



Profesores:

Alfonso López Ruiz

Práctica 3

Implementación lineal en memoria dinámica de TAD genéricos

Estructuras de datos y algoritmos

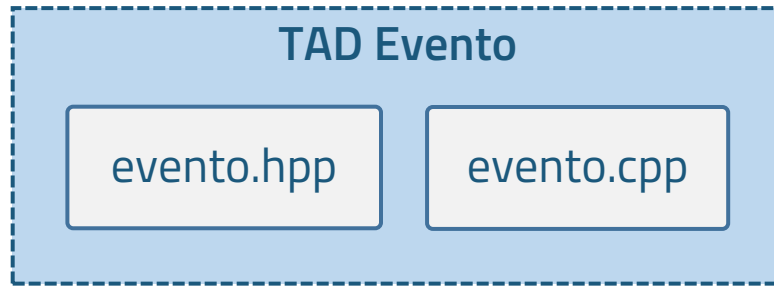
Grado en Ingeniería Informática – Curso académico 2025-2026



OBJETIVOS DE LA PRÁCTICA

- ▶ Implementar un **TAD genérico en memoria dinámica con estructura lineal**.
 - ▶ Colección de elementos interdependientes.
 - ▶ **Lista enlazada ordenada**.
- ▶ Implementar un **TAD no genérico**.
 - ▶ **Evento**.
- ▶ Comprobar de manera automática el funcionamiento del TAD genérico.
 - ▶ **Leer instrucciones y datos** de fichero.
 - ▶ **Construir un fichero de salida** con los resultados.

TAD no genérico



#include

practica3.cpp

#include

TAD Lista enlazada ordenada

lista.hpp

TAD genérico

ESPECIFICACIÓN DEL TAD NO GENÉRICO *EVENTO*

espec **Eventos**

usa **cadenas, naturales**

género **evento**

{Los valores del TAD evento representarán tuplas formadas como:
(descripción, prioridad) siendo la descripción una cadena y la prioridad un número natural.}

operaciones

...

fespec

ESPECIFICACIÓN DEL TAD NO GENÉRICO *EVENTO*

espec Eventos

...

operaciones

crearEvento: cadena descrip, natural prio -> evento  **Constructor**

{Devuelve un evento compuesto con descripción descrip y con prioridad prio.}

descripción: evento e -> cadena  **Getter**

{Dado un evento e formado como (D,P) devuelve la cadena D, i.e. la descripción en el evento e.}

cambiarDescripción: evento e, cadena nueva -> evento  **Setter**

{Dado un evento e formado como (D,P) devuelve un evento igual al compuesto como (nueva,P).}

suPrioridad: evento e -> natural  **Getter**

{Dado un evento e formado como (D,P), devuelve P, i.e. la prioridad en el evento e.}

cambiarPrioridad: evento e, natural pri -> evento  **Setter**

{Dado un evento e formado como (D,P) y un natural pri, devuelve un evento igual al compuesto como (D,pri).}

fespec

ESPECIFICACIÓN DEL TAD GENÉRICO *COLECCIÓN INTERDEPENDIENTES*

espec **coleccionesInterdependientes**

usa **booleanos, naturales**

parámetro **formal**

género **ident, val**

operación

{Se requiere que el género **ident** tenga definidas las siguientes operaciones.

Las operaciones **igual** y **anterior** definen una relación de orden total (i.e., permiten organizar los datos de la colección en forma de secuencia ordenada)}

igual: ident s1, ident s2 -> booleano

{devuelve verdad si y solo si s1 es igual que s2.}

anterior: ident s1, ident s2 -> booleano

{devuelve verdad si y solo si s1 es anterior a s2.}

fpf



ESPECIFICACIÓN DEL TAD GENÉRICO *COLECCIÓN INTERDEPENDIENTES*

espec **coleccionesInterdependientes**

usa **booleanos, naturales**

parámetro **formal**

género **ident, val**

operación

{Se requiere que el género **ident** tenga definidas las siguientes operaciones.

