PRÁCTICA 3 (sesiones prácticas 3ª y 4ª) Estructuras de Datos y Algoritmos

Título: Implementación lineal en memoria dinámica de TAD genéricos. Curso: 2025/26

Objetivos:

- Implementar en C++ un TAD genérico "*colecciones de elementos interdependientes",* en memoria dinámica y basándose en estructuras lineales.
- Implementar en C++ un TAD no genérico *evento*.
- Implementar un programa principal que, utilizando una *"colección de elementos interdependientes"* para gestionar una colección de eventos, nos permita probar automáticamente las operaciones de los TAD, leyendo datos y órdenes (a realizar sobre la colección) de un fichero de texto y generando un fichero de texto con el resultado del seguimiento de dichas órdenes.

Fecha límite de entrega: 8-11-2025 (incluido)

Material (descargar de Moodle)

- Ejemplos de ficheros de texto, con los formatos de ficheros de entrada y salida para el programa a desarrollar en la tarea 3.

Descripción detallada:

Este enunciado se utilizará para las sesiones 3ª y 4ª, y además se reutilizará en las prácticas posteriores. En la página 7 se encuentra una propuesta de reparto del trabajo a realizar a lo largo de las semanas disponibles para realizar esta práctica 3.

Se trata de implementar un TAD genérico "*colecciones de elementos interdependientes*", particularizarlo y utilizarlo con un TAD no genérico *evento*, para implementar un programa que los utilice para gestionar una colección de eventos que puedan detectarse en un sistema de seguridad.

Tienes que realizar las siguientes **tareas**:

1. Implementar en C++ el TAD *evento* que se especifica a continuación:

```
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

  crearEvento: cadena descrip, natural prio → evento
  {Devuelve un evento compuesto con descripción descrip y con prioridad prio.}
  descripción: evento e → cadena
  {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
  {Dado un evento e formado como (D,P) devuelve un evento igual al compuesto como (nueva,P).}
  suPrioridad: evento e → natural
  {Dado un evento e formado como (D,P), devuelve P, i.e. la prioridad en el evento e.}
  cambiarPrioridad: evento e, natural pri → evento
  {Dado un evento e formado como (D,P) y un natural pri, devuelve un evento igual al compuesto como
  (D,pri).}

fespec
```

2. Desarrollar una implementación en C++, **genérica y en memoria dinámica** basándose en una **lista enlazada ordenada**, de un TAD genérico *colecInterdep,* que representa "*colecciones de elementos interdependientes"*. La especificación del *TAD colecInterdep* que vamos a utilizar es:

```
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

género colecInterdep {Los valores del TAD representan colecciones de elementos formados como tuplas
   de la forma (ident, val, -, NumDepend) o bien (ident, val, identSup, NumDepend). A los elementos
   con forma (ident, val, -, NumDepend) los llamaremos en general 'elementos independientes',
   mientras que a los elementos con forma (ident, val, identSup, NumDepend), los llamaremos en
   general 'elementos dependientes'. En la colección no podrá haber dos elementos con el mismo
   ident.
   En las tuplas que representan elementos dependientes, la información identSup será la
```

identificación del elemento del que es directamente dependiente el elemento con identificación ident. Ningún elemento de la colección podrá ser directamente dependiente de sí mismo, y todo elemento dependiente debe serlo de otro elemento que exista en la colección (que a su vez puede ser un elemento independiente o dependiente).
En cada elemento, la información NumDepend de su tupla representará el número total de elementos en la colección que son directamente dependientes del elemento con identificador ident, y que será 0 si ningún elemento de la colección depende de dicho elemento.}

operaciones

crear: → colecInterdep
{Crea una colección vacía, sin elementos.}

tamaño: colecInterdep c → natural
{Devuelve el número de elementos que hay en la colección c.}

esVacía?: colecInterdep c → booleano
{Devuelve verdad si y solo si c no contiene ningún elemento.}

existe?: ident id, colecInterdep c → booleano
{Devuelve verdad si y solo si en c hay algún elemento con ident igual a id.}

existeDependiente?: ident id, colecInterdep c → booleano
{Devuelve verdad si y solo si en c hay algún elemento dependiente cuyo ident sea igual a id, es decir un elemento (id, v, idSup, NumDep).}

existeIndependiente?: ident id, colecInterdep c → booleano
{Devuelve verdad si y solo si en c hay algún elemento independiente cuyo ident sea igual a id, es decir un elemento (id, v, -, NumDep).}

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
{Si no existe?(id,c) y existe?(super,c) devuelve una colección igual a la resultante de: incrementar en 1 el número de elementos dependientes del elemento con identificador super en c, y de añadir el elemento (id,v,super,0) a la colección c.
En cualquier otro caso, devuelve una colección igual a c.}

hacerDependiente: colecInterdep c, ident id, ident super → colecInterdep
{Si no igual(id,super) y existe?(super,c) y existeDependiente?(id,c), sea su forma (id, v, superAnt, NumDep), devuelve una colección igual a la resultante de: incrementar en 1 el número de elementos dependientes del elemento con identificador super en c, decrementar en 1 el número de elementos dependientes del elemento con identificador superAnt en c, y sustituir el elemento (id, v, superAnt, NumDep) por el elemento (id, v, super, NumDep) en c.
Si no igual(id,super) y existe?(super,c) y existeIndependiente?(id,c), sea su forma (id, v, -, NumDep), devuelve una colección igual a la resultante de: incrementar en 1 el número de elementos dependientes del elemento con identificador super en c, y sustituir en c el elemento (id, v, -, NumDep) por el elemento (id, v, super, NumDep).
En cualquier otro caso, devuelve una colección igual a c.}

