Escuela de Ingeniería y Arquitectura - Depto. de Informática e Ingeniería de Sistemas
Zaragoza, 15 de junio de 2017 - 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`** que puede facilitar este trabajo ya que contiene la especificación de las dos funciones pedidas y una función principal para construir alrededor de ella programas de pruebas.

Cómo presentar el trabajo

Como resultado de este examen práctico se entregará una versión del fichero **`examenJunioT1.cc`** cuyas primeras líneas sean un **comentario con el nombre y apellidos del alumno** y, a continuación, debe figurar el código de las funciones **`transponer`**`(M)` y **`primero`**`(sec)` 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.

1º. Diseño sin bucles de la función `transponer(M)` [5.0 puntos]

Se debe diseñar la función **`transponer`**`(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. Se valorará que la función **`transponer`**`(M)` se comporte según su especificación [3.0 puntos] y se valorará también la documentación, legibilidad y calidad del diseño realizado [2.0 puntos].

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

/*
 * 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: La matriz M es igual a la transpuesta de la matriz A, es decir,
 *       (PT alfa EN [0,DIM-1]. (PT beta EN [0,DIM-1]. M[alfa][beta] = A[beta][alfa]) )
 */
template <typename T>
void transponer (T M[DIM][DIM]);
```

2º. Diseño sin bucles de la función `primero(sec)` [5.0 puntos]

Se debe diseñar la función **`primero`**`(sec)` 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, aunque siempre con rigor) y su código tampoco puede presentar bucles ni hacer uso de funciones predefinidas. Se valorará que la función **`primero`**`(sec)` se comporte según su especificación [3.0 puntos] y se valorará también la documentación, legibilidad y calidad del diseño realizado [2.0 puntos].

```
/*
 * <secuencia> ::= { <otroCaracter> } { <natural> <otroCaracter> { <otroCaracter> } }
 *                    [ <natural> ] <nulo>
 * <natural> ::= <digito> { <digito> }
 * <digito> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
 * <nulo> ::= '\0'
 * <otroCaracter> ::= cualquier carácter que no sea ni un <dígito> ni <nulo>
 */

const char NULO = '\0';   // El carácter nulo que denota el final de una cadena de caracteres

/*
 * Pre: [sec] almacena una secuencia de caracteres conforme a la sintaxis de <secuencia>;
 *      dicha secuencia que puede contener cero, uno o más naturales
 * Post: Devuelve un valor negativo si [sec] no contiene ningún natural y
 *       si [sec] contiene uno o más naturales, entonces devuelve el valor del
 *       primero de ellos. Ejemplos:
 *       1. La invocación primero("") devuelve un valor negativo
 *       2. La invocación primero("uno, dos y tres") devuelve un valor negativo
 *       3. La invocación primero("197045") devuelve 197045
 *       4. La invocación primero("La suma de 99 y 1 es 100") devuelve 99
 *       5. La invocación primero("El 007 se llama James Bond") devuelve 7
 */
int primero (const char sec []);
```

Escuela de Ingeniería y Arquitectura - Depto. de Informática e Ingeniería de Sistemas
Zaragoza, 15 de junio de 2017 - 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 las dos funciones pedidas y una función principal para construir alrededor de ella programas de pruebas.

Cómo presentar el trabajo

Como resultado de este examen práctico se entregará una versión del fichero **`examenJunioT2.cc`** cuyas primeras líneas sean un **comentario con el nombre y apellidos del alumno** y, a continuación, debe figurar el código de las funciones **`espejo`**`(M)` y **`letrasPrimera`**`(sec)` 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.

1º. Diseño sin bucles de la función `espejo(M)` [5.0 puntos]

Se debe diseñar la función **`espejo`**`(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. Se valorará que la función **`espejo`**`(M)` se comporte según su especificación [3.0 puntos] y se valorará también la documentación, legibilidad y calidad del diseño realizado [2.0 puntos].

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

/*
 * 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: La matriz M es igual a la imagen especular de la matriz A respecto de su eje de
 *       simetría vertical (o sea, respecto de su columna o columnas centrales ), es decir,
 *       (PT alfa EN [0,DIM-1]. (PT beta EN [0,DIM-1]. M[alfa][beta] = A[alfa][DIM-1-beta]) )
 */
template <typename T>
void espejo (T M[DIM][DIM]);
```

