# Examen Práctico de Programación 1 - 7 de septiembre de 2018

## Especificación del trabajo a desarrollar

Se ha de definir la estructura interna de los datos de tipo **`SL`** y el código de las cinco funciones que tienen acceso a dicha estructura interna (**`definir`**`(sec, cadena)`, **`concatenar`**`(sec, anyadido)`, **`insertar`**`(sec, c)`, **`ordenar`**`(sec)` y **`mostrar`**`(sec)`).

```
// Autor del trabajo (escriba su nombre y apellidos):

#include <iostream>
using namespace std;

// Máximo número de letras que pueden llegar a integrar una secuencia de tipo SL
const int NUM_MAX_LETRAS = 100;

/*
 * Un dato de tipo SL permite representar una secuencia de letras mayúsculas
 * de longitud menor o igual que NUM_MAX_LETRAS
 */
struct SL {
    private:
        // Estructura interna de un dato de tipo SL
        ???
    public:
        // Funciones para gestionar datos de tipo SL
        friend void definir (SL& sec, const char cadena []);
        friend void concatenar (SL& sec, const SL& anyadido);
        friend void insertar (SL& sec, const char c);
        friend void ordenar (SL& sec);
        friend void mostrar (const SL& sec);
};

/*
 * Pre: <letras> almacena una cadena de caracteres todos los cuales son letras mayúsculas.
 *      El número de caracteres de la cadena ha de ser menor o igual que NUM_MAX_LETRAS
 * Post: Ha asignado a <sec> la secuencia de letras mayúsculas almacenada en <letras>,
 *       respetando su posición en <letras>
 * Ejemplo: Tras ejecutar definir(sec, "ABCABCZYX"), <sec> almacena la secuencia de
 *          caracteres ABCABCZYX
 */
void definir (SL& sec, const char letras []) {
    ???
}

/*
 * Pre: <sec> y <anyadida> almacenan dos secuencias de letras mayúsculas. La suma
 *      de las longitudes de ambas secuencias es menor o igual que NUM_MAX_LETRAS
 * Post: <sec> almacena la secuencia de letras mayúsculas resultante de concatenar
 *       la secuencia almacenada inicialmente en <sec> seguida de la secuencia
 *       almacenada inicialmente en <anyadida>
 * Ejemplo: Sea PEREZ la secuencia que almacena <sec1> y sea GALDOS la secuencia que
 *          almacena <sec2>. Tras invocar la ejecución de concatenar(sec1,sec2) entonces
 *          <sec1> almacenará la secuencia PEREZGALDOS
 */
void concatenar (SL& sec, const SL& anyadida) {
    ???
}
```

```
/*
 * Pre: <sec> almacena una secuencia de letras mayúsculas ordenadas alfabéticamente
 *      y el valor de <c> es el de una letra mayúscula. La longitud de la secuencia
 *      almacenada en <sec> es inferior a NUM_MAX_LETRAS
 * Post: <sec> almacena una secuencia de letras mayúsculas ordenadas alfabéticamente
 *       de longitud una unidad mayor que la longitud de la secuencia que almacenaba
 *       inicialmente . En la nueva secuencia almacenada en <sec> forman parte todas
 *       las letras de su secuencia inicial a las que se ha añadido un ejemplar de
 *       la letra <c>
 * Ejemplos: Sea ABCDEF la secuencia que almacena <sec>. Tras invocar la ejecución de
 *       insertar(sec,'C') entonces <sec> almacenará ABCCDEF
 *       Sea ABCDEF la secuencia que almacena <sec>. Tras invocar la ejecución de
 *       insertar(sec,'M') entonces <sec> almacenará ABCDEFM
 *       Sea ABCXYZ la secuencia que almacena <sec>. Tras invocar la ejecución de
 *       insertar(sec,'H') entonces <sec> almacenará ABCHXYZ
 */
void insertar (SL& sec, const char c) {
    ???
}

/*
 * Pre: <sec> almacena una secuencia de letras mayúsculas
 * Post: <sec> almacena una secuencia de letras mayúsculas ordenadas alfabéticamente
 *       integrada por las mismas letras que formaban parte de su secuencia inicial
 * Ejemplo: Sea PEREZOSO la secuencia que almacena <sec>. Tras ejecutar ordenar(sec)
 *          <sec> almacenará la secuencia EEOOPRSZ
 */
void ordenar (SL& sec) {
    ???
}

/*
 * Post: Presenta por el dispositivo estándar de salida la secuencia de caracteres
 *       almacenada en <sec>, sin acabar la línea en curso
 */
void mostrar (const SL& sec) {
    ???
}
```

