Examen escrito de Programación 1. Viernes 27 de enero de 2017

- Se debe disponer de un documento de identificación con fotografía sobre la mesa.
- Se debe comenzar a resolver cada uno de los problemas del examen en una hoja de papel diferente, para facilitar su calificación por distintos profesores. Escribir en cada hoja de papel nombre y apellidos y utilizar, en su caso, ambas caras de la hoja.
- Tiempo máximo para realizar este examen: 3 horas

Problema 1º (1.5 puntos)

Se debe diseñar la función que se especifica a continuación, escribiendo su código C++. Se valorará la eficiencia del método algorítmico aplicado y la legibilidad de su código. En la solución presentada, se admite que la especificación de la función pedida sea un resumen de la especificación que se muestra a continuación.

```
/*
 * Pre: n >= 0
 * Post: Devuelve un entero cuyos dígitos son la siguiente permutación
 *       de los dígitos de n:
 *       - Los dígitos significativos de n, a partir de las decenas,
 *         se ubican en el valor devuelto una posición a la derecha.
 *       - El dígito de las unidades de n sustituye, en el valor devuelto,
 *         al dígito más significativo de n.
 *       Ejemplos:
 *         rotarDerecha(0) = 0              rotarDerecha(5) = 5
 *         rotarDerecha(10) = 1             rotarDerecha(19) = 91
 *         rotarDerecha(12000) = 1200       rotarDerecha(12003) = 31200
 *         rotarDerecha(123456) = 612345    rotarDerecha(123123) = 312312
 */
int rotarDerecha (int n);
```

Problema 2º (2.0 puntos)

Se debe diseñar la función que se especifica a continuación, escribiendo su código C++. Se valorará el planteamiento de un diseño descendente programando alguna o algunas funciones auxiliares, así como la eficiencia del método algorítmico aplicado y la legibilidad del código. Todas las funciones que integren la solución deben estar adecuadamente especificadas.

```
/*
 * Pre: n >= 3
 * Post: El valor de mayor es el mayor de los valores de suc[0,n-1],
 *       el valor de segundo es el segundo mayor de los valores de suc[0,n-1] y
 *       el valor de tercero es el tercer mayor de los valores de suc[0,n-1]
 *       Ejemplo: Si suc[0,9] almacena { 10, 17, 23, 12, 27, 23, 9, 12, 23 , 10 }
 *                tras ejecutar tresMayores(suc, 10, m, s, t) los valores de
 *                los tres últimos argumentos son: m = 27, s = 23 y t = 23
 */
void tresMayores (const int suc [], const int n, int& mayor, int& segundo, int& tercero );
```

Problema 3º (3.0 puntos)

Se debe diseñar la función que se especifica a continuación escribiendo su código C++. En este problema se valorará la eficiencia del método algorítmico aplicado y la legibilidad del código. La función pedida se presentará completa con su especificación y su código, así como, en su caso, las funciones auxiliares en las que pudiera apoyarse su diseño.

```
/*
 * Pre: n > 0
 * Post: La tabla primos[0,n-1] almacena los n primeros números
 *       primos ordenados de menor a mayor valor
 */
void calcularPrimos (const int n, int primos []);
```

Se recuerda que los números primos son: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, ...

Problema 4º (3.5 puntos)

Vamos a trabajar con secuencias de nombres de personas de longitud desconocida semejantes a la que se presenta a continuación: `Luis Abad`, `Carmen del Valle`, ..., `Manuel Espejo`.

Cada secuencia de nombres de personas se almacena en un fichero binario cuya estructura interna responde a la sintaxis definida por la regla <**ficheroNombres**>, complementada por las reglas que le siguen.

```
<ficheroNombres> ::= <nombre> { <nombre> }
<nombre> ::= <numCaracteres> { <caracter> }
<numCaracteres> ::= int
<caracter> ::= char
```

Cada nombre de persona se representa en el fichero mediante un dato entero de tipo **int**, que define su número de caracteres, seguido por la secuencia de dichos caracteres, representado cada uno de ellos por un dato de tipo **char**. Así sería la representación de la secuencia de nombres considerada anteriormente en un fichero con esta estructura (los datos del fichero de tipo **int** (9, 16, ..., 13) se han escrito aquí de forma alfanumérica para facilitar su comprensión):

```
<9> <L> <u> <i> <s> < > <A> <b> <a> <d> <16> <C> <a> <r> <m> ... <d> <e>
<l> < > <V> <a> <l> <l> <e> ... <13> <M> <a> <n> <u> ... <p> <e> <j> <o>
```

