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

La función copy( )

La función copy( )

- Definida en <algorithm>

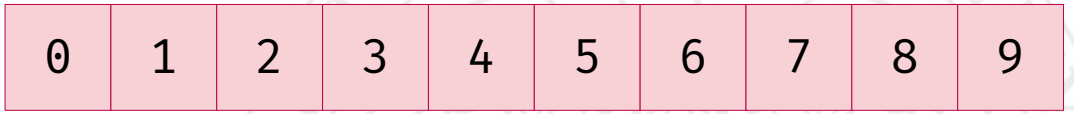
  copy(*source_begin*, *source_end*, *destination_begin*)

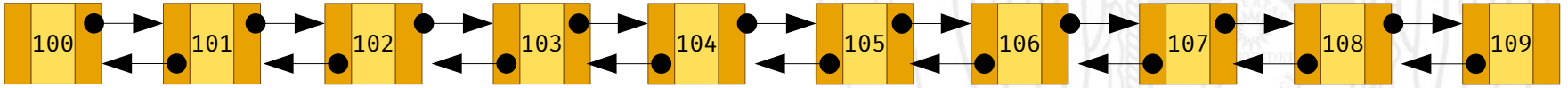
  donde:

  - 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.

Ejemplo

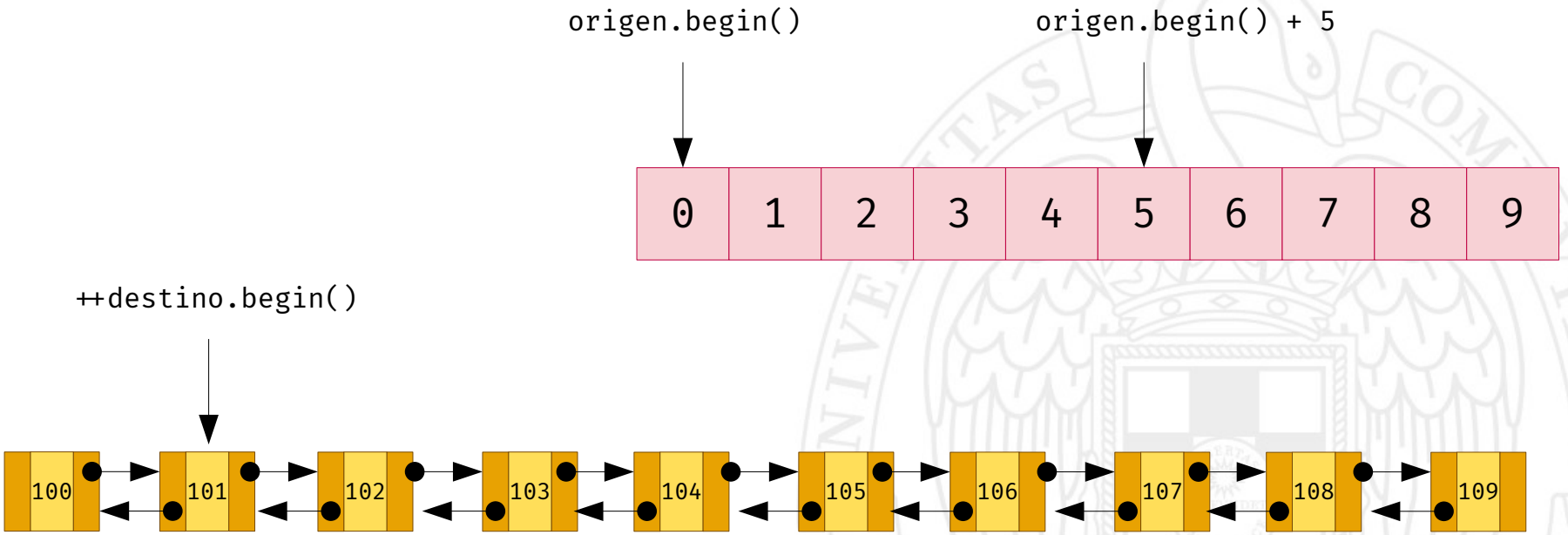
```
int main() {
  vector<int> origen;
  list<int> destino;

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





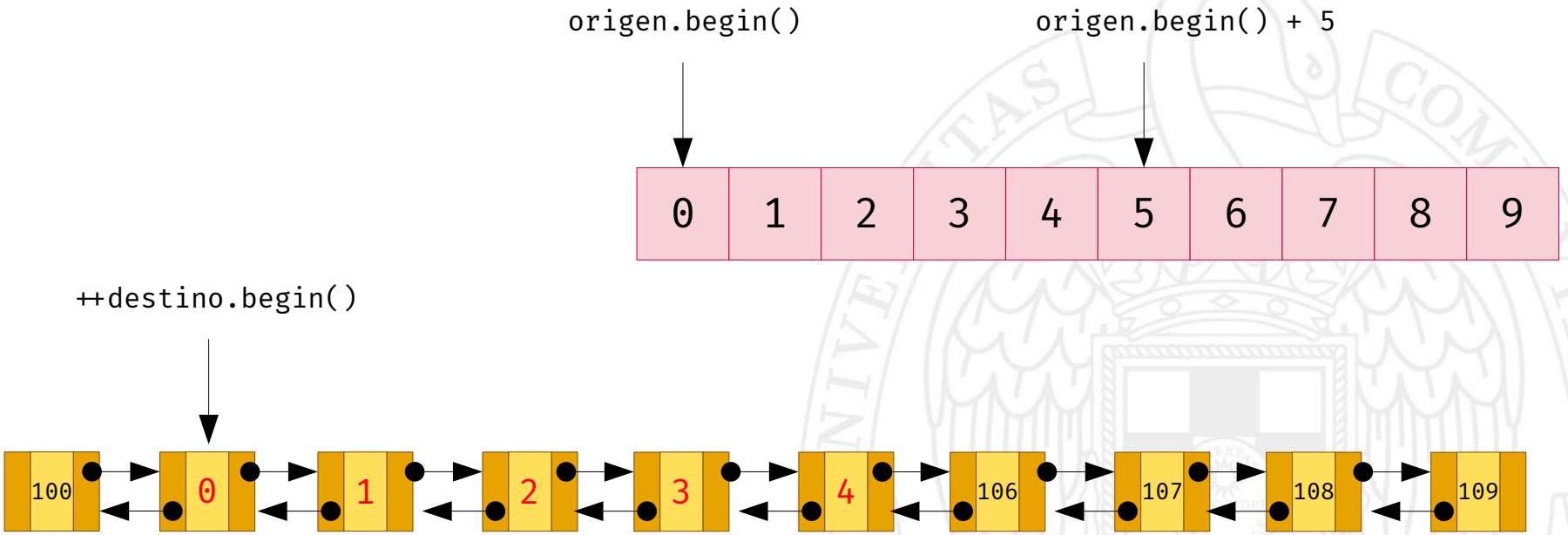
Ejemplo

