Pilas y Colas

Estructuras de Datos

Andrea Rueda

Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas

Composición

Composición

- Describe una relación de contenencia (has-a).
- Permite generar tipos de datos complejos a partir de tipos de datos más simples.
- Ejemplos:

  - Un **evento** tiene una **fecha** (día-mes-año) y una **hora** (horas-minutos-segundos).

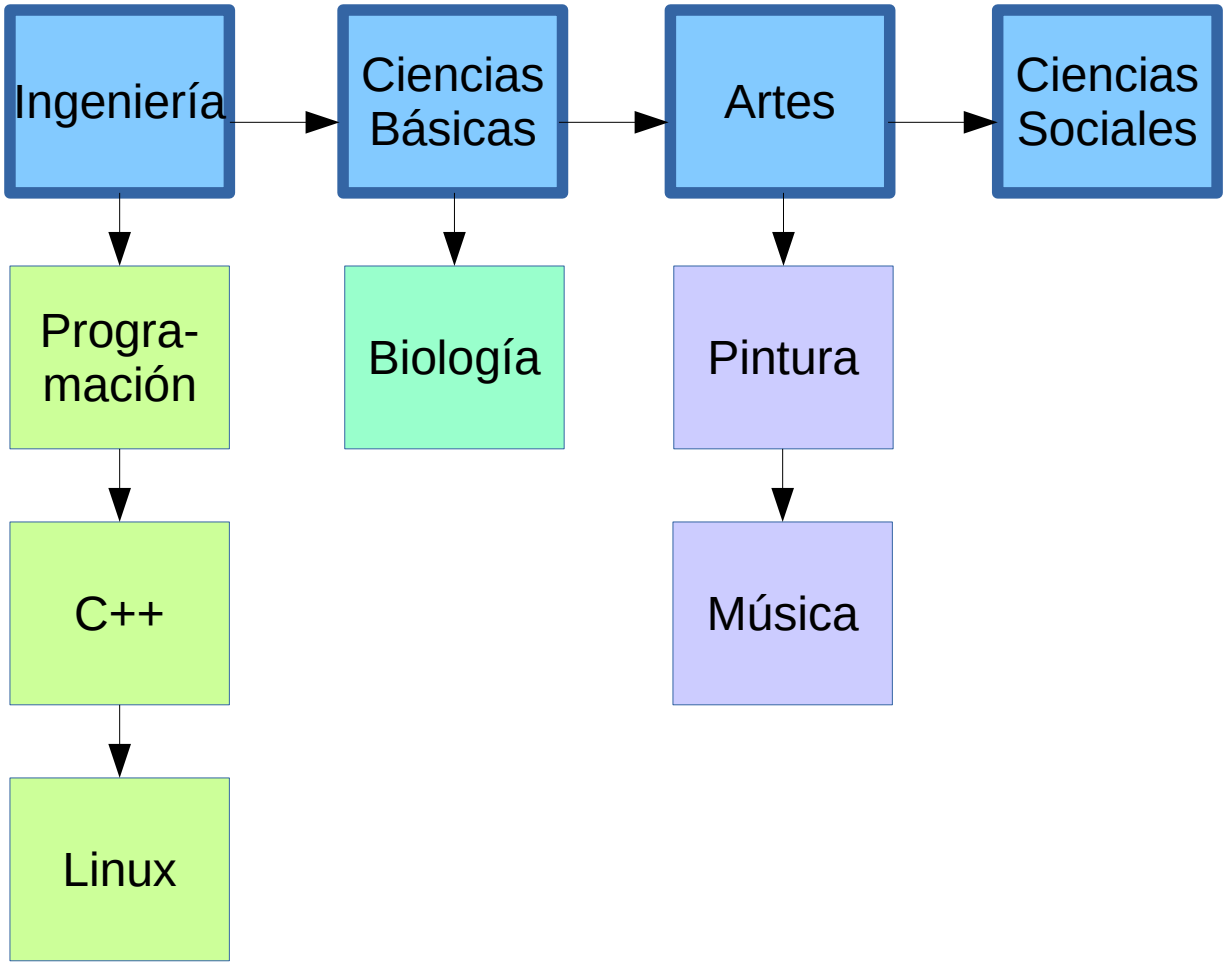
  - Una **universidad** tiene **facultades**, cada **facultad** tiene **carreras**.

¿Por qué composición?

- Cada tipo de dato simple puede enfocarse en una tarea particular.
- Cada tipo de dato simple puede reusarse en diferentes contextos.
- El tipo de dato compuesto puede dejar que cada tipo simple haga la mayor parte del trabajo, y sólo se encarga de coordinar el flujo de información.

Composición

- Ejemplo: organización de libros en una biblioteca.



Composición

- Ejemplo: organización de libros en una biblioteca.

Libro
- nombre
- cantidad ejemplares

Composición

- Ejemplo: organización de libros en una biblioteca.

Libro
- nombre
- cantidad ejemplares

Área de Conocimiento
- nombre
- *lista* de Libros

Composición

- Ejemplo: organización de libros en una biblioteca.

Libro
- nombre
- cantidad ejemplares

Área de Conocimiento
- nombre
- *lista* de Libros

Biblioteca
- nombre
- *lista* de Áreas de Conocimiento

Composición

- Organización de libros en una biblioteca.

  TAD **Libro**

  Conjunto mínimo de datos:

  Comportamiento (operaciones) del objeto:

Composición

- Organización de libros en una biblioteca.

  TAD **Libro**

  Conjunto mínimo de datos:

  nombre, cadena de caracteres, título del libro.
  num_ejempl, entero, cantidad de ejemplares.

  Comportamiento (operaciones) del objeto:

  ObtenerNombre(), retornar título del libro.
  ObtenerNumEjempl(), retornar cantidad de ejemplares.
  FijarNombre(nNom), cambiar título a nNom.
  AgregarEjemplar(), incrementar cantidad en 1.
  EliminarEjemplar(), decrementar cantidad en 1.

Composición

- Organización de libros en una biblioteca.

  TAD **AreaConocimiento**

  Conjunto mínimo de datos:

  Comportamiento (operaciones) del objeto:

Composición

- Organización de libros en una biblioteca.

  TAD **AreaConocimiento**

  Conjunto mínimo de datos:

  nombre, cadena de caracteres, nombre del área.
  l_libros, lista de **Libro**, conjunto de libros del área.

