# Multilistas

Estructuras de Datos

Andrea Rueda

Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas

# Iteradores STL

- Objeto que puede recorrer un rango de elementos predefinido a través de ciertos operadores.

# Iteradores STL

- Objeto que puede recorrer un rango de elementos predefinido a través de operadores:
  - *Input iterator* : extrae datos, movimiento hacia adelante.
  - *Output iterator* : almacena datos, movimiento hacia adelante.
  - *Forward iterator* : almacena y extrae datos, movimiento hacia adelante.
  - *Bidirectional iterator* : almacena y extrae datos, movimiento hacia adelante y hacia atrás.
  - *Random-access iterator* : almacena y extrae datos, acceso a elementos en cualquier orden.

# Iteradores STL

- Operaciones:

  - Operador *: (dereferenciación) funciona como un apuntador, retorna el contenido del iterador.

  - Operadores ++ y --: mueve el iterador a la siguiente posición o a la anterior posición.

  - Operadores == y !=: comparación de iteradores, si apuntan o no al mismo elemento.

  - Operador =: asigna una nueva posición al iterador (usualmente principio o fin del contenedor).

# Iteradores STL

| | **Operaciones** | | | | | |
|---|---|---|---|---|---|---|
| | `*` | `* =` | `++` | `--` | `== , !=` | `=` |
| *Input iterator* | ✔ | ✘ | ✔ | ✘ | ✔ | ✔ |
| *Output iterator* | ✘ | ✔ | ✔ | ✘ | ✘ | ✔ |
| *Forward iterator* | ✔ | ✔ | ✔ | ✘ | ✔ | ✔ |
| *Bidirectional iterator* | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| *Random-access iterator* * | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |

* además soporta operaciones de acceso aleatorio: `+n`, `-n`, <, >, <=, >=, `+=`, `-=`, `[ ]`

# Contenedores e Iteradores STL

- Cada contenedor incluye funciones básicas para usar con el operador =
  - **`begin():`** iterador que representa el inicio de los elementos.
  - **`end():`** iterador que representa el elemento después del final.
  - **`rbegin():`** representa el inicio en la secuencia inversa.
  - **`rend():`** representa el elemento después del final en la secuencia inversa.

# Contenedores e Iteradores STL

- Cada contenedor tiene varios tipos de iteradores:
  - ***container*`::iterator`**
    iterador de lectura/escritura (entrada/salida).
  - ***container*`::reverse_iterator`**
    iterador en secuencia inversa de lectura/escritura (entrada/salida).

# Contenedores e Iteradores STL

- `vector` con iteradores:

```
std::vector<int> miVec;

for (int i = 0; i < 10; i++)
   miVec.push_back(i+1);

std::vector<int>::iterator miIt;

for (miIt = miVec.begin();
     miIt != miVec.end(); miIt++)
   std::cout << *miIt << std::endl;
```

# Contenedores e Iteradores STL

- `vector` - impresión en reversa:

```
std::vector<int>::reverse_iterator rit;


for (rit=miVec.rbegin();
     rit!=miVec.rend(); rit++) {
  std::cout << *rit << std::endl;
}
```

# Contenedores e Iteradores STL

- `deque` con iteradores:

```
std::deque<int> miDeq;

for (int i = 0; i < 5; i++)
   miDeq.push_front(i+1);
   miDeq.push_back(i+1);

std::deque<int>::iterator miIt;

for (miIt = miDeq.begin();
     miIt != miDeq.end(); miIt++)
   std::cout << *miIt << std::endl;
```

# Contenedores e Iteradores STL

- `deque` - impresión en reversa:

```
std::deque<int>::reverse_iterator rit;


for (rit=miDeq.rbegin();
     rit!=miDeq.rend(); rit++) {
  std::cout << *rit << std::endl;
}
```

# Contenedores e Iteradores STL

- `list` con iteradores:

```
std::list<int> miList;

for (int i = 0; i < 5; i++)
   miList.push_front(i+1);
   miList.push_back(i+1);

std::list<int>::iterator miIt;

for (miIt = miList.begin();
     miIt != miList.end(); miIt++)
   std::cout << *miIt << std::endl;
```

# Contenedores e Iteradores STL

- `list` - impresión en reversa:

```
std::list<int>::reverse_iterator rit;


for (rit=miList.rbegin();
     rit!=miList.rend(); rit++) {
  std::cout << *rit << std::endl;
}
```

# Iteradores inválidos

- Después de algunas operaciones en los contenedores, los iteradores pueden resultar inválidos:
  * Cambiar el tamaño o la capacidad.
  * Algunas inserciones y/o eliminaciones:
  - Iteradores singulares: sin asociar a un contenedor.
  - Iteradores después del final.
  - Iteradores fuera de rango.
  - Iterador colgante: apunta a un elemento no existente, en otra ubicación o no accesible.

