Programación Orientada a Objetos

Standart Template Library

Pablo Abratte
03/11/11

Programación Genérica

- Ej. función para búsqueda del mayor.

```
int Mayor(int arr[], int size){
   int mayor=arr[0];
   for(int i=1; i<size; i++)
      if(arr[i]>mayor)
         mayor=arr[i];
   return mayor;
}
```

Programación Genérica

- Es un estilo de programación centrada en la lógica de los algoritmos más que en los tipos de datos que intervienen en los mismos.
- El objetivo es maximizar la generalidad y reutilización de los procedimientos.
- Para lograr esto, los algoritmos son parametrizados con los tipos de datos.
- En C++ la programación genérica se implementa mediante templates.

Templates en C++

- Ej. función para búsqueda del mayor.

```
template <class T>
T Mayor(T arr[], int size){
   T mayor=arr[0];
   for(int i=1; i<size; i++)
      if(arr[i]>mayor)
         mayor=arr[i];
   return mayor;
}
```

Templates en C++

- Al compilar, se generan copias de la plantilla para cada tipo de dato con el que es utilizada, este proceso se llama *instanciación de la plantilla*.

Standart Template Library

Colección de clases y funciones genéricas (implementadas como plantillas) para almacenar y procesar datos.

Un poco de historia...

- Desarrollada principalmente por Alexander Stepanov.
- Su investigación comenzó aproximadamente en 1987 buscando portar características del lenguaje Ada a C++ para aprovechar el modelo computacional de este último.
- Primero en laboratorios *AT&T* y más tarde en *Hewlett-Packard*.
- En 1994 STL es incorporada a la versión ANSI/ISO de C++.