En el desarrollo del trabajo se puede hacer uso de las funciones de la biblioteca predefinida **`iostream`**, pero no pueden utilizarse funciones de ninguna otra biblioteca predefinida.

En la carpeta **`examenSeptiembre`** accesible desde la web de la asignatura (seción de *`Código C+++ y datos`*) se ha dispuesto un fichero de nombre **`examenSep.cc`** que contiene el esquema anterior integrado dentro de un programa, con su correspondiente función principal, función **`main`**`()`, para facilitar la realización de pruebas.

## Presentación del trabajo y criterios de evalución

Cada alumno presentará el código completo del programa contenido en el fichero **`examenSep.cc`** a través de la plataforma **Moodle2** (https://moodle2.unizar.es/) antes de las 14:00. Este fichero vendrá encabezado por un comentario con el nombre y apellidos del alumno.

En este examen práctico **se valorará esencialmente que el programa pueda ser ejecutado y que cada función proporcione los resultados esperados**.

También se valorará el diseño del tipo de dato **`SL`** y del código de sus cinco funciones asociadas y la legibilidad del trabajo de programación realizado, atendiendo a los criterios de la **"*Guía de estilo para programar en C++*"** publicada en la web de la asignatura.

## Una solución del problema propuesto

```
// Máximo número de letras que pueden llegar a integrar una secuencia de tipo SL
const int NUM_MAX_LETRAS = 100;

/*
 * Un dato de tipo SL permite representar una secuencia de letras mayúsculas
 * de longitud menor o igual que NUM_MAX_LETRAS
 */
struct SL {
    private:
        // Estructura interna de un dato de tipo SL
        int numLetras;                          // Número de letras de la secuencia
        char letras [NUM_MAX_LETRAS];           // letras [0.. numLetras]
    public:
        // Funciones para gestionar datos de tipo SL
        friend void definir (SL& sec, const char cadena []);
        friend void concatenar (SL& sec, const SL& anyadido);
        friend void insertar (SL& sec, const char c );
        friend void ordenar (SL& sec);
        friend void mostrar (const SL& sec);
};

/*
 * Pre: <letras> almacena una cadena de caracteres todos los cuales son letras mayúsculas.
 *      El número de caracteres de la cadena ha de ser menor o igual que NUM_MAX_LETRAS
 * Post: Ha asignado a <sec> la secuencia de letras mayúsculas almacenada en <letras>,
 *       respetando su posición en <letras>
 * Ejemplo: Tras ejecutar definir (sec, "ABCABCZYX") <sec> almacena la secuencia de
 *          caracteres ABCABCZYX
 */
void definir (SL& sec, const char letras []) {
    const char NULO = '\0';
    sec.numLetras = 0;
    while ( letras [sec.numLetras] != NULO) {
        sec. letras [sec.numLetras] = letras [sec.numLetras];
        sec.numLetras = sec.numLetras + 1;
    }
}

/*
 * Pre: <sec> y <anyadida> almacenan dos secuencias de letras mayúsculas. La suma
 *      de las longitudes de ambas secuencias es menor o igual que NUM_MAX_LETRAS
 * Post: <sec> almacena la secuencia de letras mayúsculas resultante de concatenar
 *       la secuencia almacenada inicialmente en <sec> seguida de la secuencia
 *       almacenada inicialmente en <anyadida>
 * Ejemplo: Sea PEREZ la secuencia que almacena <sec1> y sea GALDOS la secuencia que
 *          almacena <sec2>. Tras invocar la ejecución de concatenar(sec1,sec2) entonces