  Comportamiento (operaciones) del objeto:

  ObtenerNombre(), retornar nombre del área.
  FijarNombre(nNom), cambiar nombre a nNom.
  ContarEjempl(), contar el total de libros del área.
  AgregarLibro(nLibro), agregar nLibro a la lista.
  EliminarLibro(nLibro), eliminar nLibro de la lista.

Composición

- Organización de libros en una biblioteca.

  TAD **Biblioteca**

  Conjunto mínimo de datos:

  Comportamiento (operaciones) del objeto:

Composición

- Organización de libros en una biblioteca.

  TAD **Biblioteca**

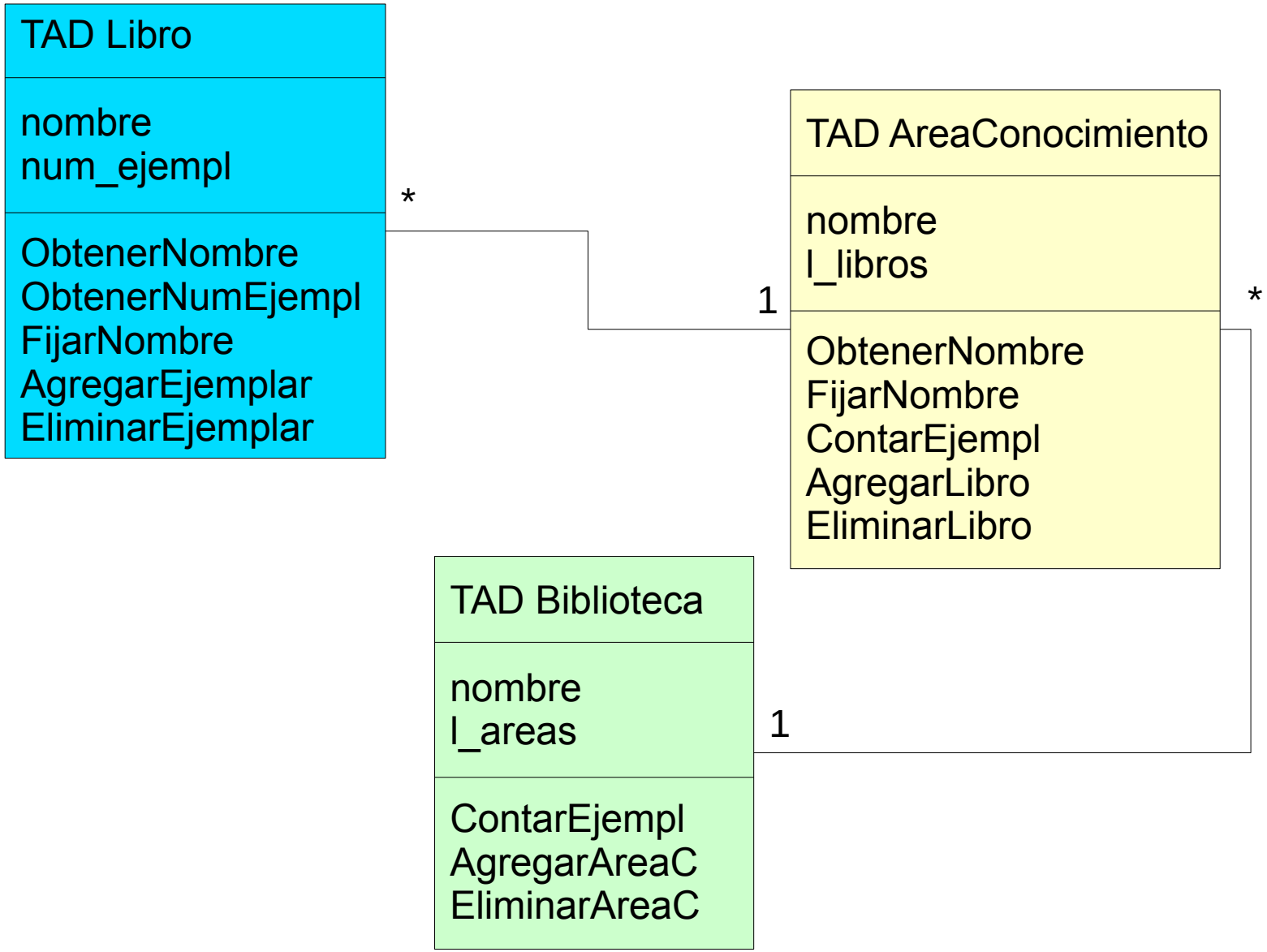
  Conjunto mínimo de datos:

  nombre, cadena de caracteres, nombre de la biblioteca.
  l_areas, lista de **AreaConocimiento**, conjunto de áreas.

  Comportamiento (operaciones) del objeto:

  ContarEjempl(), contar el total de libros de la biblioteca.
  AgregarAreaC(...), agrega un área de conocimiento.
  EliminarAreaC(...), eliminar un área de conocimiento.

Composición



Composición

- Organización de libros en una biblioteca.

```
class Libro {
  public:
   Libro();
   std::string ObtenerNombre();
   unsigned long ObtenerNumEjempl();
   void FijarNombre(std::string n_nombre);
   void AgregarEjemplar();
   void EliminarEjemplar();
  protected:
   std::string nombre;
   unsigned long num_ejempl;
};
```

https://repl.it/@Andrea_del_Pila/EDD-ejercicio-multilistas

Composición

- Organización de libros en una biblioteca.

```
class AreaConocimiento {
  public:
   AreaConocimiento();
   std::string ObtenerNombre();
   void FijarNombre(std::string n_nombre);
   void AgregarLibro(std::string n_libro);
   unsigned long ContarEjempl();
   bool EliminarLibro(std::string n_libro);
  protected:
   std::string nombre;
   std::list<Libro> l_libros;
};
```

https://repl.it/@Andrea_del_Pila/EDD-ejercicio-multilistas

Composición

- Organización de libros en una biblioteca.

```
class Biblioteca {
  public:
    Biblioteca();
    void AgregarAreaC( ... );
    long ContarEjempl() const;
    void EliminarAreaC( ... );
  protected:
    std::string nombre;
    std::list<AreaConocimiento> l_areas;
};
```

