STL

Containers

Los containers pueden ser :

Adaptors:

Stack (**pila)**

Queue (**cola**)

De secuencia:

Vector

Deque(**Como vector pero permite ingresar y sacar por el frente**)

List

Asociativos:

Map(key,valor)

Multimap

Set(**lista sin repertidos**)

Multiset

Adaptors:

Stack: es una pila;

Debo incluir <stack>

Declaración:

Stack<tipo de dato>nombre;

Ej. Stack<int> p;

Funciones miembro

p.**Push (t dato);** agrega un dato a la pila

p.**Pop ();** elimina el dato q este en top

p.**Top ();** muestra el valor de top

p.**Empty ();** me dice si la pila esta vacia

p.**size ();** me dice la cantidad de elementos de la pila

Queue: es una cola;

Debo incluir <queue>

Declaración:

Queue <tipo de dato> nombre;

Ej. Queue<int> c;

Funciones miembro

c.**Push (t dato);** agrega un dato a la pila

c**.Pop ();** elimina el dato q este en top

c.**front ();** muestra el próximo valor a sacar

c.**back ();** muestra el ultimo valor agregado

c.**Empty ();** me dice si la cola esta vacia

c.**size();** me dice la cantidad de elementos de la cola

priority queue

Debo incluir <queue>

Declaración:

Priority_queue<tipo de dato>nombre;

Ej. **Priority_queue** <int> cola_p;

Funciones miembro

cola_p.**Push (t dato);** agrega un dato a la pila

cola_p.**Pop ();** elimina el dato q este en top

cola_p.**Top ();** muestra el valor de top

cola_p.**Empty ();** me dice si la pila esta vacia

cola_p.**size ();** me dice la cantidad de elementos de la pila

Funciones miembro comunes en todos los contenedores(secuenciales y asociatvos)

`Contenedor`**`::iterator`**
crea iterador

`Contenedor.`**`begin()`**
devuelve iterator aprimera posición del contenedor

`contenedor.`**`end();`**
devuelve iterator a ultima posición;

`contenedor.`**`size()`**
devuelve cantidad de elementos del contenedor

`contenedor.`**`empty()`**
devuelve true si esta vacio

`contenedor.`**`erase(iterator)`**
borra el elemento en la posición iterator

`contenedor.`**`erase(iterator,iterator2)`**
borra los elementos entre iterator y iterator2

`contenedor.`**`clear()`**
borra todos los elementos del contenedor

`contenedor.`**`swap(cont2 )`**
intercambia contenedor y cont2(contenedor y cont dos deben ser del mismo tipo por ej dos listas)

CONTENEDORES SECUENCIALES

VECTOR

#include <vector>

DECLARACION:

vector<tipo dato> nombre;
Ej vector<char> letras;

FUNCIONES MIEMBRO:

`letras.`**push_back(dato);**
agrega un elemento al final(si esta vacio crea uno)

`letras.`**pop_back();**
saca el ultimo elemento

`letras.`**at(INT POS);**
devuelve el elemento de la posicion POS

`letras.`**capacity();**
devuelve la capacidad del vector

`letras.`**back();**
devuelve el elemento de la ultima posición

`letras.`**front();**
devuelve el elemento de la primera posición

`letras.`**insert(iterator,value);**
en la posicion del iterator copia el valor de value

Deque

es un vector que permite agregar y eliminar por el frente
#include <deque>

DECLARACION:

deque<tipo dato> nombre;
Ej deque<char> letras;

Funciones miembro

Las de vector y se agregan:

`Letras.`**Push_front(dato)**Agrega un dato en la primera posición

`Letras.`**Pop_front();**Elimina el primer element

Lista

#include <list>

DECLARACION:

list<tipo dato> nombre;
Ej list<char> letras;

FUNCIONES MIEMBRO:

`letras.`**push_back(dato);**
agrega un elemento al final(si esta vacio crea uno)

`letras.`**push_front(dato);**
agrega un elemento al principio(si esta vacio crea uno)

`letras.`**pop_back();**
saca el ultimo elemento

`letras.`**pop_front();**
saca el primer elemento

`letras.`**back();**
devuelve el elemento de la ultima posición

`letras.`**front();**
devuelve el elemento de la primera posición

`letras.`**insert(iterator,value);**
en la posicion del iterator copia el valor de value

`letras.`**merge(list L2);**
le agrega al final de letras la lista L2

`letras.`**sort()**
ordena de menor a mayor

`letras.`**reverse()**
ordena de mayor a menor

`letras.`**splice(iterator, list L2);**
en la poicion iterator de letras agrega la lista L2(L2 queda vacia)

