Examen Práctico de Programación 1 - 7/septiembre/2017

- Tiempo máximo para realizar el trabajo de programación propuesto: 120 minutos
- Entrega del trabajo a través de la plataforma *Moodle2*.

Especificación del trabajo a desarrollar

Se han de diseñar las funciones **`factorial`**`(n,fact)` [5.0 puntos] y **`primero`**`(nombreFichero,letra,nombre)` [5.0 puntos].

La función **`factorial`**`(n,fact)` debe permitir el cálculo de números factoriales, $1 \times 2 \times \cdots \times n$, que puedan llegar a tener hasta un máximo de **`NUM_MAX_DIGITOS`** dígitos. Algunos ejemplos de números factoriales:

```
5! = 120
10! = 3628800
20! = 2432902008176640000
35! = 10333147966386144929666651337523200000000
51! = 1551118753287382280224243016469303211063259720016986112000000000000
```

Se sugiere utilizar aquellas funciones desarrolladas en la práctica 4ª de la asignatura, en el módulo de biblioteca **`natGrandes`**, que puedan facilitar el diseño de la función **`factorial`**`(n,fact)`. En tal caso se deberá copiar en el fichero a entregar el código de aquellas funciones que vayan a ser utilizadas para lograr que todo el código necesario en este examen práctico esté agrupado en un único fichero de código, el fichero a entregar.

Por su parte, la función **`primero`**`(nombreFichero,letra,nombre)` debe buscar en un fichero que almacena una secuencia de nombres propios entrecomillados y separados por comas.

Es posible que se opte por un diseño descendente apoyado en alguna o algunas funciones auxiliares. En tal caso, en el documento presentado se incluirá también la especificación y el código de dichas funciones auxiliares. Un esquema del fichero `examenPractico.cc` con el código a entregar se presenta a continuación.

```
/*
 * Examen práctico de Programación 1
 * Fecha: jueves 7 de septiembre de 2017
 * Escriba aquí su nombre y apellidos : ...
 */

// Número máximo de dígitos de los números factoriales que va a
// calcular la función factorial (n, fact )
const int NUM_MAX_DIGITOS = 200;

// Aquí se incluirá , en su caso, una copia del código completo de las
// funciones convertir (n, sec ) y sumar(a,b, suma) (con sus funciones
// auxiliares si las hubiere ) definidas en la práctica 4ª en la biblioteca
// natGrandes que pueden ser utilizadas , si se desea, en el diseño de la
// función factorial (n, fact )

 . . .

// Aquí se incluirá , en su caso, el código de las funciones auxiliares
// que completen el diseño de las dos funciones pedidas

 . . .
```

```
// Finalmente, el código de las dos funciones pedidas

/*
 * Pre: n >= 0 y el número de dígitos de n! no excede el valor de la
 *      constante NUM_MAX_DIGITOS
 * Post: fact almacena la cadena de caracteres con los dígitos de n!.
 *       Ejemplos:
 *          Si n = 5 entonces fact = "120"
 *          Si n = 6 entonces fact = "720"
 *          Si n = 10 entonces fact = "3628800"
 *          Si n = 20 entonces fact = "2432902008176640000"
 */
void factorial (const int n, char fact []) {
    // ... escribir aquí el código de esta función ...
}

/*
 * Pre: <nombreFichero> define el nombre de un fichero que almacena una
 *      secuencia de nombres propios entrecomillados y separados por una coma:
 *      "MARY","PATRICIA","linda",...,"Darrel","BRODERICK","ALONSO". Los nombres
 *      propios almacenados en el fichero constan de una secuencia de letras .
 *      Cada una de estas letras puede ser mayúscula o minúscula.
 *      Ninguno de los nombres alcanza los 64 caracteres .
 *      El valor de <letra> es el de una letra mayúscula del alfabeto inglés .
 * Post: Devuelve un valor true si y solo si el fichero <nombreFichero> almacena
 *       al menos un nombre propio que comienza por <letra>, siendo indiferente
 *       que su primera letra sea minúscula o mayúscula; en tal caso <nombre>
 *       almacenará una cadena de caracteres con el primer nombre propio del fichero
 *       que comience por la letra <letra> (no importa que la letra inicial del nombre
 *       propio sea mayúscula o minúscula)
 */
bool primero (const char nombreFichero[], const char letra , char nombre[]) {
    // ... escribir aquí el código de esta función ...
}
```

En la carpeta **`examenSeptiembre`** accesible desde la web de la asignatura (`material docente común` / `código C++ y datos`) se ha dispuesto un fichero con un esquema del trabajo a entregar y una función principal para realizar algunas pruebas del comportamiento de las dos funciones pedidas.

