ESTRUCTURAS DE DATOS

NOTAS SOBRE C++

STL: Algoritmos (1)

Manuel Montenegro Montes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática – Universidad Complutense de Madrid

Vamos a ver 2 funciones MUY ÚTILES de la STL. La primera es COPY()

La función copy( )

La función copy( )

- Definida en `<algorithm>`

  `copy(source_begin, source_end, destination_begin)`

  donde:

  - `source_begin`, `source_end` son iteradores de entrada.
  - `destination_begin` es iterador de salida.

- Copia el intervalo de elementos delimitado por `source_begin` y `source_end` (excluyendo este último), a la posición apuntada por el iterador `destination_begin`.

copiaría hasta el último pero sin contar al último

Copia los elementos contenidos entre los iteradores source_begin y source_end en destination_begin

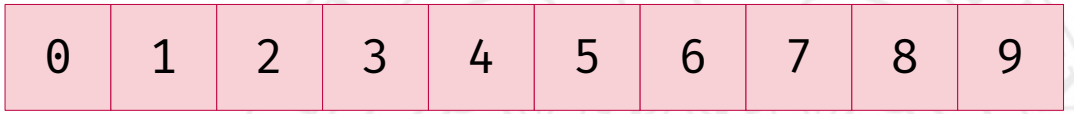
Ejemplo

```
int main() {
  vector<int> origen;
  list<int> destino;

  for (int i = 0; i < 10; i++) {
    origen.push_back(i);
    destino.push_back(100 + i);
  }
  ...
}
```

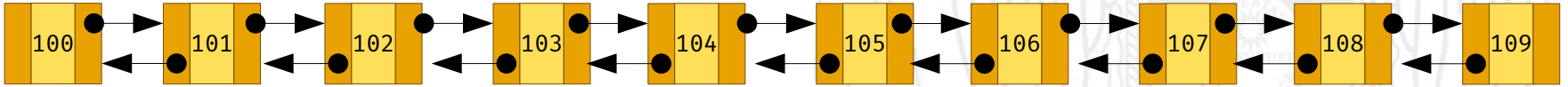
ListaNormal(vector)

Lista doblemente enlazada

origen:



destino:



