Universitat Oberta de Catalunya uoc.edu

# Fundamentos de Programación
PEC4 – 20191

Fecha límite de entrega: 21/10/2019

Estudiante
**Apellidos: Delgado Flores**
**Nombre: Pablo José**

## Objetivos

- Saber aplicar correctamente la estructura de control iterativa.
- Aprender a utilizar el tipo de datos vector para representar estructuras de datos sencillos.
- Saber utilizar las funciones de conversión de tipo cuando haga falta.

## Formato y fecha de entrega

La PEC se debe entregar antes del día **21 de octubre de 2019 a las 23:59**.

Para la entrega se deberá entregar un fichero en formato **ZIP**, que contenga:

- Este documento con las respuestas de los ejercicios 1 y 2b.
- Un workspace de Codelite que contenga los ficheros .c pedidos en el ejercicio 2a.

Hay que hacer la entrega en el apartado de entregas de EC del aula de teoría.

# Enunciado

Siguiendo con la ayuda que proporcionamos a la compañía UOCBookings, nos piden que continuemos con nuestra colaboración. En este caso nos dicen que quieren hacer una simulación con un posible algoritmo para encriptar contraseñas de clientes.

Nos proporcionan el siguiente algoritmo que debemos completar:

```
Lenguaje algorítmico
const
   MAX_CHARACTERS: integer = 10;
end const

algorithm UOCBookings
var
    password: vector[MAX_CHARACTERS] of char;
    numRepetitions: integer;
end var
    ...
end algorithm
```

## Ejercicio 1: [50%]

### Apartado a [80%]

Completa el algoritmo anterior para que encripte un número determinado de contraseñas (con un máximo de 100). Cada contraseña (*password*) está formada por 10 caracteres, cada uno de los cuales puede ser una letra minúscula o un dígito (del 0 al 9).

En primer lugar el algoritmo ha de solicitar al usuario cuantas contraseñas quiere encriptar. Este número, *numRepetitions*, debe ser mayor o igual a 1 y menor o igual a 100. En caso de que el número introducido no sea válido (inferior a 1 o superior a 100) se muestra un mensaje de error por el canal estándar de salida y se vuelve a

solicitar el número de contraseñas que se quieren encriptar. Este proceso se repite indefinidamente hasta que el usuario introduce un número válido.

Cuando se sepa cuantas contraseñas se deben encriptar, el algoritmo debe hacer lo siguiente:

1. Solicitar una contraseña mostrando un mensaje por el canal estándar de salida.
2. Leer la contraseña del canal estándar de entrada: leer uno a uno los 10 caracteres de la contraseña. Podéis suponer que los datos entrados son o bien caracteres en minúscula o dígitos y que el usuario teclea 10 exactamente. Es decir, los datos introducidos tienen el formato correcto.
3. Encriptar la contraseña: transformar cada carácter leído de la siguiente manera:
    a. Si es un dígito se multiplica por 9 y se le suma 1 y del resultado se coge el dígito menos significativo (el de las unidades).
    b. Si es una letra minúscula se cambia por el carácter que corresponde a la de tres posiciones más a la derecha del alfabeto, de manera cíclica. Es decir, la transformación sería:

        'a', 'b', 'c', ... 'x', 'y', 'z' → 'd', 'e, 'f', ... 'a', 'b', 'c'

    Estas conversiones de caracteres se deben hacer a partir del código ASCII.

4. Guardar cada carácter encriptado en el vector *password* en el orden entrado y mostrar el resultado por el canal estándar de salida.
5. Mostrar por el canal estándar de salida el número de dígitos y el número de caracteres que forman la contraseña.

<u>Observaciones:</u>

- Se deben usar estructuras iterativas, en sus diversas variantes, según cual sea la más apropiada en cada caso.
- Se deben declarar las variables auxiliares necesarias para diseñar el algoritmo.

