Listas

- Estructuras lineales
- Se puede añadir (`insert`), consultar o borrar (`erase`) en cualquier posición usando iteradores
- Su contenido se puede parametrizar: listas de números, de booleanos, etc
- Especificación: Ver fichero `list_espec.hh`

Iteradores

- `list<int>::iterator it;`
- Permiten acceder a los elementos de una lista, mediante el operador de desreferenciación: `*it`;
  - consulta: `x=*it`;
  - modificación: `*it=x`
- Si se definen como constantes, no permiten modificar las listas

```
list<int>::const_iterator it;
```

Iteradores

Se pueden mover por la lista, comparar y asignar

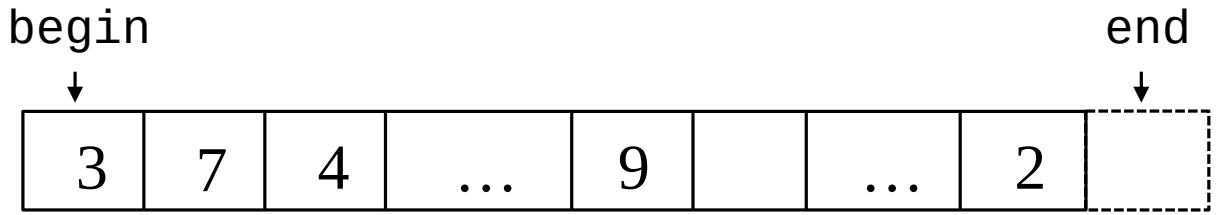
- Mover: hacia adelante: `++it`, salvo en el `end`; hacia atras `--it`, salvo en el `begin`
- Comparar: `if (it1!=it2)`

  `while (it!=l.end())`

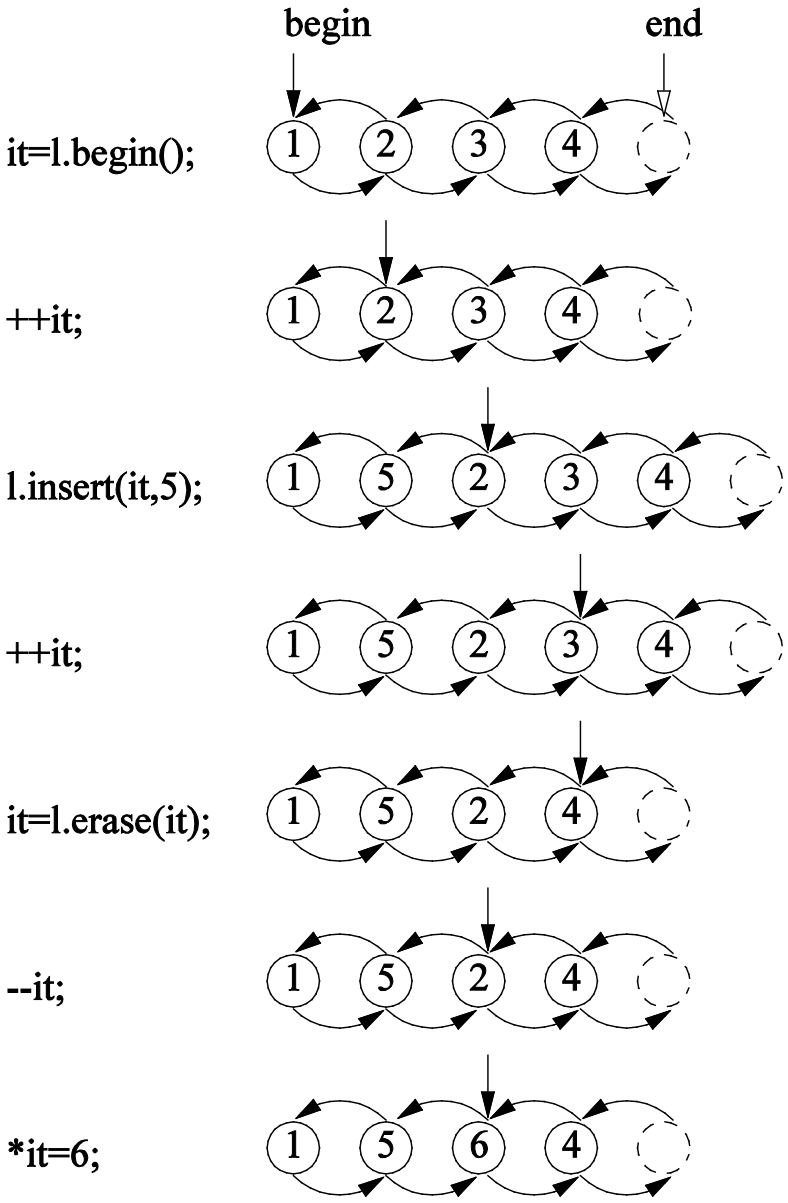
  pero los de listas no permiten `it1<it2`
- Asignar: `it1=it2` (salvo que `it2` sea `const` y `it1` no)

Iteradores

• Toda lista tiene dos iteradores especiales: `begin` y `end`



- Si la lista `l` es vacía, `l.begin() = l.end()`
- Si la lista `l` es `const`, `l.begin()` y `l.end()`son `const`
- Variantes:
  - inversos: `rbegin, rend`
  - `const: cbegin, cend, crbegin, crend`


begin
end
it=l.begin();
1
2
3
4
++it;
1
2
3
4
l.insert(it,5);
1
5
2
3
4
++it;
1
5
2
3
4
it=l.erase(it);
1
5
2
4
--it;
1
5
2
4
*it=6;
1
5
6
4

Ejemplos de uso de listas

- Suma de los elementos de una lista
- Búsqueda de un elemento en una lista
- Sumar un valor k a todos los elementos de una lista

`(ver fichero ejemplos_lista.cc)`

```
int suma_llista_int(const list<int>& l)
/* Pre: cert */
/* Post: El resultat és la suma dels elements de l */
{
  list<int>::const_iterator it; // obligatorio!
  int s=0;
  for (it=l.begin(); it != l.end(); ++it){
    s+=*it;
  }
  return s;
}
```

Ejemplos de implementación de listas

- Con punteros (lo veremos más adelante)
- Por herencia de estructuras más complejas (`STL`)

Descartamos de entrada cualquier tipo de versión con vectores porque las ops. `insert` y `erase` quedan muy ineficientes

Listas vs. vectores

- Los vectores ofrecen acceso directo a todas las posiciones, pero insertar y borrar es caro (implican desplazamientos)
- Las listas permiten insertar y borrar de forma eficiente, pero el acceso a posiciones sueltas es caro (los iterators solo pueden saltar posiciones de una en una)
- Idealmente las listas no tienen un tamaño máximo fijado al declararlas; en algunos lenguajes los vectores se pueden redimensionar pero puede ser caro
- Los vectores de c++ también tienen iterators, que permiten hacer más cosas que los de listas, p. ej. `it+=5` (acceso a posiciones sueltas)

Ejercicios

- Insertar un nuevo elemento en una lista ordenada
- Comprobar si una lista es capicúa, con y sin `size()`
- Duplicar una lista, con y sin `size()` (ver `ejemplos_lista.cc`)

Apéndice

- Operación push_back: `l.push_back(x)`
  - Equivale a un `l.insert (l.end(),x)`
  - También existe para vectores, pero por dentro son diferentes
- Operación push_front: `l.push_front(x)`
  - Equivale a un `l.insert (l.begin(),x)`

Tambien existen los correspondientes `pop_back` y `pop_front`