2º. Diseño sin bucles de la función `letrasPrimera(sec)` [5.0 puntos]

Se debe diseñar la función **`letrasPrimera`**`(sec)` 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, aunque siempre con rigor) y su código tampoco puede presentar bucles ni hacer uso de funciones predefinidas. Se valorará que la función **`letrasPrimera`**`(sec)` se comporte según su especificación [3.0 puntos] y se valorará también la documentación, legibilidad y calidad del diseño realizado [2.0 puntos].

```
/*
 * <secuencia> ::= { <otroCaracter> } { <palabra> <otroCaracter> { <otroCaracter> } }
 *                 [ <palabra> ] <nulo>
 * <palabra> ::= <letra> { <letra> }
 * <letra> ::= 'a' | 'b' | 'c' | ... | 'y' | 'z' | 'A' | 'B' | 'C' | ... | 'Y' | 'Z'
 * <nulo> ::= '\0'
 * <otroCaracter> ::= cualquier carácter que no sea ni un <letra> ni <nulo>
 */

const char NULO = '\0';   // El carácter nulo que denota el final de una cadena de caracteres

/*
 * Pre: [sec] almacena una secuencia de caracteres conforme a la sintaxis de <secuencia>;
 *      dicha secuencia puede contener cero, una o más palabras
 * Post: Si [sec] no contiene ninguna palabra entonces devuelve un valor negativo y
 *       si [sec] contiene una o más palabras, entonces devuelve el número de letras
 *       de la primera de ellas. Ejemplos:
 *       1. La invocación letrasPrimera("") devuelve un valor negativo
 *       2. La invocación letrasPrimera("2 + 2 = 4") devuelve un valor negativo
 *       3. La invocación letrasPrimera("2 mas 2 suman 4") devuelve 3
 *       4. La invocación letrasPrimera(" Felicidad ") devuelve 9
 *       5. La invocación letrasPrimera("  Estamos en el 2017") devuelve 7
 */
int letrasPrimera (const char sec []);
```

Una solución de los diseños propuestos en el turno 1º

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

/*
 * Pre:  fila  >= 1 y fila  <= DIM y columna >= 0 AND columna <= fila AND
 *       (PT alfa EN [0, fila −1]. (PT beta EN [0, fila −1]. M[alfa][beta] = A[beta][ alfa ]) AND
 *                                 (PT beta EN [ fila +1,DIM−1]. M[alfa][beta] = A[alfa][beta ]) ) AND
 *       (PT beta EN [0,columna−1]. (M[fila ][beta ] = A[beta][ fila ]) AND
 *                                  (M[beta][ fila ] = A[ fila ][beta ]) ) AND
 *       (PT beta EN [columna,DIM−1]. (M[fila][beta] = A[ fila ][beta ]) AND
 *                                    (M[beta][ fila ] = A[beta][ fila ]) )
 *       (PT alfa EN [ fila +1,DIM−1]. (PT beta EN [0,DIM−1]. M[alfa][beta] = A[alfa][beta ]) )
 * Post: (PT alfa EN [0,DIM−1]. (PT beta EN [0,DIM−1]. M[alfa][beta] = A[beta][ alfa ]) )
 */
template <typename T>
void transponer (T M[DIM][DIM], const int fila , const int columna) {
    if ( fila < DIM) {
        if (columna != fila ) {
            // Permuta los elementos M[fila ][columna] y M[columna][fila]
            T aux = M[ fila ][ columna];
            M[ fila ][ columna] = M[columna][ fila ];
            M[columna][ fila ] = aux;
            // Permuta todos los elementos situados por debajo de la diagonal
            // principal de M a partir del M[fila ][columna+1] con sus simétricos
            // respecto de dicha diagonal
            transponer (M, fila , columna + 1);
        }
        else {
            // Permuta todos los elementos de las filas ( fila + 1) y posteriores ,
            // situados por debajo de la diagonal principal de M, con sus
            // simétricos respecto de dicha diagonal
            transponer (M, fila + 1, 0);
        }
    }
}