# Contenedores STL

- Contenedores como interfaz (*container adaptors*):

  - **`queue`**: cola, primero que entra, primero que sale.
  (`std::queue` `#include <queue>`)

  - **`stack`**: pila, último que entra, primero que sale.
  (`std::stack` `#include <stack>`)

  - **`priority_queue`**: cola de prioridad.
  (`std::priority_queue` `#include <queue>`)

# Pila en STL

Stack (**`std::stack`**)

- No se puede iterar (¿Por qué?)

- push (push_back)
- pop (pop_back)
- top (back)
- size
- empty

www.cplusplus.com/reference/stack/stack

# Pila en STL

*Declaración*

```
for (int i = 0; i < 12; i++)
```

*Inserción de datos*

```
while( !aux.empty( ) ) {
```

*Extracción de datos*

```
}
```

# Pila en STL

```
std::stack< int > aux;
```

*Declaración*

```
for (int i = 0; i < 12; i++)
  aux.push( i+1 );
```

*Inserción de datos*

```
while( !aux.empty( ) ) {
  int n = aux.top( );
  aux.pop( );
  std::cout << n << std::endl;
}
```

*Extracción de datos*

# Cola en STL

Queue (**`std::queue`**)

- No se puede iterar (¿Por qué?)

- push (push_back)
- pop (pop_front)
- front
- back
- size
- empty

www.cplusplus.com/reference/queue/queue

# Cola en STL

*Declaración*

```
for (int i = 0; i < 12; i++)
```

*Inserción de datos*

```
while( !aux.empty( ) ) {
```

*Extracción de datos*

```
}
```

# Cola en STL

```
std::queue< int > aux;
```

*Declaración*

```
for (int i = 0; i < 12; i++)
  aux.push( i+1 );
```

*Inserción de datos*

```
while( !aux.empty( ) ) {
  int n = aux.front( );
  aux.pop( );
  std::cout << n << std::endl;
}
```

*Extracción de datos*

# Contenedores STL

- Contenedores asociativos:

  - **`set`**: conjunto matemático, operaciones de unión, intersección, diferencia; requiere función de comparación.
  (`std::set` `#include <set>`)

  - **`map`**: arreglo asociativo, provee mapeo de un dato (clave) a otro (valor); requiere función de comparación para la clave.
  (`std::map` `#include <map>`)

# Quiz: STL

# Quiz: STL - vector

```
#include <vector>
#include <iostream>

int main() {
  std::vector<int> arr;
  for (int i=0; i<6; i++)
    arr.push_back(10-i);
  for (int ind=0; ind<arr.size(); ind++)
    std::cout << arr[ind] << " ";
  std::cout << std::endl;
}
```

# Quiz: STL - deque

```
#include <iostream>
#include <deque>

int main() {
  std::deque<int> dcola;
  for (int i=0; i<3; i++) {
    dcola.push_back(i);
    dcola.push_front(10-i);
  }
  for (int ind=0; ind<dcola.size(); ind++)
    std::cout << dcola[ind] << " ";
  std::cout << std::endl;
}
```

# Quiz: STL - list

```
#include <iostream>
#include <list>

int main() {
  std::list<int> lista;

  for (int i=0; i<3; i++) {
    lista.push_back(i);
    lista.push_front(10-i);
  }

  std::list<int>::iterator it;

  for (it=lista.begin(); it!=lista.end(); it++)

    std::cout << *it << " ";

  std::cout << std::endl;
}
```

# Multilistas

# Multilistas

- ¿Representación de una matriz utilizando memoria dinámica?

