Universidad Autónoma de Entre Ríos | FCyT Facultad de Ciencia y Tecnología

STL
Standard Template Library

Programación Orientada a Objetos

Lic. Leonardo Brambilla

¿Por qué aprender STL?

Los errores son un componente inevitable del desarrollo de software.

> *Every line of code you don't write is bug-free.*
>
> *John Lambert*

Cada línea de código que no escribe, está libre de errores.

¿Por qué aprender STL?

Es un conjunto de clases y funciones genéricas, implementadas como plantillas (templates), que facilitan el uso de estructuras de datos y algoritmos comunes sin que el programador tenga que re-implementarlos desde cero.

Ventajas de la STL

Código reutilizable: templates.

Eficiente: implementaciones muy optimizadas.

Consistente: misma interfaz para distintos contenedores.

Flexible: se puede usar con cualquier tipo de dato, incluso definidos por el usuario.

STL - Componentes

- **Contenedores:** estructuras de datos genéricas que almacenan elementos.

  Cada uno tiene diferentes características de almacenamiento y eficiencia.

  No saben de "algoritmos", solo guardan y permiten acceso a los datos.

  std::vector, std::list, std::deque, std::set, std::map

- **Iteradores:** Son Clases que permiten señalar datos o posiciones de los contenedores. Cada contenedor tiene sus propios iteradores, pero se usan de forma genérica.

Tipos comunes:

begin() , *end()*

Algunos soportan iteradores bidireccionales (*list*), otros aleatorios (*vector*).

- **Algoritmos:** Son funciones genéricas que operan sobre rangos de elementos usando iteradores. No dependen del contenedor, sino de los iteradores que se les pasen.

std::sort, std::find, std::count, std::for_each.

STL - Contenedores

1. **Contenedores secuenciales:** Implementan estructuras de datos con acceso secuencial.

   Incluyen: vector list deque array forward_list

2. **Adaptadores de contenedor:** Implementan estructuras como colas y pilas proporcionando diferentes interfaces sobre los contenedores secuenciales.

   Incluyen: stack queue priority_queue

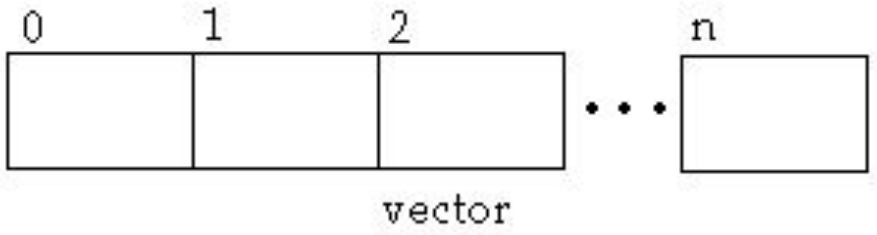
3. **Contenedores asociativos:** Se usan para almacenar datos ordenados, que pueden buscarse rápidamente mediante una clave.

   Incluyen: map multimap set multiset

STL: vector

Arreglo dinámico con acceso al azar.

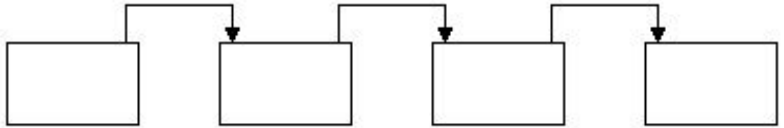
```
vector<int> v; // crea un vector de enteros vacío
v.resize(3); // el vector ahora tendrá 3 elementos
v[0]=1; v[1]=5; v[2]=7; // asigna valores a esos 3 e.
v.push_back(10); // agrega un 10 al final
v.insert(v.begin(), 1);  // Inserta al inicio
v.insert(v.begin()+2,7); // agrega un 7 en la 3ra pos.
v.erase(v.begin()+2); // elimina el 7 (3ra pos)
v.pop_back(); // elimina el último
for (int i=0;i<v.size();i++)
     cout<<v[i]<<" ";
```