hacerIndependiente: colecInterdep c, ident id → colecInterdep
{Si existeDependiente?(id,c), sea su forma (id, v, superAnt, NumDep), devuelve una colección igual a la resultante de: decrementar en 1 el número de elementos dependientes del elemento con identificador superAnt en c, y sustituir el elemento (id, v, superAnt, NumDep) por el elemento (id, v, -, NumDep). En cualquier otro caso, devuelve una colección igual a c.}

parcial actualizarVal: colecInterdep c, ident id, val nuevo → colecInterdep
{Si existeIndependiente?(id,c), sea su forma (id, v, -, NumDep), devuelve una colección igual a la resultante de sustituir el elemento (id, v, -, NumDep) por el elemento (id, nuevo, -, NumDep) en la colección c. Si existeDependiente?(id,c), sea su forma (id, v, super, NumDep), devuelve una colección igual a la resultante de sustituir el elemento (id, v, super, NumDep) por el elemento (id, nuevo, super, NumDep) en la colección c.
Parcial: la operación no está definida si no existe?(id,c).}

parcial obtenerVal: ident id, colecInterdep c → val
{Si en c hay algún elemento con ident igual a id, sea (id,v,-,num) o sea (id,v,sup,num), 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
{Si existeDependiente?(id,c), sea su forma (id, v, sup, NumDep), devuelve el dato sup que forma parte del elemento con dicho id.
Parcial: la operación no está definida si no existeDependiente?(id,c).}

parcial obtenerNúmDependientes: ident id, colecInterdep c → natural
{Si existe?(id,c), sea su forma (id, v, -, NumDep) o (id, v, sup, NumDep), devuelve el dato NumDep que forma parte del elemento con dicho id.
Parcial: la operación no está definida si no existe?(id,c).}

borrar: ident id, colecInterdep c → colecInterdep
{Si existeDependiente?(id,c), sea su forma (id, v, sup, NumDep), y obtenerNúmDependientes(id,c)=0, devuelve una colección igual a la resultante de: decrementar en 1 el número de elementos dependientes del elemento con identificador sup, y eliminar el elemento (id, v, sup, NumDep), en la colección c.
Si existeIndependiente?(id,c), sea su forma (id, v, -, NumDep), y obtenerNúmDependientes(id,c)=0, devuelve una colección igual a la resultante de eliminar el elemento (id, v, -, NumDep) en la

```
    colección c. En cualquier otro caso, devuelve una colección igual a c.}

    {Las siguientes operaciones constituyen un iterador que permite visitar los elementos de la
     colección de forma que siempre se visiten antes los elementos con ident anteriores a los que se
     visitarán más tarde:}
    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 de la
    colección (situación de no haber visitado ningún elemento).}

    existeSiguiente?: colecInterdep c → booleano
    {Devuelve verdad si queda algún elemento por visitar con el iterador de la colección c, devuelve
    falso si ya se ha visitado el último elemento.}

    parcial siguienteIdent: colecInterdep c → ident
    {Devuelve el ident del siguiente elemento a visitar con el iterador de la colección c, que será
    el elemento no visitado con ident anterior a los de todos los demás aún no visitados.
    Parcial: la operación no está definida si no quedan elementos por visitar (no existeSiguiente?(c))}

    parcial siguienteVal: colecInterdep c → val
    {Devuelve el val del siguiente elemento a visitar con el iterador de la colección c, que será el
    elemento no visitado con ident anterior a los de todos los demás aún no visitados.
    Parcial: la operación no está definida si no quedan elementos por visitar (no existeSiguiente?(c))}

    parcial siguienteDependiente?: colecInterdep c → booleano
    {Si el siguiente elemento a visitar con el iterador de la colección, que será el elemento no
    visitado con ident anterior a los de todos los demás aún no visitados, es de la forma
    (ident, val, -, numDep) devuelve falso, pero si es de la forma (ident, val, identSup, numDep)
    devuelve verdad.
    Parcial: la operación no está definida si no quedan elementos por visitar (no existeSiguiente?(c))}

    parcial siguienteSuperior: colecInterdep c → ident
    {Si el siguiente elemento a visitar con el iterador de la colección, que será el elemento no
    visitado con ident anterior a los de todos los demás aún no visitados, es de la forma
    (ident, val, identSup, numDep), devuelve su identSup.
    Parcial: la operación no está definida si no quedan elementos por visitar (no existeSiguiente?(c)),
    o bien si existeSiguiente?(c) y no es verdad siguienteDependiente?(c). }

    parcial siguienteNúmDependientes: colecInterdep c → natural
    {Devuelve el NumDep del siguiente elemento a visitar con el iterador de la colección c, que
    será el elemento no visitado con ident anterior a los de todos los demás aún no visitados.
    Parcial: la operación no está definida si no quedan elementos por visitar (no existeSiguiente?(c))}

    parcial avanza: colecInterdep c → colecInterdep
    {Avanza el iterador de la colección c para que se pueda visitar otro elemento.
    Parcial: la operación no está definida si no quedan elementos por visitar (no existeSiguiente?(c))}

fespec
```

Advertencia: nótese que la especificación del **TAD *colecInterdep* sólo requiere** que para cada elemento se pueda gestionar eficientemente la información sobre **cuántos elementos son directamente dependientes** de cada elemento de la colección (*NumDepend*), y no requiere gestionar la información sobre **cuáles son los directamente dependientes** de cada elemento. Sería factible plantear para este TAD implementaciones cuya representación interna gestione no sólo la información de cuántos son dependientes de un elemento sino también la información sobre cuáles son dependientes de él, sin embargo, tales implementaciones requerirían tener en cuenta y resolver más detalles y dificultades (en esta práctica y en la siguiente) que las que se pretenden con el enunciado tal cual está planteado. **No se recomienda intentar** resolver la práctica planteando **tales implementaciones,** y en cualquier caso optar por ellas **no permitirá obtener más nota, ni excusará comportamientos incorrectos o ineficientes de las implementaciones entregadas para su evaluación.**

3. Utilizar la implementación del TAD *evento*, implementado en la tarea 1, y la del TAD genérico *colecInterdep* implementado en la tarea 2, para implementar un programa de prueba que gestione una colección de eventos que puedan detectarse en un sistema de seguridad.