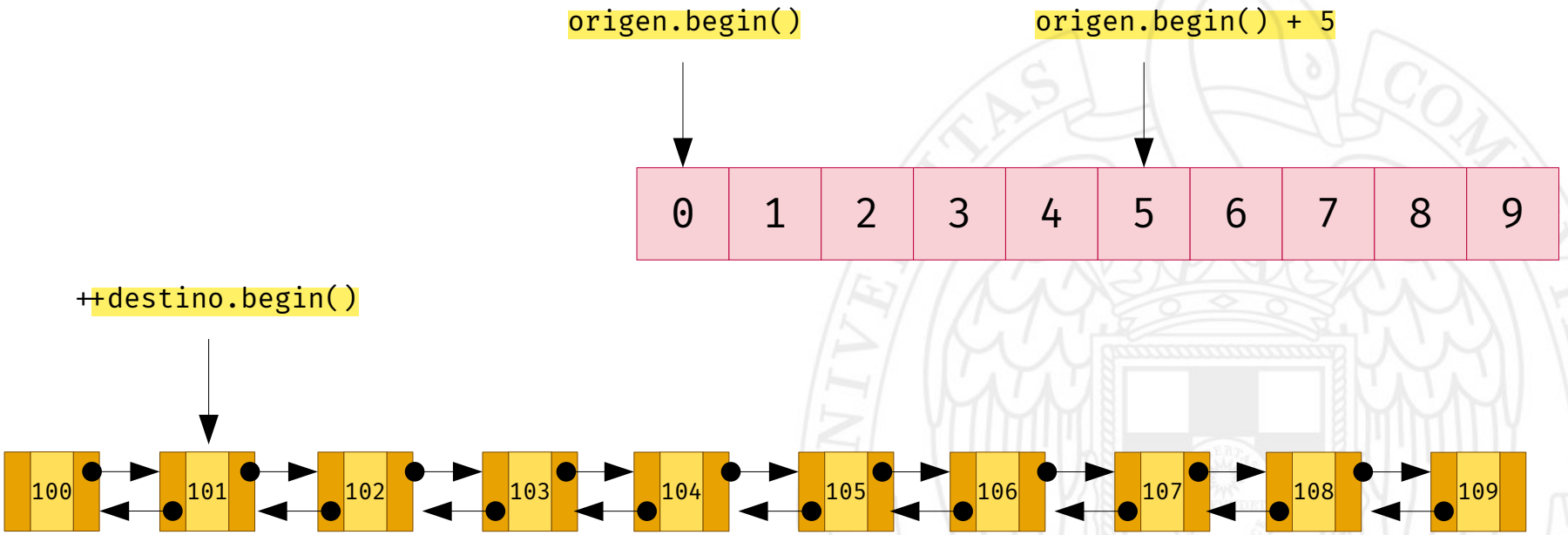
Ejemplo

source_begin()

source_end()

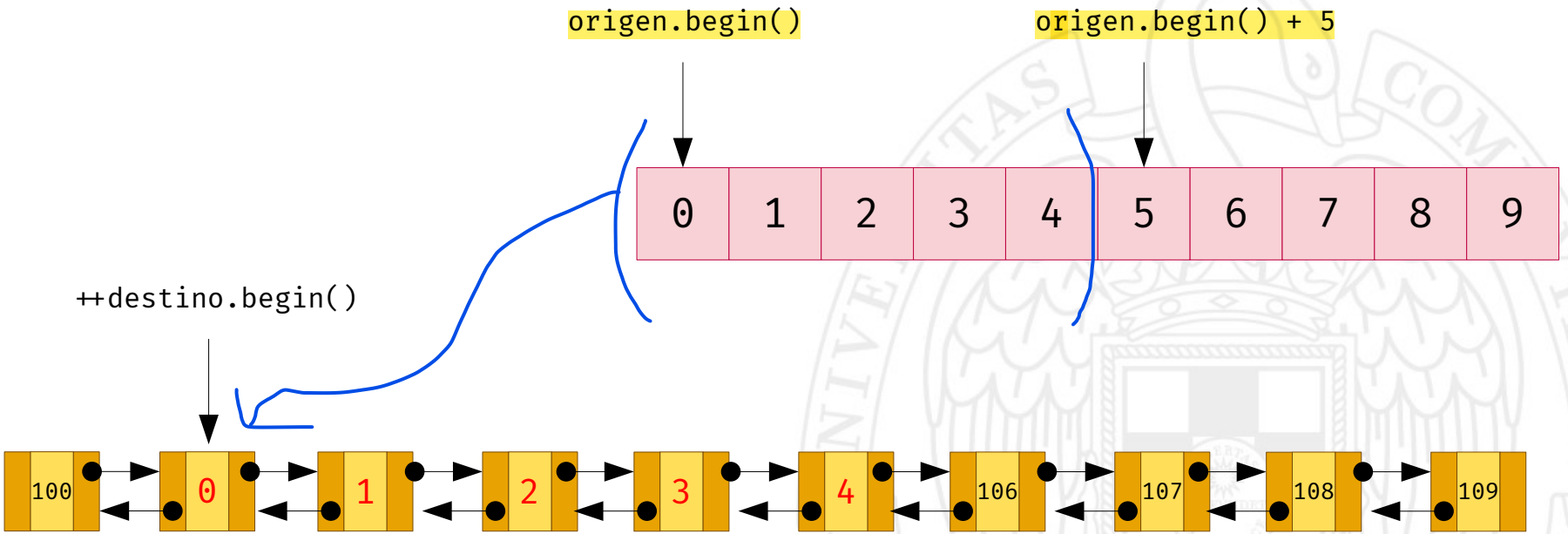
donde copiamos

```
copy(origen.begin(), origen.begin() + 5, ++destino.begin());
```



