</p> <p>Se usa como si el diccionario fuese una tabla</p> <p>Si no existe clave → se inserta Si existe clave → se modifica</p>

+

3. El tipo *pair*

- ❑ **Pair** agrupa dos valores que pueden ser de tipos distintos (T1 y T2)
- ❑ La plantilla *pair*:

```
template <class T1, class T2> struct pair;
```

clave valor

- ❑ **Atributos:**

Atributo	Descripción
<code>first</code>	El primer valor del par (clave)
<code>second</code>	El segundo valor del par (valor)

+

4. El tipo *iterator*

+ El tipo *iterator*

❑ ***Iterator*** es una abstracción que:

- Permite recorrer un contenedor y acceder a cada uno de los elementos almacenados en dicho contenedor
- Cada contenedor define su propio iterador pero todos tienen las mismas características

❑ **Ejemplo** de declaración de un iterador para un objeto tipo ***map***:

```
typedef map<int, string> miDiccionario;  
miDiccionario::iterator it;
```



❑ Características más importantes:

Expresión	Descripción
<code>constructor::iterator it</code>	Constructor vacío.
<code>a == b</code> <code>a != b</code>	Compara si los iteradores a y b apuntan al mismo (o distinto) elemento.
<code>*a</code>	Contenido del elemento al que apunta el iterador a , si iterador no está en la posición de fin.
<code>a->first</code>	Accede al atributo <i>first</i> , suponiendo que el iterador a apunta a un objeto de tipo par y no está en la posición de fin.
<code>++a</code> <code>a++</code>	Avanza el iterador a la posición del siguiente elemento en el contenedor, si iterador no está en la posición de fin.
<code>--a</code> <code>a--</code>	Cambia el iterador a la posición del elemento anterior en el contenedor, si iterador no está en la posición de inicio.

```
typedef map<int, string> miDiccionario;
```



❑ **Ejemplo:** Recorrido del diccionario *DicDatos* de tipo *miDiccionario* desde el principio hasta el final, mostrando la clave y el valor de cada par.

```
for (miDiccionario::iterator it = DicDatos.begin(); it != DicDatos.end(); it++)  
{  
    cout << it -> first;      // muestra la clave  
  
    cout << it -> second;    // muestra el valor  
}
```