# Multilistas

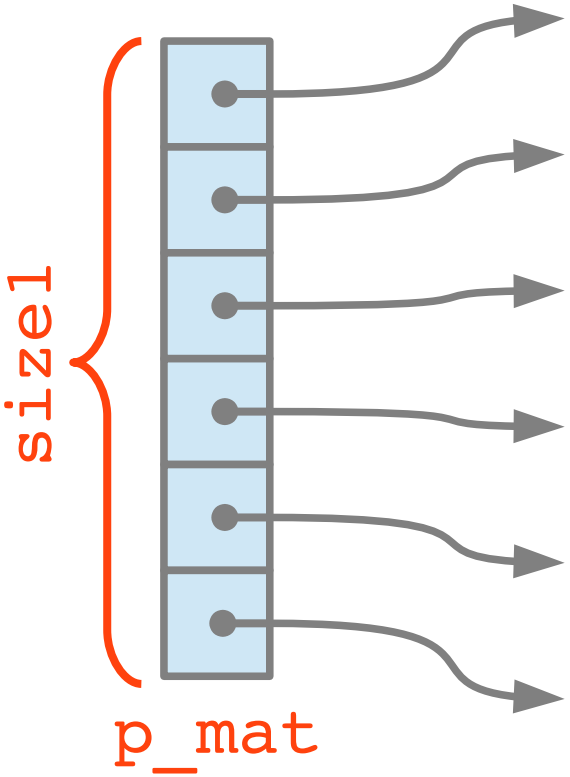
- ¿Representación de una matriz utilizando memoria dinámica?

```
int **p_mat;
p_mat = new int* [size1];

for (int i=0; i<size1; i++)
  *(p_mat+i) = new int [size2];
```

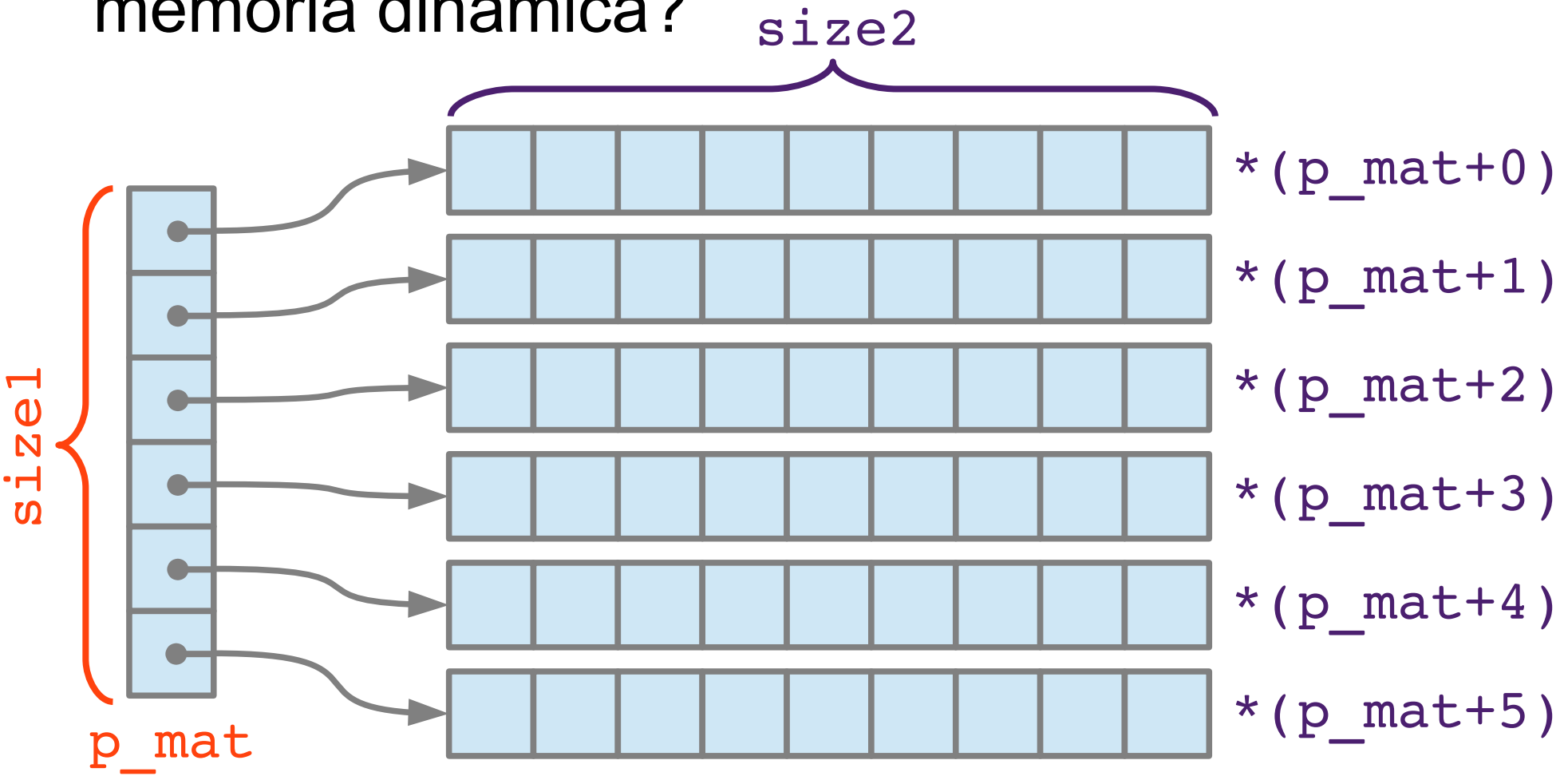
# Multilistas

- ¿Representación de una matriz utilizando memoria dinámica?



