Escuela de Ingeniería y Arquitectura - Depto. de Informática e Ingeniería de Sistemas
Zaragoza, 4 de septiembre de 2017 - Tiempo: **2 horas**

Examen práctico de Programación 2

Este examen práctico individual en laboratorio forma parte de la evaluación de la asignatura. Su calificación tiene un peso del 30 % en la calificación final de la asignatura en la convocatoria de septiembre y sustituye las calificaciones obtenidas en el trabajo obligatorio de junio (su peso era de un 15 %) y en el examen práctico de junio (su peso era de un 15 %).

En la carpeta **`examenSeptiembre`**, accesible desde la web de la asignatura, se encuentra el fichero **`examenSeptiembre.cc`** que puede facilitar este trabajo ya que contiene la especificación de la función pedida y una función principal para construir alrededor de ella algún programa de prueba.

Cómo presentar el trabajo

Se entregará un único fichero cuyas primeras líneas han de ser un **comentario con el nombre y apellidos del alumno** y, a continuación, debe figurar el código de la función **`distribuir`**`(M,FRONTERA)` precedido, en su caso, por el de sus funciones auxiliares.

La entrega se hará a través de la plataforma ***Moodle2*** (***`https://moodle2.unizar.es`***). Se recomienda no dejar el laboratorio hasta haber verificado que el comportamiento del código de la función diseñada es correcto.

Diseño sin bucles de la función `distribuir(M,FRONTERA)`

Se debe diseñar la función **`distribuir`**`(M,FRONTERA)` sin programar ningún bucle ni hacer uso de ninguna función predefinida. En caso de que el diseño se apoye en funciones auxiliares, estas deberán estar especificadas (de modo formal o no formal, pero siempre con rigor) y su código tampoco puede presentar bucles ni hacer uso de funciones predefinidas. Se valorará no solo que la función **`distribuir`**`(M,FRONTERA)` se comporte según su especificación [6.0 puntos] sino también la documentación, la legibilidad y la calidad del diseño realizado [4.0 puntos].

En el diseño no está permitido definir vectores o matrices auxiliares en los que copiar el contenido de la matriz **`M`**. Se sugiere desarrollar un algoritmo que sea una adaptación del algoritmo de distribución de los datos de un vector, estudiado en esta asignatura.

```
// Dimensión de las matrices ( definir de forma que N > 0)
const int N = ...;

/*
 * Dadas dos matrices cuadradas de la misma dimensión finita NxN, se dice que una es permutación
 * de la otra si almacenan los mismos datos y estos están repetidos el mismo número de veces en
 * cada una de ellas .
 *
 * Dada una matriz cuadrada M de dimensión NxN, definiremos la posición de cada elemento M[i][j]
 * mediante la siguiente función:
 *       posicion(M[i][j]) = i * N + j
 * Por ejemplo, al elemento M[0][0] le corresponde la primera posición (la posición 0),
 * al elemento M[0][1] le corresponde la segunda posición (la posición 1) y al
 * elemento M[N-1][N-1] le corresponde la última posición (la posición N^2-1)
 */

/*
 * Pre: M = A y los datos de tipo T admiten las seis operaciones binarias de relación
 * Post: Los elementos de M son una permutación de los elementos de A; la "posición" de
 *       cualquier elemento de M cuyo valor sea menor o igual o que FRONTERA será anterior
 *       a la de cualquier otro elemento de M cuyo valor sea mayor que FRONTERA.
 */
template <typename T>
void distribuir (T M[N][N], const T FRONTERA);
```

