Fundamentos de Programación
PEC2 - 20211

Fecha límite de entrega: **04/10/2021 a las 23:59**

Apellidos: _____
Nombre: _____

Entrega

La PEC deberá entregarse antes del **día 04/10/2021 a las 23:59**.

La entrega debe realizarse en el apartado de **entregas de EC** del Aula de teoría. Se corregirá **únicamente** la **última versión** entregada dentro del plazo establecido.

Se debe entregar un único archivo en **formato ZIP**, que contenga:

- Un documento, en **formato PDF**, con el diseño algorítmico. No es necesario incluir todo el enunciado, solo las respuestas.
- El ***workspace* de Codelite**, con el proyecto en C del Ejercicio 2, tal y como se explica en el apartado correspondiente de la *xWiki*.

No respetar el formato de entrega puede conllevar que la actividad no pueda ser corregida, y en cualquier caso **penalizará** su evaluación.

A continuación se citan algunos ejemplos de formatos de entrega incorrectos:

- Documentos en formato distinto a PDF (.docx, .odt, .rtf, .txt, .pages, etc.).
- Ficheros y carpetas sueltos del proyecto en C (.c, .h, .workspace, etc.).

Actualizaciones del enunciado

Cualquier **aclaración**, actualización o corrección de posibles errores del enunciado se publicará en los **tablones del aula de teoría**. Es importante tener en cuenta estas aclaraciones para resolver la actividad, ya que **en caso de discrepancia, tendrán preferencia sobre el enunciado original**.

Objetivos de aprendizaje

Los **objetivos de aprendizaje** de esta PEC son los que se indican a continuación. Estos objetivos de aprendizaje constituyen a su vez, los **indicadores** en los que se basará la corrección y evaluación de la actividad.

Los objetivos de aprendizaje marcados con ★ aparecen por primera vez en esta PEC.

Tratamiento de datos — 30%

- TD1 - Ser capaz de elegir los tipos básicos de datos de forma correcta. 20%
- TD2 - Ser capaz de definir y usar los tipos enumerativos de forma correcta. 20%
- TD5 - Ser capaz de declarar y usar variables de forma correcta. 10%
- TD7 - Ser capaz de leer datos por el canal estándar de forma correcta. 15%
- TD8 - Ser capaz de mostrar datos por el canal estándar de forma correcta. 15%
- ★ **TD6 - Ser capaz de definir y usar constantes de forma correcta. 20%**

Diseño algorítmico — 40%

- ★ **DA1 - Ser capaz de construir expresiones lógicas correctas y compactas para tratar la información a partir de datos de forma correcta. 50%**
- DA4 - Ser capaz de diseñar un algoritmo funcional y sencillo que dé solución al problema planteado a través un flujo de ejecución de sentencias óptimo. 30%
- DA8 - Ser capaz de utilizar la notación algorítmica de forma adecuada y aplicar las buenas prácticas en cuanto a la nomenclatura y el uso de comentarios. 20%

Codificación — 20%

- CO1 - Ser capaz de construir un programa en C plenamente funcional, partiendo de un algoritmo diseñado previamente 20%
- CO2 - Ser capaz de cumplir con los requisitos formales de entrada y salida de datos en C. 20%
- CO3 - Ser capaz de seguir las normas de estilo establecidas para el lenguaje C. 10%
- CO4 - Ser capaz de superar los juegos de pruebas y casos extremos con éxito. 50%

Herramientas y entorno — 10%

- EN1 - Ser capaz de construir un programa completo con la estructura esperada, libre de errores y *warnings.* 100%

Enunciado

La compañía *UOCoworking* nos ha encargado el desarrollo de una aplicación para gestionar una red de centro de trabajo cooperativo. En concreto, deberemos gestionar los centros de trabajo colaborativo y los trabajadores.

Para dar respuesta a esta petición, a lo largo de las distintas PEC iremos desarrollando una pequeña parte de la aplicación, que se encargará de la gestión de los centros de trabajo colaborativo mediante la definición del modelo de datos y la implementación de distintas funcionalidades y algoritmos.

En las prácticas PR1 y PR2, completaremos la aplicación con la gestión de los trabajadores.