# Multilistas

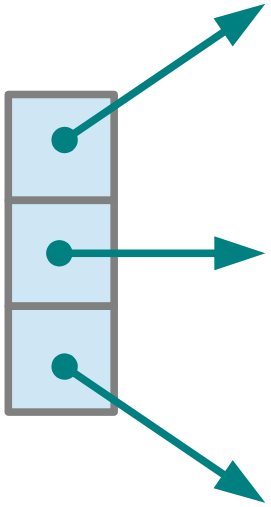
- ¿Representación de una matriz utilizando memoria dinámica?



# Multilistas

- Si una lista (secuencia) es análoga a un arreglo...

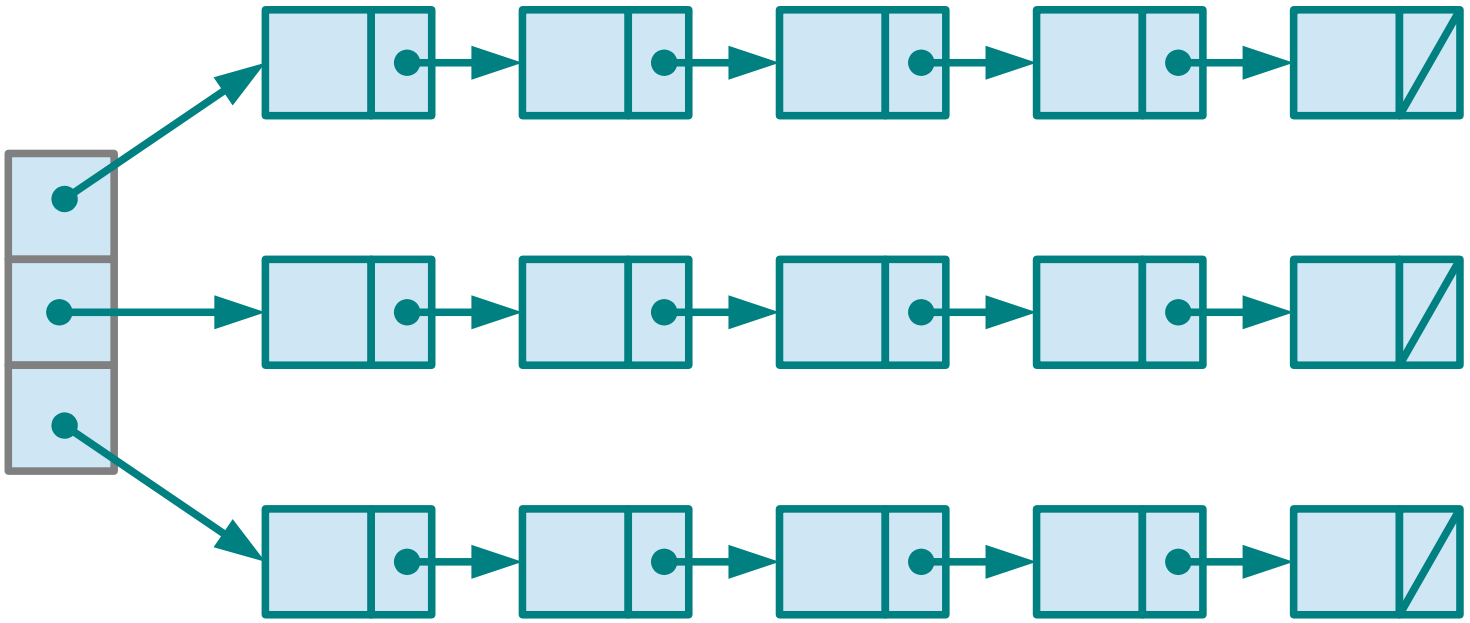
  ...¿podemos representar una matriz como un conjunto de listas (secuencias)?



# Multilistas

- Si una lista (secuencia) es análoga a un arreglo...

