Examen escrito de Programación 1. Viernes 22 de enero de 2016

- 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 (3.0 puntos)

Se debe completar el diseño de la función **`coincidencias`**`(primero,segundo,tercero)`, que se especifica a continuación, escribiendo su código `C++`. En él no se debe hacer uso de ninguna función predefinida en otros módulos. Se valorará el planteamiento de un diseño descendente programando alguna o algunas funciones auxiliares. Todas las funciones que integren la solución deben estar adecuadamente especificadas No obstante, se admite que la especificación de la función pedida sea un resumen de la especificación que se muestra a continuación.

```
/*
 * Pre: primero >= 0, segundo >= 0 y tercero >= 0
 * Post: Si una misma cifra se repite k1 veces en 'primero', k2 veces en 'segundo'
 *       y k3 veces en 'tercero', el número de coincidencias de dicha cifra en ellos
 *       es el menor de k1, k2 y k3. Así, la cifra 6 se repite 4 veces en 601866726,
 *       2 veces en 96086 y 3 veces en 6600060, por lo tanto el número de coincidencias
 *       del 6 en esos tres números es 2. Así también la cifra 0 se repite una vez en
 *       601866726, una vez en 96086 y 4 veces en 6600060, por lo tanto el número de
 *       coincidencias del 0 en esos tres números es 1. Ninguna otra cifra está repetida
 *       en los tres números citados.
 *       Devuelve la suma del número de coincidencias de cada cifra decimal en 'primero',
 *       'segundo' y 'tercero', cuando los tres números se escriben en base 10.
 *       Ejemplos:
 *
 *         primero     segundo     tercero    coincidencias(primero,segundo,tercero)
 *         =======     =======     =======    ======================================
 *           46004         239      804231                                        0
 *               0           0           0                                        1
 *            2241        1344      131112                                        1
 *        12345678      202303     8321200                                        2
 *       601866726       96086     6600060                                        3
 *           41001     1110000     1010123                                        4
 *          655623    53356623     4335656                                        5
 *       123321321   111222333   918171223                                        6
 */
int coincidencias (const int primero, const int segundo, const int tercero );
```

Problema 2º (3.0 puntos)

Se debe completar el diseño de la función **`tramoOrdenado`**`(T,n,desde,hasta)` que se especifica a continuación escribiendo su código `C++`. En él no se debe hacer uso de ninguna función predefinida en otros módulos. Se valorará el planteamiento de un diseño descendente programando alguna o algunas funciones auxiliares. Todas las funciones que integren la solución deben estar adecuadamente especificadas, incluyendo la función pedida.

```
/*
 * Pre: n > 0
 * Post: Sea T[i,j] la subtabla de T[0,n-1] que almacena un mayor número de datos ordenados
 *       de menor a mayor valor: T[i]<=T[i+1], T[i+1]<=T[i+2], ..., T[j-1]<=T[j].
 *       Asigna a 'desde' el valor del índice i y a 'hasta' el del índice j
 */
void tramoOrdenado (const double T[], const int n, int& desde, int& hasta);
```

Problema 3º (4.0 puntos)

Se desea diseñar un módulo de biblioteca denominado **lista**, programado en C++, que defina el tipo de dato **ListaOrdenada** y ofrezca una colección de funciones para trabajar con datos de tipo **ListaOrdenada**.

Las características fundamentales de un dato de tipo **ListaOrdenada** son las siguientes:

- Cada dato del tipo **ListaOrdenada** puede verse como una secuencia de elementos de la forma $< D_1, D_2, \ldots, D_k >$. Cada uno de dichos elementos es un dato de tipo **int**.
- El número **k** de elementos de una lista ordenada está comprendido entre **0** y el valor de la constante **CAPACIDAD_MAXIMA**.
- Los **k** elementos de una lista ordenada están ordenados de menor a mayor valor: $D_1 \leq D_2 \leq \cdots \leq D_{k-1} \leq D_k$
- La colección de funciones disponibles en el módulo **lista** para trabajar con datos del tipo **ListaOrdenada** es la siguiente:
  - **crear**(). Al ser invocada devuelve una lista ordenada sin ningún elemento.
  - void **insertar**(LO,nuevo). Inserta el elemento **nuevo** en la lista ordenada **LO**.
  - void **retirar**(LO). Elimina de la la lista ordenada **LO** el elemento de menor valor.
  - int **consultar**(LO,i). Devuelve el elemento que ocupa la posición i-ésimo, D_i, de la lista ordenada **LO** = $< D_1, D_2, \ldots, D_i, \ldots, D_k >$, sin modificar la lista ordenada **LO**.
  - int **numElementos**(LO). Devuelve el número de elementos de la lista ordenada **LO**, sin modificar la lista ordenada **LO**.
- Ejemplo ilustrativo del uso de los recursos definidos en el módulo **lista**:

```
ListaOrdenada L = crear ();          // L = <>
int numDatos = numElementos(L);      // numDatos = 0
insertar (L, 20);                    // L = <20>
insertar (L, 40);                    // L = <20, 40>
insertar (L, 10);                    // L = <10, 20, 40>
int elemento = consultar (L,3)       // elemento = 40
insertar (L, 30);                    // L = <10, 20, 30, 40>
numDatos = numElementos(L);          // numDatos = 4
elemento = consultar (L,3)           // elemento = 30
retirar (L);                         // L = <20, 30, 40>
elemento = consultar (L,3)           // elemento = 40
numDatos = numElementos(L);          // numDatos = 3
```