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



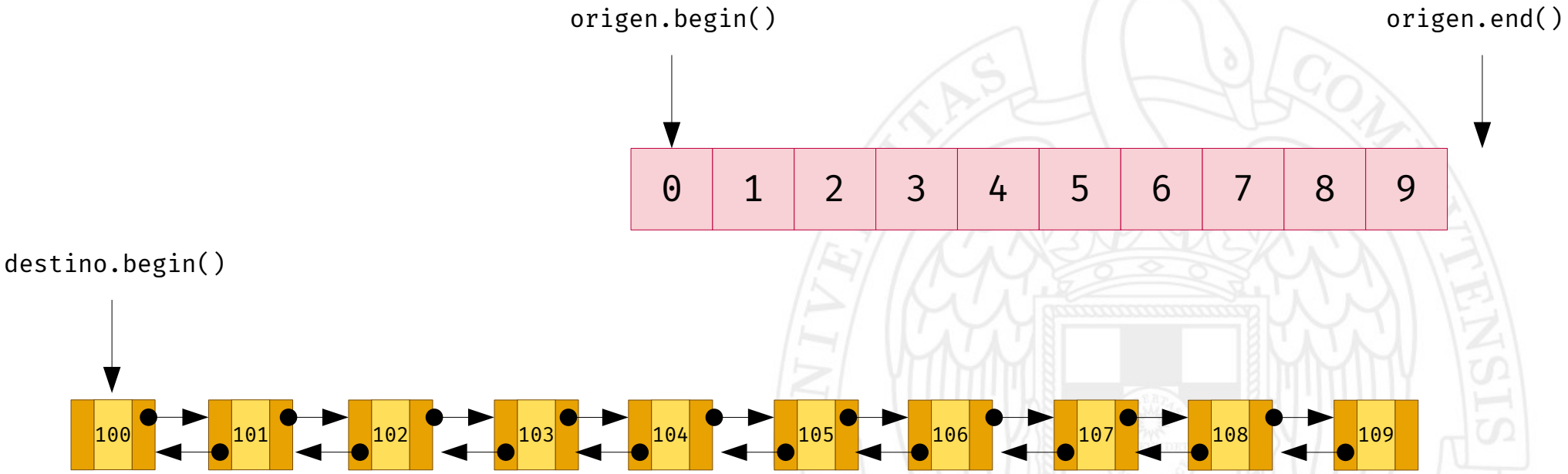
Ejemplo

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



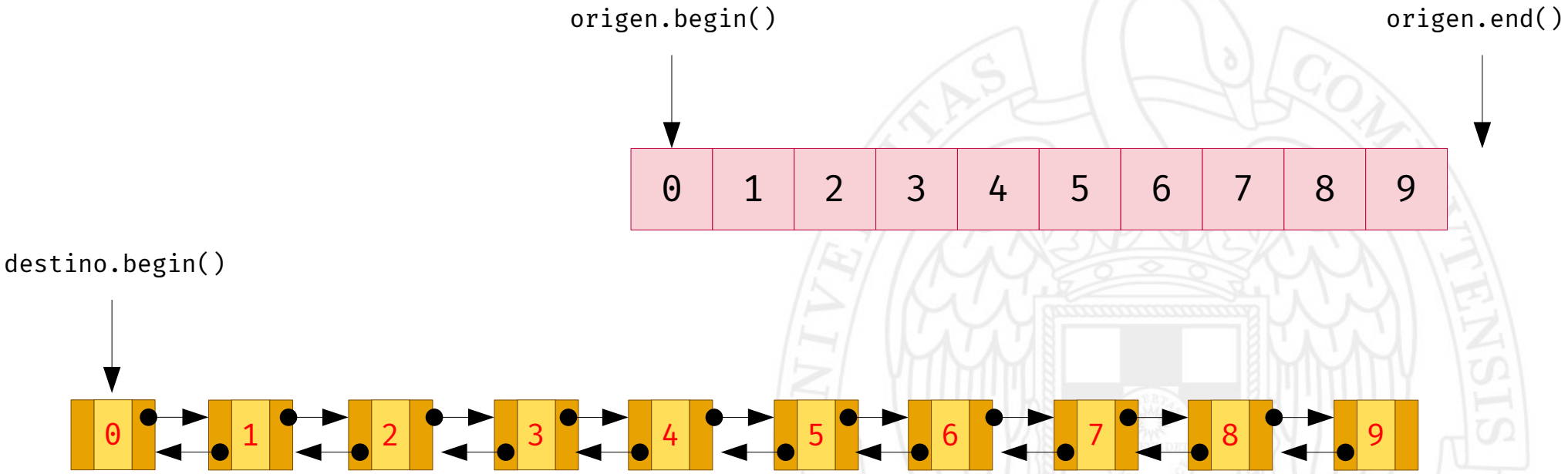
Ejemplo

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



Ejemplo