Una primera solución

```
// Dimensión de las matrices ( definir de forma que N > 0)
const int N = ...;

/*
 * Dadas dos matrices cuadradas de la misma dimensión finita NxN, se dice que una es permutación
 * de la otra si almacenan los mismos datos y estos están repetidos el mismo número de veces en
 * cada una de ellas .
 *
 * Dada una matriz cuadrada M de dimensión NxN, definiremos la posición de cada elemento M[i][j]
 * mediante la siguiente función:
 *       posicion(M[i][j]) = i * N + j
 * Por ejemplo, al elemento M[0][0] le corresponde la primera posición (la posición 0),
 * al elemento M[0][1] le corresponde la segunda posición (la posición 1) y al
 * elemento M[N-1][N-1] le corresponde la última posición (la posición N^2-1)
 */

/*
 * Pre: i >= 0 AND i <= N - 1 AND j >= 0 AND j <= N - 1
 * Post: Devuelve la posición del elemento M[i][j] de una matriz M de dimensión NxN
 *         definida del siguiente modo: posicion(M[i][j]) = i * N + j
 */
int posicion (const int i, const int j) {
    return i * N + j;
}

/*
 * Pre: fila = F AND columna = C AND F >= 0 AND F <= N - 1 AND
 *       C >= 0 AND C <= N-1 AND (F < N - 1 OR C < N - 1)
 * Post: (C < N - 1 -> fila = F AND columna = C + 1) AND
 *         (C = N - 1 -> fila = F + 1 AND columna = 0)
 */
void siguiente (int& fila, int& columna) {
    if (columna == N - 1) {
        fila = fila + 1;
        columna = 0;
    }
    else {
        columna = columna + 1;
    }
}

/*
 * Pre: fila = F AND columna = C AND F >= 0 AND F <= N - 1 AND
 *       C >= 0 AND C <= N - 1 AND (F < N - 1 OR C < N - 1)
 * Post: (C > 0 -> fila = F AND columna = C - 1) AND
 *         (C = 0 -> fila = F - 1 AND columna = N -1)
 */
void anterior (int& fila, int& columna) {
    if (columna == 0) {
        fila = fila - 1;
        columna = N - 1;
    }
    else {
        columna = columna - 1;
    }
}
```

```
/*
 * Pre: M = A y los datos de tipo T admiten las seis operaciones binarias de relación;
 *      el valor de cada uno de los elementos de M cuya "posición" sea anterior al
 *      elemento M[filaIni][columnaIni] es menor o igual a FRONTERA y el valor de
 *      cada uno de los elementos de M cuya "posición" sea posterior al elemento
 *      M[filaFin][columnaFin] es mayor que FRONTERA
 * Post: Los elementos de M son una permutación de los elementos de A; la "posición" de
 *       cualquier elemento de M cuyo valor sea menor o igual que FRONTERA será anterior
 *       a la de cualquier otro elemento de M cuyo valor sea mayor que FRONTERA.
 */
template <typename T>
void distribuir (T M[N][N], int filaIni , int columnaIni,
                 int filaFin , int columnaFin, const T FRONTERA) {
    if (posicion(filaIni ,columnaIni) < posicion(filaFin ,columnaFin)) {
        if (M[filaIni ][columnaIni] <= FRONTERA) {
            // El elemento M[filaIni ][columnaIni] está bien situado
            siguiente (filaIni , columnaIni);
        }
        else if (M[filaFin ][columnaFin] > FRONTERA) {
            // El elemento M[filaFin ][columnaFin] está bien situado
            anterior (filaFin ,columnaFin);
        }
        else {
            // Hay que permutar los elementos M[filaIni ][columnaIni]
            // y M[filaFin ][columnaFin]
            T aux = M[filaIni ][columnaIni];
            M[filaIni ][columnaIni] = M[filaFin ][columnaFin];
            M[filaFin ][columnaFin] = aux;
            siguiente (filaIni , columnaIni);
            anterior (filaFin ,columnaFin);
        }
        distribuir (M, filaIni , columnaIni, filaFin , columnaFin, FRONTERA);
    }
}

/*
 * Pre: M = A y los datos de tipo T admiten las seis operaciones binarias de relación
 * Post: Los elementos de M son una permutación de los elementos de A; la "posición" de
 *       cualquier elemento de M cuyo valor sea menor o igual que FRONTERA será anterior
 *       a la de cualquier otro elemento de M cuyo valor sea mayor que FRONTERA.
 */
template <typename T>
void distribuir (T M[N][N], const T FRONTERA) {
    distribuir (M, 0, 0, N - 1, N - 1, FRONTERA);
}
```