## SOLUCIÓN

```
const
    MAX_CHARACTERS: integer = 10;
    MIN_REPETITIONS: integer = 1;
    MAX_REPETITIONS: integer = 100;
    NUMBER1_ASCII: integer = 48;
    NUMBER9_ASCII: integer = 57;
    NUMBER_LETTER_A: integer = 97;
    ENCRYPTED_POSITION: integer = 3;
    ALPHABET: integer = 26;
end const

algorithm UOCBookings
    var
        password: vector[MAX_CHARACTERS] of char;
        numRepetitions: integer = 0;
        i: integer = 1;
        j: integer = 0;
        numberDigits: integer;
        numberLetter: integer;
    end var
    while (numRepetitions < MIN_REPETITIONS or numRepetitions >
MAX_REPETITIONS) do
        writeString(”Numero de passwords a generar: ”);
        numRepetitions:=readInteger();
        if (numRepetitions < MIN_REPETITIONS or numRepetitions >
MAX_REPETITIONS) then
            writeString(”=============================================”);
            writeString(”ERROR: debes generar entre 1 y 100 passwords.”);
            writeString(”=============================================”);
        end if
    end while
    for i:=1 to ≤ numRepetitions do
        writeString(”Introduce tu password a encriptar: ”);
        password:=readString();
```

```
        numberDigits:= 0;
        numberLetter:= 0;
        for j:=0 to < MAX_CHARACTERS do
            if ((password[j] ≥ NUMBER1_ASCII) and (password[j]
NUMBER9_ASCII)) then
                password[j]:=((password[j] – NUMBER1_ASCII) * 9 + 1) mod
10 + ‘0’;
                numberDigits:= numberDigits + 1;
            else
                password[j]:=(((password[j] – NUMBER_LETTER_A) +
ENCRYPTED_POSITION) mod ALPHABET) + NUMBER_LETTER_A;
                numberLetter:= numberLetter + 1;
            end if
        end for
    writeString(“[PASSWORD );
    writeInteger(i);
    writeString(“]La password encriptada es: “);
    writeString(password);
    writeString(“Esta formada por “);
    writeInteger(numberDigits);
    writeString(“ y “);
    writeInteger(numberLetter);
    writeString(“letras“);
    end for
end algorithm
```

## Apartado b [20%]

Tal como está definida la encriptación, ¿es posible el proceso simétrico de desencriptación? ¿Somos capaces de "deshacer" la encriptación con el algoritmo del

apartado anterior? En caso afirmativo, describe textualmente (no hace falta hacer el algoritmo) que se debería hacer.

**SOLUCIÓN**

Sinceramente, no puedo decir que si al 100%, dado que por más cálculos que he realizado, no soy capaz de transformar las ‘a’, ‘b’ y ‘c’ nuevamente en ‘x’, ’y’ y ‘z’.

Para volver a obtener los dígitos numéricos iniciales, serían tan sencillo como coger el número encriptado, multiplicarlo por nueve, sumarle uno y hacer el módulo de 10. Es decir, la misma operación para encriptar que para desencriptar.

Para las letras, el rango entre d-z es fácilmente deducible. Se coge el código ASCII de la letra encriptada, se le resta el valor de ‘a’ minúscula para operar, al resultado se le resta la cifra de codificación (que en nuestro caso es 3), y se le hace el módulo de 26. El resto más el código ASCII de ‘a’ nos dará el valor de la letra original.

Pero como digo, no soy capaz de desarrollar este problema sin anidación de if/else if en los casos de ‘a’, ‘b’ y ‘c’. Es decir, no he sido capaz de hallar una fórmula que los convierta en un solo paso todas las letras.

## Ejercicio 2: [50%]

### Apartado a [80%]  Codificación

Codificar en lenguaje C el algoritmo del ejercicio 1.

### Apartado b [20%] Pruebas / Ejecución del algoritmo

Como en las PEC anteriores, se solicita que penséis cuatro juegos de prueba para el ejercicio, explicando porque se ha escogido cada uno de ellos. Para cada juego, indicad los valores de los datos de entrada y los datos de salida esperados

| Case num. | input | | output | | |
|---|---|---|---|---|---|
| | Num tries | Password | Encrypted password | Dígits | Characters |
| 1 | 0 | ERROR | - | - | - |
| 2 | 1 | qwertyuiop | tzhuwbxlrs | 0 | 10 |
| 3 | 2 | asdfghjkl7<br>zxcv24769m | dvgijkmno4<br>cafy97452p | 1<br>5 | 9<br>5 |
| 4 | 101 | ERROR | - | - | - |

# Criterios de corrección:

## En el ejercicio 1:

- Que se sigue la notación algorítmica utilizada en la asignatura. Ver el documento *Nomenclator* en la xWiki de contenido.
- Que se siguen las instrucciones dadas y el algoritmo responda al problema planteado.
- Que se utiliza correctamente la estructura iterativa, el tipo de datos vector y, si fuera necesario, las funciones de conversión de tipo.
- Que se razona correctamente la respuesta del apartado b de la primera pregunta.

## En el ejercicio 2:

- Que el programa cumpla con las condiciones dadas.
- Que el programa compila y funciona de acuerdo con lo que se pide.
- Que se respetan los criterios de estilo de programación C. Ver la “Guía de estilo de programación en C” que tenéis en la xWiki de contenido.