https://repl.it/@Andrea_del_Pila/EDD-ejercicio-multilistas

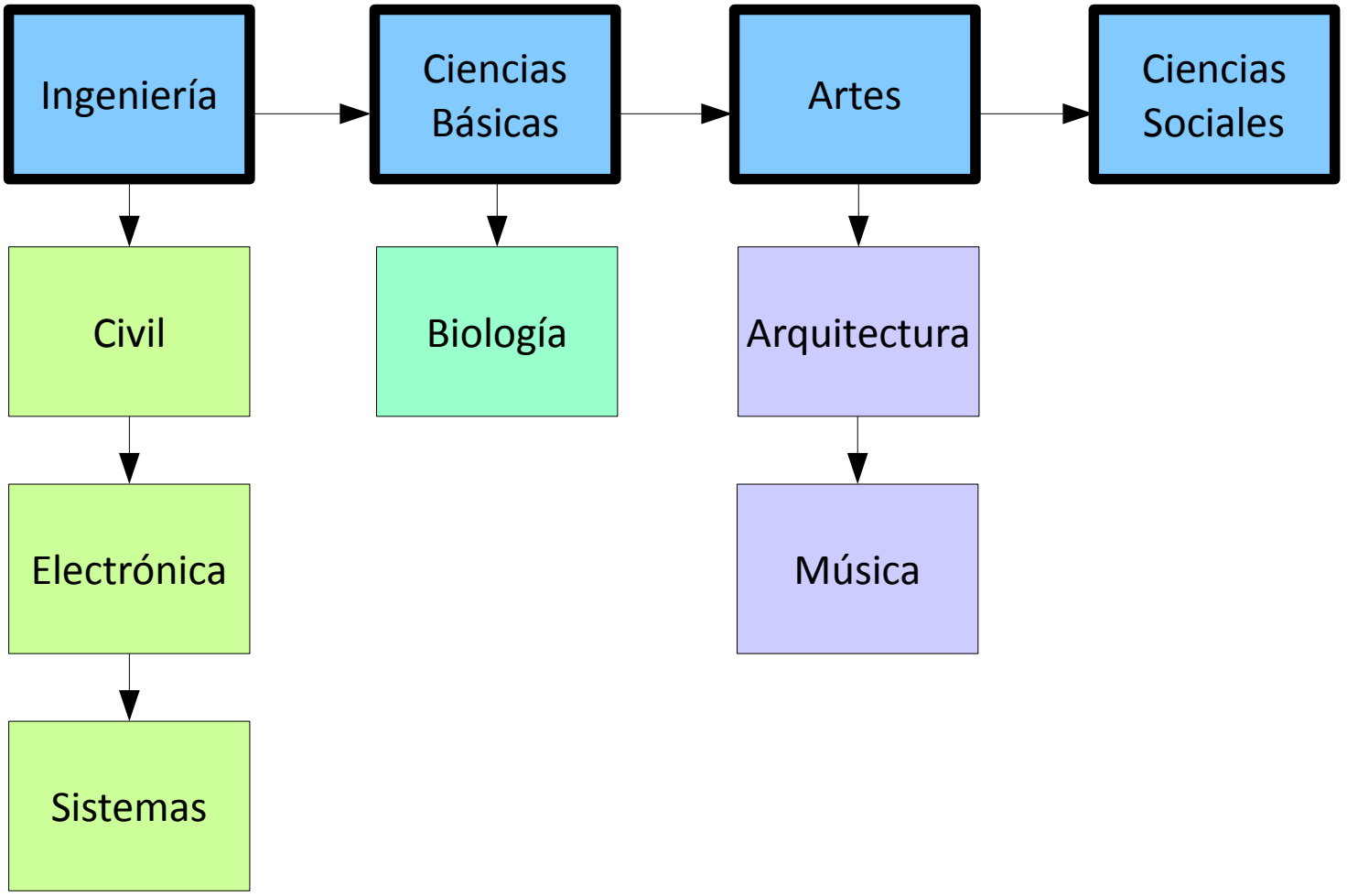
Composición

¡Multilistas!

```
typedef std::list<Libro> TArea;
typedef std::list<Area> TBiblio;

TBiblio lst = biblio.l_areas;
TBiblio::iterator lIt = lst.begin();

for ( ; lIt!=lst.end(); lIt++) {

  TArea slst = lIt->l_libros;
  TArea::iterator slIt = slst.begin( );

  for ( ; slIt!=slst.end(); slIt++)

    std::cout << slIt->nombreL << std::endl;

}
```

Ejercicio

- Carreras por facultades en una universidad :



Ejercicio

- Carreras por facultades en una universidad :

Carrera
- nombre
- cantidad estudiantes

Ejercicio

- Carreras por facultades en una universidad :

Carrera
- nombre
- cantidad estudiantes

Facultad
- nombre
- *lista* de Carrera

Ejercicio

- Carreras por facultades en una universidad :

Carrera
- nombre
- cantidad estudiantes

Facultad
- nombre
- *lista* de Carrera

Universidad
- nombre
- *lista* de Facultad

Ejercicio

- Completar los pasos faltantes:

  - Especificación de cada TAD

  - Diagrama de relación entre TADs

  - Implementación de cada TAD (cabecera y operaciones)

  - Programa principal para ingreso y visualización de datos de la Universidad

Pilas y colas

- Listas: propósito general (secuencias).
- ¿Qué pasa cuando no sólo la secuencia es importante, sino el orden de llegada?