El objetivo de este problema es realizar un diseño descendente de la función **unNombre**(nombreFichero,letra,nombrePersona) haciendo uso, únicamente, de los recursos definidos a continuación. La estructura del fichero binario que almacena una secuencia de nombres de personas, al que se hace referencia en su especificación, responde a la sintaxis definida anteriormente mediante la la regla <**ficheroNombres**>.

```
#include <fstream>

using namespace std;

/*
 * Pre:  ...
 * Post:  ...
 */
char mayuscula (const char c);

/*
 * Pre:  ...
 * Post:  ...
 */
bool leer  ( ifstream & f, char nombrePersona[]);

/*
 * Pre: nombreFichero es el nombre de un fichero binario que almacena una secuencia
 *       de nombres de personas y el valor de [ letra ] corresponde al de una letra
 *       mayúscula o minúscula
 * Post: Devuelve cierto ( true ) si y solo si el fichero de nombre nombreFichero
 *        almacena algún nombre de persona cuya inicial sea igual a letra
 *        ( resulta indiferente si la inicial del nombre y letra son minúsculas o
 *        mayúsculas). En caso afirmativo , el parámetro nombrePersona almacena una
 *        cadena de caracteres con un nombre de persona de los almacenados en el
 *        fichero y cuya letra inicial sea igual a letra ( resultando indiferente
 *        si la inicial del nombre y letra son minúsculas o mayúsculas)
 */
bool unNombre (const char nombreFichero[], const char letra , char nombrePersona[]);
```

El diseño descendente de **unNombre**(nombreFichero,letra,nombrePersona) debe apoyarse lo más posible en las funciones auxiliares **mayuscula**(c) y **leer**(f,nombrePersona).

La función **mayuscula**(c) debe permitir transformar cualquier letra minúscula en la letra mayúscula correspondiente, devolviendo una copia de cualquier otro carácter que no sea una letra minúscula. Ejemplos: mayuscula('h') = 'H', mayuscula('H') = 'H', mayuscula('7') = '7' y mayuscula(';') = ';'.

El primer parámetro de la **leer**(f,nombrePersona) está asociado a un fichero de nombres de personas del que, en su caso, ya se han podido leer algunos nombres de personas. Al ser invocada la función debe intentar la lectura de un nuevo nombre de persona del fichero. Si lo logra, entonces asigna a su segundo parámetro una cadena de caracteres con el nombre de persona leído y devuelve un valor cierto (true); si no lo logra, se limita a devolver un valor falso (false).

En este problema se debe presentar el diseño de las dos funciones auxiliares, escribiendo su especificación (pre/post) y su código, y se debe completar el diseño de la función **unNombre**(...), escribiendo su cabecera y su código (no es necesario reproducir su especificación pre/post).

Una solución del problema 1º

```
/*
 * Pre: n >= 0
 * Post: Devuelve un entero cuyos dígitos son la siguiente permutación
 *       de los dígitos de n:
 *       - Los dígitos significativos de n, a partir de las decenas,
 *         se ubican en el valor devuelto una posición a la derecha.
 *       - El dígito de las unidades de n sustituye, en el valor devuelto,
 *         al dígito más significativo de n.
 *       Ejemplos:
 *         rotarDerecha(0) = 0              rotarDerecha(5) = 5
 *         rotarDerecha(10) = 1             rotarDerecha(19) = 91
 *         rotarDerecha(12000) = 1200       rotarDerecha(12003) = 31200
 *         rotarDerecha(123456) = 612345    rotarDerecha(123123) = 312312
 */
int rotarDerecha (int n) {
    // calcula el dígito de las unidades de n
    int unidades = n % 10;
    // desplaza a la derecha los restantes dígitos de n
    n = n / 10;
    // calcula la posición del dígito más significativa de n
    int pot10 = 1;
    while (pot10 <= n) {
        pot10 = 10 * pot10;
    }
    // devuelve el numero 'rotado a la derecha'
    return unidades * pot10 + n;
}
```

Una solución del problema 2º

```
/*
 * Pre: uno = X y otro = Y
 * Post: uno = Y y otro = X
 */
void intercambiar (int& uno, int& otro) {
    int aux;
    aux = uno; uno = otro; otro = aux;
}

/*
 * Pre: a = X, b = Y y c = Z
 * Post: El valor de a es el mayor de los tres valores (X,Y,Z),
 *       el valor de c es el menor de los tres valores (X,Y,Z) y
 *       el valor de c es el intermedio de los tres valores (X,Y,Z)
 */
void ordenar (int& a, int& b, int& c) {
    if (a < b) { intercambiar(a,b); }
    if (c > a ) { intercambiar(a,c); }
    if (c > b ) { intercambiar(b,c); }
}

/*
 * Pre: n >= 3
 * Post: El valor de mayor es el mayor de los valores de suc[0,n-1],
 *       el valor de segundo es el segundo mayor de los valores de suc[0,n-1] y
 *       el valor de tercero es el tercer mayor de los valores de suc[0,n-1]
 *       Ejemplo: Si suc[0,9] almacena { 10, 17, 23, 12, 27, 23, 9, 12, 23 , 10 }
 *                tras ejecutar tresMayores(suc, 10, m, s, t) los valores de
 *                los tres últimos argumentos son: m = 27, s = 23 y t = 23
 */
void tresMayores (const int suc[], const int n, int& mayor, int& segundo, int& tercero) {
    mayor = suc[0];   segundo = suc[1];   tercero = suc[2];
    ordenar(mayor, segundo, tercero );
    for (int i = 3; i < n; ++i) {
        if (suc[i] > tercero) {
            tercero = suc[i]; ordenar(mayor, segundo, tercero );
        }
    }
}
```

