RESUMEN STL EDI 2

CONTENEDOR:

Cualquier clase capaz de contener objetos de otra clase. Operaciones:

- Un constructor por defecto y constructor copia.
- Un operador de asignación (operador=).
- Las operaciones begin() y end().
- La función size().
- La función empty(), que devuelve un valor true cuando el contenedor está vacío.

ITERADOR:

Clase especial capaz de acceder a los elementos de un contenedor concreto, para leer, escribir datos, o realizar desplazamientos sobre éstos. Operaciones:

- Un constructor por defecto y un constructor copia.
- Un operador de asignación (operador=).
- Un operador de igualdad (==) y desigualdad (!=).
- Un operador de referencia (*) para lectura y escritura del dato apuntado.
- Un operador de incremento (++).
- Un operador de decremento (--).

CONTENEDORES SECUENCIALES:

Son aquellos donde los datos están dispuestos linealmente y el acceso se realiza por posición.

- front() → Devuelve el elemento situado en la primera posición.
- back() → Devuelve el elemento situado en la última posición.
- insert() → Inserta elementos.
- erase() → Elimina elementos.
- clear () → Borra todos los elementos del contenedor.

1. - VECTORES DINÁMICOS (VECTOR)

Permite el acceso aleatorio en tiempo constante a partir de la posición. Permite inserciones en posiciones finales en tiempo constante. En cualquier otra posición es en tiempo real. Añade las siguientes operaciones a las ya citadas:

-operator[] → Acceso aleatorio de datos.
-push_back(t:T) → Inserta en el último elemento.
-pop_back():T → Elimina el último elemento.

Definición: ***vector<tipo_dato> nombre_vector;***

Definición iterador: ***vector<tipo_dato>::iterator nombre_iterador;***

2.- DEQUE (VECTORES DINÁMICOS MEJORADOS)

Similar a un vector, pero con inserción en tiempo constante tanto al principio como al final. Además permite añadir elementos sin costosas reasignaciones de memorias y copias de bloques e elementos. Añade las siguientes operaciones:

- push_front(t:T) → añade un elemento al comienzo de la estructura de datos.
- pop_front():T → saca el primer elemento del deque y lo devuelve.

Definición: ***deque<tipo_dato> nombre_deque;***

Definición iterador: ***deque<tipo_dato>:: iterator nombre_iterador;***

3.- LIST (LISTAS ENLAZADAS)

Es una lista doblemente enlazada que admite inserciones y borrados en tiempo constante en cualquier posición, siempre que esté apuntada por un iterador. Operaciones:

- push_front(t:T), pop_front():T → inserción/borrado elemento al comienzo de la lista.
- push_back(t:T), pop_back():T → inserción/borrado elemento al final de la lista.
- splice() → mueve elementos de una lista a otra.
- remove(const T &valor) → elimina todas las ocurrencias en la lista que coinciden con un valor concreto.

- unique() → elimina todos los elementos repetidos.
- merge(list<T> &x) → extrae y mezcla los elementos de x con las dos listas, proporcionando una lista ordenada.
- reserve() → invierte el orden de los elementos de la lista.
- sort() → ordena los elementos de la lista (por defecto <)

Declaración: ***list<tipo_dato> nombre_lista;***

Definición iterador: ***list<tipo_dato>::iterator nombre_iterador;***

CONTENEDORES ASOCIATIVOS

Son aquellos que tiene acceso por clave eficiente, en tiempo logarítmico (basados en árboles) o en tiempo constante (basados en tablas hash). Tipos:

-set (conjunto): El propio dato sirve como clave de ordenación.
-map (mapa): Existe diferenciación entre la clave de ordenación y el dato almacenado.

El prefijo multi indica que las claves (tanto e un map como de un set) pueden repetirse. Y el prefijo hash_ define estos mismos contenedores sobre tablas hash en lugar de árboles binarios.