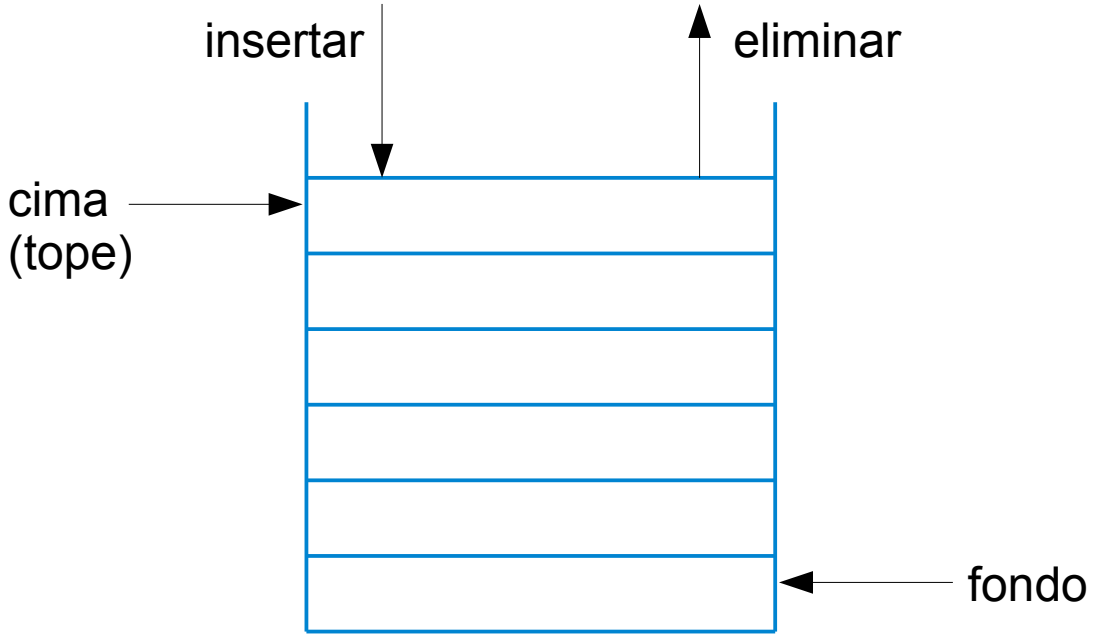
Pila
TAD Pila

Pila

- Colección ordenada de elementos con restricciones de acceso.
- Sus elementos sólo pueden accederse por un único lugar, el tope o cima de la pila.

Pila

- Entradas deben ser eliminadas en el orden inverso al que se situaron en la pila.
- “último en entrar, primero en salir” → estructura de datos LIFO (*last-in, first-out*).



Pila

Operaciones principales:

- Insertar (*push*):

  Añade un elemento y lo ubica en el tope de la pila.

- Eliminar (*pop*):

  Extrae el elemento en el tope y lo retira de la pila.

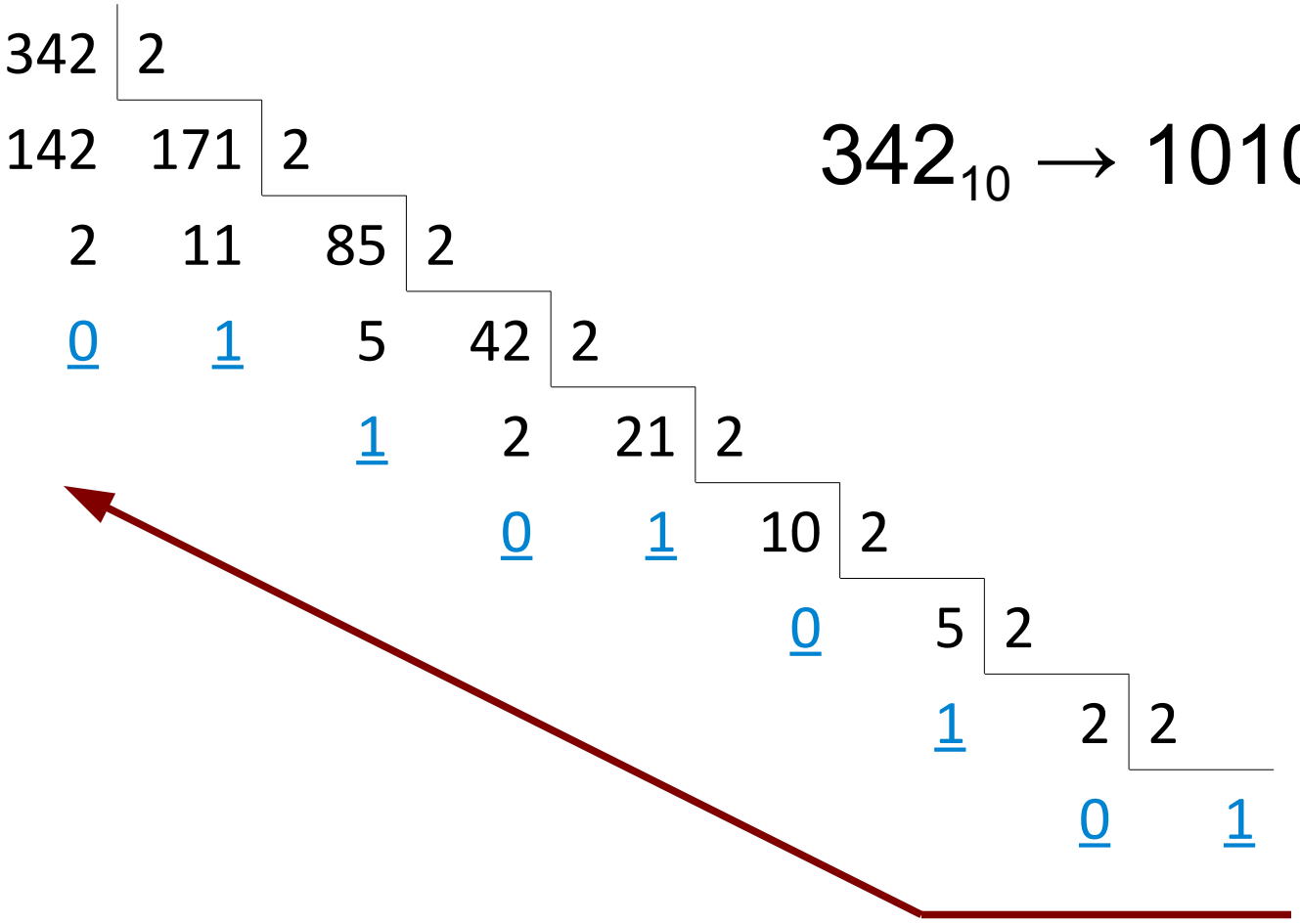
Pila

- Ejemplo de uso:
  conversión de decimal a binario.

