Metodología de la programación

30 de marzo de 2020

ugr | Universidad de Granada

Práctica 3: punteros y gestión dinámica de memoria

Curso 2019-2020

Índice

Objetivos

El objetivo de esta práctica es trabajar con los conceptos vistos en el segundo tema de teoría. Para ello debéis implementar todos 5 ejercicios de los propuestos en esta relación. Para cada uno de ellos se ofrece un conjunto de pruebas concreto que el programa debe tratar de forma correcta para que se considere la evaluación de la práctica.

Se recuerda que el trabajo en estos ejercicios debe ser personal y que se recomienda la asistencia a clase de prácticas. La copia de código no aporta nada al aprendizaje y será considerada como un incumplimiento de las normas de la asignatura con las consecuencias que ello implica, según la normativa de la Universidad de Granada.

Para todos los ejercicios debes separar el código en módulos independientes: el que contiene la función (o funciones) se llamará **utilidades** y el que aporta el programa principal se llamará como el ejercicio que corresponda. Trabaja en un directorio (llamado como el ejercicio) en el que se creen los subdirectorios para archivos de cabecera, archivos de código objeto, ejecutable y código fuente. Proporciona un archivo **Makefile** que genere el ejecutable a partir del código fuente.

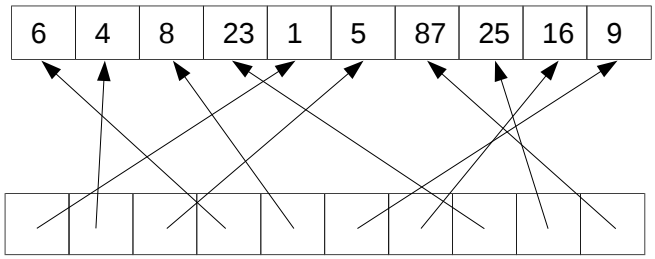
Para que quede más claro consideremos el caso del **ejercicio1**. Estará situado en un directorio llamado de la misma forma (**ejercicio1**) y que contendrá la siguiente estructura una vez esté completo y haya sido compilado y generado el ejecutable:

```
-- Makefile
-- bin
|   -- ejercicio1
-- include
|   -- utilidades.h
-- obj
|   -- ejercicio1.o
|   -- utilidades.o
-- src
|   -- ejercicio1.cpp
|   -- utilidades.cpp
```

Deben mantenerse estos nombres de carpetas para que el código pueda corregirse y compilarse de forma automática tras la entrega. También deben ajustarse los nombres de los archivos, funciones, clases,...., a las indicaciones dadas en los ejercicios. En general, se proporcionará el código básico para probar la funcionalidad pedida, con los casos de prueba indicados. En el ejemplo anterior se ofrecería la estructura básica del archivo **ejercicio1.cpp**.

Ejercicio 1

Se trata de escribir una función que reciba como entrada un vector de números, junto con su longitud, y devuelva un vector de punteros a sus elementos, de modo que los elementos apuntados por los punteros estén ordenados. Debe tenerse en cuenta que el vector de punteros será también un parámetro de la función (de forma que se ha reservado espacio para él con espacio suficiente fuera de la función). Gráficamente puede representarse de la siguiente forma:



Casos de prueba

Se proporciona un archivo de código con la función **main** que contempla 4 casos de prueba diferentes: array con valores generados de forma aleatoria, array ordenado, array invertido y array con todos los valores iguales. Para una ejecución cualquiera, el resultado debería ser similar al siguiente:

```
---------- Caso de prueba 1 ----------
-39  6  20  -30  -39  -29  8  18  23  -16
-39  -39  -30  -29  -16  6  8  18  20  23
--------------------------------------------------
---------- Caso de prueba 2 ----------
0  1  2  3  4  5  6  7  8  9
0  1  2  3  4  5  6  7  8  9
```

```
------------------------------ Caso de prueba 3 ------------------------------
9  8  7  6  5  4  3  2  1  0
0  1  2  3  4  5  6  7  8  9
------------------------------------------------------------------------------
------------------------------ Caso de prueba 4 ------------------------------
10  10  10  10  10  10  10  10  10  10
10  10  10  10  10  10  10  10  10  10
------------------------------------------------------------------------------
```

NOTA: como el primer caso de prueba se basa en la generación de números aleatorios, cada ejecución producirá un resultado diferente.

