Examen escrito de Programación 1

Escuela de Ingeniería y Arquitectura
Departamento de Informática e Ingeniería de Sistemas

7 de febrero de 2019

- Se debe disponer sobre la mesa en lugar visible un **documento de identificación** provisto de fotografía.
- Se debe escribir **nombre y dos apellidos** en cada una de las hojas de papel que haya sobre la mesa.
- Se debe comenzar a resolver cada uno de los problemas del examen **en una hoja diferente** para facilitar su corrección por profesores diferentes.
- Tiempo máximo para realizar el examen: **3 horas**.
- No está permitido utilizar dispositivos electrónicos de ningún tipo, ni consultar libros ni apuntes, excepto los dos documentos facilitados por los profesores de la asignatura: *Guía de sintaxis ANSI/ISO estándar C++*, *Resumen de recursos predefinidos en C++ que son utilizados en la asignatura* y *Guía de estilo para programar en C++*.

Parte I. Números complejos

Problema 1.º (2,5 puntos)

Un número complejo es un número que puede ser expresado de la forma $a + bi$, donde a y b son números reales e i es la solución a la ecuación $x^2 = -1$. Dado que ningún número real satisface esta ecuación, a i se le llama unidad imaginaria. Dado el número complejo $a + bi$, se denomina parte real a a y parte imaginaria a b.[1].

[1] Wikipedia contributors, «Complex number», *Wikipedia, The Free Encyclopedia,* `https://en.wikipedia.org/w/index.php?title=Complex_number&oldid=879219624` (accedido el 22 de enero de 2019).

Se pide:

- Definir un tipo de datos que permita representar números complejos.
- Escribir la **cabecera**, **especificación** (a través de su precondición y postcondición) y el **cuerpo** de las siguientes funciones:

    1. Una función que, dados dos números complejos, devuelva otro número complejo que represente la suma de los dos primeros.

       Matemáticamente, la suma de dos números complejos $a + bi$ y $c + di$ es $(a + c) + (b + d)i$.

    2. Una función que, dados dos números complejos, devuelva otro número complejo que represente el producto de los dos primeros.

       Matemáticamente, el producto de dos números complejos $a + bi$ y $c + di$ puede calcularse como $(ac - bd) + (ad + bc)i$.

    3. Una función que, dado un número complejo y un exponente entero no negativo, devuelva otro número complejo que represente la potencia del primer número elevado al exponente.

       Matemáticamente, $(a + bi)^n = 1 \underbrace{\times (a + bi) \times (a + bi) \times \ldots \times (a + bi)}_{n \text{ veces}}$

Parte II. Prototipo de un corrector de ortografía

Se desea disponer de un programa que realice la corrección de la ortografía de un texto. Se dispone ya de un diccionario en español, almacenado en un fichero de texto denominado «`diccionario.txt`» que se describe en detalle más adelante y que servirá de base para realizar dicha corrección.

Por el momento, se va a diseñar e implementar únicamente un prototipo que, en lugar de corregir la ortografía, se va a limitar a validarla. El programa solicitará al usuario el nombre de un fichero de texto, que contendrá el texto que se desea validar, y mostrará al usuario un resumen indicando el número de palabras encontradas y no encontradas en el diccionario, tal y como se muestra en los siguientes ejemplos de dos ejecuciones independientes:

```
Escriba el nombre de un fichero de texto: quijote.txt

El fichero "quijote.txt" tiene:
  364386 palabras encontradas en el diccionario.
   16829 palabras no encontradas.
```

```
Escriba el nombre de un fichero de texto: regenta.txt

El fichero "regenta.txt" tiene:
  296614 palabras encontradas en el diccionario.
   10742 palabras no encontradas.
```

En el caso de que el usuario introduzca el nombre de un fichero de texto que no exista, el programa mostrará un mensaje de error:

```
Escriba el nombre de un fichero de texto: vientos-de-invierno.txt
No se ha podido abrir el fichero "vientos-de-invierno.txt".
```

Finalmente, si el programa no es capaz de abrir el fichero «`diccionario.txt`», escribirá un mensaje de error antes incluso de pedir al usuario el nombre del fichero:

```
No se ha podido abrir el fichero "diccionario.txt".
```