$$342_{10} \rightarrow ?_{2}$$

Pila

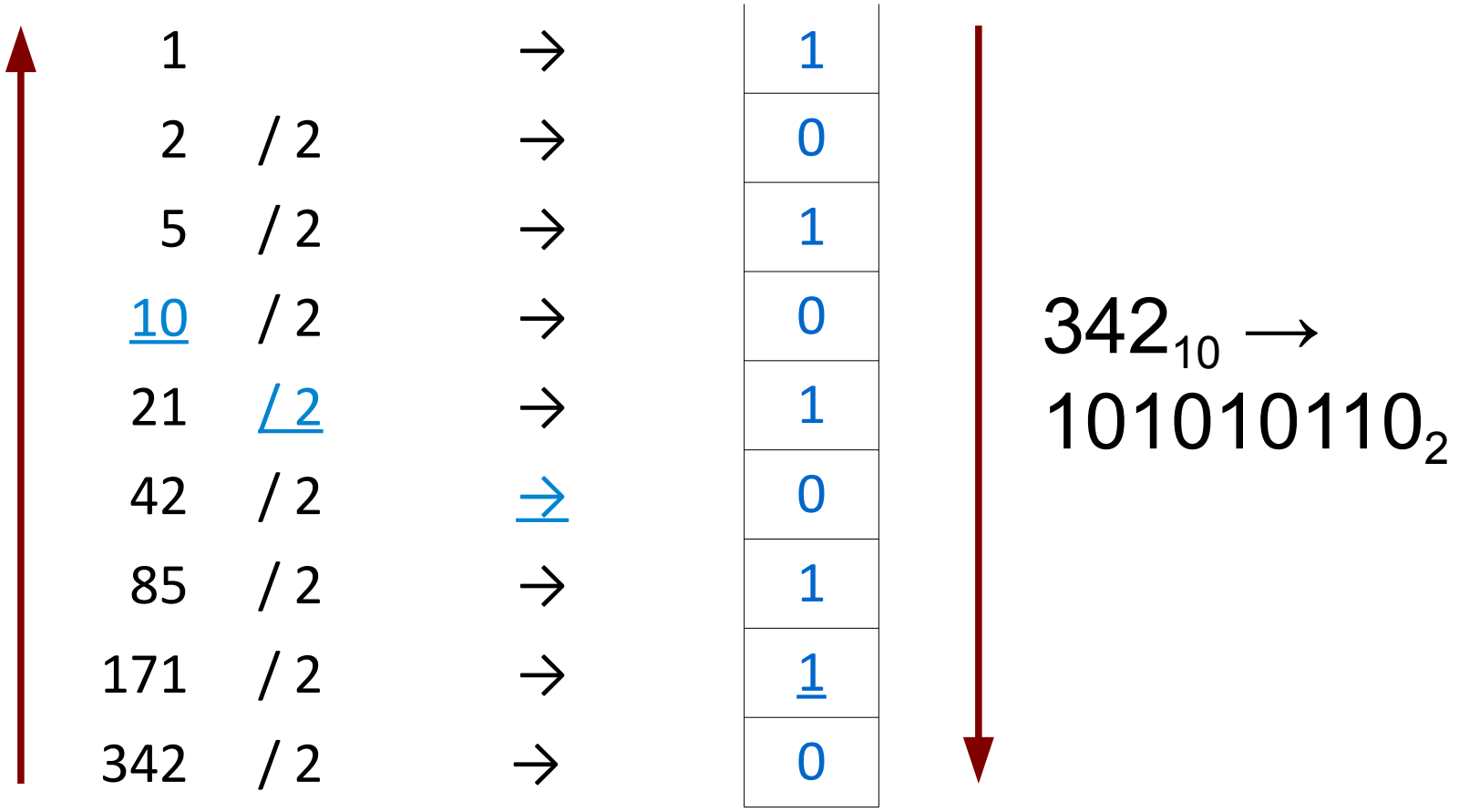
- Ejemplo de uso:
  conversión de decimal a binario.



$342_{10} \rightarrow 101010110_2$

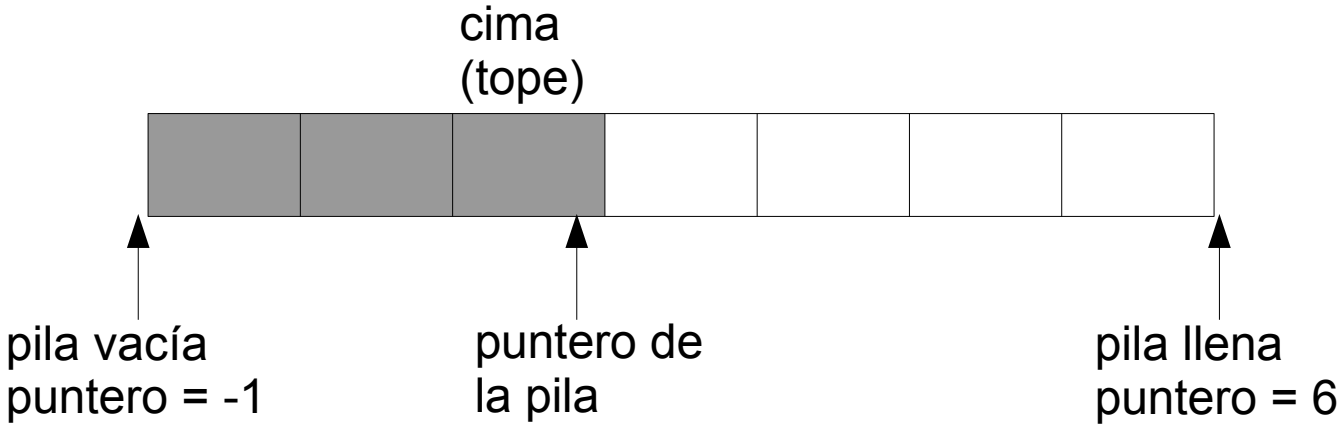
Pila

- Ejemplo de uso:
  conversión de decimal a binario.



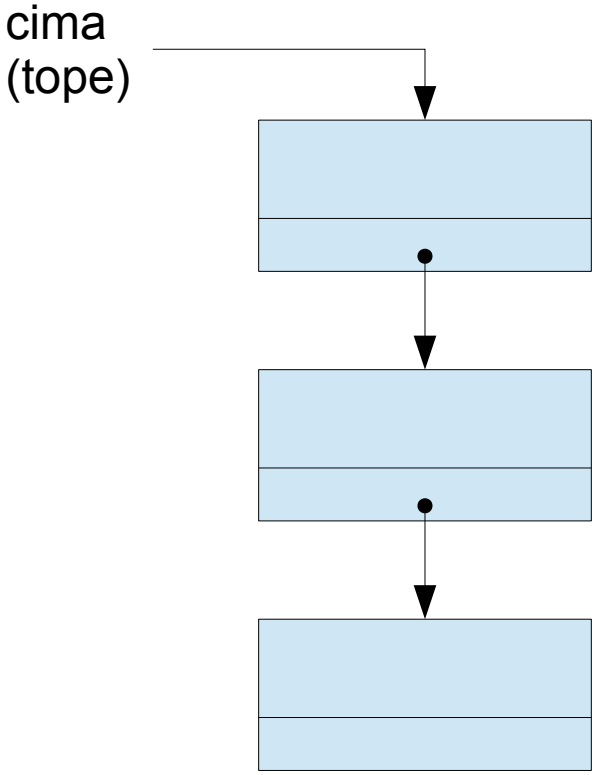
Pila

- Implementaciones:
  - Arreglos (implementación estática).