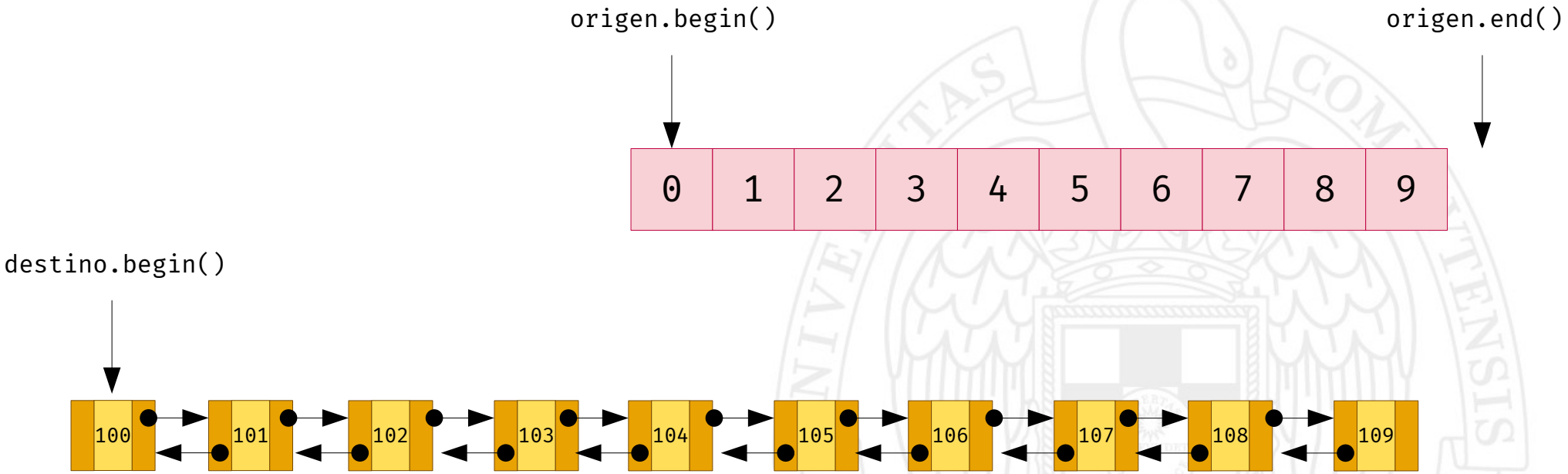
Ejemplo

```
copy(origen.begin(), origen.begin() + 5, ++destino.begin());
```



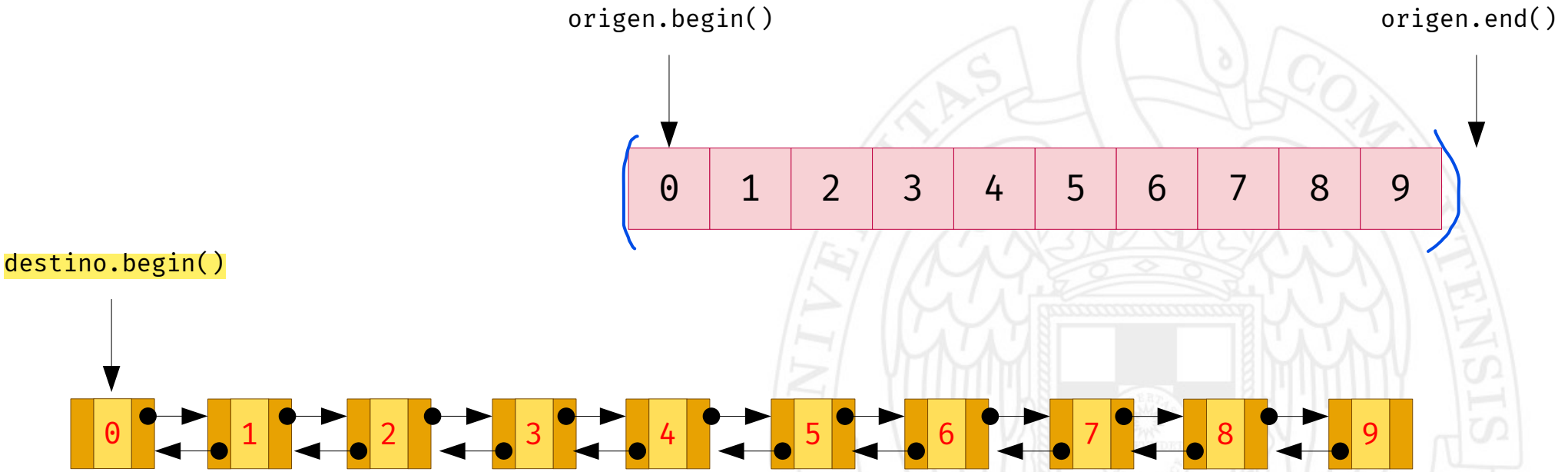
Ejemplo

```
copy(origen.begin(), origen.end(), destino.begin());
```



Ejemplo

```
copy(origen.begin(), origen.end(), destino.begin());
```



Utilidad de función copy( )

Trabaja sobre cualquier cosa que pueda tener iteradores.