Las operaciones **igual** y **anterior** definen una relación de orden total (i.e., permiten organizar los datos de la colección en forma de secuencia ordenada)}

igual: ident s1, ident s2 -> booleano

{devuelve verdad si y solo si s1 es igual que s2.}

anterior: ident s1, ident s2 -> booleano

{devuelve verdad si y solo si s1 es anterior a s2.}

fpf

Necesitamos mantener el número de dependencias, no qué elementos dependen unos de otros.

Se puede modificar la solución para mantener el conjunto de elementos dependientes pero..

- Puede conducir a errores.
- Presenta mayor dificultad.
- No se tendrá en cuenta en la evaluación.
- Las dificultades introducidas aquí también afectarán a la práctica 4.

Tupla₁

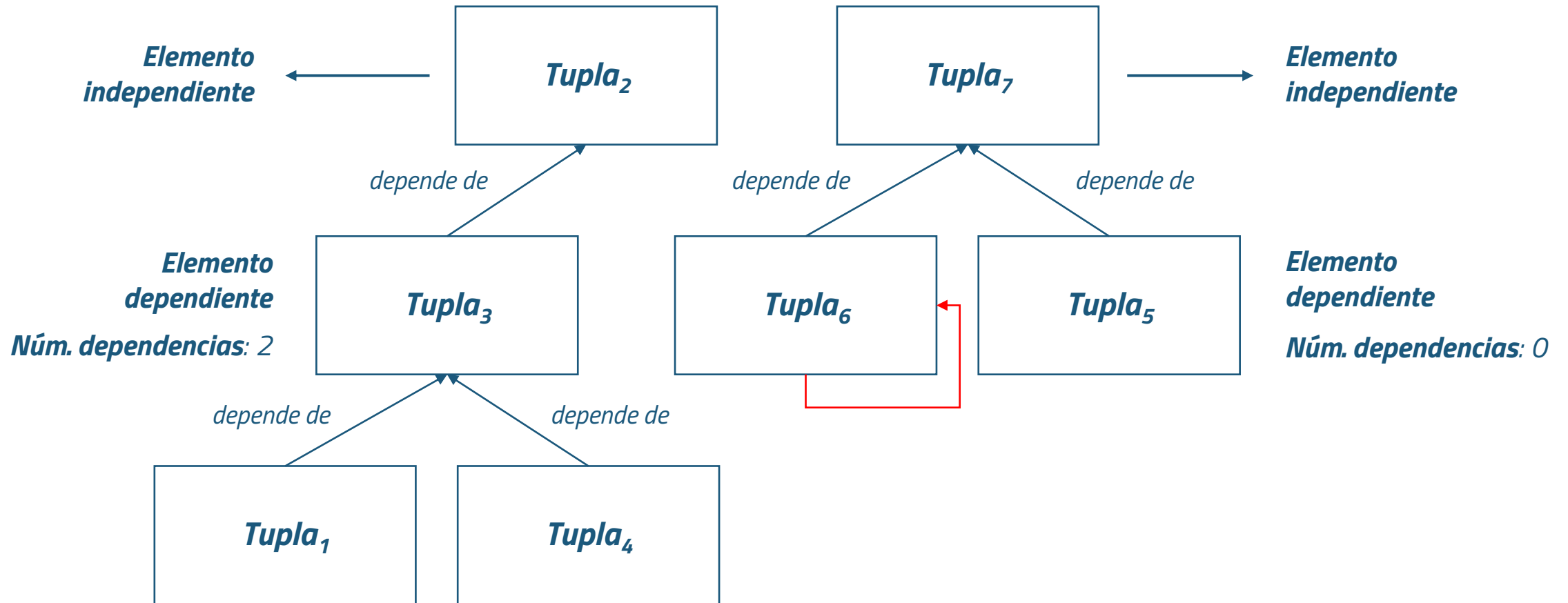
id: K
valor: V
identSup: K
numDepend: int