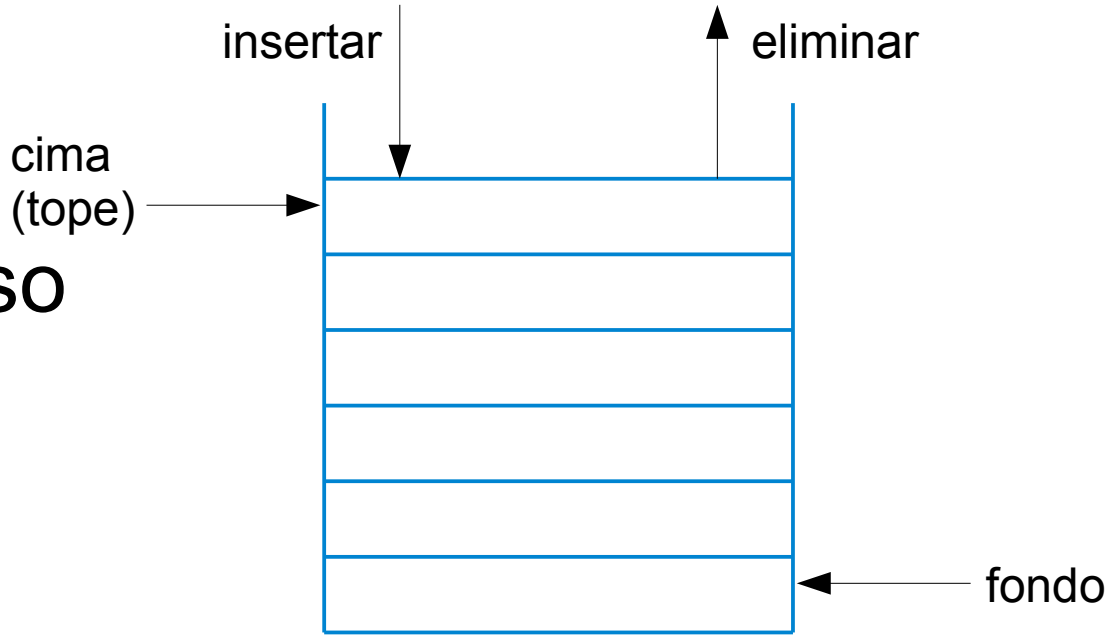
Pila

- Implementaciones:
  - Listas enlazadas (implementación dinámica).



TAD Pila

- Secuencia finita de datos.
  - Acceso sólo al elemento tope.
- Sin recorridos, acceso aleatorio restringido.
- Algoritmos:
  - Insertar, eliminar.
  - Vacía, tope.



¿Estado? ¿Interfaz?

TAD Pila

TAD Pila

Conjunto mínimo de datos:

Comportamiento (operaciones) del objeto:

TAD Pila

TAD Pila

Conjunto mínimo de datos:
- tope, ?, representa el tope (elemento accesible).

Comportamiento (operaciones) del objeto:
- esVacia(), indica si la pila está vacía.
- tope(), retorna el elemento en el tope.
- insertar(v), inserta v en la pila.
- eliminar(), elimina el elemento en el tope.
- vaciar(), elimina todos los elementos de la pila.

TAD Pila

- esVacia
  -> verificar si tope es nulo o no
- insertar
  -> crear nuevo nodo, poner dato ahí
  siguiente del nuevo nodo es el tope actual
  tope se actualiza al nuevo nodo

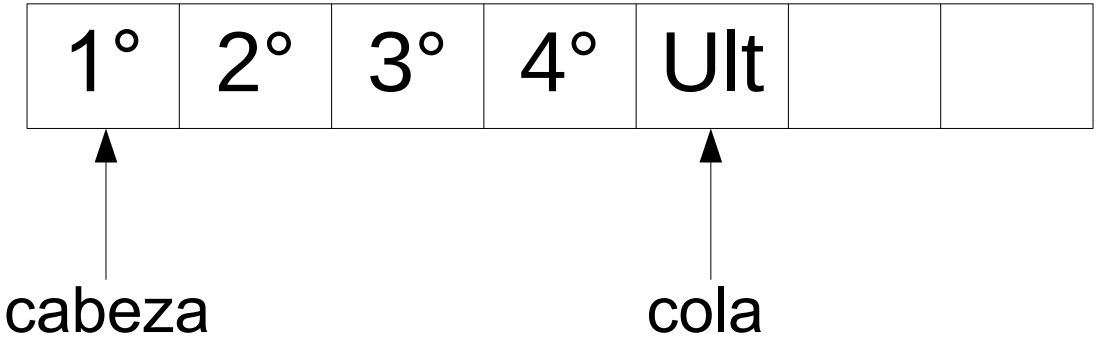
TAD Pila

- eliminar
  -> ubicar tope actual con un temporal
  tope se actualiza al siguiente del tope
  temporal se elimina de la memoria
- tope
  -> retorna tope actual
- vaciar
  -> eliminar de la pila hasta que quede vacía

Cola
TAD Cola

Cola

- Colección de elementos almacenados en una lista con restricciones de acceso.
- Los elementos sólo pueden ser insertados en la cola (final) de la lista, y sólo pueden ser eliminados por la cabeza (inicio o frente) de la lista .



Cola

- Entradas deben ser eliminadas en el mismo orden en el que se situaron en la cola.
- “primero en entrar, primero en salir” → estructura de datos FIFO (first-in, first-out).
- Ejemplos:
  - Atención a clientes en un almacén.
  - Gestión de trabajos en una impresora.

https://www.citrix.com/blogs/2013/06/28/efficient-queuing-but-where/

Cola

Operaciones principales:

- Insertar (*push*):

  Añadir un elemento por el extremo final de la cola.

- Eliminar (*pop*):

  Extraer el elemento ubicado en el extremo inicial de la cola .

Cola

- Ejemplo de uso:
  simulación de la atención de clientes en un banco con un solo cajero.

  Al banco llega un cliente cada x segundos.

  Un cajero atiende un cliente durante y segundos.

