PLANTILLAS, CONTENEDORES, ITERADORES

UCA Universidad de Cádiz

Programación Orientada a Objetos
Curso 2012 / 2013

Plantillas

- Podemos utilizar la sobrecarga para crear versiones especializadas de una misma función que trabajen con distintos tipos de datos:

```
inline int cuadrado(int x) { return x * x; }
inline double cuadrado(double x) { return x * x; }
```

- Pero C++ permite definir la función *cuadrado* de forma que permita calcular el cuadrado de cualquier objeto para el que se defina el operador *:

```
template <typename T>
inline T cuadrado(T x) { return x * x; }
```

Conviene utilizar referencias para evitar copias innecesarias:

```
template <typename T>
inline T cuadrado(const T& x) { return x * x; }
```

STL

- STL: *Standard Template Library*
- Se compone de clases contenedoras y algoritmos genéricos.
- Un contenedor almacena una colección de elementos y permite realizar operaciones con ellos.
- Los iteradores son una abstracción del concepto de puntero: permiten recorrer una colección de elementos y acceder a ellos individualmente.
- Siempre es posible usar un puntero donde se requiera un iterador.
- Los contenedores proporcionan iteradores de una determinada categoría.

Iteradores

Categorías

NOMBRE	OPERACIONES
De entrada	*, -> (ambos solo para lectura), ++, == y !=
De salida	* (solo para escritura) y ++, == y !=
Monodireccionales	Las de los de entrada y de salida
Bidireccionales	Las de los monodireccionales más --
Acceso directo	Las de los bidireccionales más [], <, <=, >, >=, + (con entero) y - (con entero, o con 2 iteradores al mismo contenedor)

Contenedores. Tipos

Secuencias	Nombre	Cabecera	Descripción
	array (C++11)	array	Formaciones
	vector	vector	Vectores
	deque	deque	Colas dobles
	list	list	Listas dobles
	forward_list (C++11)	forward_list	Listas simples

Adaptadores de secuencia	Nombre	Cabecera	Descripción
	stack	stack	Pilas
	queue, priority_queue	queue	Colas simples y de prioridades

Contenedores. Tipos

Contenedores asociativos ordenados	Nombre	Cabecera	Descripción
	set, multiset	set	Conjuntos y multiconjuntos
	map, multimap	map	Aplicaciones mono y multivalor

Contenedores asociativos no ordenados (C++11)	Nombre	Cabecera	Descripción
	unordered_set, unordered_multiset	unordered_set	Conjuntos y multicon-juntos
	unordered_map, unordered_multimap	unordered_map	Aplicacio-nes mono y multivalor

Contenedores. Características

- Por omisión construyen contenedores vacíos.
- Se pueden copiar, asignar y comparar de forma natural.
- Tienen definiciones públicas de tipos.
- Tienen métodos para construir iteradores.

```
C<T> c1, c2;
c1 = c2;
if (c1 == c2) ...
f(c1);
C<T> c;       <- C::value_type
c.empty();
c.begin();    <- C::iterator, C::const_iterator
c.end();      <- [c.begin(), c.end())
c.size()      <- C::size_type
```

Secuencias. Ejemplos

Vectores

```
    vector<int> t;
    vector<int> u(10);
    vector<int> v(10, 1);
    t = u = v;
    vector<int> w(v);
    vector<int> x = w;
```

Colas dobles

```
    deque<int> y(x.begin(), x.end());
```

Listas

```
    list<int> z(y.begin(), y.begin() + y.size() / 2);
    list<int> c;
    for (list<int>::iterator i = c.begin(); i != c.end(); ++i)
        *i = 0;
    for (list<int>::const_iterator i=c.begin(); i!=c.end(); ++i)
        cout << *i << endl;
```

Contenedores asociativos ordenados. Ejemplos

Conjuntos

```
set<string> nombres;
nombres.insert(“Romeo”);
cc.count(“Romeo”)  //devuelve 1
cc.erase(“Romeo”)
set<string>::iterator pos;
for (pos = nombres.begin(); pos!=nombres.end(); ++pos) {
    cout <<*pos<<“ “;
```

Aplicaciones

```
map<string, int> puntos;
puntos[“Romeo”]=90;
puntos.insert (make_pair (“Julieta”,100));
cout<<puntos[“Romeo”]
map<string, int> ::iterator pos = pos.find(“Julieta”);
if (pos==puntos.end()) cout<<“No hay puntos para Julieta”;
else cout <<“Julieta tiene “<< pos->second << “ puntos”;
```