   Los elementos de la colección de elementos interdependientes con la que trabajará este programa se formarán con un nombre (cadena) y un *evento,* de tal forma que en la colección no podrá haber nombres repetidos.

   El programa de prueba deberá crear una colección vacía y, a continuación, leer las instrucciones de las operaciones a realizar con la colección, desde un fichero de texto denominado “entrada.txt”. El fichero “entrada.txt” tendrá la siguiente estructura o formato:

```
<instrucción1>
<datos para la instrucción1>
<instrucción2>
<datos para la instrucción2>
...
<instrucciónÚltima>
<datos para la instrucciónÚltima>
```

Donde `<instrucciónX>` podrá ser alguna de las siguientes cadenas: *"A", "C", "O", "E", "I", "D","B", "LD", "LT",* que representarán respectivamente las ordenes de: *"**A**ñadir la información de un nuevo nombre y evento a la colección", "**C**ambiar la información de un evento en la colección para un nombre dado", "**O**btener toda la información relativa al evento de la colección correspondiente a un nombre dado", "comprobar si en la colección **E**xiste algún evento relativo a un nombre dado", "hacer **I**ndependiente el evento de la colección correspondiente a un nombre dado", "hacer **D**ependiente el evento de la colección correspondiente a un nombre dado", "**B**orrar de la colección un evento dado su nombre", "**L**istar los detalles relativos a todos los eventos que en la colección que son directamente **D**ependientes del evento con un nombre dado", y "**L**istar los detalles relativos a **T**odos los nombres y eventos, de la colección*".

El programa finalizará cuando haya procesado todas las instrucciones del fichero.

Supondremos que el fichero "entrada.txt" tendrá siempre una estructura como la descrita, sin errores, es decir:

- Si la instrucción es "*A",* las siguientes 5 líneas del fichero contendrán los valores de:
  - una línea con el nombre (a usar para el evento a añadir).
  - una línea con la descripción para crear el evento.
  - una línea con un número, a usar como la prioridad para el evento.
  - una línea con la cadena "DEPendiente" o "INDependiente", según se quiera añadir a la colección un evento dependiente o independiente, respectivamente
  - una línea, con el nombre dado al evento del cual dependerá directamente el evento que se va a añadir a la colección de eventos, o bien la cadena `"-.-.-.-.-"` si el que se va a añadir es un evento independiente
- Si la instrucción es *"C",* las siguientes 3 líneas del fichero contendrán los valores de:
  - una línea con el nombre (a usar con el evento a modificar).
  - una línea con la descripción, a usar para el evento.
  - una línea con un número, a usar como la prioridad para el evento.
- Si la instrucción es *"D",* las siguientes 2 líneas del fichero contendrán los valores de:
  - una línea con el nombre del evento que hay que convertir en dependiente.
  - una línea con el nombre del evento del cual dependerá directamente el evento con el nombre anterior.
- Si la instrucción es *"O", "E", "I", "B" o "LD",* la siguiente línea del fichero contendrá un nombre que será el del evento respecto al cual se actúa o consulta**.**
- Si la instrucción es *"LT"* no requiere que le sigan otras líneas con más datos para su ejecución, así que la siguiente línea sería la de la siguiente instrucción, o el final del fichero.

Para todas ellas, tanto en el caso de *nombres* como en el caso de *descripciones*, ninguno de ellos superará una línea, pero pueden contener cualquier número de palabras.

Para resolver cada instrucción, el programa deberá utilizar las operaciones ofrecidas por los TAD implementados en las tareas anteriores:

Cuando se procese una instrucción *"A"*, el programa intentará introducir en la colección un nuevo nombre y su evento, utilizando los datos dados en el fichero para la instrucción. Cuando se procese una instrucción *"C"*, el programa intentará actualizar el evento correspondiente de un nombre existente en la colección, utilizando los datos dados en la instrucción para actualizar los datos de su evento. Cuando se procese la instrucción *"O"*, el programa intentará obtener toda la información relativa al evento registrado en la colección con el nombre dado. Cuando se procese la instrucción *"E"*, el programa comprobará si en la colección existe un evento para el nombre dado. Cuando se procese una instrucción *"I"*, el programa intentará actualizar la colección de forma que, el evento correspondiente al nombre dado para la instrucción, pase a ser independiente. Cuando se procese una instrucción *"D"*, el programa intentará actualizar la colección de forma que, el evento correspondiente al primer nombre dado para la instrucción, pase a ser directamente dependiente del evento con el segundo nombre dado para la instrucción. Cuando se procese la instrucción *"B"*, el programa deberá eliminar de la colección el nombre, igual al dado, junto con su evento. Cuando se procese la instrucción *"LD"* el programa deberá listar en su salida todos los detalles relativos al evento de la colección registrado con el nombre dado para la instrucción, y deberá mostrar también un listado con todos los detalles relativos a todos los eventos que en la colección dependen directamente del evento con el nombre dado. Cuando se procese una instrucción *"LT"*, el programa deberá listar en su salida todos los datos de los nombres, y sus eventos, registrados en la colección.

