LECCION 14 STL

STL: Standard Template Library

_ESTRUCTURAS de DATOS LINEALES

- .vector ⟹ #include <vector>
- .list ⟹ #include <list>
- .queue, .priority-queue } ⟹ #include <queue>
- .stack ⟹ #include <stack>

① LECCION 14: STL

LISTAS ⟹ List
Hay que incluir <list> + using namespace std.

ITERADORES

- RBEGIN y REND (SACAR EL CONTENIDO EN DISTINTAS DIRECCIONES)
+EMPTY

```
#include <iostream>
#include <list>
using namespace std;
int main(){
1) list <int> milista;
2) for (int i=1; i<=5; i++) mylista.push_back(i);

3) cout << "El contenido de la lista es";
4) list<int>::reverse_iterator rit;
5) for (rit=mylista.rbegin(); rit != milista.rend(); ++rit)

6)          cout << *rit << " ";

7) list<int>::iterator ri;
8) for (ri=milista.begin(); ri != milista.end(); ++ri)
9)          cout << *ri << " ";

10) cout << "El tamaño" << milista.size();
    int sum=0;
11) while (!milista.empty()){
12)          sum += milista.front(); //milista.back()
13)          milista.pop_front(); // milista.pop_back()
    }
```

- NOTA: Para listas constantes tenemos los iteradores
  - const_iterator y const_reverse_iterator
  - Las funciones para inicializarlos y saber donde terminar
    - .cbegin() y cend() — para los const_iterator
    - .crbegin() y crend() — para los const_reverse_iterator

(2) LISTAS STL

LECCION 14: STL

ASSING, INSERT, ERASE. para LISTAS.

```
#include <iostream>
#include <list>
using namespace std;
int main(){
  int mienteros[]={13,12,10,20};
  list<int> milista;
  milista.assign(mienteros, mienteros+4);
  //milista.assign(7,100); seria 7 enteros con valor 100
  list<int> otralista;
  otralista.assign(milista.begin(), milista.end());

  list<int> l3;
  list<int>::iterator it;
  for(it=otralista.begin(); it!=otralista.end(); ++it){
        l3.push_back(*it);
  }
  //l3= 13  12  10  20
  //INSERT
  it= l3.begin(); ++it;
  l3.insert(it,2,30); // l3= 13 30 30 12 10 20
  l3.insert(it,4); // l3= 13 30 30 4 12 10 20
  it= milista.begin();
  milista.insert(it, v.begin(), v.end());
//milista= 13 30 30 4 12 10 20 13 12 10 20.
  milista.erase(milista.begin()+1, milista.begin()+3);
  milista.erase(milista.begin());
}
```

it

it

elimina el primero

③ LISTAS

LECCION 14: STL

```
#include <iostream>
#include <list>
using namespace std;
int main(){
    // Intercambiar las dos listas
    milista.swap(l3);

    //Borrar el contenido de l3
    l3.clear();
```

con el código de la pg anterior.

SPLICE: Mueve los elementos de una lista a otra

```
list <int> l1, l2;
list <int>::iterator it;

for (int i= 1; i<= 4; i++)
        l1.push_back(i); // l1= 1 2 3 4

for (int i=1; i<= 3; i++)
        l2.push_back (i*10); // l2= 10 20 30

it = l1.begin(); ++it;
l1.splice (it, l2); // l1= 1 10 20 30 2 3 4
                    // l2 = empty
                    // it todavia apunta a 2

l2.splice (l2.begin(), l1, it); // l1= 1 10 20 30 34
                                // l2= 2
                                // it ahora se invalida

it = l1.begin();
advance (it, 3); // it apunta a 30
l1.splice ( l1.begin(), l1, it, l1.end()); // l1=30 3 4 1 10 20
```

it ↓

1 10 20 30 2 3 4

1 10 20 30 34

4 Listas LECCION 14 STL

REMOVE

```
int mientras[] = { 3, 4, 79, 15 };
list<int> l(mientras, mientras+4);

l.remove(79);

list<int>::iterator it;
for (~~int~~ it= l.begin(); it!= l.end(); ++it)
        cout<< *it;
```

REMOVE_IF: Elimina los elementos que cumplan una condición

```
bool   Par(int v){
        return (v%2==0);
}
int main(){
    list<int> l;
    for(int i=1; i<10; i++)
        l.push_back(i); //1 2 3 4 5 6 7 8 9

    l.remove_if(Par); // l= 1 3 5 7 9
}
```