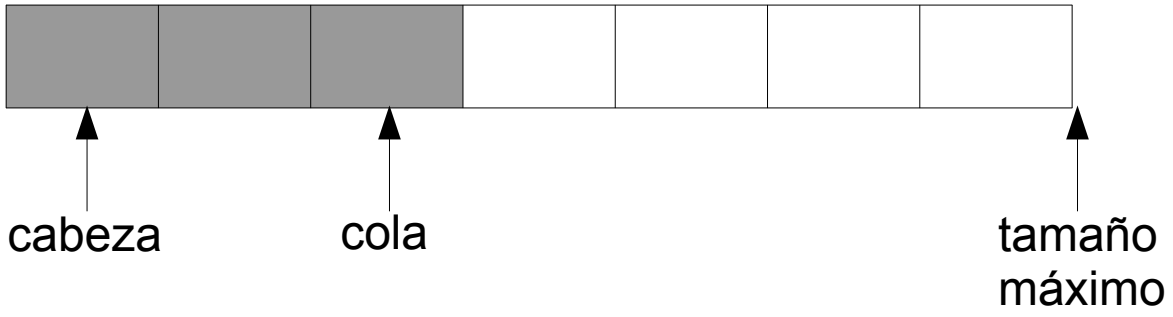
Cola

```
Segundos = 0
mientras (Segundos < 3600)
    si (Segundos % x == 0)
        cola.insertar(cliente)
    si (Segundos % y == 0)
        cola.eliminar()
    Segundos++
fin_mientras
```

Cola

- Implementaciones:

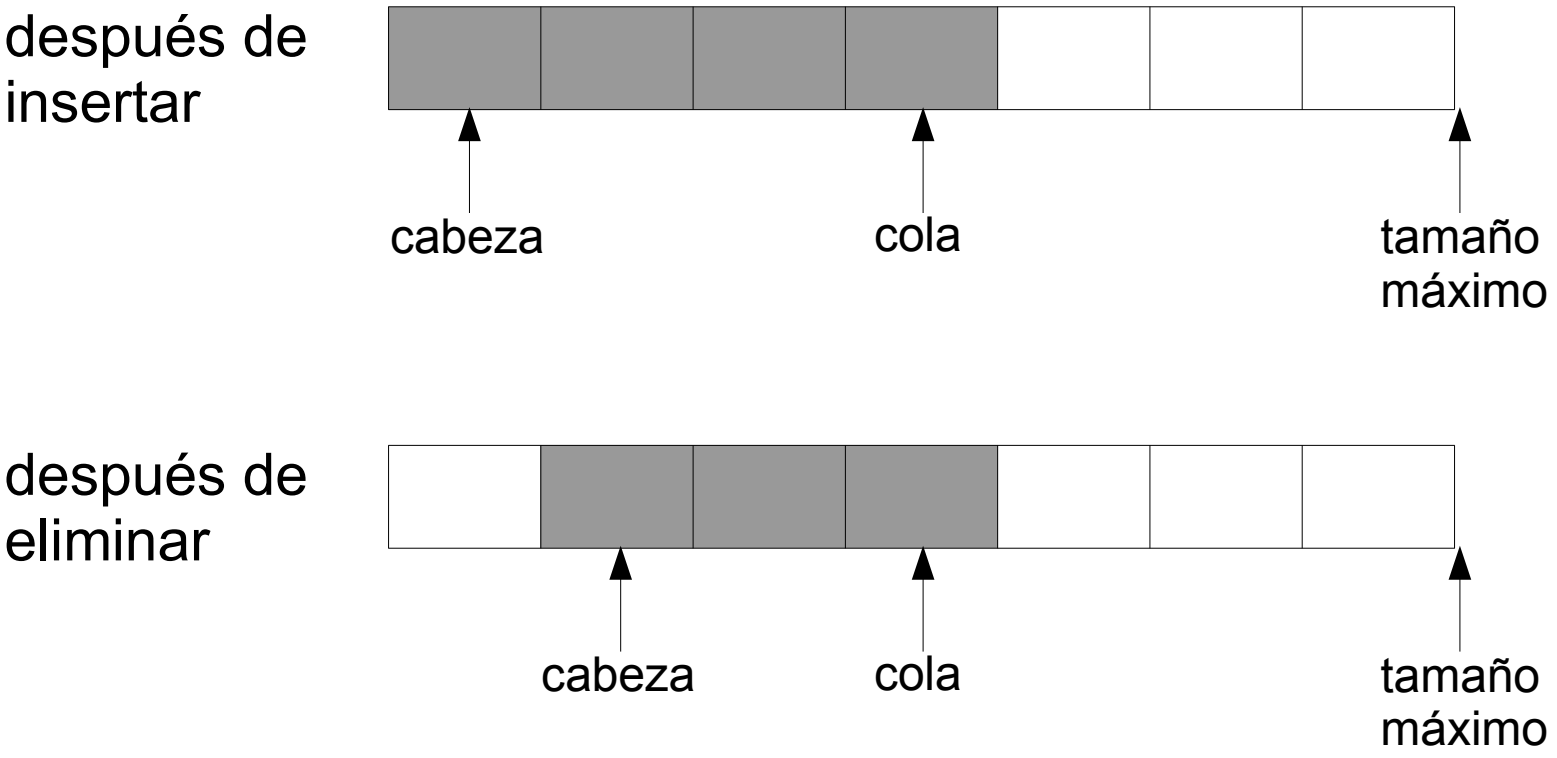
  - Arreglos (implementación estática).
  La cola no puede crecer indefinidamente, requiere un indicador de tamaño máximo.