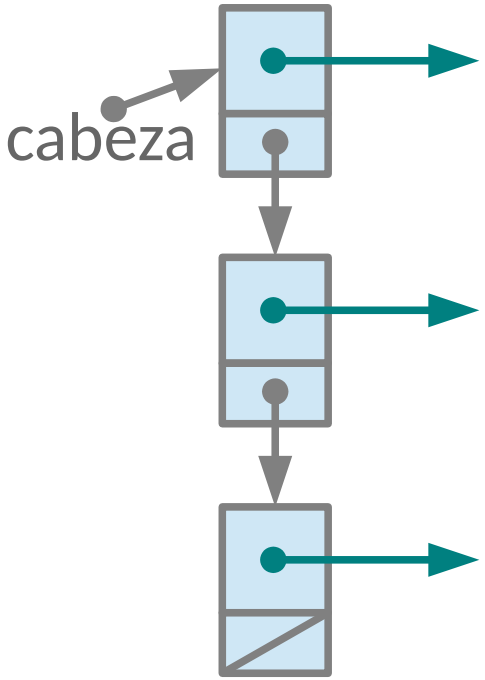
  ...¿podemos representar una matriz como un conjunto de listas (secuencias)?



# Multilistas

- Si una lista (secuencia) es análoga a un arreglo...

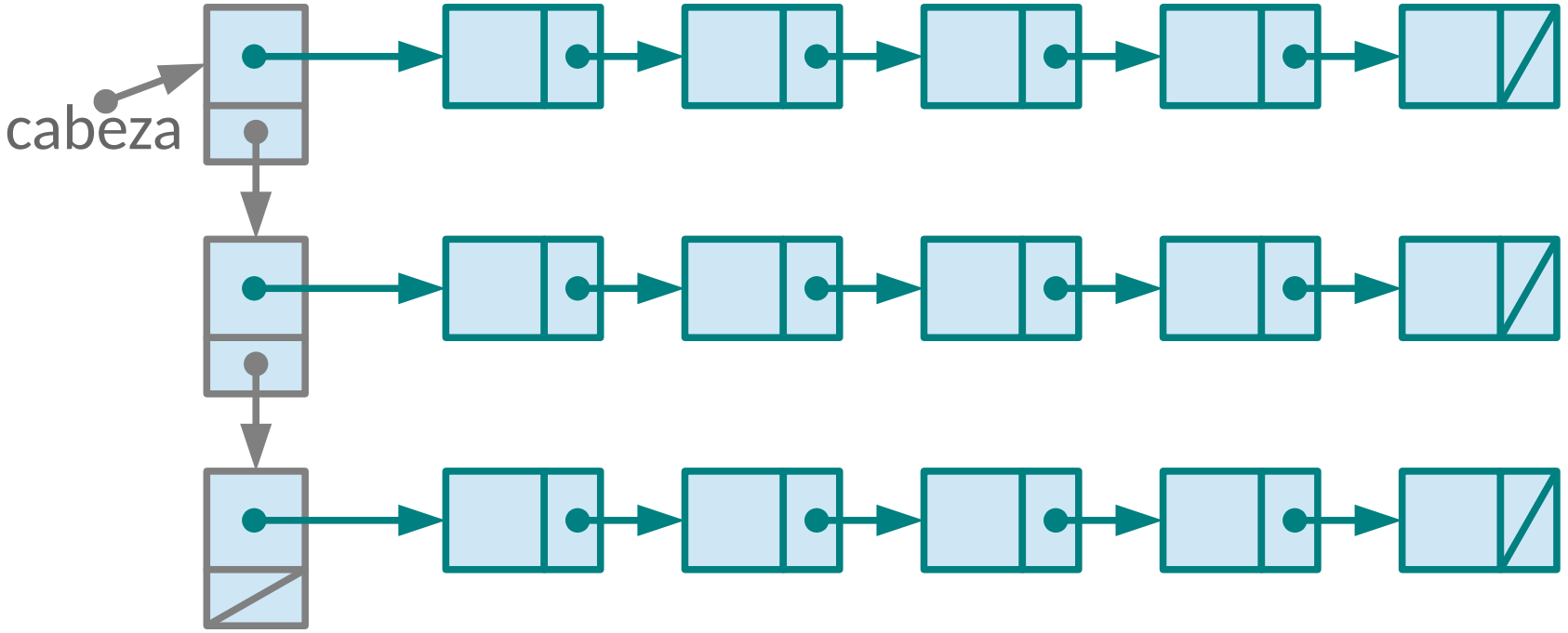
  ...¿podemos representar una matriz como un conjunto de listas (secuencias)?