 *          <sec1> almacenará la secuencia PEREZGALDOS
 */
void concatenar (SL& sec, const SL& anyadida) {
    for (int i = 0; i < anyadida.numLetras; ++i) {
        sec. letras [sec.numLetras + i] = anyadida. letras [i];
    }
    sec.numLetras = sec.numLetras + anyadida.numLetras;
}
```

```
/*
 * Pre: <sec> almacena una secuencia de letras mayúsculas ordenadas alfabéticamente
 *      y el valor de <c> es el de una letra mayúscula. La longitud de la secuencia
 *      almacenada en <sec> es inferior a NUM_MAX_LETRAS
 * Post: <sec> almacena una secuencia de letras mayúsculas ordenadas alfabéticamente
 *       de longitud una unidad mayor que la longitud de la secuencia que almacenaba
 *       inicialmente . En la nueva secuencia almacenada en <sec> forman parte todas
 *       las letras de su secuencia inicial a las que se ha añadido un ejemplar de
 *       la letra <c>
 * Ejemplos: Sea ABCDEF la secuencia que almacena <sec>. Tras invocar la ejecución de
 *       insertar(sec,'C') entonces <sec> almacenará ABCCDEF
 *       Sea ABCDEF la secuencia que almacena <sec>. Tras invocar la ejecución de
 *       insertar(sec,'M') entonces <sec> almacenará ABCDEFM
 *       Sea ABCXYZ la secuencia que almacena <sec>. Tras invocar la ejecución de
 *       insertar(sec,'H') entonces <sec> almacenará ABCHXYZ
 */
void insertar (SL& sec, const char c) {
    int i = 0;
    bool seguir = true;
    while (seguir && i < sec.numLetras) {
        if (sec.letras[i] <= c) {
            i = i + 1;
        }
        else {
            seguir = false;
        }
    }
    for (int j = sec.numLetras; j > i; --j) {
        sec.letras[j] = sec.letras[j-1];
    }
    sec.letras[i] = c;
    sec.numLetras = sec.numLetras + 1;
}
```

```
/*
 * Pre: <sec> almacena una secuencia de letras mayúsculas
 * Post: <sec> almacena una secuencia de letras mayúsculas ordenadas alfabéticamente
 *       integrada por las mismas letras que formaban parte de su secuencia inicial
 * Ejemplo: Sea PEREZOSO la secuencia que almacena <sec>. Tras ejecutar ordenar(sec)
 *          <sec> almacenará la secuencia EEOOPRSZ
 */
void ordenar (SL& sec) {
    // Algoritmo de ordenación de un vector por selección
    for (int i = 0; i < sec.numLetras - 1; ++i) {
        // Determina el indice iPrimera de la primera letra, en orden alfabébetico,
        // sec.letras[iPrimera], de sec.letras[i..sec.numLetras-1]
        int iPrimera = i;
        int j = i + 1;
        while (j < sec.numLetras) {
            if (sec.letras[j] < sec.letras[iPrimera]) {
                iPrimera = j;
            }
            j = j + 1;
        }
        // sec.letras[iPrimera] es la primera letra, en orden alfabébetico,
        // de sec.letras[i..sec.numLetras - 1]

        // Permuta sec.letras[i] y sec.letras[iPrimera]
        char aux = sec.letras[i];
        sec.letras[i] = sec.letras[iPrimera];
        sec.letras[iPrimera] = aux;
        // La letras sec.letras[0..i] están colocadas en orden alfabébetico
        // y preceden alfabéticamente a las letras sec.letras[i+1..sec.numLetras-1]
    }
}

/*
 * Post: Presenta por el dispositivo estándar de salida la secuencia de caracteres
 *       almacenada en <sec>, sin acabar la línea en curso
 */
void mostrar (const SL& sec) {
    for (int j = 0; j < sec.numLetras; ++j) {
        cout << sec.letras[j];
    }
}
```