Escuela de Ingeniería y Arquitectura - Depto. de Informática e Ingeniería de Sistemas
Zaragoza, 20 de junio de 2018 - Turno 1º - Tiempo hasta las **16:45**

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 15 % en la calificación final de la asignatura en la convocatoria de junio.

En la carpeta **`examenJunio`**, accesible desde la web de la asignatura, se encuentra el fichero **`examenJunioT1.cc`** para facilitar este trabajo ya que contiene la especificación de la función pedida y una función principal para construir alrededor de ella programas de pruebas.

Cómo presentar el trabajo

Como resultado del trabajo se entregará una versión del fichero **`examenJunioT1.cc`** con el siguiente contenido: en sus primeras líneas habrá un **comentario con el nombre y apellidos del alumno** y, a continuación, debe figurar el código de la función **`seleccion`**`(M)` precedido, en su caso, por el de sus funciones auxiliares.

La entrega se hará a través de la plataforma ***Moodle2*** (***`https://moodle2.unizar.es`***) antes de las 16:45. Se recomienda no dejar el laboratorio hasta haber verificado que el comportamiento del código de las funciones diseñadas es correcto.

Diseño sin bucles de la función `seleccion(M)`

El algoritmo de ordenación de vectores por selección ha sido estudiado en la asignatura. En este problema se trata de desarrollar un algoritmo genérico para la ordenación de los datos de una matriz cuadrada, la función **`seleccion`**`(M)`, que sea una adaptación fiel del algoritmo de ordenación por selección de los datos de un vector.

Se debe diseñar la función **`seleccion`**`(M)` sin programar ningún bucle. En caso de que el diseño se apoye en funciones auxiliares, estas deberán estar especificadas (de modo formal o no formal, aunque siempre con rigor) y su código tampoco puede presentar bucles.

En el diseño de **`seleccion`**`(M)` y de sus funciones auxiliares no está permitido definir nuevos vectores, matrices o ficheros auxiliares en los que copiar el contenido de la matriz **`M`**.

Se valorará que la función **`seleccion`**`(M)` se comporte según su especificación [6.0 puntos] y se valorará también la documentación, legibilidad y calidad del diseño realizado [4.0 puntos].

```
const int DIM = ... ;          // Definida de forma que DIM > 0

/*
 * Sea M una matriz cuadrada de dimensión DIMxDIM. Con relación a la ubicación de
 * sus DIM^2 elementos, diremos que su primer elemento es el M[0][0], su segundo
 * elemento es el M[0][1] y, así sucesivamente, su último elemento es el M[DIM-1][DIM-1].
 * De esta forma, cualquier elemento de la fila i-ésima precede (es anterior) a cualquier
 * elemento de la fima (i+1)-ésima y, dentro de una misma fila, el elemento j-ésimo de
 * la fila precede (es anterior) al elemento (j+1)-ésimo de la misma fila.
 *
 * Sea M una matriz cuadrada de dimensión DIMxDIM. Diremos que la matriz M está
 * ordenada de menor a mayor valor si el valor de un elemento que precede a otro
 * es siempre igual o menor que el de éste.
 *
 * Sean A y M dos matrices cuadradas de dimensión finita DIMxDIM. Diremos que ambas
 * matrices almacenan una permutación de los datos de la otra y viceversa si, para
 * cualquier elemento de la matriz A, el número de veces que está repetido este
 * elemento en A es igual al número de veces que está repetido en M.
 */

/*
 * Pre: Sea M una matriz de dimensión DIMxDIM y sea M = A, es decir,
 *      (PT alfa EN [0,DIM-1]. (PT beta EN [0,DIM-1]. M[alfa][beta] = A[alfa][beta]) )
 * Post: Los elementos de M son una permutación de los elementos de A y
 *       los elementos comprendidos entre el M[0][0] y el M[DIM-1][DIM-1]
 *       están ordenados de menor a mayor valor
 */
template <typename T>
void seleccion (T M[DIM][DIM]) {
    // Ordena los elementos de la matriz M aplicando el algoritmo de ordenación
    // por selección
    . . .
}
```

Escuela de Ingeniería y Arquitectura - Depto. de Informática e Ingeniería de Sistemas
Zaragoza, 20 de junio de 2018 - Turno 2º - Tiempo hasta las **18:45**

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 15 % en la calificación final de la asignatura en la convocatoria de junio.

