Fundamentos de Programación
PEC4 - 20211

Fecha límite de entrega: **18/10/2021 a las 23:59**

Apellidos: _____
Nombre: _____

Entrega

La PEC deberá entregarse antes del **día 18 de octubre 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 **en negrita** aparecen por primera vez en esta PEC.

Tratamiento de datos 20%

- TD3 - Ser capaz de declarar y utilizar vectores de forma correcta. 50%
- Indicadores de E/S 25%:
  - TD7 - Ser capaz de leer datos por el canal estándar de forma correcta.
  - TD8 - Ser capaz de mostrar datos por el canal estándar de forma correcta.
- Resto de indicadores de tratamiento de datos 25 %:
  - TD1 - Ser capaz de elegir los tipos básicos de datos de forma correcta.
  - TD2 - Ser capaz de definir y usar los tipos enumerativos de forma correcta.
  - TD5 - Ser capaz de declarar y usar variables de forma correcta
  - TD6 - Ser capaz de definir y usar constantes de forma correcta

Diseño algorítmico 50%

- DA1 - Ser capaz de construir expresiones lógicas correctas y compactas para tratar la información a partir de datos de forma correcta. 10%
- DA2 - Ser capaz de construir estructuras alternativas compactas y funcionales, y diseñar árboles de decisión. 10%
- **DA3 - Ser capaz de construir estructuras iterativas de forma correcta y óptima. 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. 20%
- 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. 10%

Codificación 20%

- CO1 - Ser capaz de construir un programa en C plenamente funcional, partiendo de un algoritmo diseñado previamente adaptándolo a las particularidades del lenguaje C 40%
- 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%

- 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 una red de centros de trabajo ideales, partiendo de datos guardados en vectores.**

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

    MIN_COWORKCENTERS: integer = 1;          {Min. Number of cowork centers}
    MAX_COWORKCENTERS: integer = 5;          {Max. Number of cowork centers}
    MAX_DISTANCE: real = 5.0;                {Max. acceptable distance}
    MAX_PRICE: real = 100.0;                 {Max. acceptable price}
end const

type
    tCoworkingType = {STARTUPS, FREELANCERS, RURAL, SPECIALIZED, GENERALIST}
```

```
end type

algorithm UOCoworking
    {Variable definition}
    var
        coworkCenterIdVector: vector[MAX_COWORKCENTERS] of integer;
        coworkTypeVector: vector[MAX_COWORKCENTERS] of tCoworkingType;
        priceVector: vector[MAX_COWORKCENTERS] of real;
        distanceFromCityCenterVector: vector[MAX_COWORKCENTERS] of real;
        hasKitchenVector: vector[MAX_COWORKCENTERS] of boolean;

        numCoworkCenters: integer;
        coworkType: tCoworkingType;

        acceptableCoworkCentersList: vector[MAX_COWORKCENTERS] of integer;
        numAcceptableCoworkCenters: integer;
        {...}

    end var

    {Data input}
    {Exercise 2.1}
    writeString("INPUT DATA");
    writeString("NUMBER OF COWORK CENTERS ? (1-5) >>");
    readInteger(numCoworkCenters);    {num. of cowork centers in the city}
    {Data validation}
    {...}

    {Exercise 2.2}
    {...}

    {Exercise 2.3}
    writeString("TYPE? (1-STARTUPS, 2-FREELANCERS, 3-RURAL, 4-SPECIALIZED,
        5-GENERALIST) >>");
    coworkType := readType();