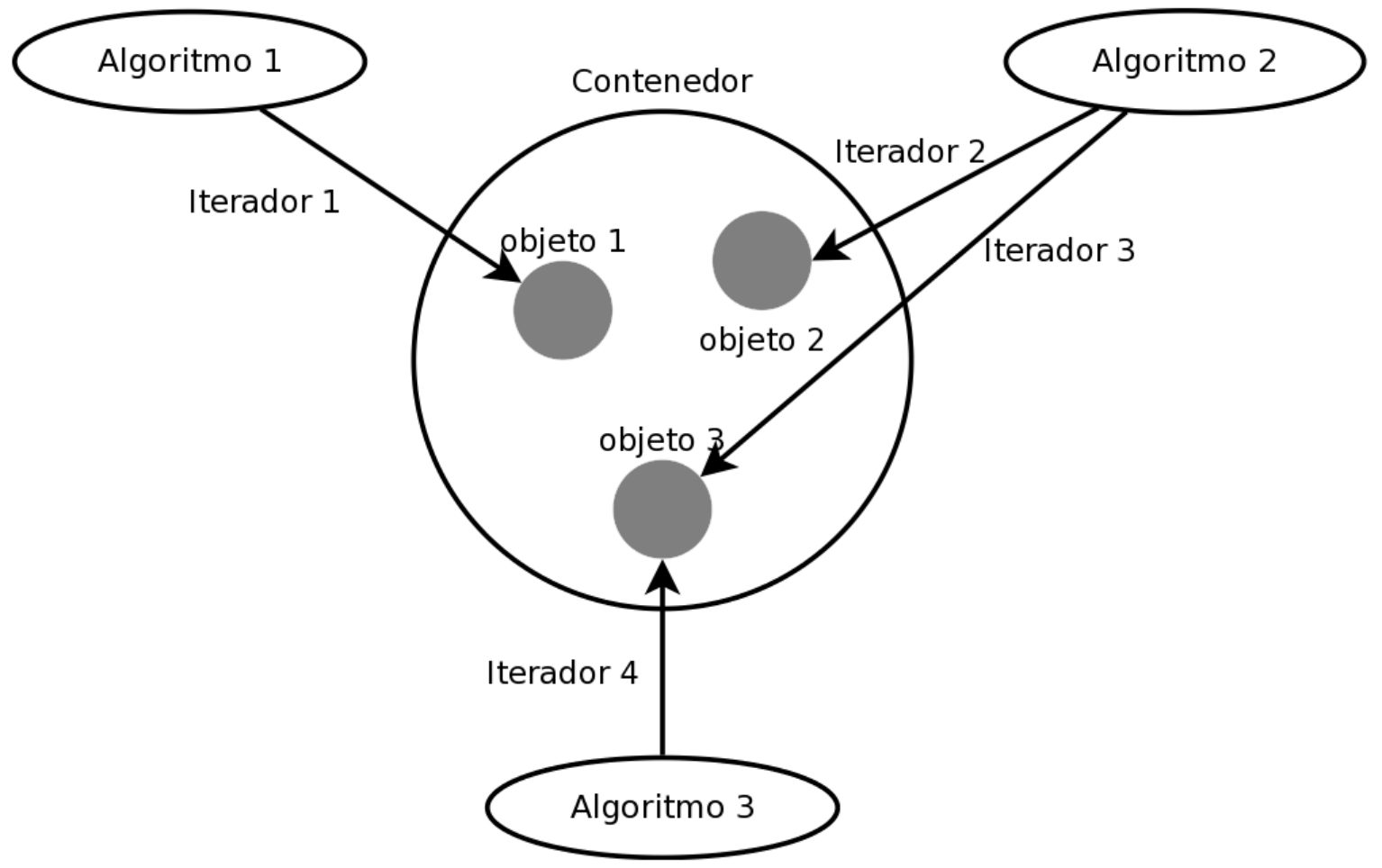
Standart Template Library

- Consta de varios componentes, pero estudiaremos tres que son clave:
  - Contenedores
  - Algoritmos
  - Iteradores

Standart Template Library

- **Contenedores:** objetos que se encargan de almacenar los datos y organizarlos en memoria. Se implementan mediante clases templatizadas para poder personalizar el tipo de dato que se desea almacenar.
- **Algoritmos:** son procedimientos utilizados para procesar los datos almacenados en contenedores. Se implementan como funciones templatizadas.
- **Iteradores:** son objetos que apuntan a un elemento de un contenedor (punteros). Permiten a los algoritmos actuar sobre los contenedores.