Una segunda solución

```
// Dimensión de las matrices ( definir de forma que N > 0)
const int N = ...;

/*
 * Dadas dos matrices cuadradas de la misma dimensión finita NxN, se dice que una es permutación
 * de la otra si almacenan los mismos datos y estos están repetidos el mismo número de veces en
 * cada una de ellas .
 *
 * Dada una matriz cuadrada M de dimensión NxN, definiremos la posición de cada elemento M[i][j]
 * mediante la siguiente función:
 *      posicion(M[i][j]) = i * N + j
 * Por ejemplo, al elemento M[0][0] le corresponde la primera posición (la posición 0),
 * al elemento M[0][1] le corresponde la segunda posición (la posición 1) y al
 * elemento M[N-1][N-1] le corresponde la última posición (la posición N^2-1)
 */

/*
 * Pre: i >= 0 AND i <= N - 1 AND j >= 0 AND j <= N - 1
 * Post: Devuelve la posición del elemento M[i][j] de una matriz M de dimensión NxN
 *         definida del siguiente modo:
 *            posicion(M[i][j]) = i * N + j
 */
int posicion (const int i, const int j) {
    return i * N + j;
}

/*
 * Pre: M = A y los datos de tipo T admiten las seis operaciones binarias de relación;
 *       el valor de cada uno de los elementos de M cuya "posición" sea anterior al
 *       elemento M[filaIni ][columnaIni] es menor o igual a FRONTERA y el valor de cada
 *       uno de los elementos de M cuya "posición" sea posterior al elemento
 *       M[filaFin ][columnaFin] es mayor que FRONTERA
 * Post: Los elementos de M son una permutación de los elementos de A; la "posición" de
 *         cualquier elemento de M cuyo valor sea menor o igual que FRONTERA será anterior
 *         a la de cualquier otro elemento de M cuyo valor sea mayor que FRONTERA.
 */
template <typename T>
void distribuir (T M[N][N], const int filaIni , const int columnaIni,
                 const int filaFin , const int columnaFin, const T FRONTERA) {
    if (posicion( filaIni ,columnaIni) < posicion( filaFin ,columnaFin)) {
        if (M[ filaIni ][columnaIni] <= FRONTERA) {
            // El elemento M[ filaIni ][columnaIni] está bien situado
            if (columnaIni < N - 1) {
                distribuir (M, filaIni , columnaIni + 1, filaFin , columnaFin, FRONTERA);
            }
            else {
                distribuir (M, filaIni + 1, 0, filaFin , columnaFin, FRONTERA);
            }
        }
        . . .
```

```
        . . .
        else if (M[filaFin][columnaFin] > FRONTERA) {
            // El elemento M[filaFin][columnaFin] está bien situado
            if (columnaFin > 0) {
                distribuir (M, filaIni , columnaIni, filaFin , columnaFin − 1, FRONTERA);
            }
            else {
                distribuir (M, filaIni , columnaIni, filaFin − 1, N − 1, FRONTERA);
            }
        }
        else {
            // Hay que permutar los elementos M[filaIni][columnaIni]
            // y M[filaFin][columnaFin]
            T aux = M[filaIni][columnaIni];
            M[filaIni][columnaIni] = M[filaFin][columnaFin];
            M[filaFin][columnaFin] = aux;
            if (columnaIni < N − 1) {
                if (columnaFin > 0) {
                    distribuir (M, filaIni , columnaIni + 1, filaFin , columnaFin − 1, FRONTERA);
                }
                else {
                    distribuir (M, filaIni , columnaIni + 1, N − 1, columnaFin − 1, FRONTERA);
                }
            }
            else {
                if (columnaFin > 0) {
                    distribuir (M, filaIni + 1, 0, filaFin , columnaFin − 1, FRONTERA);
                }
                else {
                    distribuir (M, filaIni + 1, 0, filaFin − 1, N − 1, FRONTERA);
                }
            }
        }
    }
}

/*
 * Pre: M = A y los datos de tipo T admiten las seis operaciones binarias de relación
 * Post: Los elementos de M son una permutación de los elementos de A; la "posición" de
 *       cualquier elemento de M cuyo valor sea menor o igual que FRONTERA será anterior
 *       a la de cualquier otro elemento de M cuyo valor sea mayor que FRONTERA.
 */
template <typename T>
void distribuir (T M[N][N], const T FRONTERA) {
    distribuir (M, 0, 0, N − 1, N − 1, FRONTERA);
}
```