Ficheros utilizados

Los ficheros cuyo nombre se le va a pedir al usuario para que sea verificada su ortografía son simples ficheros de texto. El contenido de esos ficheros son textos escritos en español como, por ejemplo, novelas.

```
En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho
tiempo que vivia un hidalgo de los de lanza en astillero, adarga antigua,
rocin flaco y galgo corredor. Una olla de algo mas vaca que carnero,
salpicon las mas noches, duelos y quebrantos los sabados, lantejas los
viernes, algun palomino los domingos, consumian las tres partes de su
hacienda.

El resto della concluian sayo de velarte, calzas de velludo para las
fiestas, con sus pantuflos de lo mesmo, y los dias de entresemana se honraba
con su vellori de lo mas fino. Tenia en su casa una ama que pasaba de [...]
```

El fichero que contiene el diccionario es también un fichero de texto que, en este caso, contiene una única palabra por línea, escrita en minúsculas. Cada variación de cada palabra de la lengua española (por género, número o persona) está presente de forma independiente en el fichero de diccionario. Las palabras del fichero están ordenadas alfabéticamente por orden ascendente. Se muestra a continuación parte del contenido de dicho fichero:

```
a
ababol
ababoles
abacera
abaceras
abad
abadejo
abadejos
abades
abadesa
abadesas
...
programamos
programan
programando
programar
...
zurzo
zutana
zutanas
zutano
zutanos
```

El fichero de diccionario no tiene más de 300 000 palabras y cada una de ellas de menos de 30 caracteres de longitud.

Tanto los ficheros cuya ortografía el usuario quiera validar como el fichero de diccionario utilizan la misma codificación de caracteres. Como simplificación del problema, se informa de que ni los ficheros a validar ni en el fichero de diccionario hay letras distintas a las del alfabeto inglés.

Entorno

En los problemas que siguen, se puede suponer que se han declarado las siguientes constantes simbólicas con ámbito global:

```
const int MAX_LONG_FICH = 128;
const char FICH_DICCIONARIO[] = "diccionario.txt";
const int MAX_PALABRAS = 300000;
const int MAX_LONG = 30;
```

Cualquiera de las funciones que se piden en los siguientes problemas pueden utilizarse en la solución de cualquier otro problema.

Problema 2.º (0,5 puntos)

El cuerpo de la función `buscar` que se presenta a continuación contiene un **único** error **sintáctico**:

```
#include <cstring>
using namespace std;
const int MAX_LONG = 30;

/*
 * Pre:  «palabra» es una cadena de caracteres acabada en el carácter '\0' y
 *       de menos de MAX_LONG caracteres; numPalabras > 0; el vector
 *       «diccionario» tiene al menos «numPalabras» componentes, que son
 *       cadenas de caracteres, cada una de ellas acabadas en el carácter '\0'
 *       y de menos de MAX_LONG caracteres; las
 *       «numPalabras» primeras componentes del vector «diccionario» están
 *       ordenadas alfabéticamente de forma ascendente.
 * Post: Si entre las cadenas de caracteres del vector «diccionario» hay una
 *       igual al valor del parámetro «palabra», ha devuelto el índice de
 *       dicha cadena en el vector; en caso contrario, ha devuelto un valor
 *       negativo.
 */
int buscar(const char palabra[], const char diccionario[][MAX_LONG],
           const int numPalabras) {
    int inf = 0;
    int sup = numPalabras - 1;
    while (inf < sup) {
        med = (inf + sup) / 2;
        if (strcmp(palabra, diccionario[med]) > 0) {
            inf = med + 1;
        }
        else {
            sup = med;
        }
    }

    if (strcmp(palabra, diccionario[inf]) == 0) {
        return inf;
    }
    else {
        return -1;
    }
}
```

Identifica el error. Para ello, indicando la línea en la que se encuentra, en qué consiste y cómo solucionarlo.

Problema 3.º (2,5 puntos)

Escribe el código de la función `limpiar` cuya especificación se muestra a continuación:

```
/*
 * Pre:  «palabra» es una cadena de caracteres acabada en el carácter '\0' y
 *       de menos de MAX_LONG caracteres que no contiene ningún
 *       carácter blanco (ni espacios en blanco, tabulaciones o caracteres de
 *       cambio de línea).
 * Post: Cuando se ha terminado de ejecutar esta función, la cadena de
 *       caracteres «palabra» contiene únicamente las letras del alfabeto
 *       inglés que contenía inicialmente (cuando se comenzó a ejecutar esta
 *       función). Cualquier otro carácter ha sido eliminado.
 *
 *       Ejemplos:
 *
 *       Valor inicial de «palabra»         Valor final de «palabra»
 *       ------------------------------------------------------------
 *       ""                                 ""
 *       "En"                               "en"
 *       "un"                               "un"
 *       "Mancha,"                          "mancha"
 *       "corredor."                        "corredor"
 *       "-¡Oh!"                            "oh"
 *       "¿Duermen?"                        "duermen"
 *       "1604"                             ""
 *       "H2SO4"                            "hso"
 */
void limpiar(char palabra[]);
```

De la solución que se presente se valorará, entre otros aspectos, que se trabaje directamente sobre el parámetro `palabra` sin utilizar cadenas de caracteres adicionales.