En la carpeta **`examenJunio`**, accesible desde la web de la asignatura, se encuentra el fichero **`examenJunioT2.cc`** que puede facilitar este trabajo ya que contiene la especificación de la función pedida y una función principal para construir alrededor de ella programas de pruebas.

Cómo presentar el trabajo

Como resultado del trabajo se entregará una versión del fichero **`examenJunioT2.cc`** con el siguiente contenido: en sus primeras líneas habrá un **comentario con el nombre y apellidos del alumno** y, a continuación, debe figurar el código de la función **`intercambio`**`(M)` precedido, en su caso, por el de sus funciones auxiliares.

La entrega se hará a través de la plataforma ***Moodle2*** (***`https://moodle2.unizar.es`***) antes de las 18:45. Se recomienda no dejar el laboratorio hasta haber verificado que el comportamiento del código de las funciones diseñadas es correcto.

Diseño sin bucles de la función `intercambio(M)`

El algoritmo de ordenación de vectores por intercambio ha sido estudiado en la asignatura. En este problema se trata de desarrollar un algoritmo genérico para la ordenación de los datos de una matriz cuadrada, la función **`intercambio`**`(M)`, que sea una adaptación fiel del algoritmo de ordenación por intercambio de los datos de un vector. El método de ordenación aplicado en este algoritmo se conoce también como método de la burbuja.

Se debe diseñar la función **`intercambio`**`(M)` sin programar ningún bucle. En caso de que el diseño se apoye en funciones auxiliares, estas deberán estar especificadas (de modo formal o no formal, aunque siempre con rigor) y su código tampoco puede presentar bucles.

En el diseño de **`intercambio`**`(M)` y de sus funciones auxiliares no está permitido definir nuevos vectores, matrices o ficheros auxiliares en los que copiar el contenido de la matriz **`M`**.

Se valorará que la función **`intercambio`**`(M)` se comporte según su especificación [6.0 puntos] y se valorará también la documentación, legibilidad y calidad del diseño realizado [4.0 puntos].

```
const int DIM = ... ;          // Definida de forma que DIM > 0

/*
 * Sea M una matriz cuadrada de dimensión DIMxDIM. Con relación a la ubicación de
 * sus DIM^2 elementos, diremos que su primer elemento es el M[0][0], su segundo
 * elemento es el M[0][1] y, así sucesivamente, su último elemento es el M[DIM-1][DIM-1].
 * De esta forma, cualquier elemento de la fila i-ésima precede (es anterior) a cualquier
 * elemento de la fima (i+1)-ésima y, dentro de una misma fila, el elemento j-ésimo de
 * la fila precede (es anterior) al elemento (j+1)-ésimo de la misma fila.
 *
 * Sea M una matriz cuadrada de dimensión DIMxDIM. Diremos que la matriz M está
 * ordenada de menor a mayor valor si el valor de un elemento que precede a otro
 * es siempre igual o menor que el de éste.
 *
 * Sean A y M dos matrices cuadradas de dimensión finita DIMxDIM. Diremos que ambas
 * matrices almacenan una permutación de los datos de la otra y viceversa si, para
 * cualquier elemento de la matriz A, el número de veces que está repetido este
 * elemento en A es igual al número de veces que está repetido en M.
 */

/*
 * Pre: Sea M una matriz de dimensión DIMxDIM y sea M = A, es decir,
 *      (PT alfa EN [0,DIM-1]. (PT beta EN [0,DIM-1]. M[alfa][beta] = A[alfa][beta]) )
 * Post: Los elementos de M son una permutación de los elementos de A y
 *       los elementos comprendidos entre el M[0][0] y el M[DIM-1][DIM-1]
 *       están ordenados de menor a mayor valor
 */
template <typename T>
void intercambio (T M[DIM][DIM]) {
    // Ordena los elementos de la matriz M aplicando un algoritmo de ordenación
    // por intercambio o algoritmo de la burbuja
    . . .
}
```