Standart Template Library

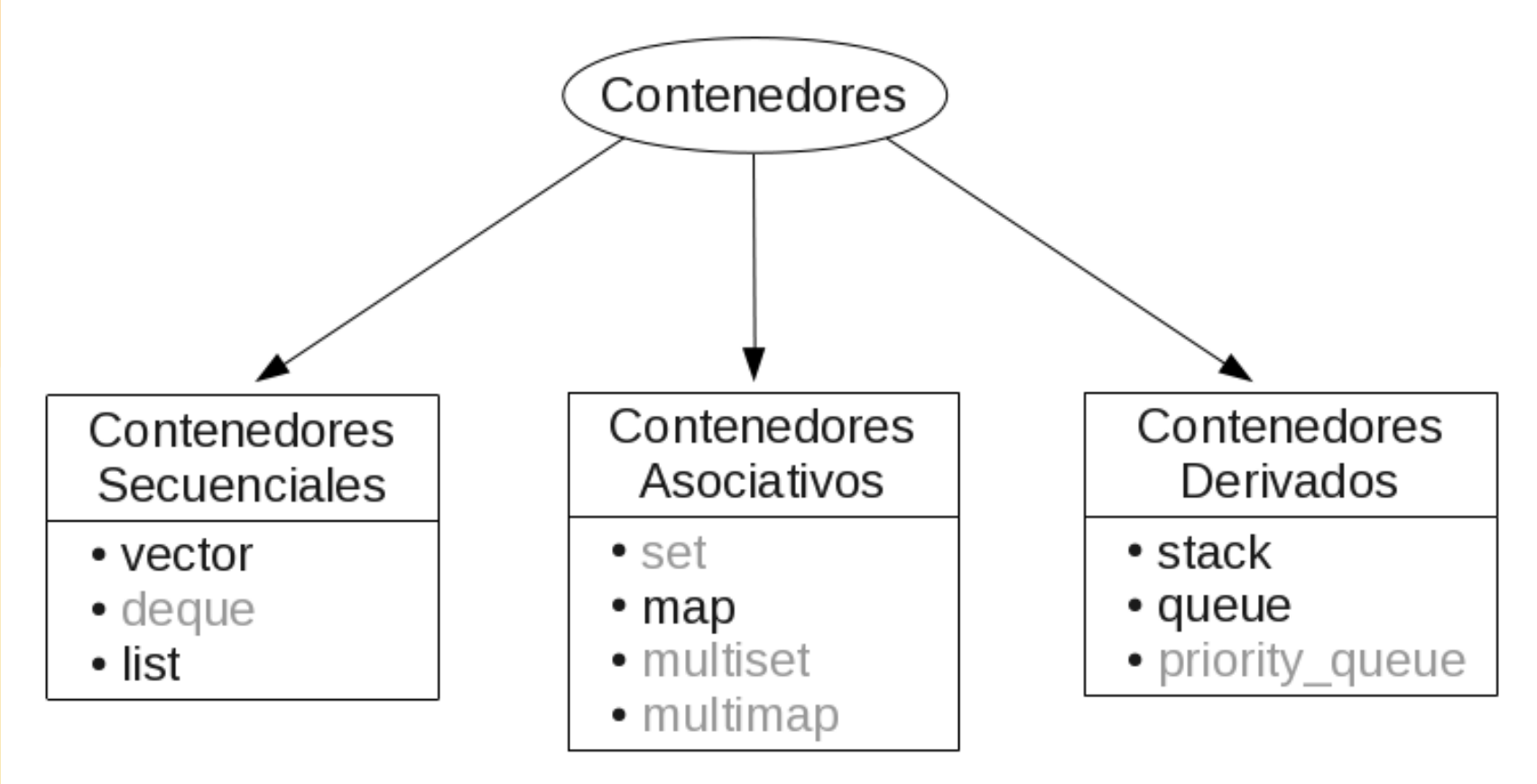


Standart Template Library

Ejemplo:

Cree dos vectores de 10 elementos e inicialicelos con valores aleatorios. Luego muéstrelos.

Standart Template Library



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

Contenedores

- **Secuenciales:** almacenan elementos en una secuencia lineal, uno tras otro. Cada elemento está relacionado con los restantes mediante su posición.
- **Asociativos:** permiten el acceso directo a sus elementos mediante claves.
- **Derivados:** son implementados a partir de contenedores secuenciales. También se los conoce como contenedores adaptados.

Vector

Es un array que varía su tamaño de forma dinámica y automática, permite acceso directo a sus elementos e insertar o eliminar en cualquier posición.

http://www.cplusplus.com/reference/stl/vector

Vector

```
// constructor por defecto
vector<T>();

// constructor con tamaño inicial y valor
vector<T>(size_type n, const T& value);

// operador= para copiar vectores
vector<T>& operator=(const vector<T>& x);

// retorna la cantidad de elementos en el vector
size_type size();

// devuelve verdadero si el vector está vacío
bool empty();
```

Vector

```
// cambia el tamaño del vector
void resize(size_type sz);

// permite acceder al elemento n para
// lectura o escritura. No controla que el
// elemento se encuentre dentro del rango
T &operator[](size_type n);

// retorna un iterador al primer elemento del vector
vector<T>::iterator begin();

// devuelve un iterador a la posición final del vector
// (posición posterior al último elemento)
vector<T>::iterator end();
```

Vector

```
// insertar un elemento x en la posición p,
// agrandando el tamaño del vector, sí es
// necesario. Devuelve un iterador a la
// posición del elemento insertado.
vector<T>::iterator insert(   vector<T>::iterator p,
                              const T& x);

// elimina del vector el elemento en la posición p o
// los elementos en el rango [first, last).
// Devuelve un iterador a la posición siguiente del
// último de los elementos eliminados.
vector<T>::iterator erase( vector<T>::iterator p);
vector<T>::iterator erase( vector<T>::iterator first,
                           vector<T>::iterator last);
```