# Multilistas

- Si una lista (secuencia) es análoga a un arreglo...

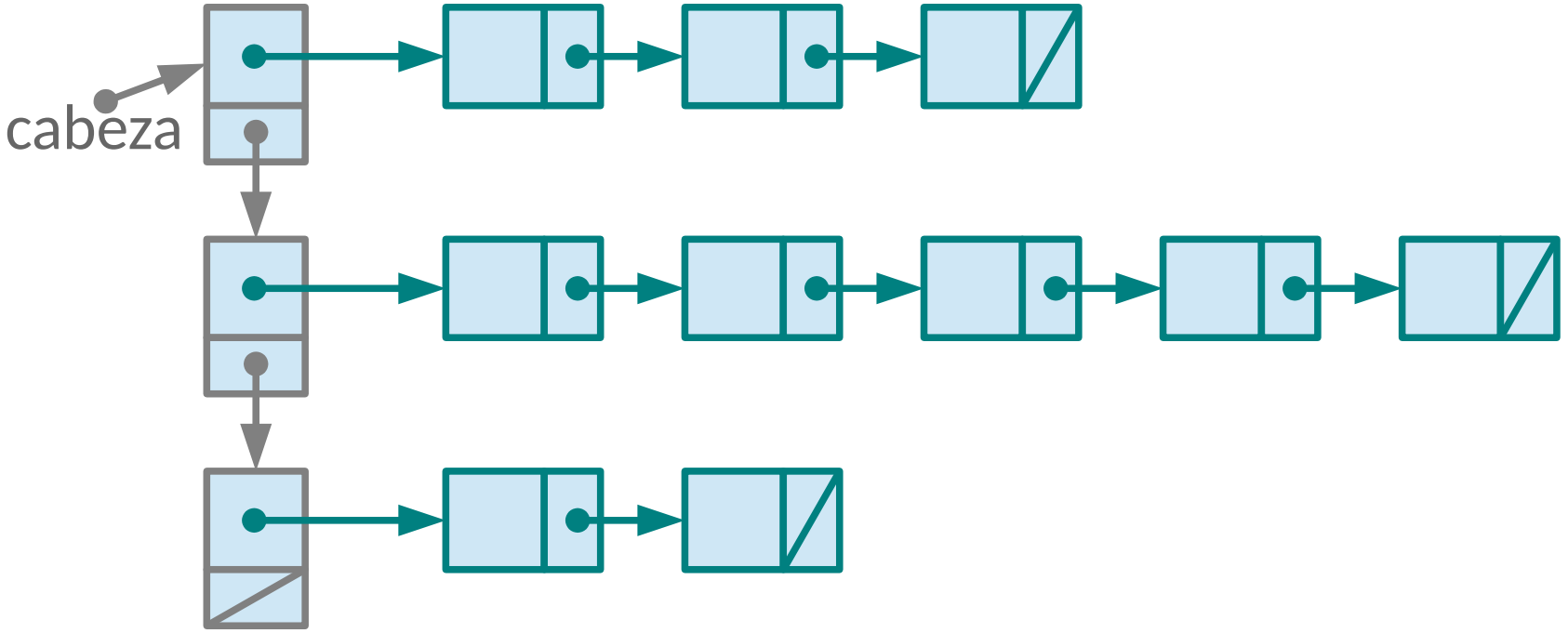
  ...¿podemos representar una matriz como un conjunto de listas (secuencias)?



# Multilistas

- Esto es una multilista: una lista de listas (secuencia de secuencias).

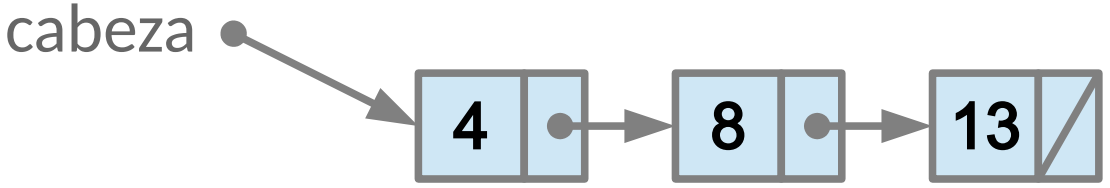
  Ventaja: al crearse nodo a nodo, no todas deben ser del mismo tamaño.