`letras.`**remove(dato)**
remueve todas las ocurrencias de dato en la lista

`letras.`**remove_if()**

remueve si se cumple una condicion

CONTENEDORES ASOCIATIVOS

set:

set es una lista de elementos que no tiene elementos repetidos
#include <set>

Declaración

set<tipo> nombre;
set<char> conjunto;

funciones miembro:

conjunto.**insert(tipo dato)**;
devuelve un pair <iterator,bool>
el first indica la posición donde se quiso insertar y el second si se logro o no insertar

conjunto.**count(dato);**
cantidad de veces q aparece dato en el conjunto(debería ser 1)

conjunto.**equal_range(dato);**
devuelve un pair<iterator,iterator> con el rango donde aparece dato
(pair<primera_aparición, ultima_aparicion>)

conjunto.**find(dato);**
busca la posición del dato devuelve un iterator a la posición

conjunto.**lower_bound(dato);**
devuelve iterator (busca dato y devuelve la posición anterior)

conjunto.**upper_bound(dato);**
devuelve itertor(busca dato y devuevle la posición siguiente a dato)

Map:

Contenedor asociativo asocia una clave a un valor
#include <map>

Declaración

Map<tipo1 clave,tipo2 valor>nombre;
Map<string,int>palabras;

Funciones miebro

Palabras.**find(key);** busca la posición de una key devuelve iterator a posición

Palabras.**upper_bound(key);** devuelve laposicion siguiente a la ultima aparición de la key

Palabras.**lower_bound(key)** devuelve la posición anterior a la key buscada

Palabras.**erase(iterator,iterator2);** borra un rango de valores entre iterator y iterator2

Palabras.**equal_range(key);** encuentra el rango en el q aparece la key buscada devuelve un pair<iter ator,iterator2> , first es la posición de la primera aparición y el second es la ultima aparición de la key

Palabras.**count(key)** cuenta cuantas veces aparece una key

Palabras.**insert(make_pair(dato tipo 1, dato de tipo2));** agrega un pair a mi map

ALGORITMOS

#include <algorithms>

No mutating

For_each(iterator principio,iterator final,función f){}
sirve para recorrer contenedores

Find(iterator principio,iterator final,valor);
busca el valor desde indicado en el rango(princio,final) devuelve true si encuentra

Find_if((iterator principio,iterator final,condicion);)
mismo q el anterior pero devuelve true si se cumple la condición

Count (iterador principio , iterador final, valor, n)
cuenta cunatas veces aparece valor desde principio a final

Count_if(iterador principio, iterador final , condición , n)
cuenta si se cumple una condición

Mutating

Copy(iterator de entrada primero, iterator de entrada final, iterador de salida primero2)
copia desde primero hasta final y lo guarda en primero2

Remplace(iterator primero, iterator ultimo, valor viejo, valor nuevo)
Remplaza valor viejo por valor nuevo

Remplace_if(iterator primero, iterator ultimo, condición , valor nuevo)
Remplaza por valor nuevo si cumple con la condición

Fill(primero, ultimo, valor)
Llenar un contenedor desde primero hasta ultimo con valor

Fill_n(primero,valor ,n)
Mismo q el anterior pero desde primero posicion n

Remove(primero,ultimo,valor)
Borra valor desde primero hasta ultimo

Remove_if(primero,ultimo,condicion)
Borra si se cumple una condición

Unique(primero, ultimo)
elimina elementos repetidos

Reverse(primero, ultimo)
Ordena al revés

De busqueda

Sort(primero, ultimo);
Ordena

Búsqueda binaria

Binary_search(primero, ultimo, valor)
Devuelve true si encuentra valor en el rango

Lower_bound(primero, ultimo,valor)
Devuelve un iterator a la posicion anterior a la primera aparición de valor

Upper_bound(primero, ultimo , valor)
Devuelve un iterator a la posicion siguiente a la ultima aparición de valor

Equal_range(primero, ultimo, valor)
Devuelve un pair<iterator,iterator2> donde el first del pair indica la primera posicion del rango y el second la ultima

De conjuntos

Set_union(primero1, ultimo1, primero2,ultimo2, resultado)
Recorre dos contenedores desde primero a ultimo , hace la unión y lo guarda en resultado

Set_intercection(primero1, ultimo1, primero2,ultimo2, resultado)
Recorre dos contenedores desde primero a ultimo , hace la interseccion y lo guarda en resultado

Set_difference(primero1, ultimo1, primero2,ultimo2, resultado)

Recorre dos contenedores desde primero a ultimo , hace la diferencia y lo guarda en resultado

`Set_symmetric_difference(primero1, ultimo1, primero2,ultimo2, resultado)`

Recorre dos contenedores desde primero a ultimo , hace la diferencia simétrica y lo guarda en resultado

Min y max

`Min_element(primero,ultimo)`
Devuelve Iterador a la posicion del elemento mínimo

`Max_element(primero, ultimo)`
Devuelve Iterador a la posicion del elemento máximo

Computacional

`Accumulate(primero, ultimo,valor)`
Recorre y va acumulando en valor

ITERATORS

#Include <iterators>

Tipos de iteradores

Input

Output

Forward

Bidirectional

random acces

stream iterators(istream iterators , ostream iterators**)**

adaptors iterators

reverse iterators

insert iterators(back_inserted_iterator, front_inserted_iterator,insert_iterator**)**