- Se puede utilizar para multitud de casos:
  - De un `vector` a un `list` y viceversa.
  - De `vector` a `vector`.
  - De `list` a `deque`.
  - De un array a `vector` y viceversa.
  - De un array a `list` y viceversa.
  - De un `vector/list/`array a un `ostream_iterator`.

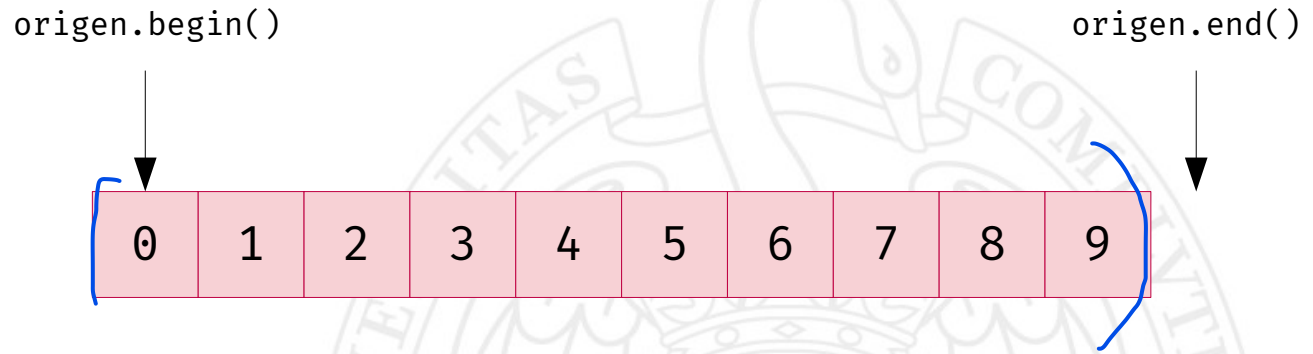
pasar de un vector/list/array a un ostream_operator,