STL: list

Lista enlazada, acceso secuencial

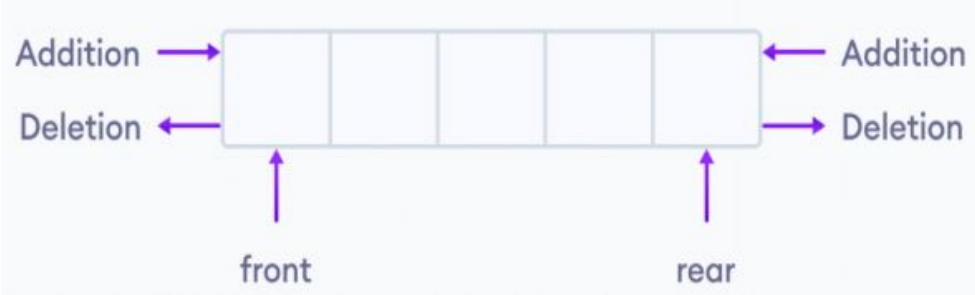
```
list<int> l; // crea un vector de enteros vacío
list<int>::iterator it; // iterador
v.push_back(10); // agrega un 10 al final
v.push_back(20); // agrega un 20 al final
v.push_front(5); // agrega un 5 al principio
it=l.begin(); l++; l++; // apunta con l al 3er elemento
it=l.insert(it,7); // agrega un 7 en la 3ra pos.
it=l.erase(it); // elimina el 7 (3ra pos.)
for (it=l.begin();it!=l.end();it++)
     cout<<*it<<" ";
```



STL: deque

Cola doblemente enlazada, acceso al azar

```
deque<int> d;  // crea un deque de enteros vacío

// Insertamos elementos

d.push_back(10);   // agrega un 10 al final

d.push_back(20);   // agrega un 20 al final

d.push_front(5);   // agrega un 5 al principio

// Insertar en la 3ra posición (índice 2)

auto it = d.begin() + 2;

it = d.insert(it, 7);  // agrega un 7 en la 3ra pos.

// Eliminar el elemento insertado

it = d.erase(it);      // elimina el 7 (3ra pos.)
```



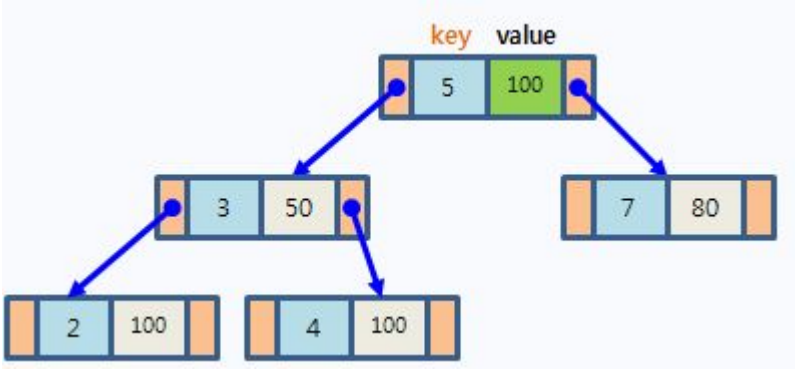
STL: map

Un map es un contenedor asociativo ya que dado un valor (key) , encontrará el otro valor (value):

```
map<string, int> //es un mapeo de strings a int
map<int, int> //es un mapeo de int a int
```

maps son 1:1

multimap son 1:n



STL: map


```
map<string,int> calif;
calif["Fundamentos de Programación"]=9;
calif["POO"]=7;
calif["Matemática Básica"]=8;
calif["Física I"]=7;
calif["Álgebra"]=7;
calif["Ingeniería de Software"]=7;
string materia;
getline(cin,materia);
if (calif.find(materia)==calif.end())
     cout<<"No se registra nota para esa materia.";
else
     cout<<"La nota es: "<<calif[materia];
```

STL: <algoritm>

Colección de algoritmos genéricos que operan sobre rangos (como los de vector, list, array, etc.)

```
bool es_par(int x) {
     return x%2==0;
     }


list<int> l;
list<int>::iterator it;
// busca el valor 10 en la lista
it = find( l.begin() , l.end() , 10 );
// busca el primer numero par
it = find_if( l.begin() , l.end() , es_par );
```

STL: <algoritm>


```
// elimina todas las veces que aparece el valor 10
remove( l.begin() , l.end() , 10 );
// elimina todos los números pares de la lista
remove_if( l.begin() , l.end() , es_par );

// reemplaza todos los 15 por 30
replace( l.begin() , l.end() , 15 , 30 );
// reemplaza todos los pares por 0
replace_if( l.begin() , l.end() , es_par , 0 );
```

STL: <algoritm>


```
// cuenta cuantas veces aparece el valor 10
int n1 = count( l.begin() , l.end() , 10 );
// cuenta cuantos números pares hay en la lista
int n2 = count_if( l.begin() , l.end() , es_par );

// suma todo el contenido de la lista
n = accumulate( l.begin() , l.end() , 0 );

// elimina los repetidos de la lista
it = unique( l.begin() , l.end() ); // los coloca al final
l.erase( it , l.end() ); // despues hay que borrarlos
```

STL: <algoritm>


```
// busca el maximo y minimo
it = max_element( l.begin() , l.end() );
it = min_element( l.begin() , l.end() );

// desordena el vector
random_shuffle( v.begin() , v.end() );

// invierte el orden de la lista
reverse( l.begin() , l.end() );
```