Problema 4.º (1,5 puntos)

Escribe el código de la función `contarPalabras` cuya especificación se muestra a continuación:

```
/*
 * Pre:  «nombreFichero» es una cadena de caracteres cuyo valor representa el
 *       nombre de un fichero de texto que utiliza la misma codificación de
 *       caracteres que el fichero «FICH_DICCIONARIO» y que no
 *       contiene letras que no pertenezcan al alfabeto inglés (pero que
 *       puede contener caracteres que no sean letras, como dígitos y signos
 *       de puntuación);
 *       «numPalabras» > 0;
 *       el vector «diccionario» tiene al menos «numPalabras» componentes,
 *       que son cadenas de caracteres, cada una de ellas acabadas en el
 *       carácter '\0' y de menos de MAX_LONG caracteres; las
 *       «numPalabras» primeras componentes del vector «diccionario» están
 *       ordenadas alfabéticamente de forma ascendente;
 *       los valores de los parámetros «encontradas» y «noEncontradas» no
 *       están definidos.
 * Post: Ha asignado a los parámetros «encontradas» y «noEncontradas» el
 *       número de palabras contenidas en el fichero de nombre
 *       «nombreFichero» que, respectivamente, se encuentran y no se han
 *       encontrado en las primeras «numPalabras» primeras componentes del
 *       vector «diccionario», entendiendo como palabra una secuencia de una
 *       o más letras minúsculas del alfabeto inglés. Ha devuelto «true»
 *       cuando el fichero de nombre «nombreFichero» ha podido procesarse
 *       correctamente, y «false» en caso contrario.
 */
bool contarPalabras(const char nombreFichero[],
                    const char diccionario[][MAX_LONG],
                    const int numPalabras,
                    int& encontradas, int& noEncontradas);
```

Se recuerda que cualquiera de las funciones presentadas en este enunciado puede ser utilizada en la solución de este problema.

Problema 5.º (3 puntos)

Especifica a través de una precondición y una postcondición la función `main` correspondiente al programa presentado en la introducción de esta parte II del enunciado.

Escribe también su código. En la solución presentada se exigirá, además de su corrección y legibilidad, la aplicación de la metodología de diseño descendente y la minimización del esfuerzo de desarrollo por haber hecho uso adecuado de las funciones cuya cabecera y especificación se han mostrado en los problemas anteriores. Si se estima necesario, se pueden definir también otras funciones auxiliares, cuyo código y especificación deberá proporcionarse. Se puede (y recomienda) utilizar también las constantes simbólicas definidas en la introducción.

Solución al problema 1.º

```
struct Complejo {
    double parteReal, parteImaginaria;
};

/*
 * Pre:  ---
 * Post: Ha devuelto la suma de los números complejos «a» y «b».
 */
Complejo sumar(Complejo a, Complejo b) {
    Complejo suma = {a.parteReal + b.parteReal,
                     a.parteImaginaria + b.parteImaginaria};
    return suma;
}

/*
 * Pre:  ---
 * Post: Ha devuelto el producto de los números complejos «a» y «b».
 */
Complejo multiplicar(Complejo a, Complejo b) {
    Complejo producto = {a.parteReal * b.parteReal
                                - a.parteImaginaria * b.parteImaginaria,
                         a.parteReal * b.parteImaginaria
                                + a.parteImaginaria * b.parteReal};
    return producto;
}

/*
 * Pre:  exponente  0
 * Post: Ha devuelto la potencia resultante de elevar el número complejo
 *       «base» al exponente indicado por el valor del parámetro «exponente».
 */
Complejo elevar(Complejo base, int exponente) {
    // Se inicializa el resultado con la unidad
    Complejo potencia = {1.0, 0.0};

    // Se multiplica «exponente» veces el resultado por la base
    for (int i = 1; i <= exponente; i++) {
        potencia = multiplicar(potencia, base);
    }

    // Se devuelve la potencia
    return potencia;
```

```
}
```

Solución al problema 2.º

El error se encuentra en la instrucción de asignación de la línea 23 y consiste en que la variable `med` no ha sido declarada. La solución es convertir la instrucción de asignación «`med = (inf + sup) / 2;`» en una instrucción de declaración «**`int`** `med = (inf + sup) / 2;`», en la que `med` se declara e inicializa.