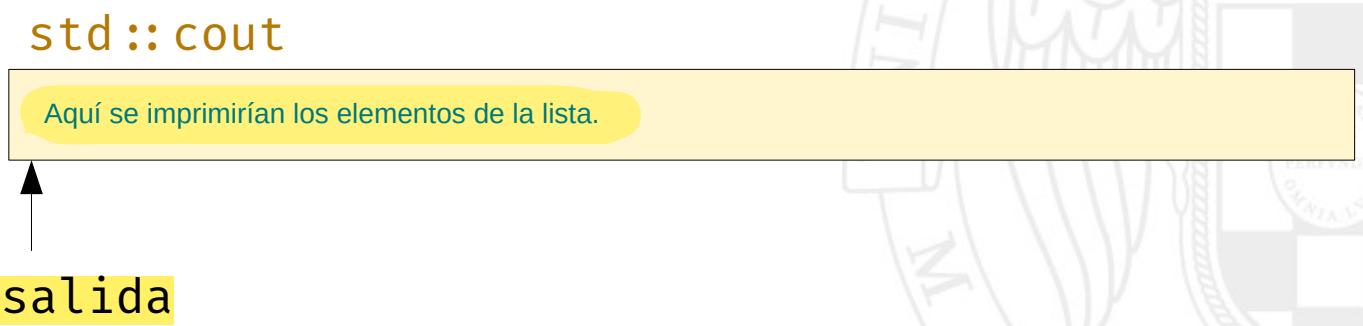
Vamos a ver un ejemplo de esto último

Para enviar el contenido de una estructura de datos a un fichero por ejemplo

Otro ejemplo

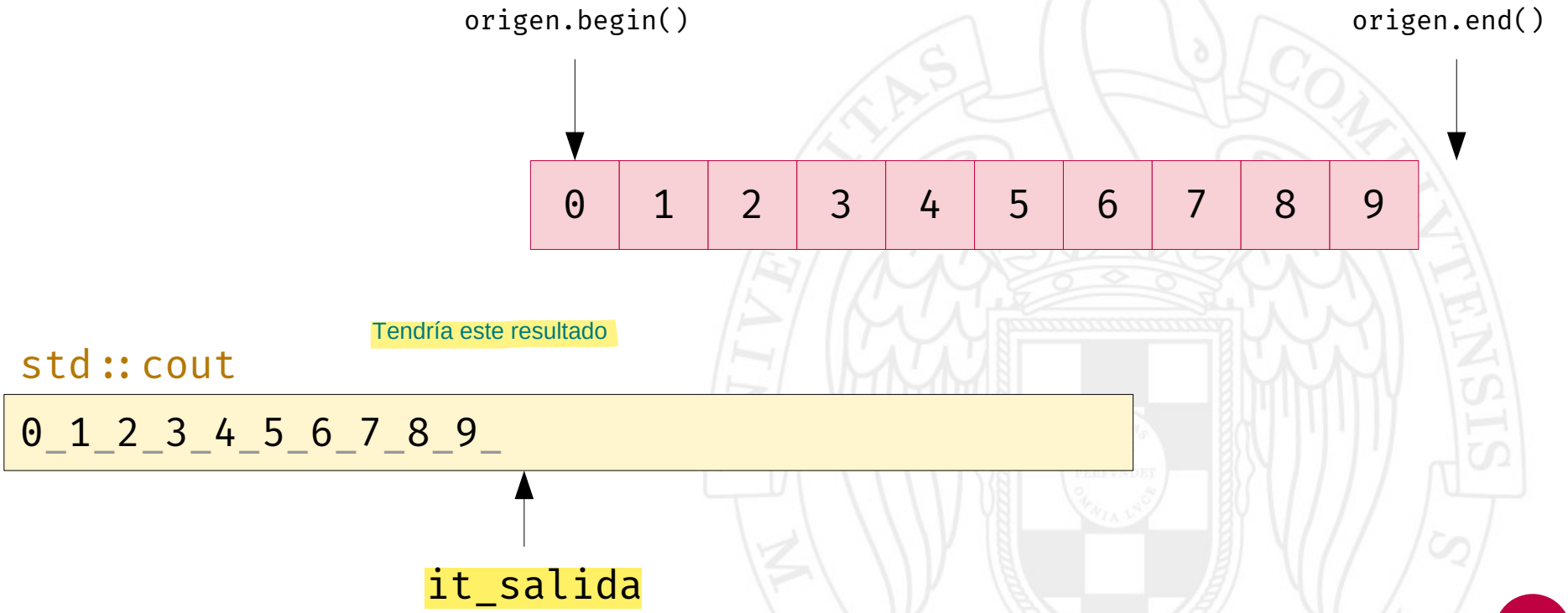
```
ostream_iterator<int> it_salida(cout, " ");
copy(origen.begin(), origen.end(), it_salida);
```





Otro ejemplo

```
ostream_iterator<int> it_salida(cout, " ");
copy(origen.begin(), origen.end(), it_salida);
```