Como resultado de su ejecución, el programa creará un fichero de texto "salida.txt" en el que irá escribiendo el resultado de cada orden ejecutada, y que constará de las siguientes líneas:

- Por cada instrucción de tipo *"A"*, **si es posible introducir los datos de un nuevo nombre y su evento en la colección**, se escribirá una línea en el fichero de salida que empiece con la cadena `"INTRODUCIDO: "`; **si no es posible introducir el nombre y su evento en la colección**, se escribirá una línea en el fichero de salida que empiece con la cadena `"NO INTRODUCIDO: "`. En cualquiera de los dos casos, se seguirá escribiendo a continuación, y en la misma línea del fichero, una cadena de acuerdo a uno de los dos formatos que se describen a continuación, utilizando el que corresponda según se haya intentado añadir en colección como independiente o como dependiente:

```
[ nombre ] --- descrip --- ( prioridad )
[ nombre -de-> nomSup ] --- descrip --- ( prioridad )
```

siendo `nombre`, `descrip`, y `prioridad` respectivamente, el nombre, la descripción, y la prioridad, dados para la instrucción, y siendo en su caso `nomSup` el segundo nombre dado para la instrucción y respecto al cual se quería hacer directamente dependiente el de nombre `nombre`, a añadir en la colección.

- Por cada instrucción de tipo *"C"*, **si es posible actualizar en la colección los datos del evento correspondiente al nombre dado**, se escribirá una línea en el fichero de salida que empiece con la cadena "`CAMBIADO: `", seguida de la concatenación de todos los datos relativos al nombre encontrado y su evento en la colección, de acuerdo a uno de los dos formatos que se describen a continuación, utilizando el que corresponda según se haya encontrado que en colección se encuentra como independiente o como dependiente:

```
[ nombre --- NumDepend ] --- descrip --- ( prioridad )
[ nombre -de-> nomSup ;;; NumDepend ] --- descrip --- ( prioridad )
```

siendo `nombre, descrip` y `prioridad` respectivamente, el nombre, la descripción y la prioridad dados para la instrucción, y siendo `NumDepend` el número de elementos de la colección directamente dependientes de aquel con el nombre dado para la instrucción. En el caso de que en la colección este sea independiente, aparecerá con el formato descrito en la primera línea, pero si en la colección no es independiente, aparecerá con el formato de la segunda línea, siendo `nomSup` el nombre del elemento de la colección del que depende el que tiene el nombre dado para la instrucción.

Si no es posible actualizar el evento de dicho nombre en la colección, se escribirá una línea en el fichero de salida que empiece con la cadena "`NO CAMBIADO: `", seguida del nombre dado para la instrucción.

- Por cada instrucción de tipo *"O"*, **si existía** algún evento en la colección para un nombre igual al dado, se escribirá una línea que empiece con la cadena "`LOCALIZADO: `", seguida de la concatenación de todos los datos relativos al nombre encontrado y su evento en la colección, de acuerdo a uno de los dos formatos que se describen a continuación, utilizando el que corresponda según se haya encontrado que en colección se encuentra como independiente o como dependiente:

```
[ nombre --- NumDepend ] --- descrip --- ( prioridad )
[ nombre -de-> nomSup ;;; NumDepend ] --- descrip --- ( prioridad )
```

siendo `nombre`, el dato dado para la instrucción; siendo `descrip` y `prioridad` respectivamente, la descripción y la prioridad del evento correspondiente a dicho nombre en la colección, y siendo `NumDepend` el número de elementos en la colección directamente dependientes de aquel con el nombre dado para la instrucción. En el caso de que en la colección este sea independiente, aparecerá con el formato descrito en la primera línea, pero si en la colección no es independiente, aparecerá con el formato de la segunda línea, siendo `nomSup` el nombre del elemento de la colección del que depende el que tiene el nombre dado para la instrucción.

Si no existía ningún evento en la colección para un nombre igual al dado, se escribirá en el fichero una línea que empiece con la cadena "`NO LOCALIZADO: `", seguida del nombre dado para la instrucción.