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

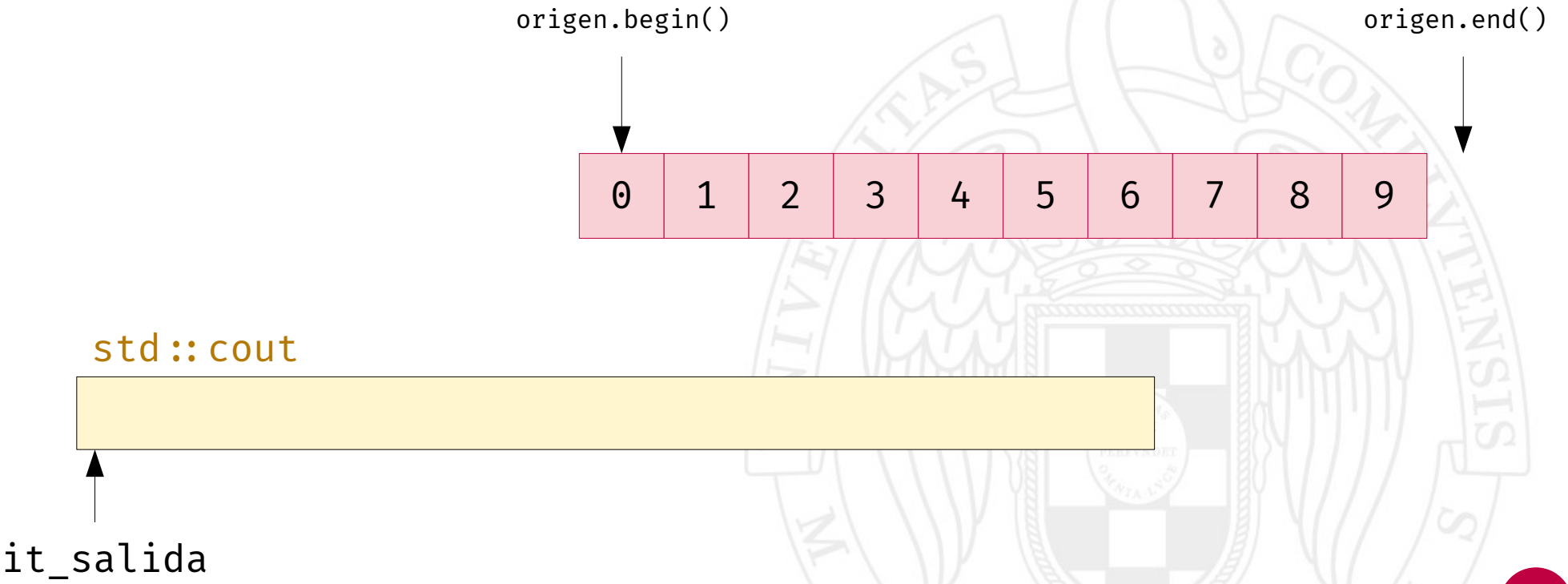


Utilidad de función copy( )

- 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`.

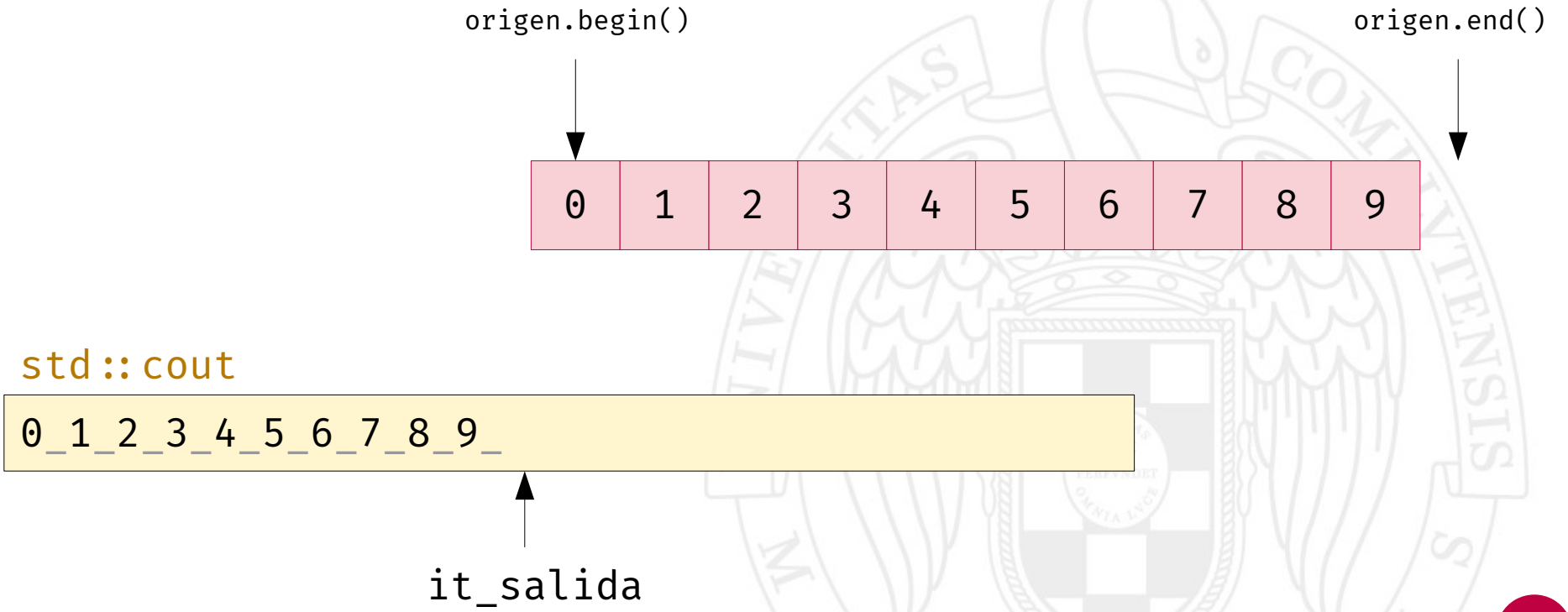
Otro ejemplo

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