    {Data processing}
    {Exercise 2.4}
    {...}

    {Data output}
    {Exercise 2.5}
    {...}

end algorithm
```

La estructura del algoritmo, así como los tipos de datos, constantes y variables que ya están declaradas 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. Lectura de datos. Leer por el canal estándar de entrada, la siguiente información necesaria para la aplicación:

- El número de centros de trabajo colaborativo que existen en una determinada ciudad.

Para la lectura y validación de esta información, se aplicarán las siguientes reglas:

- El número de centros de trabajo colaborativo debe ser **superior a cero e inferior o igual al máximo permitido**. En caso contrario, debe mostrarse un mensaje de error y volver a pedir el dato. El proceso se **repetirá indefinidamente** hasta que el usuario introduzca un valor válido.

Este apartado se encuentra desarrollado parcialmente en el algoritmo del enunciado.

Para guardar la información, podrán declararse las variables, constantes y tipos de datos que se consideren necesarios, pudiéndose combinar con las que ya están declaradas en el algoritmo parcial del enunciado.

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

2.2. Lectura de datos. Leer por el canal estándar de entrada, los siguientes datos **para cada uno de los centros de trabajo colaborativo** de la ciudad, y almacenarlos en el **vector** correspondiente.

Variable	Descripción	Tipo / validación
coworkCenterId	Código identificador del centro	Valor de tipo entero
coworkType	Tipo de centro	Valor de tipo enumerado, que puede ser uno de los siguientes: *STARTUPS*, *FREELANCERS*, *RURAL*, *SPECIALIZED*, *GENERALIST*.
price	Precio del alquiler mensual de una sala de trabajo de trabajo [eur]	Valor de tipo real
distanceFromCityCenter	Distancia del centro al centro de la ciudad [km]	Valor de tipo real
hasKitchen	Indica si el centro tiene cocina	Valor de tipo booleano

Para guardar la información, deberán declararse las variables, constantes y tipos de datos que se consideren necesarios, pudiéndose combinar con las que ya están declaradas en el algoritmo parcial del enunciado.

En este punto de desarrollo del algoritmo, el número de centros a introducir es un dato conocido y válido, que ha sido comprobado anteriormente.

El resto de datos de cada centro no son necesarios para el desarrollo de la aplicación en esta PEC.

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 ? (...)");
coworkType := readType();
{...}
```

En el proceso de lectura de datos de este apartado, 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.

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

2.3. Lectura de datos. Leer por el canal estándar de entrada, la siguiente información adicional necesaria para la aplicación:

- Un tipo de centro de trabajo determinado, que puede ser: *STARTUPS, FREELANCERS, RURAL, SPECIALIZED, GENERALIST*.

Este apartado se encuentra desarrollado en el algoritmo del enunciado.

2.4. Procesamiento de datos. Calcular los siguientes datos, mediante la combinación de expresiones y estructuras algorítmicas, basándose en la información introducida por el usuario:

- Un vector que contenga los identificadores de los centros que sean del **mismo tipo que el indicado por el usuario** en el apartado 2.3, y que además tengan condiciones ideales. Para ser considerado ideal, un centro de trabajo debe estar a una distancia **inferior a 5 km** del centro de la ciudad, el precio de alquiler de una sala de trabajo del centro debe ser **inferior o igual a 100 €** y debe disponer de cocina. Este **vector conformará la red de centros ideales**.
- El número total de centros de trabajo que conforman la red de centros ideales.

Para obtener y guardar el resultado, deberán declararse las variables, constantes y tipos de datos que se consideren necesarios, pudiéndose combinar con las que ya están declaradas en el algoritmo parcial del enunciado.

En lenguaje algorítmico, los tipos de datos enumerados no tienen una correspondencia numérica, y por este motivo no pueden manipularse como si fueran enteros. En su lugar, hay que usar siempre sus correspondientes etiquetas literales.

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

2.5. Salida de datos. Completar el algoritmo y mostrar por el canal estándar de salida los resultados, de acuerdo con el siguiente flujo de ejecución:

- En caso de que se pueda conformar una red de centros ideales (esto es, que existan centros de trabajo que sean del mismo tipo solicitado al usuario y que cumplan con las condiciones ideales), mostrar el número total y los identificadores de todos ellos. En caso contrario, mostrar un mensaje informativo y finalizar la ejecución.

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.
- Los tipos enumerados en C, tienen una correspondencia numérica. Por este motivo, podrán leerse y mostrarse como si fueran enteros, siguiendo las reglas establecidas y siempre mediante la ayuda de una interfaz de usuario para interpretar los datos a leer/mostrar.

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.