Vector

```
// elimina todos los elementos del vector
void clear();

// inserta un elemento con valor x al final del
// vector, redimensionándolo si es necesario.
// Es equivalente a v.insert(v.end(), x).
void push_back(const T& x);

// quita el último elemento del vector
// es equivalente a v.erase(v.end())
void pop_back(const T& x);
```

List

Es una lista lineal y bidireccional (secuencial como toda lista). Permite insertar y eliminar en cualquier posición.

http://www.cplusplus.com/reference/stl/list

List

```
// constructor por defecto, crea una lista vacía.
list<T>();

// constructor: crea una lista con n elementos
inicializados con el valor value.
list<T>(size_type n, const T& value);

// permite asignar o crear una copia de
// objetos de tipo list.
list<T>& operator=(const list<T>& x);

// retorna la cantidad de elementos de la lista.
size_type size();
```

List

```
// retorna un iterador al primer elemento de la lista
list<T>::iterator begin();

// devuelve un iterador a la posición final de la
// lista (posición posterior al último valor)
list<T>::iterator end();

// devuelve verdadero si la lista está vacía, o falso,
// cuando ocurre lo contrario.
bool empty();

// elimina todos los elementos de la lista,
// dejándola vacía.
void clear();
```

List

```
// insertar un elemento x en la posición position.
// Devuelve un iterador a la posición del elemento
// insertado.
list<T>::iterator insert(  list<T>::iterator position,
                           const T& x);

// elimina del vector el elemento en la posición
// position o los elementos en el rango [first, last).
// Devuelve un iterador a la posición siguiente del
// último de los elementos eliminados.
list<T>::iterator erase(list<T>::iterator position);
list<T>::iterator erase(   list<T>::iterator first,
                           list<T>::iterator last);
```

List

```
// inserta un elemento con valor x al final de la
// lista. Es equivalente a l.insert(l.end(), x)
void push_back(const T& x);

// quita el último elemento de la lista.
// Es equivalente a l.erase(l.end())
void pop_back(const T& x);

// inserta un elemento con valor x al principio de
// la lista. Es equivalente a l.insert(l.begin(), x)
void push_front(const T& x);

// quita el primer elemento de la lista.
// Es equivalente a l.erase(l.begin())
void pop_front(const T& x);
```

List

STL provee un conjunto de algoritmos genéricos que pueden ser aplicados a cualquier estructura de datos, sin embargo estos algoritmos requieren que la estructura sea de acceso aleatorio. Este no es el caso de la lista, por lo cual la misma implementa estos algoritmos como funciones miembro.

List

```
// Ordena la lista.
void sort();

// Invierte el orden de los elementos en la lista.
void reverse();

// Elimina de la lista los elementos repetidos,
// dejando sólo la primera ocurrencia de cada valor.
// Por ejemplo, dada una lista con los elementos
// (5, 5, 10, 20, 20, 21), luego de aplicar unique(),
// se obtiene (5, 10, 20, 21). La lista debe estar
// ordenada
void unique();
```

List

```
// mezcla la lista actual con la lista x, manteniendo
// el orden. Ambas listas deben estar previamente
// ordenadas. Por ejemplo, dadas las listas
// A=(1, 2, 3, 4) y B=(3, 4, 5, 6), luego de hacer
// A.merge(B), A=(1, 2, 3, 3, 4, 4, 5, 6).
void merge(list<T> &x);
```

List

```
// la primera versión de esta función mueve los
// elementos de la lista x a la posición p de la lista
// actual. La segunda versión permite realizar la
// misma operación, pero no sobre la lista completa,
// sino en el rango [first, last).
void splice(   list<T>::iterator p, list<T>& x);
void splice(   list<T>::iterator p, list<T>& x,
               list<T>::iterator first,
               list<T>::iterator last);
```

List

Ejemplo:

Generar una lista e insertarle 10 valores enteros aleatorios entre 0 y 15, ordenarlos y mostrar la lista ordenada. Luego eliminar los valores repetidos y mostrar la lista.

