Escuela de Ingeniería y Arquitectura - Depto. de Informática e Ingeniería de Sistemas
Zaragoza, 4 de septiembre de 2018 - 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 las funciones pedidas, una función principal **`main`**`(..)` y un par de funciones auxiliares adicionales para facilitar la realización de pruebas de las dos funciones pedidas.

Diseño sin bucles de un par de funciones

Se deben diseñar en C++ las funciones **`contarDivisores`**`(n)` [5.0 puntos] y **`primerNumeroProgresion`**`(n)` [5.0 puntos], que se especifican a continuación, sin programar ningún bucle.

```
/*
 * Nombre y apellidos del alumno: ... escribir aquí nombre y apellidos ...
 * Examen práctico de PROGRAMACIÓN 2 del 4 de septiembre de 2018
 */

/*
 * DEFINICIONES:
 *     Para simplificar la escritura de las especificaciones de las funciones que siguen
 *     se define el predicado esProgresion(n), que determina si un natural <n> es o no
 *     un número 'progresión', y la función entera numDivisores(n), que determina el número
 *     de divisores de un natural <n>:
 *           esProgresion(n) = (EX alfa EN [1,n]. n = (SIGMA beta EN [1,alfa]. beta))
 *           numDivisores(n) = (NUM alfa EN [1,n]. n % alfa = 0)
 */

/*
 *  Pre: n > 0
 *  Post: contarDivisores(n) = (NUM alfa EN [1,n]. n % alfa = 0)
 */
int contarDivisores (const int n);

/*
 * Pre: nDiv > 0
 * Post: primerNumeroProgresion(nDiv) = N AND
 *       esProgresion(N) AND numDivisores(N) > nDiv AND
 *       (PT alfa EN [1,N-1]. NOT esProgresion(alfa) OR numDivisores(alfa) <= nDiv)
 */
int primerNumeroProgresion (const int nDiv);
```

En caso de que el diseño se apoye en funciones auxiliares, éstas deberán estar especificadas (de modo formal o no formal, pero siempre con rigor) y su código tampoco puede presentar bucles. Se valorará no

solo que las funciones diseñadas se comporten según su especificación sino también la documentación, la legibilidad y, muy importante, la eficiencia de los algoritmos programados.

En el diseño de cualquiera de las dos funciones pedidas se podrá invocar cualquiera de las funciones desarrolladas en el ámbito de este trabajo.

También se podrá hacer uso, si se estima necesario, de las funciones predefinidas en la biblioteca estándar `<cmath>`, pero no se podrá hacer uso de funciones predefinidas en ninguna otra biblioteca estándar.

Para facilitar el análisis de la primera función aquí se presenta una tabla que muestra los divisores de algunos naturales.

NÚMERO NATURAL	DIVISORES	NÚMERO DE DIVISORES
1	1	1
6	1, 2, 3 y 6	4
10	1, 2, 5 y 10	4
25	1, 5 y 25	3
29	1 y 29	2
36	1, 2, 3, 4, 6, 9, 12, 18 y 36	9
64	1, 2, 4, 8, 16, 32 y 64	7
75	1, 3, 5, 15, 25 y 75	6
77	1, 7, 11 y 77	4
121	1, 11, 121	3

Diremos que un número es *'progresión'* si y solo si es igual a la suma de los **k** primeros naturales. Para facilitar el análisis de la segunda función aquí se presenta una tabla que muestra los primeros números *'progresión'* con un listado de sus divisores.

NÚMERO PROGRESIÓN	EQUIVALENCIA	DIVISORES	Nº DIVISORES
1	1 = 1	1	1
3	3 = 1 + 2	1 y 3	2
6	6 = 1 + 2 + 3	1, 2, 3 y 6	4
10	10 = 1 + 2 + 3 + 4	1, 2, 5 y 10	4
15	15 = 1 + 2 + 3 + 4 + 5	1, 3, 5 y 15	4
21	21 = 1 + 2 + 3 + 4 + 5 + 6	1, 3, 7 y 21	4
28	28 = 1 + 2 + 3 + 4 + 5 + 6 + 7	1, 2, 4, 7, 14 y 28	6
36	36 = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8	1, 2, 3, 4, 6, 9, 12, 18 y 36	9

Cómo presentar el trabajo

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

No es necesario que el fichero incluya cláusulas `#include` ni el código de las funciones utilizadas exclusivamente para la realización de pruebas.

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

Una solución

```
/*
 * DEFINICIONES:
 *    Para simplificar la escritura de las especificaciones de las funciones que siguen
 *    se define el predicado esProgresion(n), que determina si un natural <n> es o no
 *    un número 'progresión, y la función entera numDivisores(n), que determina el número
 *    de divisores de un natural <n>:
 *          esProgresion(n) = (EX alfa EN [1,n]. n = (SIGMA beta EN [1,alfa]. beta))
 *          numDivisores(n) = (NUM alfa EN [1,n]. n % alfa = 0)
 */

/*
 *  Pre: n > 0 AND desde > 0 AND hasta = int(sqrt(n)) AND
 *       cuenta = (NUM alfa EN [1,desde-1]. n % alfa = 0)
 *  Post: contarDivisores(n, desde, hasta, cuenta) = (NUM alfa EN [1,n]. n % alfa = 0)
 */
int contarDivisores (const int n, const int desde, const int hasta,
                     const int cuenta) {
    if (desde <= hasta) {
        if (n % desde == 0) {
            if (desde != n / desde) {
                // Contabiliza los divisores <desde> y <n>/<desde>
                return contarDivisores (n, desde + 1, hasta, cuenta + 2);
            }
            else {
                // Contabiliza el divisor <desde>, que es el último
                return cuenta + 1;
            }
        }
        else {
            // No contabiliza <desde> como divisor
            return contarDivisores (n, desde + 1, hasta, cuenta);
        }
    }
    else {
        // <cuenta> contabiliza todos los divisores de <n>
        return cuenta;
    }
}

/*
 *  Pre: n > 0
 *  Post: contarDivisores(n) = (NUM alfa EN [1,n]. n % alfa = 0)
 */
int contarDivisores (const int n) {
    return contarDivisores (n, 1, int(sqrt(n)), 0);
}
```

```
/*
 * Pre: nDiv > 0  AND n > 0  AND i > 0  AND
 *       esProgresion(n) AND n = (SIGMA alfa EN [1,i]. alfa )  AND
 *       (PT alfa EN [1, n−1]. NOT esProgresion(alfa ) OR numDivisores(alfa) <= nDiv)
 * Post: primerNumeroProgresion(nDiv,n,i ) = NUM AND
 *        esProgresion(NUM) AND numDivisores(NUM) > nDiv AND
 *        (PT alfa EN [1,NUM−1]. NOT esProgresion(alfa) OR numDivisores(alfa) <= nDiv)
 */
int primerNumeroProgresion (const int nDiv, const int n, const int i) {
    if ( contarDivisores (n) <= nDiv) {
        return primerNumeroProgresion(nDiv, n + i + 1, i + 1);
    }
    else {
        return n;
    }
}

/*
 * Pre: nDiv > 0
 * Post: primerNumeroProgresion(nDiv) = N AND
 *        esProgresion(N)  AND numDivisores(N) > nDiv  AND
 *        (PT alfa EN [1, N−1]. NOT esProgresion(alfa) OR numDivisores(alfa) <= nDiv)
 */
int primerNumeroProgresion (const int nDiv) {
    return primerNumeroProgresion(nDiv, 1, 1);
}
```