Aplicación a desarrollar en esta PEC: **obtener características especiales de un centro de trabajo colaborativo partiendo de sus datos básicos.**

El desarrollo de la aplicación en esta PEC tiene **cuatro partes**:

1. Interpretación de un algoritmo.
2. Diseño algorítmico.
3. Codificación en C.
4. Prueba del programa en C.

1. Interpretación de un algoritmo

Leer e interpretar el algoritmo desarrollado parcialmente que se expone a continuación.

```
const
    CATEGORY1: integer = 1;                  {Category 1 id}
    CATEGORY2: integer = 2;                  {Category 2 id}
    CATEGORY3: integer = 3;                  {Category 3 id}
end const

type
    tCoworkingType = {STARTUPS, FREELANCERS, RURAL, SPECIALIZED, GENERALIST};
end type

algorithm UOCoworking
    {Variable definition}
    var
        id: integer;                         {center id}
        coworkingType: tCoworkingType;                {center type}
        category: integer;                   {center category}
        price: real;                         {center price [Eur/month per space]}
        distanceFromCityCenter: real;        {center distance from city center [km]}
        hasMeetingRooms: boolean;
        hasKitchen: boolean;
        hasAuditorium: boolean;
        percentOccupation: real;
        {...}
    end var
    {...}
```

```
    {Data input}
    {Exercise 2.1}
    writeString("INPUT DATA");
    writeString("ID? (AN INTEGER) >>");
    id := readInteger();
    {...}
    writeString("PRICE [EUR]? (A REAL) >>");
    price := readReal();                    {center price [Eur/month per space]}
    writeString("DISTANCE FROM CITY CENTER [KM]? (A REAL) >>");
    distanceFromCityCenter := readReal();
    {...}

    {Exercise 2.2}
    {...}

    {Data processing}
    {Exercise 2.3}
    {...}

    {Exercise 2.4}
    {...}

    {Exercise 2.5}
    {...}

    {Data output}
    {Exercise 2.6}
    {...}
end algorithm
```

La estructura del algoritmo, así como los tipos de datos, constantes y variables que ya están declaradas en el algoritmo anterior servirán de base para el diseño algorítmico posterior.

2. Diseño algorítmico

Diseñar un algoritmo que incluya las funcionalidades que se detallan a continuación.

2.1. Entrada de datos. Diseña un algoritmo que lea por el canal estándar de entrada los siguientes datos correspondientes a **un centro de trabajo colaborativo,** almacenándolos en las variables ya declaradas, con el siguiente formato y significado:

Variable	Descripción	Tipo / validación
coworkingType	Tipo de centro	Valor de tipo enumerado, que puede ser uno de los siguientes: *STARTUPS*, *FREELANCERS*, *RURAL*, *SPECIALIZED*, *GENERALIST*.
category	Categoría del centro	Valor de tipo entero, que sigue el siguiente convenio (no se trata de un enumerado): - 1: Categoría 1 - BASIC - 2: Categoría 2 - STANDARD - 3: Categoría 3 - PREMIUM

hasKitchen	Indica si el centro tiene cocina	Valor de tipo booleano
hasAuditorium	Indica si el centro tiene auditorio	Valor de tipo booleano
percentOccupation	Porcentaje de ocupación total del centro (salas de trabajo ocupados respecto el total)	Valor de tipo real

La lectura de datos implica una interacción con el usuario y, por lo tanto, éste deberá ser informado de lo que se pide, el tipo de datos básico de la variable, las unidades, el rango válido o los valores posibles (en caso que los hubiera).

Por ejemplo, para leer la edad de una persona y guardarla en una variable de nombre age*, una posible implementación sería la siguiente:*

```
writeString("CURRENT AGE [YEARS]? (AN INTEGER) >>");
age := readInteger();
```

En lenguaje algorítmico, los tipos de datos enumerados no tienen una correspondencia numérica, y por este motivo no pueden leerse como si fueran enteros. En su lugar, en esta PEC podemos usar la/s siguiente/es función/es, que podemos considerar definida/s "adhoc":

```
{...}
writeString("TYPE ? (...)");
coworkingType := readType();
{...}
```

2.2. Entrada de datos. Una empresa quiere alquilar salas de trabajo en el centro para un determinado número de trabajadores. Diseña un algoritmo que lea por el canal estándar de entrada la siguiente información adicional necesaria para la aplicación:

- El número total de trabajadores de la empresa, que se distribuirán en las salas de trabajo alquiladas en el centro.
- Un porcentaje de descuento sobre el precio de alquiler mensual de **una sala de trabajo en el centro**, que puede tener decimales [0.00% - 100.00%].

Para guardar la información, deberán declararse las variables que se consideren necesarias.