1.- CONJUNTOS (SET): La clave y dato coinciden. (si no se indica función de comparación, se toma less<T> por defecto). Operaciones:

- pair<itrador,bool>
- insert() → inserción de un dato.
- erase(T t) → elimina un elemento.
- iterator find(T t) → busca el dato t devolviendo la posición o end() si no está el dato.
- iterator lower_bound(T t) → busca a t, si no está retorna la posición siguiente.
- iterator upper_bound(T t) → busca al dato siguiente a t.

Definición: ***set<tipo_dato, [clase_comparación]>nombre_set;***

2.- MULTICONJUNTOS (MULTISET): Un conjunto que permite datos repetidos. Diferencias respecto a set:

- insert(T t) → devuelve un iterador apuntado al dato insertado.
- find(), lower_bound(), upper_bound() → devuelven la primera ocurrencia del dato (si existen varias).

Definición: ***multiset<tipo_dato, [clase_comparación]> nombre_multiset;***

3.- MAPAS (MAP): La clave y el dato son distintos. Difrencias respecto a las operaciones de set:

- pair<iterator,bool>insert(pair<Tclave, Tdato>t) → donde se indica el valor clave y el dato utilizando un pair.
- Tdato& operator[](Tclave c) → permite un acceso intuitivo a los elementos, para lectura/escritura.

Declaración: ***map<tipo_clave, tipo_dato, [clase_comparación]> nombre_map;***

4.- MULTIMAPAS (MULTIMAP): Admite múltiples valores con idéntica clave. Diferencias respecto a map:

- iterator insert(pair<Tclave, Tdato>t) → inserta el elemento t, que no devuelve un pair, sólo un iterador apuntando al dato insertado.
- No está disponible el operador[]

Declaración: ***multimap<tipo_clave, tipo_dato, [clase_comparación]>nombre_multimap;***

CONTENEDORES ASOCIATIVOS BASADOS EN DISPERSIÓN

Este tipo de contenedores proporciona un tiempo hipotético de acceso de O(1). Implementa la dispersión abierta mediante un vector de listas enlazadas. Se verifica siempre que tamaño_vector >= num_elementos.
Para ello el vector crece automáticamente al doble de su tamaño actual; necesitando una redispersión de todos los valores (proceso bastante costoso).

Para evitar redispersiones es conveniente indicar al principio el máximo número de elementos que van a manejarse (operación resize()). La definición de este tipo de contenedor necesita indicar una **clase de dispersión y una clase de comparación**. Por defecto la clase de comparación es equal_to<T> y la clase de dispersión es hash<T> que proporciona dispersión por división del dato. El recorrido secuencial con iteradores sobre el contenedor está garantizado, aunque el resultado no es un conjunto ordenado.

Las operaciones son similares a los de sus correspondientes Contenedores Asociativos basados en árboles.

1.- CONJUNTOS DISPERSOS (HASH_SET)

```
#include <iostream>
#include <hash_set>
using namespace std;
int main() {

hash_set<int, hash<int>, equal_to<int> > c,d;
```

2.- MULTICONJUNTOS DISPERSOS (HASH_MULTISET)

```
#include <iostream>
#include <hash_multiset>
using namespace std;
int main() {
hash_multiset<int, hash<int>, equal_to<int> > c,d;
```

3.- MAPAS DISPERSOS (HASH_MAP)

```
#include <iostream>
#include <hash_map>
using namespace std;
typedef char nombre[20];
typedef unsigned long dni;

int main() {
hash_map<dni, nombre, hash<dni>, equal_to<dni> > m;
```

4.- MULTIMAPAS DISPERSOS (HASH_MULTIMAP)

```
#include <iostream>
#include <hash_multimap>
using namespace std;
typedef unsigned int dni;
typedef unsigned int proyecto;
int main() {
hash_multimap<proyecto, dni, hash<proyecto>, equal_to<proyecto> > m;
m.insert(pair<proyecto,dni>(668, 33398776));
```