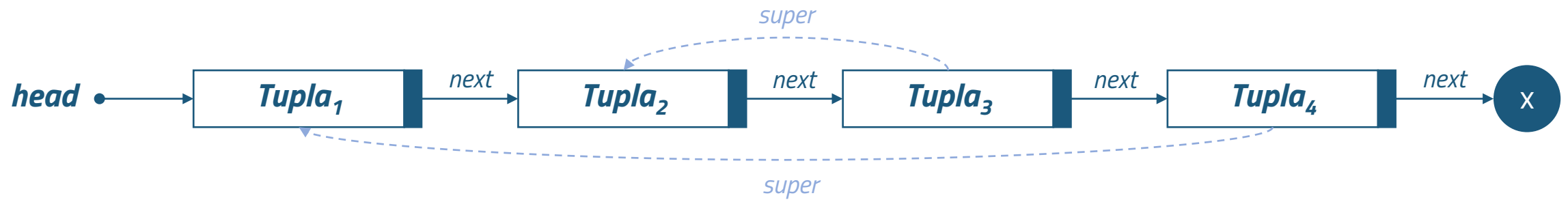
<
==

Tupla₂

id: K
valor: V
identSup: K
numDepend: int

ESPECIFICACIÓN DEL TAD GENÉRICO *COLECCIÓN INTERDEPENDIENTES*





ESPECIFICACIÓN DEL TAD GENÉRICO *COLECCIÓN INTERDEPENDIENTES*

espec **coleccionesInterdependientes**

usa **booleanos**, **naturales**

parámetro **formal**

género **ident**, **val**

operación

{Se requiere que el género **ident** tenga definidas las siguientes operaciones.

Las operaciones **igual** y **anterior** definen una relación de orden total (i.e., permiten organizar los datos de la colección en forma de secuencia ordenada)}

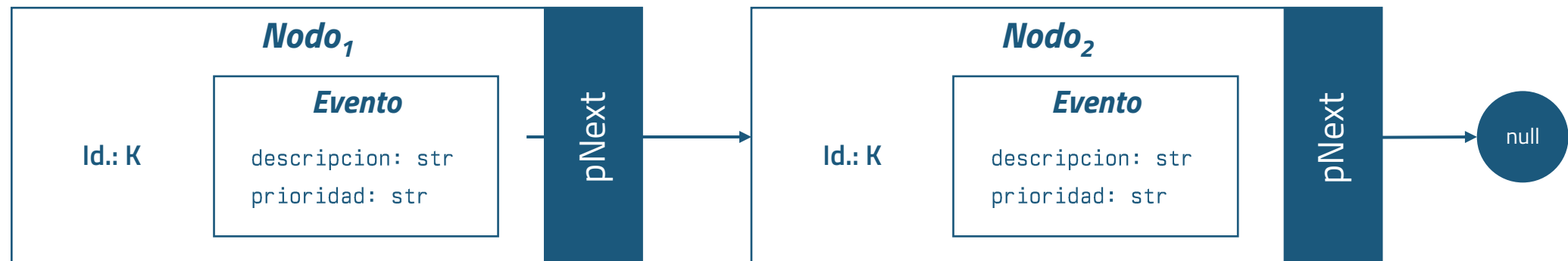
igual: **ident** **s1**, **ident** **s2** -> **booleano**

{devuelve verdad si y solo si **s1** es igual que **s2**.}

anterior: **ident** **s1**, **ident** **s2** -> **booleano**

{devuelve verdad si y solo si **s1** es anterior a **s2**.}

fpf



ESPECIFICACIÓN DEL TAD GENÉRICO *COLECCIÓN INTERDEPENDIENTES*

operaciones

crear: $\rightarrow \text{colecInterdep}$

{Crea una colección vacía, sin elementos.}

tamaño: $\text{colecInterdep } c \rightarrow \text{natural}$

{Devuelve el número de elementos que hay en la colección c .}

esVacía?: $\text{colecInterdep } c \rightarrow \text{booleano}$

{Devuelve verdad si y solo si c no contiene ningún elemento.}

existe?: $\text{ident id, colecInterdep } c \rightarrow \text{booleano}$

{Devuelve verdad si y solo si en c hay algún elemento con ident igual a id .}

existeDependiente?: $\text{ident id, colecInterdep } c \rightarrow \text{booleano}$