En primer lugar se debe definir la estructura interna del tipo **ListaOrdenada**, escribiendo los comentarios que faciliten la comprensión de cómo se representan los datos de este tipo.

```
// Capacidad máxima de una lista ordenada
const int CAPACIDAD_MAXIMA = 375; // Modificar en caso necesario

/*
 * Un dato del tipo ListaOrdenada almacena una secuencia de datos
 * de tipo 'int', con capacidad máxima CAPACIDAD_MAXIMA.
 *    ... completar, en su caso, la explicación de la estructura ...
 */
struct ListaOrdenada {
    ... definir y explicar la estructura interna de este tipo de dato ...
};
```

A continuación, se deben especificar las cinco funciones que siguen, atendiendo a las explicaciones previas, y se debe escribir el código de cada una de ellas. En las precondiciones de las funciones se debe atender a garantizar la viabilidad de la ejecución de cada función.

```
/*
 * Pre: ... escribir la precondición de la función ...
 * Post: ... escribir la postcondición de la función ...
 */
ListaOrdenada crear ( ... escribir su lista de parámetros ... ) {
    ... escribir el código de la función ...
}

/*
 * Pre: ... escribir la precondición de la función ...
 * Post: ... escribir la postcondición de la función ...
 */
void insertar ( ... escribir su lista de parámetros ... ) {
    ... escribir el código de la función ...
}

/*
 * Pre: ... escribir la precondición de la función ...
 * Post: ... escribir la postcondición de la función ...
 */
void retirar ( ... escribir su lista de parámetros ... ) {
    ... escribir el código de la función ...
}

/*
 * Pre: ... escribir la precondición de la función ...
 * Post: ... escribir la postcondición de la función ...
 */
int consultar ( ... escribir su lista de parámetros ... ) {
    ... escribir el código de la función ...
}

/*
 * Pre: ... escribir la precondición de la función ...
 * Post: ... escribir la postcondición de la función ...
 */
int numElementos ( ... escribir su lista de parámetros ... ) {
    ... escribir el código de la función ...
}
```

Una solución del problema 1º

```
// Base numérica en la que se va a trabajar
const int BASE = 10;

/*
 * Pre:  n>=0 y «veces» tiene al menos «BASE» componentes.
 * Post: Al terminar su ejecución, «veces[i]» almacena el número de veces que el
 *       dígito «i» aparece en el número «n» cuando se escribe en base «BASE».
 */
void calcularVecesDigito (const int n, int veces[]) {
    // Inicialización de la tabla
    for (int i = 0; i < BASE; i++) {
        veces[i] = 0;
    }

    if (n == 0) {
        // Caso particular: el 0 se representa con un dígito 0
        veces[0] = 1;
    }
    else {
        // Caso general: actualización del número de veces que aparece el último
        // dígito de m y división del mismo por la base para la siguiente
        // iteración.
        int m = n;
        while (m > 0) {
            veces[m % BASE]++;
            m = m / BASE;
        }
    }
}

/*
 * Pre:  ---
 * Post: Ha devuelto el menor de los valores {a, b, c}.
 */
int menor (const int a, const int b, const int c) {
    int menor = a;
    if (b < menor) {
        menor = b;
    }
    if (c < menor) {
        menor = c;
    }
    return menor;
}

/*
 * Pre:  primero >= 0, segundo >= 0 y tercero >= 0.
 * Post: Ha duevuelto la suma del número de coincidencias de cada cifra decimal
 *       en «primero», «segundo» y «tercero», cuando los tres números se
 *       escriben en base «BASE».
 */
int coincidencias (const int primero, const int segundo, const int tercero) {
    int vecesPrimero[BASE];
    int vecesSegundo[BASE];
    int vecesTercero[BASE];

    calcularVecesDigito (primero, vecesPrimero);
    calcularVecesDigito (segundo, vecesSegundo);
```

```
    calcularVecesDigito ( tercero ,  vecesTercero );

    int sumaCoincidencias = 0;
    for (int digito = 0; digito < BASE; digito++) {
        sumaCoincidencias += menor(vecesPrimero[ digito ], vecesSegundo[ digito ],
                                   vecesTercero[ digito ]);
    }
    return sumaCoincidencias;
}
```