NOTA: conviene implementar métodos para mostrar el array original y también para mostrar por pantalla el array de punteros obtenido como resultado.

Ejercicio 2

Modifica el ejercicio 2 del guión anterior (sobre arrays, cadenas estilo **C** y matrices), de forma que todas las funciones reciban como argumento punteros y no se usen corchetes en el acceso a los valores de los arrays manejados. Considera la declaración más adecuada para cada función, haciendo uso de **const** para proteger punteros y valores cuando sea necesario.

El archivo **ejercicio2.cpp** puede mantenerse sin cambios respecto del ejercicio del guión previo. (Ejercicio de examen, Septiembre 2016)

Ejercicio 3

Implementar una función llamada **mezclarUnico** que devuelva un valor de tipo entero y reciba como argumento dos arrays de valores de tipo **double** (pasados como punteros) y los combine almacenando el resultado en un tercer array, pasado también como argumento. Debe considerarse que:

- los vectores de entrada no están ordenados. Debes de implementar la función ordenar.
- el vector de salida tendrá los elementos ordenados y sin repetidos.
- podemos asumir que el array de salida tiene capacidad suficiente. para almacenar todos los elementos que deba contener (se ha dimensionado de forma correcta antes de llamarla).
- el valor devuelto por la función indica el número de elementos contenido en el array de resultado.

Casos de prueba

La salida esperada de los casos de prueba incluidos en el **main** del ejercicio son los siguientes:

```
------------------------------ Caso de prueba 1 ------------------------------
1 2 10 15 20 25 30 35 40 45
0 1 2 3 4 5 6 9 10 21
```

```
Tam. resultante de mezclar array1 y array2: 17
0 1 2 3 4 5 6 9 10 15 20 21 25 30 35 40 45
--------------------------------------------------------------------------------
---------------------- Caso de prueba 2 ----------------------------------------
1
0 1 2 3 4 5 6 9 10 21

Tam. resultante de mezclar array1 y array2: 10
0 1 2 3 4 5 6 9 10 21
--------------------------------------------------------------------------------
---------------------- Caso de prueba 3 ----------------------------------------
0 1 2 3 4 5 6 9 10 21
21

Tam. resultante de mezclar array1 y array2: 10
0 1 2 3 4 5 6 9 10 21
--------------------------------------------------------------------------------
---------------------- Caso de prueba 4 ----------------------------------------
0 1 2 3
4 5 6 7

Tam. resultante de mezclar array1 y array2: 8
0 1 2 3 4 5 6 7
--------------------------------------------------------------------------------
---------------------- Caso de prueba 5 ----------------------------------------
4 5 6 7
0 1 2 3

Tam. resultante de mezclar array1 y array2: 8
0 1 2 3 4 5 6 7
--------------------------------------------------------------------------------
```

(Ejercicio de examen, Junio 2016)

Ejercicio 4

Dadas las siguientes definiciones de clase:

```
class Celda{
     private:
      double info;
      Celda *sig;
     public:
       ............
   };

   class Lista{
     private:
       Celda * contenido;
     .......
   };
```

donde **info** es una variable de tipo **double** y **sig** es un puntero que apunta a un objeto de tipo **Celda**. Con esta declaración un objeto de la clase **Lista** contiene básicamente

un puntero a un objeto de la clase **Celda**, que se corresponde con el primer valor que almacena. En base a estas clases se pide:

- método para mostrar el contenido de una celda (**mostrar**).
- método que muestra por pantalla el contenido de una lista (**mostrar**).
- método que devuelve la longitud de lista (**obtenerLongitud**).
- un método (**agregarFinal**) que permita añadir una nueva celda al final de la lista.
- un método que permita eliminar la última celda de una lista (**eliminarFinal**).
- método para insertar una nueva celda al principio de la lista (**agregarInicio**).
- método que obtenga un puntero a la celda que ocupa una posición determinada (**obtener**).
- método para insertar una celda en una posición concreta de la lista (**agregarPosicion**).
- método para liberar todo el espacio de memoria usado por la lista (**liberarEspacio**).