{Devuelve verdad si y solo si en c hay algún elemento dependiente cuyo ident sea igual a id .}

existeIndependiente?: $\text{ident id, colecInterdep } c \rightarrow \text{booleano}$

{Devuelve verdad si y solo si en c hay algún elemento independiente cuyo ident sea igual a id .}

ESPECIFICACIÓN DEL TAD GENÉRICO *COLECCIÓN INTERDEPENDIENTES*

operaciones

añadirIndependiente: colecInterdep c, ident id, val v -> colecInterdep

{Si no existe?(id,c), devuelve una colección igual a la resultante de añadir el elemento independiente (id,v,-,0) a la colección c. En caso contrario, devuelve una colección igual a c.}

añadirDependiente: colecInterdep c, ident id, val v, ident super -> colecInterdep

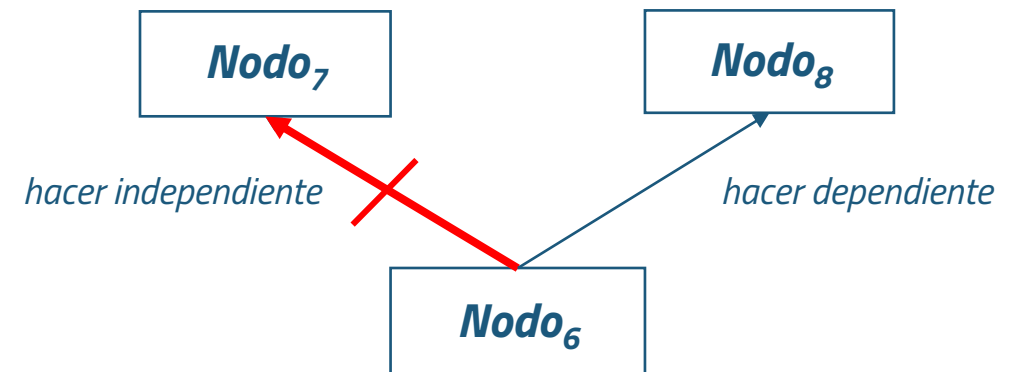
{Comprobar existe?(id,c) y existe?(super,c).}

hacerIndependiente: colecInterdep c, ident id -> colecInterdep

{Comprobar existeDependiente?(id,c).}

hacerDependiente: colecInterdep c, ident id, ident super -> colecInterdep

{Comprobar igual(id,super), existe?(super,c) y existeDependiente?(id,c). }



ESPECIFICACIÓN DEL TAD GENÉRICO *COLECCIÓN INTERDEPENDIENTES*

operaciones

```
parcial actualizarVal: colecInterdep c, ident id, val nuevo -> colecInterdep  
{Comprobar existe?(id,c).}
```

```
parcial obtenerVal: ident id, colecInterdep c -> val  
{Comprobar existe?(id,c). Devuelve el dato v que forma parte del elemento con dicho id.  
Parcial: la operación no está definida si no existe?(id,c).}
```

```
parcial obtenerSupervisor: ident id, colecInterdep c -> ident  
{Comprobar existeDependiente?(id,c).}
```