- Por cada instrucción de tipo *"E"*, **si existía** algún evento en la colección para un nombre igual al dado y que no es directamente dependiente de ningún otro, se escribirá una línea en el fichero que empiece con la cadena "`INDEPendiente: `", **si existía** algún evento en la colección para un nombre igual al dado y que resulta ser directamente dependiente de algún otro, se escribirá una línea en el fichero que empiece con la cadena "`DEPendiente: `". **Si no existía** ningún evento en la colección para un nombre igual al dado, se escribirá en el fichero una línea que empiece con la cadena "`DESCONOCIDO: `". En cualquiera de los tres casos, se seguirá escribiendo a continuación, y en la misma línea del fichero, el nombre dado para la instrucción.
- Por cada instrucción de tipo *"I"*, **si existía** algún evento en la colección para un nombre igual al dado y que no fuese independiente, se escribirá una línea que empiece con la cadena "`INDEPENDIZADO: `", **si existía** algún evento en la colección para un nombre igual al dado pero ya era independiente, se escribirá una línea que empiece con la cadena "`YA ERA INDepend.: `", y **si no existía** ningún evento en la colección para un nombre igual al dado, se escribirá en el fichero de salida una línea con la cadena "`NO INDEPENDIZADO: `". En cualquiera de los tres casos, se seguirá escribiendo a continuación, y en la misma línea del fichero, el nombre dado para la instrucción.
- Por cada instrucción de tipo *"D"*, **si existían** en la colección eventos para los nombres iguales a los dados para la instrucción, se escribirá una línea que empiece con la cadena "`INTENTANDO hacer depend.: `", y **si no existía** ningún evento en la colección para alguno de los nombres dados para la instrucción, se escribirá en el fichero de salida una línea con la cadena "`IMPOSIBLE hacer depend.: `". En cualquiera de los dos casos, se seguirá escribiendo a continuación, y en la misma línea del fichero: el primer nombre dado para la instrucción, seguido de una cadena igual a "` -de-> `", seguido del segundo nombre dado para la instrucción.
- Por cada instrucción de tipo *"B"*, **si existía** algún evento en la colección para un nombre igual al dado y este es borrado de la colección, se escribirá una línea que empiece con la cadena "`BORRADO: `", y **si no existía** ningún evento en la colección para un nombre igual al dado o este no es borrado de la colección, se escribirá en el fichero una línea que empiece con la cadena "`NO BORRADO: `". En cualquiera de los dos casos, se seguirá escribiendo a continuación, y en la misma línea del fichero, el nombre dado para la instrucción.
- Por cada instrucción de tipo *"LD"*, se escribirá una línea que empiece con la cadena "`****DEPENDIENTES: `" seguida del nombre dado para la instrucción. **Si no existía** ningún evento en la colección con un nombre igual al dado, se escribirá en el fichero una línea que empiece con la cadena "`****DESCONOCIDO`" finalizando con ello el tratamiento de la instrucción *"LD"*.

Si existe en la colección un nombre igual al dado para la instrucción, a continuación, se escribirá en el fichero de salida: una línea con todos los datos relativos al nombre y su evento en la colección, de acuerdo a uno de los dos formatos que se describen a continuación, utilizando el que corresponda según sea independiente o dependiente:

```
[ nombre --- NumDepend ] --- descrip --- ( prioridad ) ****
[ nombre -de-> nomSup ;;; NumDepend ] --- descrip --- ( prioridad ) ****
```

siendo `nombre` el nombre dado para la instrucción, siendo `descrip` y `prioridad` respectivamente, la descripción y la prioridad del evento para dicho nombre en la colección, y siendo `NumDepend` el número de elementos de la colección directamente dependientes del que tiene el nombre `nombre`. En el caso de que en la colección este sea independiente, aparecerá con el formato descrito en la primera línea, pero si en la colección no es independiente, aparecerá con el formato de la segunda línea, siendo `nomSup` el nombre del elemento de la colección del que es directamente dependiente el que tiene el nombre `nombre` en la colección. A continuación, se escribirán en el fichero de salida una línea con los datos relativos a cada uno de los elementos que en la colección son directamente dependientes del que tiene el nombre dado para la instrucción (siguiendo el orden lexicográfico creciente, según el nombre de cada uno ellos), y para cada uno de ellos una línea de acuerdo a uno de los dos formatos que se describen a continuación, utilizando el que corresponda según sea independiente o dependiente:

```
[p -> nombreDep --- NumDependDp ] --- descrip --- ( prioridad ) ;;;;
[p -> nombreDp -de-> nomSup ;;; NumDependDp ] --- descrip --- ( prioridad ) ;;;;
```

siendo `p` el número que representa el puesto de dicho elemento en este listado (puesto 1 para el primer dependiente), siendo `nombreDep` el nombre del dependiente, siendo `descrip` y `prioridad` respectivamente la descripción y la prioridad su evento en la colección, y siendo `NumDependDp` el número de elementos de la colección directamente dependientes del que tiene el nombre `nombreDep`. En el caso de que en la colección este sea independiente, aparecerá con el formato descrito en la primera línea, pero si en la colección no es independiente, aparecerá con el formato de la segunda línea, siendo `nomSup` el nombre del elemento de la colección del que es directamente dependiente el que tiene el nombre `nombreDep` en la colección.
El listado se finalizará escribiendo en el fichero de salida una línea con una cadena con el formato:

```
****FINAL dependientes -de-> nombre
```

siendo `nombre` el dado para la instrucción.

- Por cada instrucción de tipo *"LT",* se escribirá una primera línea con la cadena "`-----LISTADO: `", seguida del número total de nombres registrados en la colección, y a continuación, por cada nombre almacenado en la colección, y siguiendo el orden (lexicográfico) creciente según el nombre, se escribirá otra línea con la información de todos los datos relativos al nombre y su evento en la colección, de acuerdo a uno de los dos formatos que se describen a continuación, utilizando el que corresponda según sea independiente o dependiente:

```
[ nombre --- NumDepend ] --- descrip --- ( prioridad )
[ nombre -de-> nomSup ;;; NumDepend ] --- descrip --- ( prioridad )
```

siendo `nombre, descrip` y `prioridad` respectivamente, el nombre, la descripción y la prioridad del evento, y siendo `NumDepend` el número de elementos de la colección directamente dependientes del que tiene el nombre `nombre`. En el caso de que en la colección este sea independiente, aparecerá con el formato descrito en la primera línea, pero si en la colección no es independiente, aparecerá con el formato de la segunda línea, siendo `nomSup` el nombre del elemento de la colección del que depende el que tiene el nombre `nombre`.
El listado finalizará con una línea con la cadena "`-----`".

Las implementaciones de las ordenes *"LT"* y *"LD"*, deberán utilizar obligatoriamente las operaciones del iterador de la implementación del TAD *colecciónInterdep*.