Otro ejemplo

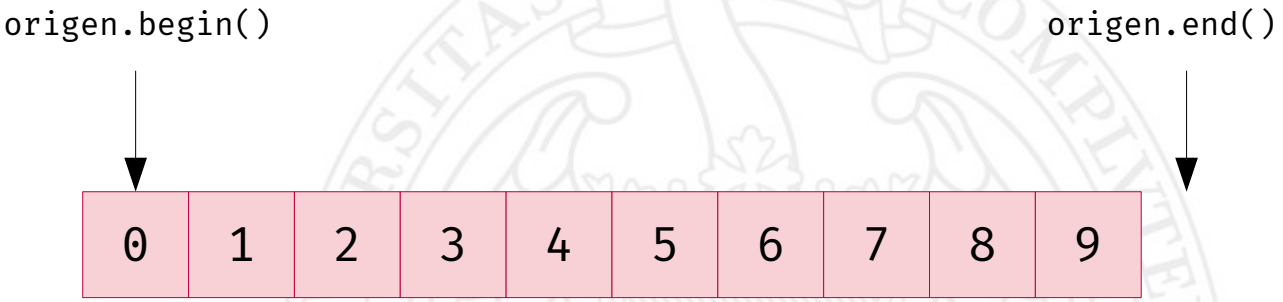
```
ostream_iterator<int> it_salida(cout, " ");
copy(origen.begin(), origen.end(), it_salida);
```

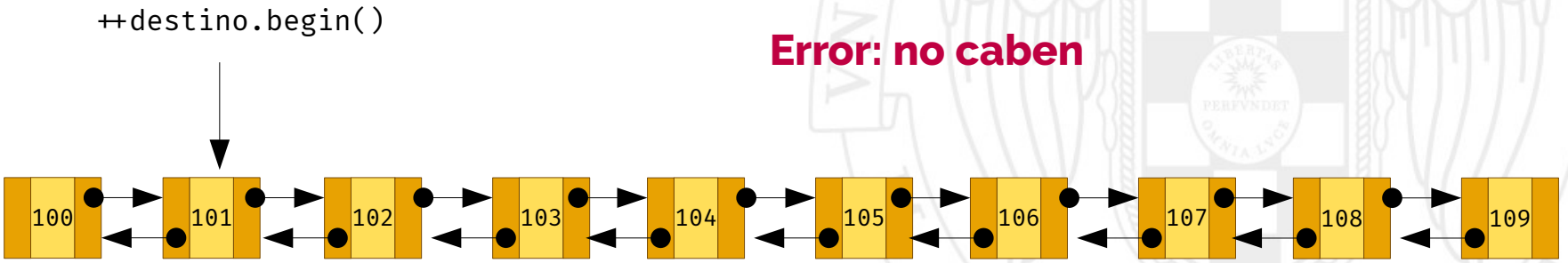


Cuidado!

- 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());
```