Vector vs List

	Vector	**List**
Organización en memoria	Contigua	No contigua
Tipo de acceso	Aleatorio	Secuencial
Inserción/eliminación al principio	Muy lenta	Rápida
Inserción/eliminación al final	Rápida	Rápida
Inserción/eliminación en el medio	Lenta	Rápida

Stack

Una pila estándar. El último en entrar es el primero en salir (LIFO).

http://www.cplusplus.com/reference/stl/stack

Stack

```
// constructor, crea una pila vacía
stack<T>();

// inserta el elemento x en el tope de la pila
void push(const T &x);

// quita el elemento que está en el tope de la pila
void pop();

// devuelve el elemento en el tope de la pila
T top();

// devuelve verdadero si la pila está vacía
bool empty();
```

Stack

Ejemplo:

Cree una pila para almacenar nombres de personas. Muestre el tamaño de la pila y luego desapile cada elemento y muéstrelo.

Queue

Una cola estándar. El primero en entrar es el primero en salir (FIFO).

http://www.cplusplus.com/reference/stl/queue

Queue

```
// constructor, crea una cola vacía.
queue<T>();

// inserta el elemento x final de la cola
void push(const T &x);

// quita el elemento que está en el frente de la cola
void pop();

// devuelve verdadero si la cola está vacía
bool empty();
```

Queue

```
// devuelve el elemento del frente de la cola,
// es decir, el próximo por salir.
T front();

//  devuelve el elemento del final de la cola,
// es decir, el último en entrar.
T back();
```

Queue

Ejemplo:

Cree una cola para almacenar datos de la estructura mostrada debajo. Encole los elementos y muestrelos a medida que son eliminados.

```
struct Paciente{
    string nombre;
    int numero;
}
```

Map

Contenedor asociativo para almacenar pares de la forma: clave/valor. Cada clave está asociada a un único valor y no se permite la existencia de claves repetidas (correspondencia uno a uno). Permite realizar búsquedas por clave.

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

Map

```
// constructor, crea un mapeo vacío
// con claves de tipo T y valores de tipo S
map<class T, class S> map();

// Permite asignar o realizar una copia entre mapeos
map<class T, class S>&
operator=(const map<class T, class S>& x);

// devuelve la cantidad de elementos en el mapeo.
size_type size();

// devuelve verdadero si el mapeo está vacío
bool empty();
```

Map

```
// devuelve un iterador al comienzo del mapeo
map<class T, class S>::iterator begin();

// devuelve un iterador a la posición final del mapeo
// (posición posterior al último valor)
map<class T, class S>::iterator end();

// elimina todos los elementos del mapeo
void clear();
```

Map

```
// inserta un nuevo elemento en el mapeo. El mismo
// consiste en un par formado por una clave de tipo T
// y un valor de tipo S, modelado mediante otro tipo
// de contenedor: pair<class T, class S>.
// El mapeo no permite la existencia de claves
// duplicadas, por lo cual el elemento no se insertará
// en el mapeo en caso de que ya exista otro con
// la misma clave y tampoco se modificará el valor
// mapeado existente para la clave.
// La función devuelve otro par de valores, con un
// iterador a la posición del elemento insertado y un
// booleano, que indica si el valor ha sido insertado
pair<map<class T, class S>::iterator, bool>
insert(const pair<class T, class S>& x);
```

Map

```
// permite eliminar un solo elemento o un rango de
// ellos
map<class T, class S>::iterator
erase(map<T, S>::iterator position);
map<class T, class S>::iterator
Erase(   map<T, S>::iterator first,
         map<T, S>::iterator last);

// devuelve el valor mapeado asociado a la clave key.
// Si no existe un valor con dicha clave, el mapeo
// inserta un nuevo elemento con clave key y un valor
// mapeado inicializado por defecto.
S &operator[](T &key);
```

Map

```
// busca el elemento, cuya clave es igual a key, y
// devuelve un iterador al mismo, en caso de haberlo
// encontrado, o map::end(), si ocurre lo contrario.
map<class T, class S>::iterator find(T &key);
```

Map

Ejemplo:

Utilice un mapeo para manejar el conjunto de calificaciones de un grupo de alumnos. Pida al usuario que ingrese el nombre de un alumno e informe su nota o un mensaje alusivo en caso de no encontrar al alumno.