El código correcto se muestra a continuación:

```
/*
 * Especificación en el enunciado.
 */
int buscar(const char palabra[], const char diccionario[][MAX_LONG],
           const int numPalabras) {
    int inf = 0;
    int sup = numPalabras - 1;
    while (inf < sup) {
        int med = (inf + sup) / 2;
        if (strcmp(palabra, diccionario[med]) > 0) {
            inf = med + 1;
        }
        else {
            sup = med;
        }
    }

    if (strcmp(palabra, diccionario[inf]) == 0) {
        return inf;
    }
    else {
        return -1;
    }
}
```

Solución al problema 3.º

```
/*
 * Especificación en el enunciado.
 * Nota: Por error, en la postcondición del enunciado no se menciona que la
 *       palabra, una vez limpiada debe contener únicamente letras
 *       minúsculas, aunque se puede deducir de los ejemplos. Al corregir, no
 *       se va a penalizar ni por convertir las letras a minúsculas, ni por
 *       no hacerlo. En todo caso, la solución al problema propuesto sí que
 *       requiere que, en algún momento, las palabras leídas del fichero del
 *       texto que se revisa se pasen a minúsculas antes de buscarlas en el
 *       diccionario. El lugar puede ser esta función o la función
 *       «contarPalabras».
 */
void limpiar(char palabra[]) {
    // Índice con el que consultar cada carácter del valor original de la
    // cadena «palabra». Se va a incrementar de forma constante en 1 unidad
    // en el bucle que procesará palabra.
    int indOriginal = 0;

    // Índice con el que escribir cada carácter del valor final de «palabra».
    // Se va a incrementar en 1 o 0 componentes por iteración, dependiendo de
    // si palabra[indOriginal] es una letra o no. Por ello, se va a cumplir
    // que indLimpia <= indOriginal.
    int indLimpia = 0;

    while (palabra[indOriginal] != '\0') {
        if (isalpha(palabra[indOriginal])) {
            palabra[indLimpia] = tolower(palabra[indOriginal]);
            indLimpia++;
        }
        indOriginal++;
    }
    // palabra[indOriginal] != '\0'

    // Como indLimpia <= indOriginal, hay que asegurarse de que la palabra
    // limpia termina con un carácter nulo en la posición indexada por
    // «indLimpia».
    palabra[indLimpia] = '\0';
}
```

Solución al problema 4.º

```
/*
 * Especificación en el enunciado.
 */
bool contarPalabras(const char nombreFichero[],
                    const char diccionario[][MAX_LONG],
                    const int numPalabras,
                    int& encontradas, int& noEncontradas) {
    ifstream texto(nombreFichero);
    if (texto.is_open()) {
        encontradas = 0;
        noEncontradas = 0;

        char palabra[MAX_LONG];
        // Lectura con el operador de alto nivel, puesto que puede habar dos
        // delimitadores de «palabras» válidos: ' ' y '\n'. «getline» solo
        // permite especificar  un delimitador, por lo que las soluciones que
        // hacen uso de «getline» son o erróneas o más complejas.
        // Como el método «eof» solo devuelve true cuando se han leído o
        // intentado leer más bytes de los que tiene el fichero, se intenta
        // leer una nueva palabra del texto antes de entrar en el bucle y
        // otra al final.
        texto >> palabra;

        while (!texto.eof()) {
            // texto.eof() es falso: el último intento de lectura (antes de
            // entrar en el bucle o antes de acabar la iteración anterior),
            // leyó correctamente una «palabra». Se procesa a continuación:
            limpiar(palabra);
            if (buscar(palabra, diccionario, numPalabras) >= 0) {
                // El índice devuelto corresponde a una componente válida del
                // vector: «palabra» se ha encontrado en el diccionario.
                encontradas++;
            }
            else {
                // Se ha devuelto -1: «palabra» no está en el diccionario.
                noEncontradas++;
            }
            texto >> palabra;
        }
        texto.close();
        return true;
```

```
    }
    else {
        return false;
    }
}
```