Una solución del problema 2º

```
/*
 * Pre:  n > 0, desde >= 0, desde <= n-1 y «T» tiene al menos «n» componentes.
 * Post: Ha devuelto el mayor de los índices «i» comprendidos en el intervalo
 *       [desde, n - 1] tal que T[desde, i] es una tabla cuyos datos están
 *       ordenados de menor a mayor valor.
 */
int finTramoOrdenado(const double T[], const int n, const int desde) {
    int finTramo = desde;
    while (finTramo < n - 1 && T[finTramo] <= T[finTramo+1]) {
        // Los datos de T[desde, finTramo + 1] están ordenados de menor a mayor
        finTramo++;
        // Los datos de T[desde, finTramo] están ordenados de menor a mayor
    }
    // finTramo >= n-1 || T[finTramo] > T[finTramo+1]
    // Los datos de T[desde, finTramo] están ordenados de menor a mayor
    // o no hay siguiente componente, o su valor es mayor que la de T[finTramo]
    return finTramo;
}

/*
 * Pre:  n > 0 y «T» tiene al menos «n» componentes.
 * Post: Sea T[i, j] la subtabla de T[0, n-1] que almacena un mayor número de
 *       datos ordenados de menor a mayor valor: T[i] <= T[i+1],
 *       T[i+1] <= T[i+2], ..., T[j-1] <= T[j].
 *       Se ha asignado a «desde» el valor del índice «i» y a «hasta» el del
 *       índice «j».
 */
void tramoOrdenado(const double T[], const int n, int& desde, int& hasta) {
    // Establecimiento del primer intervalo ordenado
    desde = 0;
    hasta = finTramoOrdenado(T, n, 0);

    // Establecimiento del inicio del siguiente intervalo ordenado
    int nuevoDesde = hasta + 1;
    while (nuevoDesde < n) {
        // Establecimiento del final del siguiente intervalo ordenado
        int nuevoHasta = finTramoOrdenado(T, n, nuevoDesde);

        // Determinación de si el último intervalo ordenado tiene mayor longitud
        // que el mayor hasta el momento
        if (nuevoHasta - nuevoDesde > hasta - desde) {
            desde = nuevoDesde;
            hasta = nuevoHasta;
        }
        // Establecimiento del inicio del siguiente intervalo ordenado
        nuevoDesde = nuevoHasta + 1;
    }
}
```

Una solución del problema 3º

```
/*
 * Un dato del tipo ListaOrdenada almacena una secuencia de
 * datos de tipo 'int', con capacidad máxima CAPACIDAD_MAXIMA.
 * La lista se gestiona en función del valor de los datos
 * almacenados en ella. Cuando se retira un dato de la lista,
 * el dato retirado es siempre el de menor valor.
 */
struct ListaOrdenada {
    // Número de elementos de la lista ordenada
    int numElementos;
    // Los elementos de la lista están almacenados en elemento[0,numElementos-1]
    // y están ordenados de menor a mayor valor: elemento[0] <= elemento[1],
    // elemento[1] <= elemento[2], ..., elemento[numElementos-2]
    //                              <= elemento[numElementos-1]
    int elemento[CAPACIDAD_MAXIMA];
};
```

```
/*
 * Fichero listaOrdenada.cc de implementación del módulo listaOrdenada
 */

#include "listaOrdenada.h"

/*
 * Pre: ---
 * Post: Devuelve una lista ordenada sin ningún elemento
 */
ListaOrdenada crear () {
    ListaOrdenada LO;
    LO.numElementos = 0;
    return LO;
}

/*
 * Pre: numElementos(L) < CAPACIDAD_MAXIMA
 * Post: Añade 'nuevo' como nuevo elemento de la lista ordenada LO
 */
void insertar (ListaOrdenada& LO, const int nuevo) {
    bool encontrado = false;
    int i = 0;
    while (!encontrado && i < LO.numElementos) {
        if (LO.elemento[i] > nuevo) {
            encontrado = true;
        }
        else {
            i = i + 1;
        }
    }
    if (encontrado) {
        for (int j = LO.numElementos - 1; j >= i; --j) {
            LO.elemento[j+1] = LO.elemento[j];
        }
        LO.elemento[i] = nuevo;
    }
    else {
        LO.elemento[LO.numElementos] = nuevo;
    }
```

```
    LO.numElementos = LO.numElementos + 1;
}

/*
 * Pre: numElementos(L) > 0
 * Post: Elimina de la lista ordenada LO el menor de sus elementos
 */
void retirar (ListaOrdenada& LO) {
    for (int i = 1; i < LO.numElementos; ++i) {
        LO.elemento[i-1] = LO.elemento[i];
    }
    LO.numElementos = LO.numElementos - 1;
}

/*
 * Pre: i >= 1 y i <= numElementos(LO)
 * Post: Devuelve el valor de su i-ésimo menor elemento
 */
int consultar (const ListaOrdenada& LO, const int i) {
    return LO.elemento[i-1];
}

/*
 * Pre: --
 * Post: Devuelve el número de elementos de la lista ordenada LO
 */
int numElementos (const ListaOrdenada& LO) {
    return LO.numElementos;
}
```