/*
 * 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: La matriz M es igual a la transpuesta de la matriz A, es decir,
 *       (PT alfa EN [0,DIM−1]. (PT beta EN [0,DIM−1]. M[alfa][beta] = A[beta][ alfa ]) )
 */
template <typename T>
void transponer (T M[DIM][DIM]) {
    // Permuta todos los elementos situados por debajo de la diagonal principal de M
    // a partir del elemento M[1][0] con sus simétricos respecto de dicha diagonal
    transponer (M, 1, 0);
}
```

```
const char NULO = '\0';   // El carácter nulo que denota el final de una cadena de caracteres

/*
 * Pre: [sec] almacena una secuencia de caracteres conforme a la sintaxis de <secuencia>.
 *      El valor de i ha de ser igual o mayor que cero y menor o igual al del índice de
 *      [sec] que almacena un valor NULO
 *      Que reconocido sea negativo significa que en la subsecuencia sec[0.. i-1] no hay
 *      ningún dígito. Que reconocido sea nulo o positivo significa sec[i-1] es un dígito
 *      integrante del primer natural de [sec] y que el valor de reconocido define el natural
 *      formado por el dígito sec[i-1] y los dígitos que le preceden.
 * Post: Devuelve un valor negativo si [sec] no contiene ningún natural y si [sec] contiene
 *       uno o más naturales, entonces devuelve el valor del primero de ellos.
 */
int primero (const char sec[], const int i, const int reconocido) {
    if (sec[i] != NULO) {
        if (sec[i] < '0' || sec[i] > '9') {
            // sec[i] no es el carácter NULO ni es un dígito
            if (reconocido >= 0) {
                // Se ha completado el reconocimiento del primer natural de [sec]
                return reconocido;
            }
            else {
                // Aún no se ha localizado ninguno de los dígitos del primer natural
                // de [sec]. Continúa la exploración a partir del carácter sec[i+1]
                return primero(sec, i + 1, reconocido);
            }
        }
        else {
            // sec[i] es un dígito perteneciente al primer natural de [sec]
            if (reconocido >= 0) {
                // sec[i] no es el primer dígito de la secuencia. Se actualiza el valor
                // del primer natural reconocido y continúa la exploración a partir del
                // carácter sec[i+1]
                return primero(sec, i + 1, 10 * reconocido + sec[i] -'0');
            }
            else {
                // sec[i] es el primer dígito del primer natural de la secuencia.
                // Se actualiza el valor del primer natural reconocido y continúa
                // la exploración a partir del carácter sec[i+1]
                return primero(sec, i + 1, sec[i] -'0');
            }
        }
    }
    else {
        // sec[i] es el carácter NULO; la exploración de la secuencia debe
        // concluir devolviendo el valor [reconocido]
        return reconocido;
    }
}
```

```
/*
 * <secuencia> ::= { <otroCaracter> } { <natural> <otroCaracter> { <otroCaracter> } }
 *                    [ <natural> ] <nulo>
 * <natural> ::= <digito> { <digito> }
 * <digito> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
 * <nulo> ::= '\0'
 * <otroCaracter> ::= cualquier carácter que no sea ni un <dígito> ni <nulo>
 */