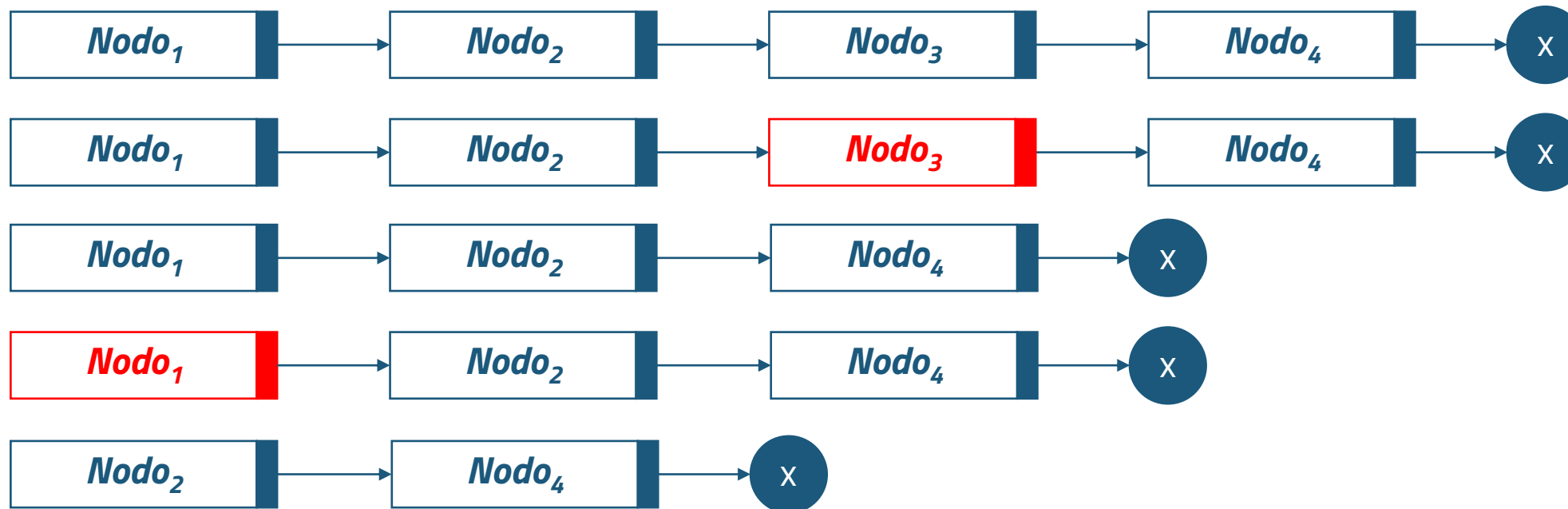
```
parcial obtenerNumDependientes: ident id, colecInterdep c -> natural  
{Comprobar existe?(id,c).}
```

ESPECIFICACIÓN DEL TAD GENÉRICO *COLECCIÓN INTERDEPENDIENTES*

operaciones

borrar: ident id, colecInterdep c -> colecInterdep

{Comprobar existeDependiente?(id,c).}

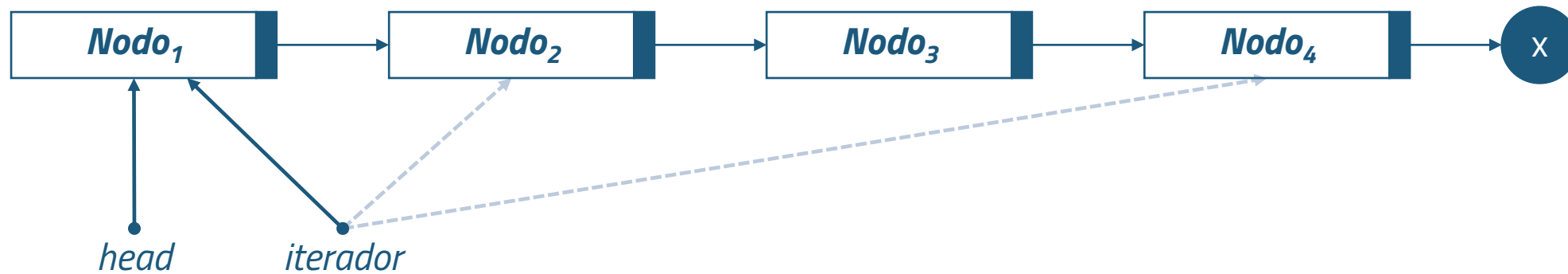


ESPECIFICACIÓN DEL TAD GENÉRICO *COLECCIÓN INTERDEPENDIENTES*

operaciones

iniciarIterador: `colecInterdep c -> colecInterdep`

{Inicializa el iterador para recorrer los elementos de la colección *c*, de forma que el siguiente elemento a visitar sea el que tiene un *ident* anterior a los de todos los demás elementos.}



ESPECIFICACIÓN DEL TAD GENÉRICO *COLECCIÓN INTERDEPENDIENTES*

operaciones

existeSiguiente?: colecInterdep c -> booleano

{Devuelve verdad si queda algún elemento por visitar con el iterador de la colección c.}

parcial siguienteIdent: colecInterdep c -> ident

{Devuelve el ident del siguiente elemento a visitar con el iterador de la colección c. Comprobar existeSiguiente?(c)}

parcial siguienteVal: colecInterdep c -> val

{Devuelve el val del siguiente elemento a visitar con el iterador de la colección c. Comprobar existeSiguiente?(c)}