Los números que representan la prioridad de un evento, se escribirán siempre en el fichero de salida en el formato numérico habitual: como una secuencia de dígitos decimales, y sin que aparezcan caracteres '+', '-', '.', ni espacios o separadores entre los dígitos.

Los ficheros que se leen y se escriben deberán respetar escrupulosamente los formatos que se describen en el enunciado, o la práctica no será evaluada. Al final de este documento se muestran los formatos de los ficheros con un ejemplo de fichero "entrada.txt", y su correspondiente fichero "salida.txt".

Detalles sobre la implementación con C++:

- Ejemplo de un esquema básico, de lectura de flujo y de línea a línea, que puede servir para leer el fichero de instrucciones:

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

Propuesta de reparto de trabajo entre las dos sesiones y semanas de la práctica 3:

Recuerda que las prácticas de EDA tienen planificado tiempo que hay que dedicar a trabajar en ellas antes, durante y después de la sesión de prácticas presencial. Desde la publicación del enunciado hasta la fecha límite de entrega de la práctica, las dos sesiones de prácticas se distribuyen, para todos los grupos de laboratorio, en un periodo de algo más de cuatro semanas lectivas. Aquí se sugiere una posible división del trabajo entre semanas, para tener una guía de aproximadamente qué parte de la práctica deberías tener hecha y totalmente acabada, antes de la segunda mitad de esta práctica.

- **En las dos primeras semanas:** Implementar y probar completamente el TAD *evento*. Del TAD *colecciónInterdep*, tener en cuenta todas las operaciones para elegir la representación en memoria, pero de momento únicamente implementar y probar el correcto funcionamiento de las operaciones: *crear*, *tamaño, esVacia?, añadirIndependiente, existe, existeIndependiente, existeDependiente,* y todas las operaciones del *iterador*. Del programa de la tarea 3, implementar solo lo necesario para el correcto funcionamiento de las instrucciones *"A"* (pero probarla de momento sólo cuando se trate de datos que serán independientes), *"E"* y *"LT"*.

- **Semanas tercera y cuarta:** Implementar todas las demás operaciones del TAD *colecciónInterdep*, y probar a fondo el correcto funcionamiento de todas las operaciones del TAD. Del programa de la tarea 3, terminar su implementación y probar a fondo el correcto funcionamiento de todas las instrucciones del programa. Asegurarse de cumplir escrupulosamente con todos los formatos indicados para los ficheros de "entrada.txt" y "salida.txt".

Observaciones.

- **El código fuente de las prácticas evaluables será compilado y probado en *lab000.unizar.es*, que es donde deberá compilar y funcionar correctamente.** Es recomendable que te asegures de que se podrá compilar y ejecutar correctamente en *lab000* dedicando a ello el tiempo y las pruebas suficientes.
- **El código deberá compilar correctamente con la opción** `-std=c++11` **activada**.
  - Esto significa que, si se trabaja con la línea de comandos, deberá compilarse con:

    ```
    g++ -std=c++11  *.cpp
    ```

- **No se impondrá ninguna ruta concreta para los ficheros de código fuente a compilar, ni para los ficheros que leerá/escribirá el programa**. De esta forma, el fichero de entrada será siempre **"entrada.txt"** (no "entrada2.txt", "datos/entrada.txt" o similares), y el fichero de salida se llamará **"salida.txt"**, no "salida2.txt", "misalida.txt", etc.
- Todos los ficheros con código fuente deberán estar **correctamente documentados**.
- En el **comentario inicial de cada fichero** de código fuente se añadirán los **nombres completos y NIP[1] de las personas autoras de dicho código fuente.**
- La salida debe cumplir escrupulosamente las especificaciones del enunciado. Por ejemplo, cuando se indica que en el fichero de salida se escribirá "`NO INTRODUCIDO: `", está escrito en mayúsculas, con un espacio en blanco tras ':', y lo mismo para el resto de instrucciones. **Es obligatorio cumplir todos los formatos descritos en este enunciado, o la práctica no será evaluada**.
- **Los TAD deberán implementarse siguiendo las instrucciones dadas en las clases y prácticas de la asignatura, no se permite utilizar Programación Orientada a Objetos, y tampoco el uso o lanzamiento de excepciones.**
- **No se permite usar** las clases o componentes de la Standard Template Library (STL), ni otras bibliotecas similares, y tampoco se permite utilizar *Smart Pointers*. Tampoco se permite utilizar código ajeno, es decir, no implementado **integra y personalmente** por las personas integrantes del equipo de prácticas.

Entrega de la práctica. Instrucciones.

- **La práctica solo deberá entregarla uno de los miembros del equipo de prácticas**.
- La práctica deberá ser entregada en la tarea correspondiente en el curso Moodle, y antes del plazo límite establecido. **No se aceptarán entregas fuera de plazo, ni entregas por ninguna otra vía que no sea la entrega en la tarea Moodle para esta práctica.**
- Crea un fichero comprimido en formato `Zip` y de nombre `practica3_NIP1_NIP2.zip`, que contenga todos los ficheros de código fuente desarrollados para resolver la práctica, y dos ficheros de texto ***entrada.txt*** y ***salida.txt***, con los formatos descritos en el enunciado, pero que sean significativamente diferentes a los proporcionados como ejemplo, y con los que habréis probado la implementación realizada en vuestra práctica. Los ficheros no deben estar distribuidos en subdirectorios, y no debe entregarse ningún fichero de código objeto, ni ejecutables, etc.
- El nombre utilizado para el fichero comprimido: `practica3_NIP1_NIP2.zip`, debe contener los NIP de ambos integrantes del equipo, siendo `NIP1<NIP2`. Si el trabajo se ha hecho de forma individual el nombre del fichero deberá ser: `practica3_NIP1.zip` , siendo `NIP1` el NIP de la persona que ha realizado la práctica.
- A la hora de evaluar la práctica se utilizará tanto el fichero de prueba *entrada.txt* y *salida.txt* que se entreguen, como ficheros de prueba entregados por otros compañeros, o ficheros propios de los profesores.