UNIQUE: Elimina valores duplicados que se encuentran consecutivamente dejando una única ocurrencia.

```
bool iguales_enteros(double v1, double v2){
        return (int)v1 == (int)v2;
}
```

(5) LISTAS LECCION 14: STL

```
int main(){
  double m[] = { 12.15, 2.72, 73.0, 12.77, 3.14,
                 12.77, 73.35, 72.25, 15.3, 72.254};

  list <doubles> l1(m, m+10);

  l1.sort();   //l1 = 2.72, 3.14, 12.15, 12.77, 12.77, 15.3
                     72.25, 72.25, 73.0, 73.35

  l1.unique(); //l1 = 2.72, 3.14, 12.15, 12.77, 15.3, 72.25
                     73, 73.35

  l1.unique(iguales_enteros); l2 = 2.72, 3.14, 12.15, 15.3,
                                   72.25, 73

};
```

MERGE → mezcla dos listas ordenadas.
REVERSE: Invierte.

```
bool micomparacion(double v1, double v2){
    return (int)v1 < (int)v2;
}
int main(){
  list <double> l1, l2;
  l1.push_back(3.1); l1.push_back(2.2); l1.push_back(2.9);

  l2.push_back(3.7); l2.push_back(7.1); l2.push_back(1.4);

  l1.sort(); //l1 = 2.2  2.9  3.1
  l2.sort(); // l2 = 1.4  3.7  7.1        se elimina de l2
  l1.merge(l2); //l1 = 1.4  2.2  2.9  3.1  3.7  7.1
  l2.push_back(2.1); // l2 = 2.1
  l1.merge(l2, micomparacion); //l1 = 1.4  2.2  2.9  2.1  3.1
                                     3.7  7.1
  l1.reverse();
```

LECCION 14: STL

Pares de valores: Pair

La clase pair es un struct que contiene dos pares de valores con tipos iguales o diferentes.

El struct se define como

```
template <class T1, class T2>
struct pair {
    typedef T1 first_type; //alias de T1
    typedef T2 second_type; //alias de T2

    T1 first;
    T2 second;
    pair(): first(T1()), second(T2()) {}
    pair(const T1& x, const T2& y): first(x), second(y) {}

    template <class U, class V>
    pair(const pair<U,V>& p): first(p.first), second(p.second) {}
};
```

Ejemplo

```
#include <vector>
#include <iostream>
#include <utility> //permite hacer comparaciones de dos objetos pair
#include <string>
using namespace std;
int main() {
    pair <string, double> product1("tomates", 3.25);
    pair <string, double> product2;
    pair <string, double> product3;
    product2.first = "naranjas"; product2.second = 1.3;
    product3 = make_pair("ciruelas", 2.2);
    vector < pair <string, double> > miv={product1, product2, product3};
    for (int i=0; i <miv.size(); i++)
        cout << "Producto " << i << ": " << miv[i].first << "Precio: " << miv[i].
                                                                    second<<en

}
```

Funciones Miembro

			Contenedores	
Cabeceras			<vector>	<list>
Funciones Miembro		Eficiencia	**vector**	list
	constructor	*	constructor	constructor
	destructor	O(n)	destructor	destructor
	operator=	O(n)	operator=	operator=
iteradores	begin	O(1)	begin	begin
	end	O(1)	end	end
	rbegin	O(1)	rbegin	rbegin
	rend	O(1)	rend	rend
capacidad	size	*	size	size
	max_size	*	max_size	max_size
	empty	O(1)	empty	empty
	resize	O(n)	resize	resize
acceso a los elementos	front	O(1)	front	front
	back	O(1)	back	back
	operator[]	*	operator[]	
	at	O(1)	at	
modificadores	assign	O(n)	assign	assign
	insert	*	insert	insert
	erase	*	erase	erase
	swap	O(1)	swap	swap
	clear	O(n)	clear	clear
	push_front	O(1)		push_front
	pop_front	O(1)		pop_front
	push_back	O(1)	push_back	push_back
	pop_back	O(1)	pop_back	pop_back

observadores	`key_comp`	O(1)		
	`value_comp`	O(1)		
operaciones	`find`	O(log n)		
	`count`	O(log n)		
	`lower_bound`	O(log n)		
	`upper_bound`	O(log n)		
	`equal_range`	O(log n)		
miembros únicos			capacity reserve	splice remove remove_if unique merge sort reverse

O(1) constante < O(log n) logaritmica < O(n) lineal; *=depende del contendor