La lectura de datos implica una interacción con el usuario y, por lo tanto, deberá ser informado de lo que se pide, el tipo de datos básico de la variable, las unidades, el rango de valores esperado o conjunto de valores posibles (en caso que los hubiera y fuera necesario).
Por ejemplo, para leer la edad de una persona y guardarla en una variable de nombre age, una posible implementación sería la siguiente:

```
writeString("CURRENT AGE [YEARS]? (AN INTEGER, 0-199) >>");
age := readInteger();
```

En el proceso de lectura de datos, se presupondrá que el usuario respetará el tipo de datos, rango de valores esperado o conjunto de valores posibles y, por lo tanto, no será necesario realizar ninguna comprobación al respecto.

2.3. Procesamiento de datos. Diseñar una expresión que, basándonos en los datos introducidos por el usuario, devuelva como resultado si el centro de trabajo colaborativo tiene o no condiciones ideales.
Se considera que un centro tiene condiciones ideales si cumple con **todas** las condiciones siguientes:

- Está a una distancia inferior a 5 km del centro de la ciudad.
- El precio de alquiler mensual de una sala de trabajo es inferior o igual a 100€.
- Dispone de cocina.

2.4. Procesamiento de datos. Diseñar una expresión que, basándonos en los datos introducidos por el usuario, devuelva como resultado si el centro de trabajo necesita una campaña publicitaria de promoción.
Se considera que el centro necesita una campaña si cumple con **todas** la condiciones siguientes:

- Tiene una ocupación inferior al 50%.
- Se encuentra en las afueras de la ciudad (la distancia al centro de la ciudad es igual o superior a 5 km)

2.5. Procesamiento de datos. Diseñar una expresión que, basándonos en los datos introducidos por el usuario, devuelva como resultado el precio **total** del alquiler mensual del centro para la empresa del apartado 2.2. Para ello, debe tenerse en cuenta que:

- En cada sala de trabajo del centro puede haber **un máximo de dos trabajadores**.

- El precio base (leído en el algoritmo del enunciado) es para **cada una de las salas** de trabajo alquiladas.
- Sobre el precio base debe aplicarse el descuento de promoción informado en el apartado 2.2.

Para el cálculo del precio total, se presupondrá que el centro dispone de suficientes salas de trabajo libres para albergar todos los trabajadores introducidos por el usuario.

En el proceso de obtención de los resultados anteriores, no es necesario el uso de ninguna estructura de control alternativa if...else *(y de hecho se considerará un error de diseño).*

Se podrán definir las variables auxiliares y constantes que se consideren necesarias para guardar el resultado de cada una de las expresiones solicitadas, escogiendo el tipo de datos más apropiado.

Hay que evitar siempre que sea posible el uso de valores numéricos directos en el algoritmo, y utilizar en su lugar constantes previamente definidas.

2.6. Salida de datos. Completar el algoritmo y **mostrar por el canal estándar de salida**, los resultados de las expresiones diseñadas anteriormente, indicando claramente a qué corresponde cada resultado, así como sus unidades (si las hubiera), rango de valores u otra información relevante.

Siguiendo con el ejemplo anterior, una posible solución para la interacción con el usuario sería la siguiente:

```
writeString("AGE [YEARS]:");
writeInteger(age);
```

3. Codificación en C

Codificar en C el algoritmo diseñado anteriormente.

El programa en C debe cumplir con las siguientes particularidades:

- Los números reales deben mostrarse con una precisión de dos decimales.

En el enunciado se declara un conjunto de variables, pero solo se leen por teclado algunas de ellas. En C, esto originará uno o más warning *debido a la existencia de variables declaradas sin utilizar. Estos* warning *son normales en esta PEC y no supondrán ninguna penalización.*

4. Prueba del programa en C

Ejecutar y superar los juegos de prueba automáticos disponibles en la herramienta ACME.

*El proceso de validación y corrección de la herramienta ACME se basa en una comprobación **literal** de la salida obtenida por el programa sometido a prueba, con los resultados esperados de los juegos de prueba introducidos previamente. Por ello, los textos de la interfaz de usuario deben ser exactamente idénticos a los esperados.*

Hay que tener en cuenta que el proceso de copiar y pegar textos entre distintos editores, entornos y herramientas puede generar caracteres ocultos que hagan que la comparación de los textos literales sea incorrecta a pesar de que los textos literales aparentemente sean idénticos.

*A modo de ejemplo, a continuación se muestra una ejecución del programa con datos aleatorios, así como la entrada y salida de la herramienta ACME, donde pueden verse los textos literales esperados. En caso de discrepancia, los textos literales que aparecen en la herramienta ACME **tendrán preferencia** respecto a los textos literales que aparecen en este enunciado.*