Solución al problema 5.º

```
/*
 * Pre:  Existe un fichero de texto denominado «nombreFichero» que contiene
 *       una única palabra por línea, escrita exclusivamente en minúsculas
 *       del alfabeto inglés. Las palabras del mismo están ordenadas
 *       alfabéticamente de forma ascendente. En el fichero no hay más de
 *       «MAX_PALABRAS» y todas ellas son de menos de «MAX_LONG»
 *       caracteres. El vector «diccionario» tiene al menos «MAX_PALABRAS»
 *       componentes.
 * Post: Ha asignado al parámetro «numPalabras» el número de palabras que
 *       contiene el fichero «nombreFichero», y a las «numPalabras» primeras
 *       componentes del vector «diccionario», las palabras contenidas en el
 *       fichero, en el mismo orden en que aparecen en el fichero (ordenadas
 *       alfabéticamente). Ha devuelto «true» si el fichero «nombreFichero»
 *       se ha podido leer, y «false» en caso contrario.
 */
bool cargarDiccionario(const char nombreFichero[],
                       char diccionario[][MAX_LONG],
                       int& numPalabras) {
    ifstream f(nombreFichero);
    if (f.is_open()) {
        numPalabras = 0;
        f.getline(diccionario[numPalabras], MAX_LONG);
        while (!f.eof() && numPalabras < MAX_PALABRAS - 1) {
            numPalabras++;
            f.getline(diccionario[numPalabras], MAX_LONG);
            // también sería válido f >> diccionario[numPalabras];
        }
        f.close();
        return true;
    }
    else {
        return false;
    }
}


/*
 * Pre:  Existe un fichero de texto denominado «FICH_DICCIONARIO»
 *       que contiene una única palabra por línea, escrita exclusivamente en
 *       minúsculas del alfabeto inglés. Cada variación de cada palabra está
 *       presente de forma independiente en el fichero. Las palabras del mismo
```

```
 *       están ordenadas alfabéticamente de forma ascendente. En el fichero
 *       de diccionario no hay más de «MAX_PALABRAS» y todas ellas son de
 *       menos de «MAX_LONG» caracteres.
 * Post: El programa ha solicitado al usuario el nombre de un fichero de texto
 *       que utiliza la misma codificación de caracteres que el fichero
 *       «FICH_DICCIONARIO» y que no contiene letras que no
 *       pertenezcan al alfabeto inglés (pero que puede contener caracteres
 *       que no sean letras, como dígitos y signos de puntuación). Si el
 *       fichero existe, ha informado al usuario del número de palabras
 *       contenidas en el fichero cuyo nombre haya escrito el usuario que se
 *       encuentran y no se han encontrado en el fichero de diccionario,
 *       entendiendo como palabra una secuencia de una o más letras
 *       minúsculas del alfabeto inglés. En caso contrario, o si el fichero
 *       de diccionario no ha podido abrirse, escribe en la pantalla un
 *       mensaje de error.
 */
int main() {
    // Declaración del diccionario como una matriz de caracteres, que va a ser
    // tratada como un vector de cadenas de caracteres (el primer índice
    // indexa una cadena, el segundo índice, indexa un caracter de una cadena
    // concreta.
    char diccionario[MAX_PALABRAS][MAX_LONG];
    int numPalabras;

    if (cargarDiccionario(FICH_DICCIONARIO,
                          diccionario, numPalabras)) {
        // Si se ha podido cargar el diccionario, se pide el nombre del
        // fichero a verificar. Se lee con «getline» para permitir que haya
        // espacios en blanco en el nombre del fichero.
        cout << "Escriba el nombre de un fichero de texto: " << flush;
        char nombreFichero[MAX_LONG_FICH];
        cin.getline(nombreFichero, MAX_LONG_FICH);
        cout << endl;

        int encontradas, noEncontradas;
        if (contarPalabras(nombreFichero, diccionario, numPalabras,
                           encontradas, noEncontradas)) {
            // Si se ha podido leer el fichero cuyo nombre indica
            // «nombreFichero», se muestra en la pantalla los resultados
            cout << "El fichero \"" << nombreFichero << "\" tiene: " << endl;
            cout << setw(8) << encontradas
                 << " palabras encontradas en el diccionario." << endl;
            cout << setw(8) << noEncontradas << " palabras no encontradas."
                 << endl;
```

```
            return 0;
        }
        else {
            cerr << "No␣se␣ha␣podido␣abrir␣el␣fichero␣\"" << nombreFichero
                 << "\"." << endl;
            return 1;
        }
    }
    else {
        cerr << "No␣se␣ha␣podido␣abrir␣el␣fichero␣\"" << FICH_DICCIONARIO
             << "\"." << endl;
        return 2;
    }
}
```