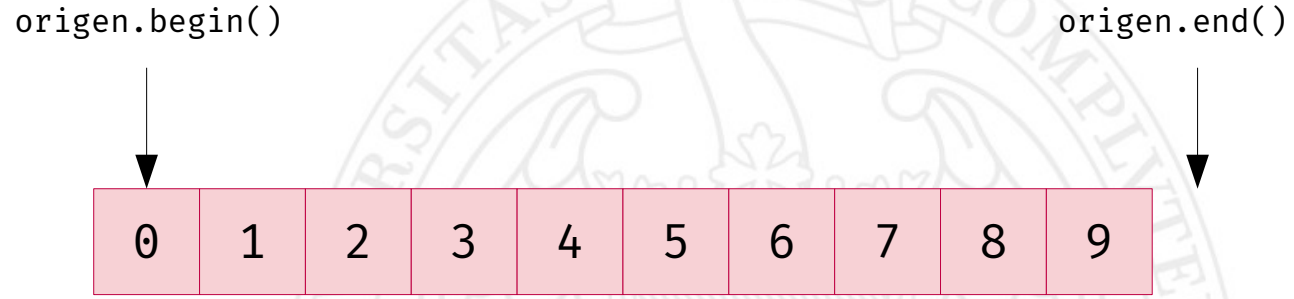


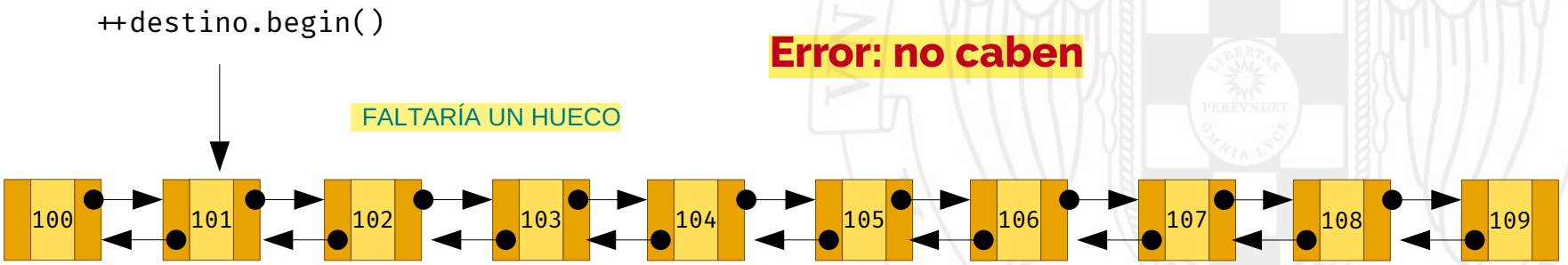
Cuidado!

- Para que la copia tenga éxito, el iterador de destino debe poderse incrementar tantas veces como elementos deseen copiarse.

```
copy(origen.begin(), origen.end(), ++destino.begin());
```

Es decir, DEBER HABER ESPACIO LIBRE.





Error: no caben

Los iteradores back_insert_iterator

- Son iteradores de salida que van asociados a un contenedor secuencial (list, vector, deque, etc).
- Cuando se escribe en el iterador, se añade un elemento al contenedor.
- Cuando se incrementa el iterador, no se hace nada.

Escribe los elementos desde el final de la lista.

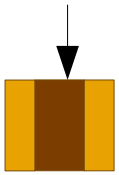
Ejemplo

```
int main() {
  vector<int> origen;
  list<int> lista_destino;

  // inicializar origen
  ...
  // suponemos que lista_destino queda vacía

  back_insert_iterator<list<int>> it_dest(lista_destino);
  copy(origen.begin(), origen.end(), it_dest);

  imprimir(cout, lista_destino);
}
```

Apunta al final de la lista que es el nodo fantasma

0	1	2	3	4	5	6	7	8	9

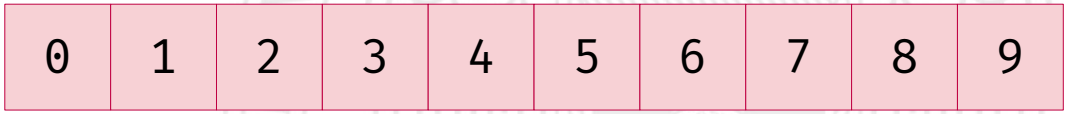
it_dest

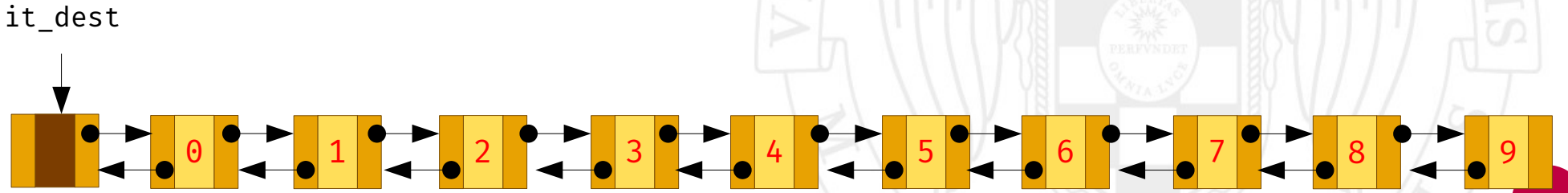


Ejemplo

```
int main() {
  vector<int> origen;
  list<int> lista_destino;

  // inicializar origen
  ...
  // suponemos que lista_destino queda vacía

  back_insert_iterator<list<int>> it_dest(lista_destino);
  copy(origen.begin(), origen.end(), it_dest);

  imprimir(cout, lista_destino);
}
```

Al hacer la copia se insertan todos en la lista enlazada.