ESPECIFICACIÓN DEL TAD GENÉRICO *COLECCIÓN INTERDEPENDIENTES*

operaciones

parcial siguienteDependiente?: colecInterdep c -> booleano

{Si el siguiente elemento a visitar es de la forma (ident, val, -, numDep) devuelve falso. Comprobar existeSiguiente?(c)}

parcial siguienteSuperior: colecInterdep c -> ident

{Si el siguiente elemento a visitar con el iterador de la colección es de la forma (ident, val, identSup, numDep), devuelve su identSup. Comprobar existeSiguiente?(c) y siguienteDependiente?(c). }

parcial siguienteNumDependientes: colecInterdep c -> natural

{Devuelve el NumDep del siguiente elemento a visitar con el iterador. Comprobar existeSiguiente?(c). }

parcial avanza: colecInterdep c -> colecInterdep

{Avanza el iterador. Comprobar existeSiguiente?(c)}

► Tipos de instrucciones:

- ▷ **L**istas **D**etalles de todos los elementos de la colección.
- ▷ **C**ambiar información de un evento en la colección.
 - ▷ Nombre, descripción y prioridad.
- ▷ **A**ñadir un evento a la colección.
 - ▷ Nombre, descripción, prioridad, (in)dependiente e id. del evento al que depende.
- ▷ **H**acer **D**ependiente un evento de la colección.
 - ▷ Nombre de evento dependiente y nombre del evento del que depende el anterior.
- ▷ **L**istar detalles de los eventos dependientes de un evento.
 - ▷ Nombre del evento.

► Tipos de instrucciones:

- ▷ **O**btener información de un evento.
 - ▷ Nombre del evento.
- ▷ Comprobar si **E**xiste un evento.
 - ▷ Nombre del evento.
- ▷ Hacer **I**ndependiente un evento.
 - ▷ Nombre del evento.
- ▷ **B**orrar un evento.
 - ▷ Nombre del evento.

```
ifstream f;  
f.open("entrada.txt");  
  
string instruccion, salto;  
while (f >> instruccion) {  
    getline(f,salto);  
    if (instruccion == "A") {  
        //... sigue el programa  
    } else if (instruccion == "0")  
    {  
        //... sigue el programa  
    }  
} //... sigue el programa
```

► Semana 1-2:

- ▷ Implementar operaciones de **añadir un evento**.
- ▷ Implementar **iterador**.
- ▷ Implementar consultas: ***esVacia?, tamaño, existe, existeIndependiente, existeDependiente...***
- ▷ **A, E, LT.**

► Semana 3-4:

- ▷ Implementar resto de operaciones de la colección.
- ▷ Implementar lectura de ***"entrada.txt"*** y comprobar formato de ***"salida.txt"***.

ENTREGA DE LA PRÁCTICA

- ▶ Comprobar resultados en **Hendrix**.
 - ▷ **C++ 11**.
- ▶ Utilizar siempre ***“entrada.txt”*** y ***“salida.txt”*** como rutas.
- ▶ **La salida debe coincidir exactamente** con la exigida en el guion.
 - ▷ Mayúscula, minúsculas, caracteres, etc.
- ▶ **No** se pueden utilizar **excepciones**.
- ▶ **No** se pueden utilizar ***smart pointers***.
- ▶ **Documentación** de código.
- ▶ **Incluir nombres completos y NIP de ambos autores al comienzo de cada fichero.**

ENTREGA DE LA PRÁCTICA

- ▶ Solo **entregará un miembro** del equipo.
- ▶ **practica3_NIP1_NIP2.zip**.
- ▶ Dos ficheros ***"entrada.txt"*** y ***"salida.txt"***.
 - ▷ **Significativamente diferentes** a los propuestos.
- ▶ **Estructura plana: solo código fuente y los dos ficheros anteriores.**