[1] El NIP, también llamado NIA; es el número de 6 dígitos que la Universidad de Zaragoza asigna a cada estudiante para su identificación.

entrada.txt

```
LT
C
apertura_2025_10_6
apertura del edificio 2025-10-7
1
A
cierre_2025_10_6
Cierre del edificio
2
INDependiente
-.-.-.-.-
D
cierre_2025_10_6
cierre_2025_10_6
A
ON_luces Z008
Encendidas luces008, deteccion sensor presencia
2
DEPendiente
presencia_S008
LD
presencia_S008
LT
A
Validacion_P001_exterior
Validacion acceso en puerta P001, desde exterior
3
INDependiente
-.-.-.-.-
A
apertura_P001
Abierta puerta P001
3
DEPendiente
Validacion_P001_exterior
A
presencia_S008
Deteccion presencia, sensor S008
3
INDependiente
-.-.-.-.-
A
ON_luces Z008
Encendidas luces008, deteccion sensor presencia
4
DEPendiente
presencia_S008
A
OFF_luces Z008
Apagadas luces zona 008, por expiracion tiempo
2
DEPendiente
ON_luces Z008
LT
O
presencia_S008
O
OFF_luces Z008
O
apertura_2025_10_7
E
presencia_S008
E
OFF_luces Z008
E
apertura_2025_10_7
LD
Validacion_P001_exterior
C
apertura_P001
Abierta puerta P001, previa validacion
2
```

(continuación de) entrada.txt

```
O
apertura_P001
C
presencia_S008
presencia, sensor S008
4
LD
presencia_S008
LT
D
OFF_luces Z008
presencia_S008
LD
presencia_S008
I
Validacion_P001_exterior
I
apertura_2025_10_6
I
apertura_P001
LD
Validacion_P001_exterior
C
apertura_P001
Abierta puerta P001 sin validacion acceso
5
D
presencia_S008
apertura_P001
D
cierre_2025_10_6
apertura_2025_10_6
LT
B
apertura_2025_10_7
B
OFF_luces Z008
B
presencia_S008
LD
presencia_S008
LD
apertura_P001
LT
```