Una solución del diseño propuesto en el turno 1º

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

/*
 * Sea M una matriz cuadrada de dimensión DIMxDIM. Con relación a la ubicación de
 * sus DIM^2 elementos, diremos que su primer elemento es el M[0][0], su segundo
 * elemento es el M[0][1] y, así sucesivamente, su último elemento es el M[DIM-1][DIM-1].
 * De esta forma, cualquier elemento de la fila i-ésima precede (es anterior) a cualquier
 * elemento de la fima (i+1)-ésima y, dentro de una misma fila, el elemento j-ésimo de
 * la fila precede (es anterior) al elemento (j+1)-ésimo de la misma fila.
 *
 * Sea M una matriz cuadrada de dimensión DIMxDIM. Diremos que la matriz M está
 * ordenada de menor a mayor valor si el valor de un elemento que precede a otro
 * es siempre igual o menor que el de éste.
 *
 * Sean A y M dos matrices cuadradas de dimensión finita DIMxDIM. Diremos que ambas
 * matrices almacenan una permutación de los datos de la otra y viceversa si, para
 * cualquier elemento de la matriz A, el número de veces que está repetido este
 * elemento en A es igual al número de veces que está repetido en M.
 */

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

/*
 * Pre: (f,c) y (fMax,cMax) son dos índices de la matriz cuadrada M de dimensión
 *      DIMxDIM. El índice (f,c) es igual o posterior al índice (fMax,cMx) y
 *      sea DATO el valor del elemento M[fMax][cMax]
 * Post: Si algún elemento comprendido entre M[f][c] y M[DIM-1][DIM-1] es mayor
 *       que DATO entonces los valores finales de fMax y CMax corresponden al de
 *       los índices del elemento comprendido entre M[f][c] y M[DIM-1][DIM-1] con
 *       mayor valor; en caso contrario fMax y CMax conservan su valor inicial.
 */
template <typename T>
void buscarMaximo (const T M[DIM][DIM], int f, int c, int& fMax, int& cMax) {
    if (DIM * f + c < DIM * DIM - 1) {
        siguiente (f, c);
        if (M[f][c] < M[fMax][cMax]) {
            fMax = f;    cMax = c;
        }
        buscarMaximo(M, f, c, fMax, cMax);
    }
}
```

```
/*
 * Pre: M = A, siendo matrices cuadradas de dimensión DIMxDIM y los elementos
 *      comprendidos entre el M[0][0] y el que precede al M[f][c] están
 *      ordenados de menor a mayor valor y el valor de cualquiera de ellos es
 *      igual o inferior al de cualquiera de los elementos comprendidos entre
 *      el elemento M[f][c] y el elemento M[DIM-1][DIM-1]
 * Post: Los elementos de M son una permutación de los elementos de A y
 *       los elementos comprendidos entre el M[0][0] y el M[DIM-1][DIM-1]
 *       están ordenados de menor a mayor valor
 */
template <typename T>
void seleccion (T M[DIM][DIM], int f, int c) {
    if (DIM * f + c < DIM * DIM - 1) {
        int fMax = f, cMax = c;
        buscarMaximo(M, f, c, fMax, cMax);
        // Permuta los elementos M[f][c] y M[fMax][cMax]
        T aux = M[f][c];
        M[f][c] = M[fMax][cMax];
        M[fMax][cMax] = aux;
        // Determina los índices (f,c) del siguiente elemento de M a ordenar
        siguiente (f, c);
        // Reaplica el algoritmo de ordenación desde el elemento M[f,c]
        seleccion (M, f, c);
    }
}