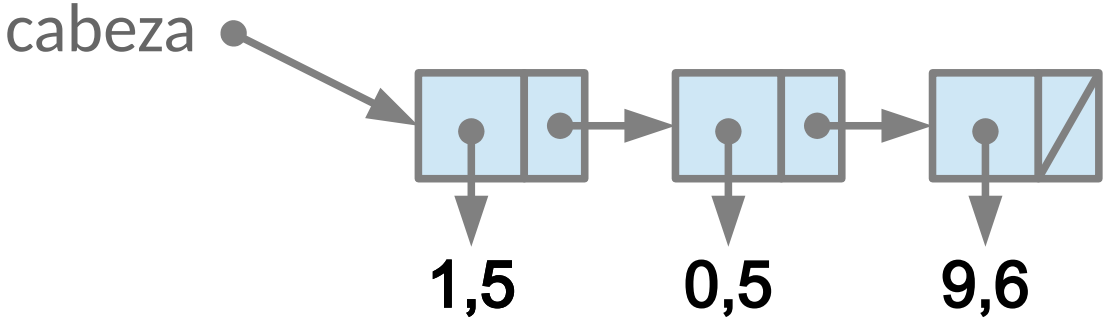
# Multilistas

- Implementación:

  Lista de números enteros:
  `std::list<int> enteros;`




  Lista de apuntadores a números reales:
  `std::list<float*> reales;`



# Multilistas

- Implementación:

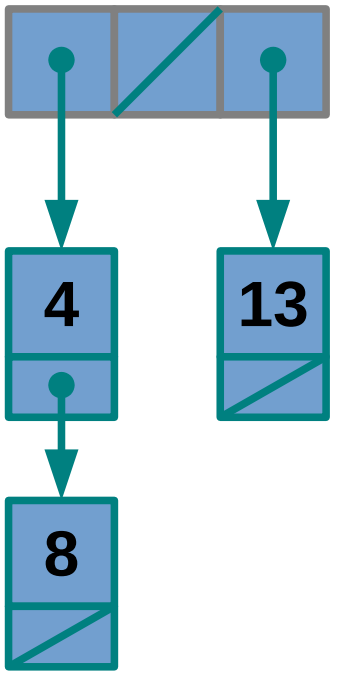
  ¿Vector de listas?

# Multilistas

- Implementación:

  ¿Vector de listas?

```
std::vector< std::list<int> > arr;
arr[0].push_back(4);
arr[2].push_back(13);
arr[0].push_back(8);
```



# Multilistas

- Implementación:

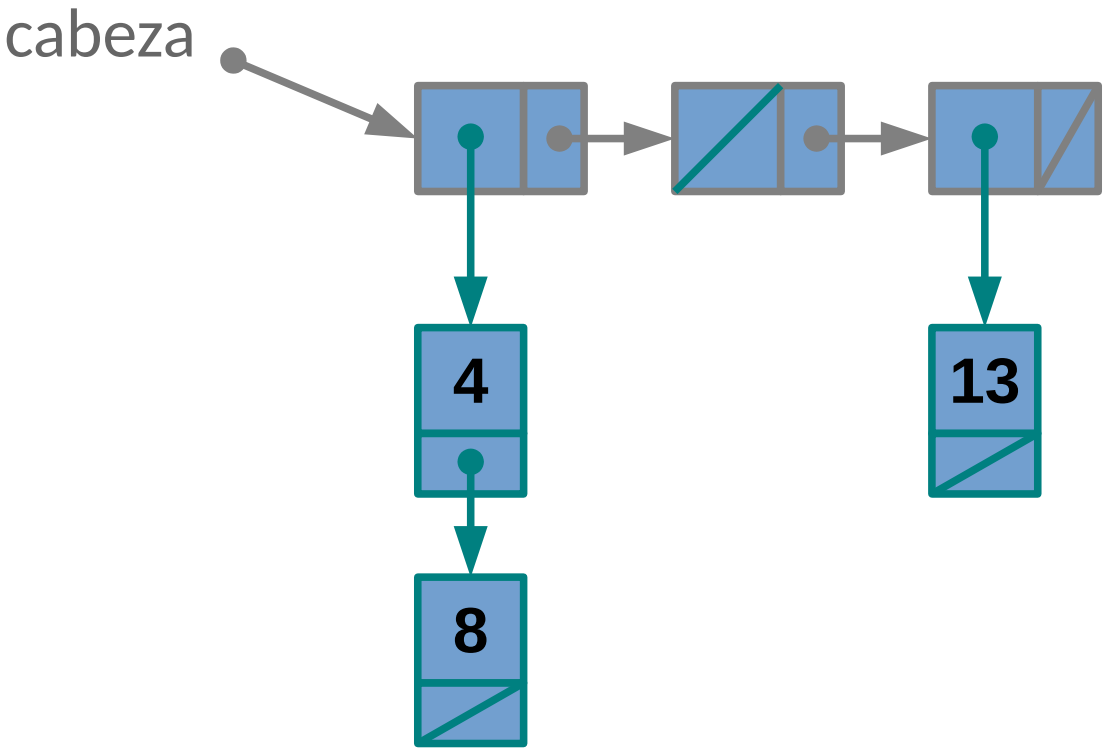
  ¿Lista de listas?