salida.txt

```
-----LISTADO: 0
-----
NO CAMBIADO: apertura_2025_10_6
INTRODUCIDO: [ cierre_2025_10_6 ] --- Cierre del edificio --- ( 2 )
INTENTANDO hacer depend.: cierre_2025_10_6 -de-> cierre_2025_10_6
NO INTRODUCIDO: [ ON_luces Z008 -de-> presencia_S008 ] --- Encendidas luces008, deteccion sensor presencia --- ( 2 )
****DEPENDIENTES: presencia_S008
****DESCONOCIDO
-----LISTADO: 1
[ cierre_2025_10_6 --- 0 ] --- Cierre del edificio --- ( 2 )
-----
INTRODUCIDO: [ Validacion_P001_exterior ] --- Validacion acceso en puerta P001, desde exterior --- ( 3 )
INTRODUCIDO: [ apertura_P001 -de-> Validacion_P001_exterior ] --- Abierta puerta P001  --- ( 3 )
INTRODUCIDO: [ presencia_S008 ] --- Deteccion presencia, sensor S008 --- ( 3 )
INTRODUCIDO: [ ON_luces Z008 -de-> presencia_S008 ] --- Encendidas luces008, deteccion sensor presencia --- ( 4 )
INTRODUCIDO: [ OFF_luces Z008 -de-> ON_luces Z008 ] --- Apagadas luces zona 008, por expiracion tiempo --- ( 2 )
-----LISTADO: 6
[ OFF_luces Z008 -de-> ON_luces Z008 ;;;  0 ] --- Apagadas luces zona 008, por expiracion tiempo --- ( 2 )
[ ON_luces Z008 -de-> presencia_S008 ;;;  1 ] --- Encendidas luces008, deteccion sensor presencia --- ( 4 )
[ Validacion_P001_exterior --- 1 ] --- Validacion acceso en puerta P001, desde exterior --- ( 3 )
[ apertura_P001 -de-> Validacion_P001_exterior ;;;  0 ] --- Abierta puerta P001  --- ( 3 )
[ cierre_2025_10_6 --- 0 ] --- Cierre del edificio --- ( 2 )
[ presencia_S008 --- 1 ] --- Deteccion presencia, sensor S008 --- ( 3 )
-----
LOCALIZADO: [ presencia_S008 --- 1 ] --- Deteccion presencia, sensor S008 --- ( 3 )
LOCALIZADO: [ OFF_luces Z008 -de-> ON_luces Z008 ;;; 0 ] --- Apagadas luces zona 008, por expiracion tiempo --- ( 2 )
NO LOCALIZADO: apertura_2025_10_7
INDEPendiente: presencia_S008
DEPendiente: OFF_luces Z008
DESCONOCIDO: apertura_2025_10_7
****DEPENDIENTES: Validacion_P001_exterior
[ Validacion_P001_exterior --- 1 ] --- Validacion acceso en puerta P001, desde exterior --- ( 3 ) ****
[1 -> apertura_P001 -de-> Validacion_P001_exterior ;;;  0 ] --- Abierta puerta P001  --- ( 3 ) ;;;;
****FINAL dependientes -de-> Validacion_P001_exterior
CAMBIADO: [ apertura_P001 -de-> Validacion_P001_exterior ;;; 0 ] --- Abierta puerta P001, previa validacion --- ( 2 )
LOCALIZADO: [ apertura_P001 -de-> Validacion_P001_exterior ;;; 0 ] --- Abierta puerta P001, previa validacion --- ( 2 )
CAMBIADO: [ presencia_S008 --- 1 ] --- presencia, sensor S008 --- ( 4 )
****DEPENDIENTES: presencia_S008
[ presencia_S008 --- 1 ] --- presencia, sensor S008 --- ( 4 ) ****
[1 -> ON_luces Z008 -de-> presencia_S008 ;;;  1 ] --- Encendidas luces008, deteccion sensor presencia --- ( 4 ) ;;;;
****FINAL dependientes -de-> presencia_S008
-----LISTADO: 6
[ OFF_luces Z008 -de-> ON_luces Z008 ;;;  0 ] --- Apagadas luces zona 008, por expiracion tiempo --- ( 2 )
[ ON_luces Z008 -de-> presencia_S008 ;;;  1 ] --- Encendidas luces008, deteccion sensor presencia --- ( 4 )
[ Validacion_P001_exterior --- 1 ] --- Validacion acceso en puerta P001, desde exterior --- ( 3 )
[ apertura_P001 -de-> Validacion_P001_exterior ;;;  0 ] --- Abierta puerta P001, previa validacion --- ( 2 )
[ cierre_2025_10_6 --- 0 ] --- Cierre del edificio --- ( 2 )
[ presencia_S008 --- 1 ] --- presencia, sensor S008 --- ( 4 )
-----
INTENTANDO hacer depend.: OFF_luces Z008 -de-> presencia_S008
****DEPENDIENTES: presencia_S008
[ presencia_S008 --- 2 ] --- presencia, sensor S008 --- ( 4 ) ****
[1 -> OFF_luces Z008 -de-> presencia_S008 ;;;  0 ] --- Apagadas luces zona 008, por expiracion tiempo --- ( 2 ) ;;;;
[2 -> ON_luces Z008 -de-> presencia_S008 ;;;  0 ] --- Encendidas luces008, deteccion sensor presencia --- ( 4 ) ;;;;
****FINAL dependientes -de-> presencia_S008
YA ERA INDepend.: Validacion_P001_exterior
NO INDEPENDIZADO: apertura_2025_10_6
INDEPENDIZADO: apertura_P001
****DEPENDIENTES: Validacion_P001_exterior
[ Validacion_P001_exterior --- 0 ] --- Validacion acceso en puerta P001, desde exterior --- ( 3 ) ****
****FINAL dependientes -de-> Validacion_P001_exterior
CAMBIADO: [ apertura_P001 --- 0 ] --- Abierta puerta P001 sin validacion acceso --- ( 5 )
INTENTANDO hacer depend.: presencia_S008 -de-> apertura_P001
IMPOSIBLE hacer depend.: cierre_2025_10_6 -de-> apertura_2025_10_6
-----LISTADO: 6
[ OFF_luces Z008 -de-> presencia_S008 ;;;  0 ] --- Apagadas luces zona 008, por expiracion tiempo --- ( 2 )
[ ON_luces Z008 -de-> presencia_S008 ;;;  0 ] --- Encendidas luces008, deteccion sensor presencia --- ( 4 )
[ Validacion_P001_exterior --- 0 ] --- Validacion acceso en puerta P001, desde exterior --- ( 3 )
[ apertura_P001 --- 1 ] --- Abierta puerta P001 sin validacion acceso --- ( 5 )
[ cierre_2025_10_6 --- 0 ] --- Cierre del edificio --- ( 2 )
[ presencia_S008 -de-> apertura_P001 ;;;  2 ] --- presencia, sensor S008 --- ( 4 )
-----
NO BORRADO: apertura_2025_10_7
BORRADO: OFF_luces Z008
NO BORRADO: presencia_S008
****DEPENDIENTES: presencia_S008
[ presencia_S008 -de-> apertura_P001 ;;; 1 ] --- presencia, sensor S008 --- ( 4 ) ****
[1 -> ON_luces Z008 -de-> presencia_S008 ;;;  0 ] --- Encendidas luces008, deteccion sensor presencia --- ( 4 ) ;;;;
****FINAL dependientes -de-> presencia_S008
****DEPENDIENTES: apertura_P001
[ apertura_P001 --- 1 ] --- Abierta puerta P001 sin validacion acceso --- ( 5 ) ****
[1 -> presencia_S008 -de-> apertura_P001 ;;;  1 ] --- presencia, sensor S008 --- ( 4 ) ;;;;
****FINAL dependientes -de-> apertura_P001
-----LISTADO: 5
[ ON_luces Z008 -de-> presencia_S008 ;;;  0 ] --- Encendidas luces008, deteccion sensor presencia --- ( 4 )
[ Validacion_P001_exterior --- 0 ] --- Validacion acceso en puerta P001, desde exterior --- ( 3 )
[ apertura_P001 --- 1 ] --- Abierta puerta P001 sin validacion acceso --- ( 5 )
[ cierre_2025_10_6 --- 0 ] --- Cierre del edificio --- ( 2 )
[ presencia_S008 -de-> apertura_P001 ;;;  1 ] --- presencia, sensor S008 --- ( 4 )
-----
```