NOTA: pueden agregarse los métodos auxiliares que se consideren oportunos. Los métodos implementados deberían proporcionar una salida similar a la mostrada a continuación para las operaciones realizadas sobre las listas disponibles en la función **main**.

NOTA: este ejercicio debe realizarse usando gestión dinámica de memoria.

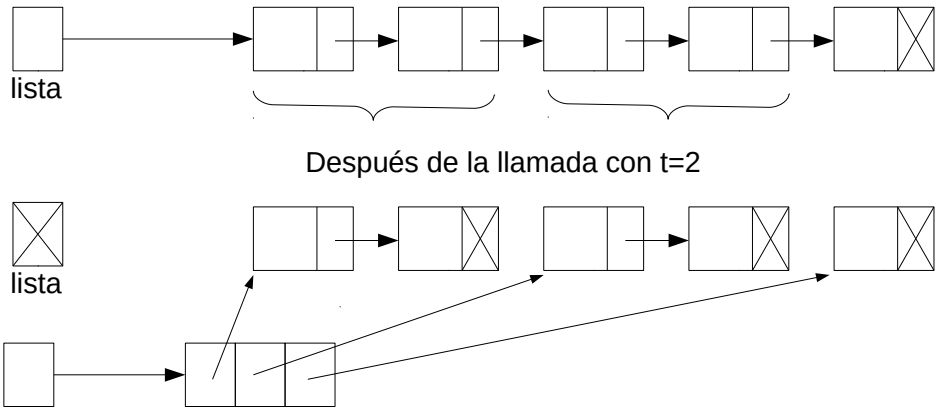
Casos de prueba

La salida obtenida al ejecutar el **main** proporcionado para este ejercicio es la siguiente:

```
Lista inicial:
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10

Lista con longitud 10

Lista tras liberar tres elementos del final:
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7

Lista con longitud 7

Lista tras insertar 0 al inicio:
0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7

Lista con longitud 8

Elemento en posicion 3 es 3.0

Elemento en primera posicion es 0.0

Elemento en ultima posicion es 7.0
```

Ejercicio 5

Se desea dividir una lista de celdas enlazadas en secuencias de listas de tamaño **t**. Escribe una función que reciba como argumento una lista de celdas enlazadas y devuelve un array de listas (una para cada secuencia) y el número de listas generadas. Por ejemplo, imaginemos la siguiente lista de 5 celdas siendo **t=2**, el resultado sería un array de 3 listas: las dos primeras de tamaño 2 y la última de tamaño 1,como se aprecia en la figura siguiente:



A considerar:

- la lista original queda vacía.
- no es necesario reservar ni liberar ninguna celda.
- la última lista podría quedar con menos de **t** casillas, como ocurre en el gráfico de arriba.
- es necesario reservar el vector para almacenar las sublistas obtenidas de la división.
- cada lista es una secuencia de celdas enlazadas que finaliza en un puntero nulo (puede utilizarse la misma estructura de clases del ejercicio 4).
- podemos asumir que la lista de partida no está vacía.

(Ejercicio de examen, Julio 2014)

Casos de prueba

Los casos de prueba a considerar son: lista de tamaño impar y división en listas de tamaño 2, lista de tamaño par y división con el mismo valor de **t** y lista de tamaño unidad:

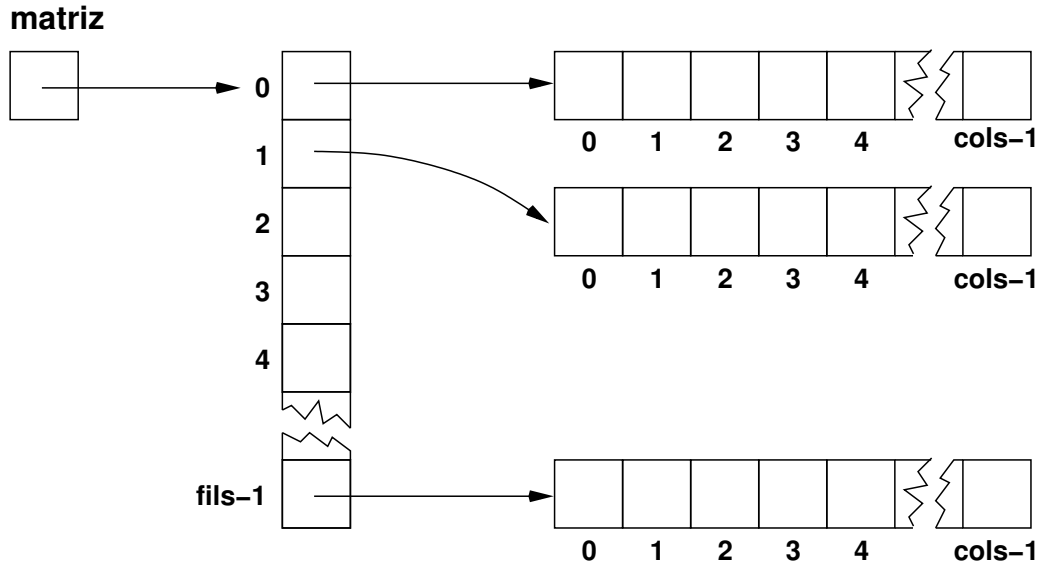
```
------------------- Caso de prueba 1 -------------------
5 4 3 2 1

Seccion 0: 5 4
```

```
-------------------------

Seccion 1: 3 2
-------------------------

Seccion 2: 1
-------------------------
---------------------- Caso de prueba 2 ----------------------
6 5 4 3 2 1

Seccion 0: 6 5
-------------------------

Seccion 1: 4 3
-------------------------

Seccion 2: 2 1
-------------------------
---------------------- Caso de prueba 3 ----------------------
1

Seccion 0: 1
-------------------------
```

NOTA: se pueden agregar métodos auxiliares, por ejemplo, para obtener la longitud de la lista, para agregar elementos al inicio y permitir la prueba, etc.

Ejercicio 6

Supongamos que para tratar con matrices bidimensionales con gestión dinámica de memoria usamos una estructura como la indica en la siguiente figura:



Se pide:

- definir cómo sería la clase para representar una matriz con esta estructura.
- añadir el constructor de la clase para crear una matriz con un determinado número de filas y de columnas.

- incluir en la clase un método que permita mostrar por pantalla el contenido de la matriz.
- implementar un método que haga una copia del objeto sobre el que se hace la llamada. La declaración del método será entonces:

```
Matri2D * copiarMatriz();
```

  Observad que la única forma posible de devolver un objeto creado en un modo (o función) es crearlo en memoria dinámica, lo que justifica el tipo de devolución indicado.

- implementar un método que extraiga una submatriz de una matriz. El método recibirá como argumento la fila y columna de inicio y final. La declarión es:

```
Matri2D * extraerSubmatriz(int filaInicio, int columnaInicio,
    int filaFin, int ColumnaFin);
```

- incorporar a la clase un método que elimina una fila dada (el resto de filas se desplazarán hacia arriba para que no haya huecos). El método no genera una nueva matriz; el cambio se hace en el objeto sobre el que se hace la llamada. Su declaración es:

```
void eliminarFila(int fila);
```

- agregar un método para eliminar una columna dada (las columnas siguientes se moverán a la derecha para no dejar huecos). Igual que en el apartado anterior, el cambio se hace sobre el objeto para el que se hace la llamada. Su declaración es:

```
void eliminarColumna(int columna);
```

Casos de prueba

Al ejecutar el código del programa principal del ejercicio debemos obtener la salida mostrada a continuación:

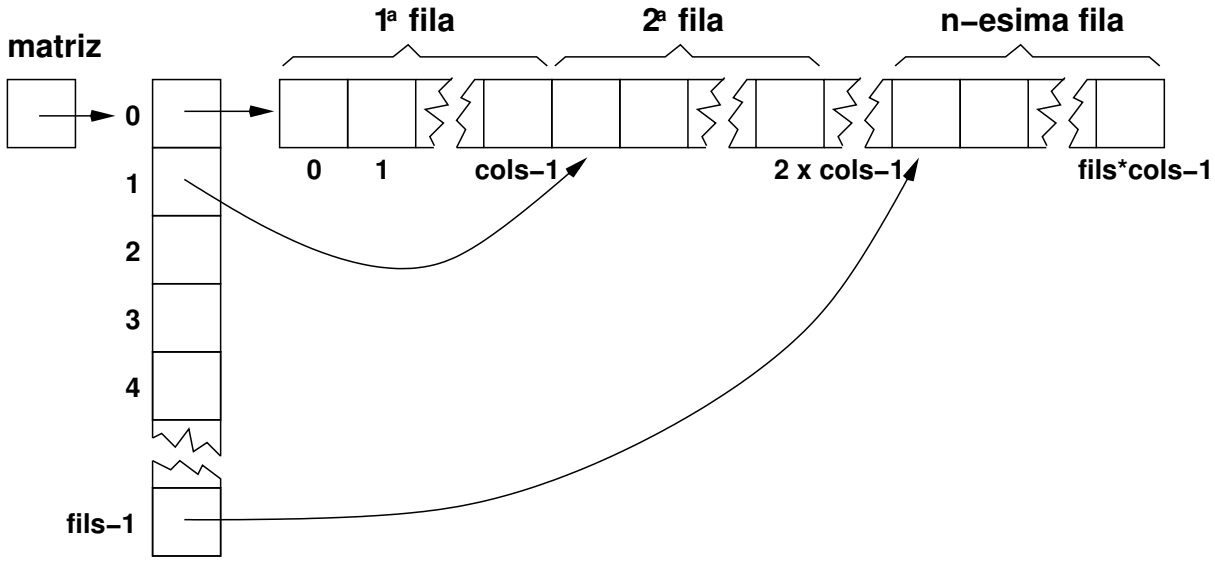
```
Matriz original:

1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
4 8 12 16 20
5 10 15 20 25

Matriz copiada a partir de la original:
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
4 8 12 16 20
5 10 15 20 25

Submatriz desde elemento (2,1) hasta (4,3)
```

```
6 9 12
8 12 16
10 15 20

Matriz obtenida al eliminar la primera fila:

2 4 6 8 10
3 6 9 12 15
4 8 12 16 20
5 10 15 20 25

Matriz obtenida al eliminar la columna 1 sobre la anterior:

2 6 8 10
3 9 12 15
4 12 16 20
5 15 20 25
```

Ejercicios opcionales

Ejercicio 7

Imaginemos que ahora cambiamos la forma de almacenar las matrices, de acuerdo a la siguiente estructura:



Usando esta representación, implementad todos los métodos pedidos en el ejercicio anterior. Si el nombre de la clase usada es el mismo que en el ejercicio anterior, el programa principal del mismo (**ejercicio6.cpp**) puede copiarse directamente para este ejercicio cambiando el nombre a **ejercicio7.cpp**. Observa qué métodos has tenido que cambiar y cuáles permanecen sin modificaciones.

Ejercicio 8

Reaprovechando el código empleado para el ejercicio 4, modifica la implememtación del método que muestra por pantalla el contenido de la lista (**mostrar**) para que funcione de forma recursiva. Implementa otro método análogo, llamado **mostraInverso** que muestre el contenido de la lista, pero en orden inverso y usando recursividad.

Entrega

La entrega se hará mediante la plataforma **PRADO**. Debéis entregar todo el código desarrollado para los ejercicios. Se recomienda organizar todo el código en un directorio (**guion3**) dentro del que figuran a su vez directorios para cada uno de los ejercicios entregados. A la plataforma se subirá un comprimido (archivo con extensión **zip**) con todo el contenido del directorio base (**guion3**).