Presentación del trabajo

Cada una de las dos funciones pedidas se valorará sobre 5.0 puntos. Se evaluará, en primer lugar, el correcto comportamiento de ambas funciones. En caso de que su comportamiento sea el esperado, se valorará la calidad de su diseño y la legibilidad del código, tanto de las funciones pedidas como, en su caso, de sus funciones auxiliares.

Cada alumno presentará un fichero, de nombre **`examenPractico.cc`**, a través de la plataforma **Moodle2** (https://moodle2.unizar.es/) antes de las 14:00. El contenido del fichero será el siguiente: 1) un comentario con nombre y apellidos, 2) una copia de la especificación y el código de las funciones del módulo **`natGrandes`** que hayan sido utilizadas en este trabajo, 3) la especificación y el código de las funciones auxiliares de las funciones pedidas, si las hubiera, y 4) la especificación y el código de las dos funciones pedidas.

Una solución del trabajo propuesto

```
// Número máximo de dígitos de los naturales con los que
// va a trabajar la función factorial (n, fact)
const int NUM_MAX_DIGITOS = 200;


/*
 * Pre: n >= 0 y el número de dígitos de n! no excede el valor de la
 *      constante NUM_MAX_DIGITOS
 * Post: fact almacena la cadena de caracteres con los
 *       dígitos de n!. Ejemplos:
 *         Si n = 5 entonces fact = "120"
 *         Si n = 6 entonces fact = "720"
 *         Si n = 10 entonces fact = "3628800"
 *         Si n = 20 entonces fact = "2432902008176640000"
 */
void factorial (const int n, char fact []) {
    char sumaParcial[NUM_MAX_DIGITOS], sumando[NUM_MAX_DIGITOS];
    convertir (1, fact );                                   // fact = 0!
    for (int i = 1; i <= n; ++i) {
        // fact = (i-1)!
        strcpy (sumaParcial, fact );                        // sumaParcial = (i-1)!
        strcpy (sumando, fact );                            // sumando = (i-1)!
        for (int veces = 2; veces <= i; ++veces) {
            // sumaParcial = (veces-1) x ((i-1)!) y sumando = (i-1)!
            sumar(sumaParcial, sumando, fact );             // fact = veces x ((i-1)!)
            strcpy (sumaParcial, fact );                    // sumaParcial = veces x ((i-1)!)
        }
        // fact = i x (i-1)! = i!
    }
    // fact = n!
}
```

```
/*
 * Pre: <nombreFichero> define el nombre de un fichero que almacena una
 *      secuencia de nombres propios entrecomillados y separados por una coma:
 *      "MARY","PATRICIA","linda",...,"Darrel","BRODERICK","ALONSO". Los nombres
 *      propios almacenados en el fichero constan de una secuencia de letras.
 *      Cada una de estas letras puede ser mayúscula o minúscula.
 *      Ninguno de los nombres alcanza los 64 caracteres.
 *      El valor de <letra> es el de una letra mayúscula del alfabeto inglés.
 * Post: Devuelve un valor true si y solo si el fichero <nombreFichero> almacena
 *       al menos un nombre propio que comienza por <letra>, siendo indiferente
 *       que su primera letra sea minúscula o mayúscula; en tal caso <nombre>
 *       almacenará una cadena de caracteres con el primer nombre propio del fichero
 *       que comience por la letra <letra> (no importa que la letra inicial del nombre
 *       propio sea mayúscula o minúscula)
 */
bool primero (const char nombreFichero[], const char letra, char nombre[]) {
    // Asocia el fichero <nombreFichero> al flujo de entrada f
    ifstream f (nombreFichero);
    if (f.is_open ()) {
        // Intenta leer el primer nombre pripio del fichero asociado a f
        f.ignore (10, '"'); f.get(nombre, 36, '"');
        // Provisionalmente no hay éxito en la búsqueda
        bool encontrado = false;
        while (!encontrado && !f.eof ()) {
            // Analiza si el último nombre propio leído comienza por <letra>
            char primeraLetra = nombre[0];
            if (primeraLetra >= 'a' && primeraLetra <= 'z') {
                primeraLetra = 'A' + primeraLetra - 'a';
            }
            if (primeraLetra == letra) {
                // El último nombre propio leído comienza por letra
                // La búsqueda ha terminado con éxito
                encontrado = true;
            }
            else {
                // Intenta leer un nuevo nombre propio del fichero asociado a f
                f.ignore (10, ','); f.ignore (10, '"');
                f.get(nombre, 36, '"');
            }
        }
        f.close ();
        // Devuelve el valor de <encontrado> que denota si la búsqueda
        // ha sido exitosa o no lo ha sido
        return encontrado;
    }
    else {
        // La búsqueda ha fracasado al no poder asociar el fichero a
        // un flujo de entrada para su lectura
        return false;
    }
}
```