Una solución del problema 3º

```
/*
 * Pre:  candidato > 0 e impar, numPrimos > 1 y las primeras «numPrimos»
 *       componentes de la tabla «primos» almacena los «numPrimos» primeros
 *       primos, ordenados de menor a mayor.
 * Post: Ha devuelto true si y solo si «candidato» es primo.
 */
bool esPrimo(const int candidato, const int primos[], const int numPrimos) {
    bool divisible = false;
    // Índice para la tabla de primos.
    // Se excluye el primero, puesto que «candidato» es par
    int i = 1;
    while (! divisible && i < numPrimos && primos[i]*primos[i] <= candidato) {
        divisible = candidato % primos[i] == 0;
        i++;
    }
    // divisible || i >= numPrimos || primos[i]^2 > candidato
    return ! divisible ;
}

/*
 * Pre: n > 0
 * Post: La tabla primos[0,n-1] almacena los n primeros números
 *       primos ordenados de menor a mayor valor
 */
void calcularPrimos(const int n, int primos[]) {
    // Almacena el primer número primo en primos[0]
    primos[0] = 2;
    // Inicia el contador de números primos almacenados en primos[0,numPrimos-1]
    int numPrimos = 1;
    // A partir de 2, solo los impares pueden ser primos
    int candidato = 3;
    // Almacena en primos[1,n-1] los n-1 números primos que siguen al 2
    while (numPrimos < n) {
        // Determina el (numPrimos+1)-ésimo número primo
        if (esPrimo(candidato, primos, numPrimos)) {
            // «candidato», que es el (numPrimos+1)-ésimo número primo,
            // es almacenado en primos
            primos[numPrimos] = candidato;
            numPrimos++;
        }
        // Se prepara a considerar el siguiente candidato a número primo
        candidato += 2;
    }
}
```

Una solución del problema 4º

```
/*
 * Pre: --
 * Post: Si c es una letra minúscula entonces devuelve la letra mayúscula
 *       equivalente, en caso contrario devuelve el valor de c
 */
char mayuscula (const char c) {
    if (c >= 'a' && c <= 'z') { return 'A' + c - 'a'; }
    else { return c; }
}

/*
 * Pre: f está asociado a un fichero que almacena una secuencia
 *      de nombres de personas del que se han podido leer algunos
 *      de los nombres de personas que almacena
 * Post: Si ha podido leer del fichero asociado a f el siguiente nombre
 *       de persona, asigna a nombrePersona una cadena de caracteres
 *       con dicho nombre y devuelve un valor cierto; en caso contrario
 *       se limita a devolver un valor falso
 */
bool leer (ifstream& f, char nombrePersona[]) {
    int n;
    f.read( reinterpret_cast<char*>(&n), sizeof(int));
    if (!f.eof()) {
        for (int i = 0; i < n; ++i) {
            f.read( reinterpret_cast<char*>(&nombrePersona[i]), sizeof(char));
        }
        // Sería válido sustituir el bucle anterior por: f.read(nombrePersona, n);
        nombrePersona[n] = '\0';
    }
    return !f.eof();
}
```

```
/*
 * Pre: nombreFichero es el nombre de un fichero binario que almacena una secuencia
 *      de nombres de personas y el valor de [letra] corresponde al de una letra
 *      mayúscula o minúscula
 * Post: Devuelve cierto (true) si y solo si el fichero de nombre nombreFichero
 *       almacena algún nombre de persona cuya inicial sea igual a letra
 *       (resulta indiferente si la inicial del nombre y letra son minúsculas o
 *       mayúsculas). En caso afirmativo, el parámetro nombrePersona almacena una
 *       cadena de caracteres con un nombre de persona de los almacenados en el
 *       fichero y cuya letra inicial sea igual a letra (resultando indiferente
 *       si la inicial del nombre y letra son minúsculas o mayúsculas)
 */
bool unNombre (const char nombreFichero[], const char letra, char nombrePersona[]) {
    ifstream f;
    f.open(nombreFichero, ios :: binary);
    if (f.is_open ()) {
        bool encontrado = false;
        char letraMay = mayuscula(letra);
        while (!encontrado && leer(f, nombrePersona)) {
            if (letraMay == mayuscula(nombrePersona[0])) {
                encontrado = true;
            }
        }
        f.close ();
        return encontrado;
    }
    else {
        return false;
    }
}
```