La función `sort()`

La función `sort()`

- También definida en `<algorithm>`.

  `sort(begin, end)`

  Luego no lo vamos a poder utilizar para listas enlazadas

  donde:

  - `begin`, `end` son iteradores con acceso aleatorio.

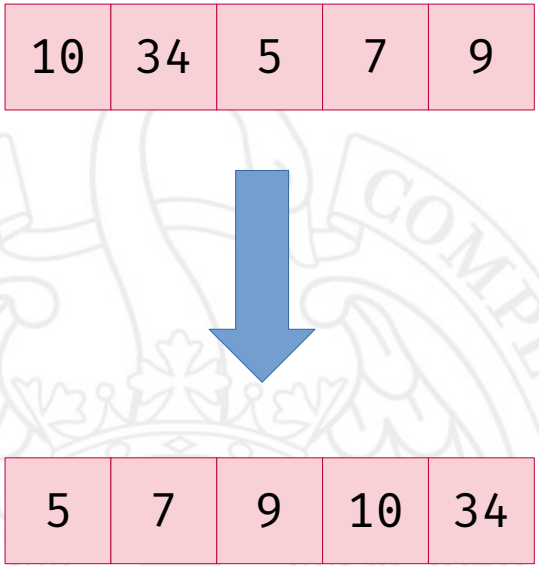
- Ordena ascendentemente los elementos contenidos entre los iteradores `begin` y `end` (excluyendo este último).
- Utiliza el operador `<` para comparar los elementos.

  Tendremos que asegurarnos de que este operador este definido entre los elementos que queramos ordenar.

Ejemplo

```
int main() {
   vector<int> v;
   v.push_back(10);
   v.push_back(34);
   v.push_back(5);
   v.push_back(7);
   v.push_back(9);

   sort(v.begin(), v.end());
}
```

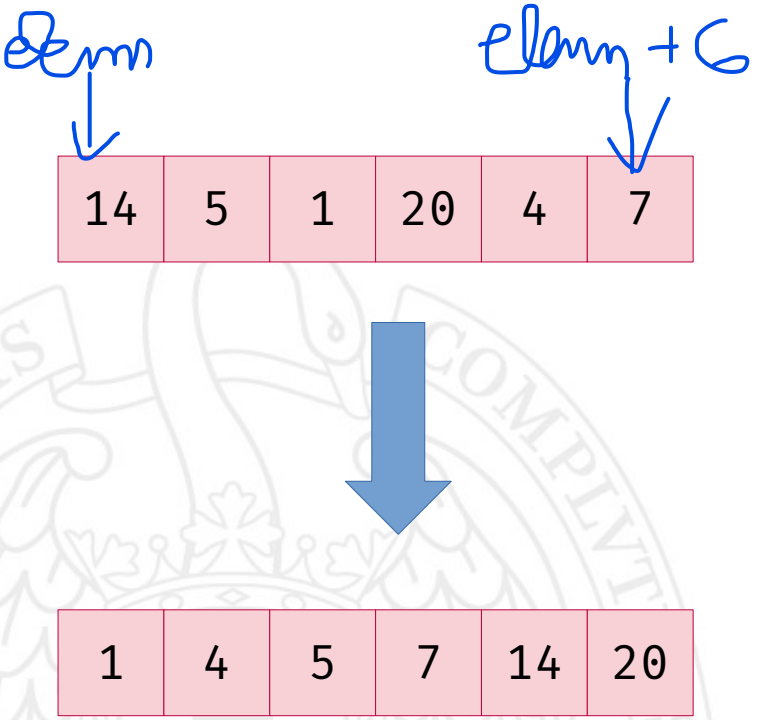
ordena el array



Otro ejemplo

```
int main() {
  int elems[] = {14, 5, 1, 20, 4, 7};
  sort(elems, elems + 6);
}
```

ordena de menor a mayor



Más funciones en <algorithm>

- `find(begin, end, value)` busca un elemento dentro del vector o lista enlazada.
- `fill(begin, end, value)` llenar todos los elementos de un determinado intervalo con un número
- `unique(begin, end)` unique para eliminar duplicados en una sentencia que esté ordenada.
- `binary_search(begin, end, value)`
- `max(begin, end)`