/*
 * Pre: [sec] almacena una secuencia de caracteres conforme a la sintaxis de <secuencia>;
 *      dicha secuencia que puede contener cero, uno o más naturales
 * Post: Devuelve un valor negativo si [sec] no contiene ningún natural y
 *       si [sec] contiene uno o más naturales, entonces devuelve el valor del
 *       primero de ellos. Ejemplos:
 *       1. La invocación primero("") devuelve un valor negativo
 *       2. La invocación primero("uno, dos y tres") devuelve un valor negativo
 *       3. La invocación primero("197045") devuelve 197045
 *       4. La invocación primero("La suma de 99 y 1 es 100") devuelve 99
 *       5. La invocación primero("El 007 se llama James Bond") devuelve 7
 */
int primero (const char sec[]) {
    // Valor negativo que denota que no ha habido éxito por el momento
    // en la localización de naturales en la secuencia [sec]
    const int NO_LOCALIZADO = -1;
    // Se lanza la exploración de la secuencia [sec] a partir de sec[0]
    return primero (sec, 0, NO_LOCALIZADO);
}
```

Una solución de los diseños propuestos en el turno 2º

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

/*
 * Pre:  fila  >= 0 AND fila <= DIM AND columna >= 0 AND columna <= DIM/2 AND
 *       (PT alfa EN [0, fila −1]. (PT beta EN [0,DIM−1]. M[alfa][beta] = A[alfa][DIM−1−beta]) ) AND
 *       (PT beta EN [0,columna−1]. M[fila][beta] = A[fila][DIM−1−beta] ) AND
 *       (PT beta EN [columna,DIM−1−columna]. M[fila][beta] = A[fila][beta] ) AND
 *       (PT beta EN [DIM−columna,DIM−1]. M[fila][beta] = A[fila][DIM−1−beta] ) AND
 *       (PT alfa EN [fila +1,DIM−1]. (PT beta EN [0,DIM−1]. M[alfa][beta] = A[alfa][beta]) ) AND
 * Post: (PT alfa EN [0,DIM−1]. (PT beta EN [0,DIM−1]. M[alfa][beta] = A[alfa][DIM−1−beta]) )
 */
template <typename T>
void espejo (T M[DIM][DIM], const int fila , const int columna) {
    if ( fila < DIM) {
        if (columna < DIM / 2) {
            // Permuta el elemento M[fila][columna] con su simétrico respecto del eje
            // de simetría vertical , M[fila][DIM−1−columna]
            T aux = M[fila][columna];
            M[fila][columna] = M[fila][DIM−1−columna];
            M[fila][DIM−1−columna] = aux;
            // Permuta todos los elementos situados a la izquierda del eje de simetría vertical
            // de M, a partir del elemento M[fila][columna + 1], con sus simétricos respecto
            // de dicho eje
            espejo(M, fila , columna + 1);
        }
        else {
            // Permuta todos los elementos situados a la izquierda del eje de simetría vertical
            // de M, ubicados en las filas ( fila +1) y posteriores , con sus simétricos respecto
            // de dicho eje
            espejo(M, fila + 1, 0);
        }
    }
}

/*
 * 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: La matriz M es igual a la imagen especular de la matriz A respecto de su eje de
 *       simetría vertical (o sea, respecto de su columna o columnas centrales ), es decir,
 *       (PT alfa EN [0,DIM−1]. (PT beta EN [0,DIM−1]. M[alfa][beta] = A[alfa][DIM−1−beta]) )
 */
template <typename T>
void espejo (T M[DIM][DIM]) {
    // Permuta todos los elementos situados a la izquierda del eje de simetría vertical
    // de M, a partir del elemento M[0][0], con sus simétricos respecto de dicho eje
    espejo(M, 0, 0);
}
```

```
const char NULO = '\0';   // El carácter nulo que denota el final de una cadena de caracteres

/*
 * Pre: [sec] almacena una secuencia de caracteres conforme a la sintaxis de <secuencia>.
 *      El valor de i ha de ser igual o mayor que cero y menor o igual al del índice de
 *      [sec] que almacena un valor NULO