Cola

- Implementaciones:

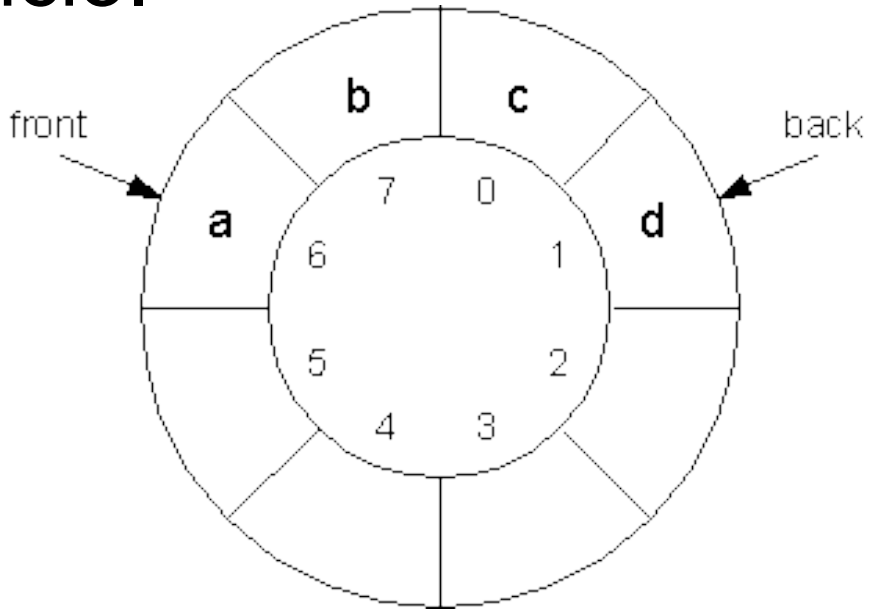
  - Arreglos (implementación estática).



Cola

- Implementaciones:

  - Arreglo circular (implementación estática). Se une el extremo final del arreglo con su extremo cabeza, para evitar las posiciones libres al inicio.

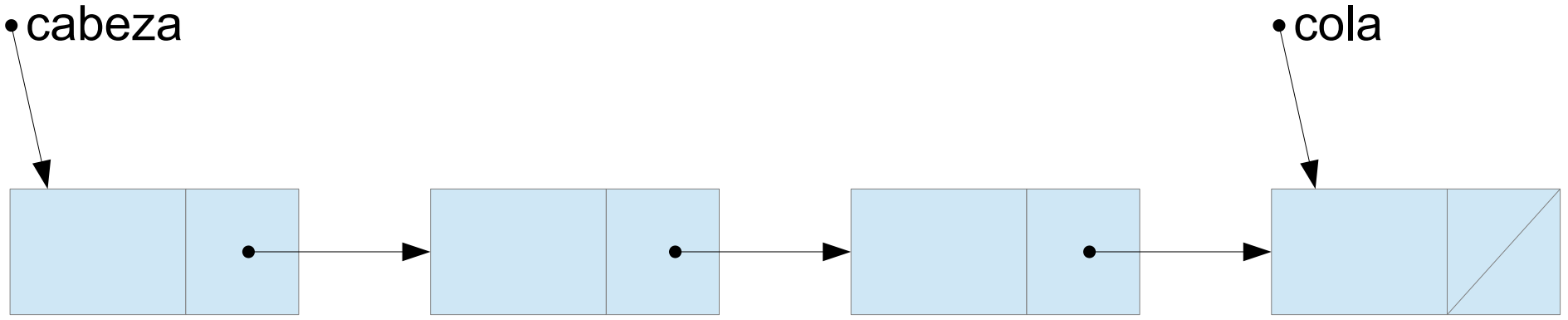


http://pages.cpsc.ucalgary.ca/~jacobs/Courses/cpsc331/F10/tutorials/tutorial8.html

Cola

- Implementaciones:

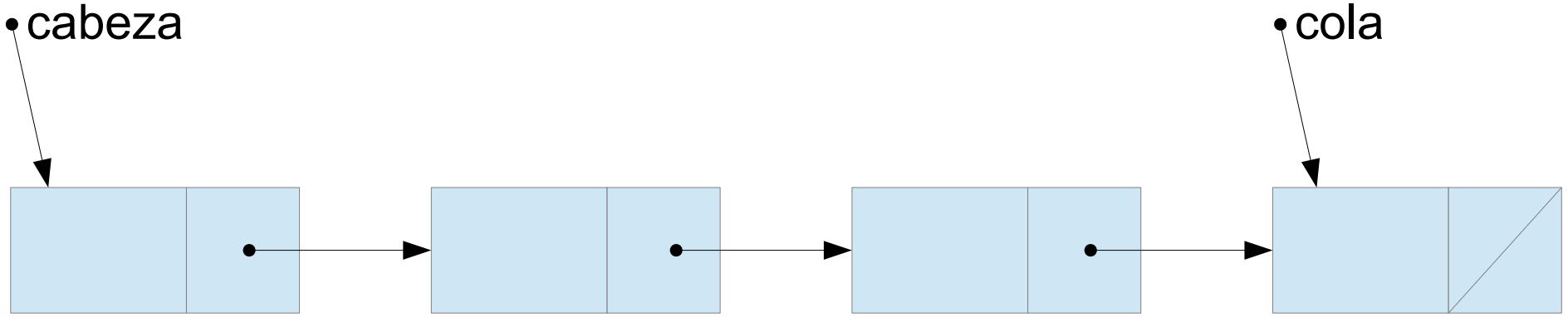
  - Lista enlazada (implementación dinámica).



TAD Cola

- Secuencia finita de datos.
  - Inserción por un extremo, eliminación por el otro.
- Sin recorridos, acceso aleatorio restringido
- Algoritmos:
  - Insertar, eliminar.
  - Vacía, cabeza.

¿Estado? ¿Interfaz?



TAD Cola

TAD Cola

Conjunto mínimo de datos:

Comportamiento (operaciones) del objeto:

TAD Cola

TAD Cola

Conjunto mínimo de datos:
- cabeza, ?, representa el extremo inicial.
- cola, ?, representa el extremo final.

Comportamiento (operaciones) del objeto:
- esVacia(), indica si la cola está vacía.
- cabeza(), retorna el elemento en la cabeza.
- insertar(v), inserta v en la cola.
- eliminar(), elimina el elemento en la cabeza.
- vaciar(), elimina todos los elementos en la cola.

TAD Cola

- esVacia
  -> verificar si cabeza y cola son nulos o no
- insertar
  -> crear nuevo nodo, poner dato ahí
  siguiente de la cola actual es el nuevo nodo
  cola se actualiza al nuevo nodo

TAD Cola

- eliminar
  -> ubicar cabeza actual con un temporal
  cabeza se actualiza al siguiente de la cabeza
  temporal se elimina de la memoria
- cabeza
  -> retorna cabeza actual
- vaciar
  -> eliminar de la cola hasta que quede vacía

Implementación
TADs Pila y Cola

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;

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

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

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;

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

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

Referencias

- L. Joyanes Aguilar, I. Zahonero. Algoritmos y estructuras de datos: una perspectiva en C. McGraw-Hill, 2004.
- www.sgi.com/tech/stl
- www.cplusplus.com