# Multilistas

- Implementación:

  ¿Lista de listas?

```
std::list< std::list<int> > multilista;
```



# Multilistas

- Implementación:

  ¿Lista de listas?

```
std::list< std::list<int> > multilista;
std::list< std::list<int> >::iterator it;
it = multilista.begin();
it->push_back(4);
it->push_back(8);
it++;
it++;
it->push_back(13);
```

# Multilistas

```cpp
#include <iostream>
#include <list>

int main() {
  std::list< std::list<int> > miCont = {{0},{0},{0}};

  std::list< std::list<int> >::iterator it;

  it = miCont.begin();
  it->push_back(4);
  it->push_back(8);

  it++;

  it++;
  it->push_back(13);

  for (it = miCont.begin(); it != miCont.end(); it++) {
    std::list<int>::iterator it2;
    for (it2 = it->begin(); it2 != it->end(); it2++) {
      std::cout << *it2 << " ";
    }
    std::cout << std::endl;
  }
}
```

https://replit.com/@Andrea_del_Pila/contenedoresSTL

# Multilistas

- Ejercicio:

  Considere el TAD Vehiculo, implementado:

```
class Vehiculo {
  protected:
    unsigned short modelo;
    string placa;
    unsigned int tarifaDiaria;
  public:
    unsigned short obtenerModelo();
    string obtenerPlaca();
    unsigned int obtenerTarifa();
    ...
};
```

# Multilistas

- Ejercicio:

  … y considere una entidad que administra diferentes parqueaderos, cada uno visto como una lista (secuencia) de vehículos.

# Multilistas

- Ejercicio:

  1. ¿Cómo se representaría la entidad?

# Multilistas

- Ejercicio:

  1. Representar la entidad como una lista de listas de vehículos.

```
std::list< std::list<Vehiculo> > entidad;
```

# Multilistas

- Ejercicio:

  2. Asumiendo que los parqueaderos se identifican con un número entero consecutivo, ¿Cómo se insertaría un vehículo en un parqueadero dado?

# Multilistas

- Ejercicio:

  2. Insertar un vehículo en un parqueadero dado.

```
void InsertarVeh( Vehiculo nVeh, int idParq,
    std::list< std::list<Vehiculo> > &entidad )
{
  std::list< std::list<Vehiculo> >::iterator itP;
  itP = entidad.begin();
  for (int i = 1; i < idParq; i++)
    itP++;
  itP->push_back(nVeh);
}
```

# Multilistas

- Ejercicio:

  3. ¿Cómo se calcularía la cantidad total de vehículos en la entidad?

# Multilistas

- Ejercicio:

  3. Cantidad total de vehículos en la entidad.

```
unsigned int totalVehs(
  std::list< std::list<Vehiculo> > &entidad )
{
  unsigned int total = 0;
  std::list< std::list<Vehiculo> >::iterator itP;
  itP = entidad.begin();
  for ( ; itP != entidad.end(); itP++)
    total += itP->size();
  return total;
}
```

# Multilistas

- Ejercicio:

  4. ¿Cómo se calcularía el valor total diario que recibe la entidad por los vehículos parqueados?

# Multilistas

- Ejercicio:

  4. Valor total diario de vehículos parqueados.

```
unsigned int totalDiario(
  std::list< std::list<Vehiculo> > &entidad )
{
  unsigned int total = 0;
  std::list< std::list<Vehiculo> >::iterator itP;
  std::list<Vehiculo>::iterator itV;
  itP = entidad.begin();
  for ( ; itP != entidad.end(); itP++) {
    itV = itP->begin();
    for ( ; itV != itP->end(); itV++)
      total += itV->obtenerTarifa();
  }
  return total;
}
```

https://replit.com/@Andrea_del_Pila/EDD-ejercicio-multilistas-vehiculo

# Referencias


- T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein. Introduction to Algorithms, 3rd edition. MIT Press, 2009.
- L. Joyanes Aguilar, I. Zahonero. Algoritmos y estructuras de datos: una perspectiva en C. McGraw-Hill, 2004.