 *      Que el valor de [cuenta] sea negativo significa que en la subsecuencia sec[0.. i-1] no hay
 *      ninguna letra. Que el valor de [cuenta] sea positivo significa sec[i-1] es un carácter
 *      integrante de la primera palabra de [sec] y que el valor de [cuenta] es igual al número
 *      de letras de la palabra de la que forma parte sec[i-1] contadas desde su primera letra
 *      hasta la letra sec[i-1].
 * Post: Si [sec] no contiene ninguna palabra entonces devuelve un valor negativo y si [sec] contiene
 *       una o más palabras, entonces devuelve el número de letras de la primera de ellas.
 */
int letrasPrimera (const char sec[], const int i, const int cuenta) {
    if (sec[i] != NULO) {
        if (!(sec[i] >= 'a' && sec[i] <= 'z') && !(sec[i] >= 'A' && sec[i] <= 'Z')) {
            // sec[i] no es el carácter NULO ni es una letra
            if (cuenta > 0) {
                // Se ha completado el reconocimiento de la primera palabra de [sec]
                return cuenta;
            }
            else {
                // Aún no se ha localizado ninguna de las letras de la primera palabra
                // de [sec]. Continúa la exploración a partir del carácter sec[i+1]
                return letrasPrimera(sec, i + 1, cuenta);
            }
        }
        else {
            // sec[i] es una letra perteneciente a la primera palabra de [sec]
            if (cuenta > 0) {
                // sec[i] no es la primera letra de primera palabra de [sec].
                // Se actualiza la cuenta de letras de la primera palabra y
                // continúa la exploración de [sec] a partir de sec[i+1]
                return letrasPrimera(sec, i + 1, cuenta + 1 );
            }
            else {
                // sec[i] es la primera letra de la primera palabra de [sec].
                // Inicia la cuenta de letras y continúa la exploración de [sec]
                // a partir del carácter sec[i+1]
                return letrasPrimera(sec, i + 1, 1);
            }
        }
    }
    else {
        // sec[i] es el carácter NULO; la exploración de la secuencia debe
        // concluir devolviendo el valor de [cuenta]
        return cuenta;
    }
}
```

```
/*
 * <secuencia> ::= { <otroCaracter> } { <palabra> <otroCaracter> { <otroCaracter> } }
 *                    [ <palabra> ] <nulo>
 * <palabra> ::= <letra> { <letra> }
 * <letra> ::= 'a' | 'b' | 'c' | ... | 'y' | 'z' | 'A' | 'B' | 'C' | ... | 'Y' | 'Z'
 * <nulo> ::= '\0'
 * <otroCaracter> ::= cualquier carácter que no sea ni un <letra> ni <nulo>
 */

/*
 * Pre: [sec] almacena una secuencia de caracteres conforme a la sintaxis de <secuencia>;
 *      dicha secuencia puede contener cero, una o más palabras
 * Post: Si [sec] no contiene ninguna palabra entonces devuelve un valor negativo y
 *       si [sec] contiene una o más palabras, entonces devuelve el número de letras
 *       de la primera de ellas. Ejemplos:
 *       1. La invocación letrasPrimera("") devuelve un valor negativo
 *       2. La invocación letrasPrimera("2 + 2 = 4") devuelve un valor negativo
 *       3. La invocación letrasPrimera("2 mas 2 suman 4") devuelve 3
 *       4. La invocación letrasPrimera("Felicidad") devuelve 9
 *       5. La invocación letrasPrimera("  Estamos en el 2017") devuelve 7
 */
int letrasPrimera (const char sec []) {
    // Valor negativo que denota que no ha habido éxito por el momento
    // en la localización de una palabra en la secuencia [sec]
    const int NINGUNA = -1;
    // Se lanza la exploración de la secuencia [sec] a partir de sec[0]
    return letrasPrimera (sec, 0, NINGUNA);
}
```