/*
 * Pre: Sea M una matriz de dimensión DIMxDIM y sea M = A, es decir,
 *      (PT alfa EN [0,DIM-1]. (PT beta EN [0,DIM-1]. M[alfa][beta] = A[alfa][beta]) )
 * Post: Los elementos de M son una permutación de los elementos de A y
 *       los elementos comprendidos entre el M[0][0] y el M[DIM-1][DIM-1]
 *       están ordenados de menor a mayor valor
 */
template <typename T>
void seleccion (T M[DIM][DIM]) {
    // Ordena los elementos de la matriz M aplicando el algoritmo de ordenación
    // por selección
    seleccion (M, 0, 0);
}
```

Una solución del diseño propuesto en el turno 2º

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

/*
 * Sea M una matriz cuadrada de dimensión DIMxDIM. Con relación a la ubicación de
 * sus DIM^2 elementos, diremos que su primer elemento es el M[0][0], su segundo
 * elemento es el M[0][1] y, así sucesivamente, su último elemento es el M[DIM-1][DIM-1].
 * De esta forma, cualquier elemento de la fila i-ésima precede (es anterior) a cualquier
 * elemento de la fima (i+1)-ésima y, dentro de una misma fila, el elemento j-ésimo de
 * la fila precede (es anterior) al elemento (j+1)-ésimo de la misma fila.
 *
 * Sea M una matriz cuadrada de dimensión DIMxDIM. Diremos que la matriz M está
 * ordenada de menor a mayor valor si el valor de un elemento que precede a otro
 * es siempre igual o menor que el de éste.
 *
 * Sean A y M dos matrices cuadradas de dimensión finita DIMxDIM. Diremos que ambas
 * matrices almacenan una permutación de los datos de la otra y viceversa si, para
 * cualquier elemento de la matriz A, el número de veces que está repetido este
 * elemento en A es igual al número de veces que está repetido en M.
 */

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

```
/*
 * Pre: M = A, siendo matrices cuadradas de dimensión DIMxDIM.
 *      n es igual al número de elementos de M que ya están ordenados
 *      de menor a mayor valor: los n elementos con índices más altos.
 *      Los valores de los primeros DIM^2 − n elementos de M son menores
 *      o iguales que los n últimos
 * Post: Los elementos de M son una permutación de los elementos de A y
 *       los elementos comprendidos entre el M[0][0] y el M[DIM−1][DIM−1]
 *       están ordenados de menor a mayor valor
 */
template <typename T>
void intercambiar (T M[DIM][DIM], const int f, const int c, const int n) {
    if (DIM * f + c < DIM * DIM − n − 1) {
        int fSig = f, cSig = c;
        siguiente (fSig, cSig);
        if (M[f][c] > M[fSig][cSig]) {
            // Permuta los elementos M[f][c] y M[fSig][cSig]
            T aux = M[f][c];
            M[f][c] = M[fSig][cSig];
            M[fSig][cSig] = aux;
        }
        intercambiar (M, fSig, cSig, n);
    }
    else if (n < DIM * DIM) {
        intercambiar (M, 0, 0, n + 1);
    }
}

/*
 * Pre: Sea M una matriz de dimensión DIMxDIM y sea M = A, es decir,
 *      (PT alfa EN [0,DIM−1]. (PT beta EN [0,DIM−1]. M[alfa][beta] = A[alfa][beta]) )
 * Post: Los elementos de M son una permutación de los elementos de A y
 *       los elementos comprendidos entre el M[0][0] y el M[DIM−1][DIM−1]
 *       están ordenados de menor a mayor valor
 */
template <typename T>
void intercambio (T M[DIM][DIM]) {
    // Ordena los elementos de la matriz M aplicando el algoritmo de ordenación
    // por intercambio o algoritmo de la burbuja
    intercambiar (M, 0, 0, 0);
}
```