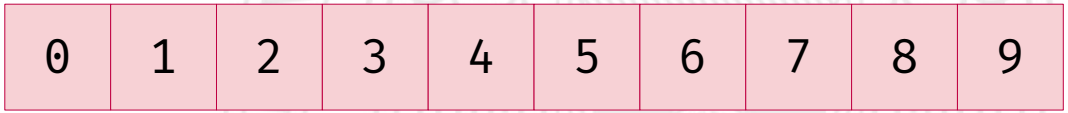


Los iteradores `back_insert_iterator`

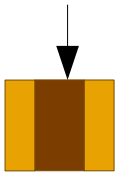
- Son iteradores de salida que van asociados a un contenedor secuencial (list, vector, deque, etc).
- Cuando se escribe en el iterador, se añade un elemento al contenedor.
- Cuando se incrementa el iterador, no se hace nada.

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);
}
```

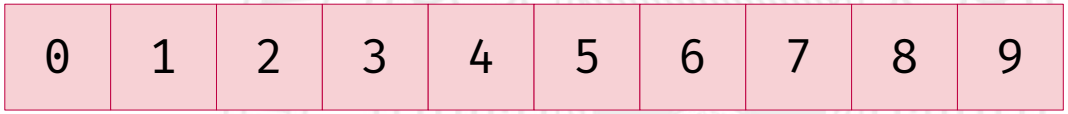


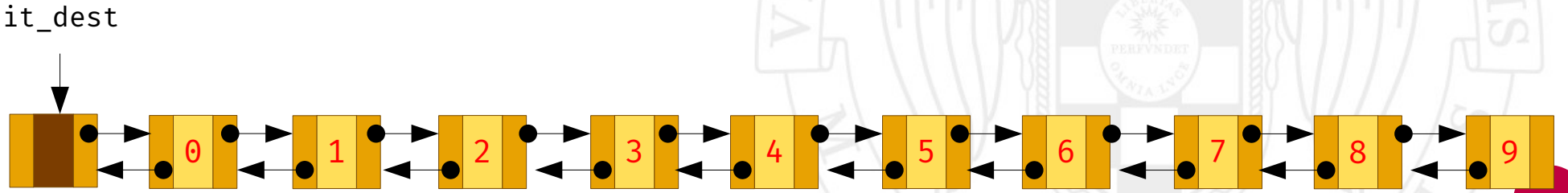
it_dest



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);
}
```





La función `sort()`

La función `sort()`

- También definida en `<algorithm>`.

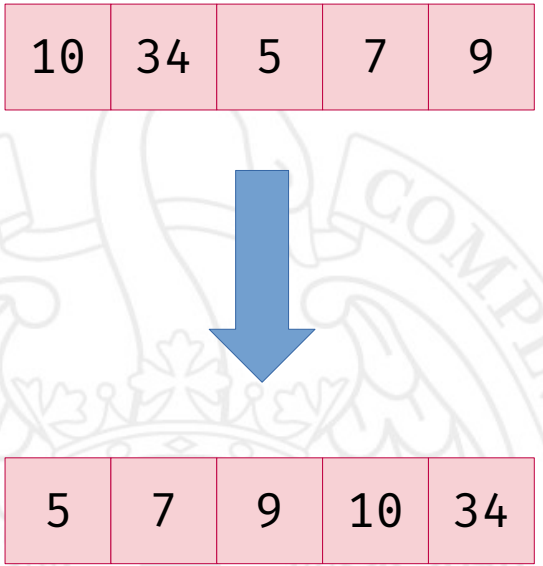
  ```
  sort(begin, end)
  ```

  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.

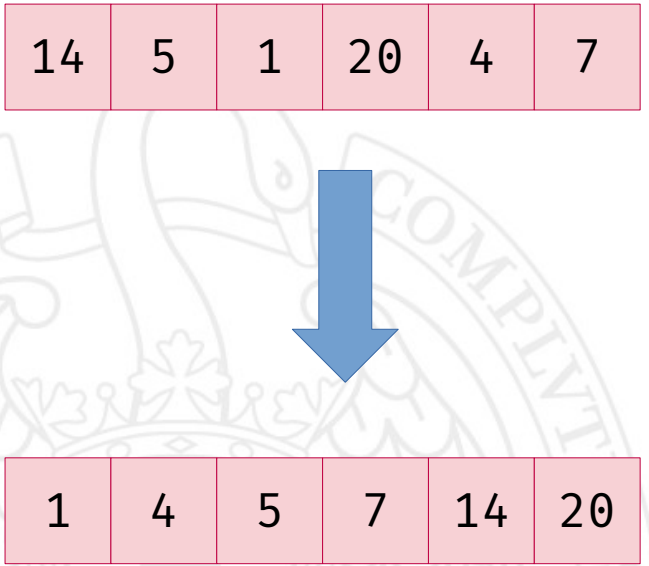
Ejemplo

```
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());
}
```



Otro ejemplo

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



Más funciones en <algorithm>

- `find(`*begin*`, `*end*`, `*value*`)`
- `fill(`*begin*`, `*end*`, `*value*`)`
- `unique(`*begin*`, `*end*`)`
- `binary_search(`*begin*`, `*end*`, `*value*`)`
- `max(`*begin*`, `*end*`)`