Fundamentos de Programación
PEC7 - 20211

Fecha límite de entrega: **15/11/2021 a las 23:59**

Apellidos: _____
Nombre: _____

Entrega

La PEC deberá entregarse antes del **día 15 del 11 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%

- TD4 - Ser capaz de definir y utilizar tipos estructurados de datos de forma correcta 40%
- Indicadores de E/S 40%:
  - 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 20%:
  - 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 15%
- DA2 - Ser capaz de construir estructuras alternativas compactas y funcionales, y diseñar árboles de decisión 15%
- 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 10%
- DA5 - Ser capaz de declarar y utilizar acciones y funciones de forma correcta 50%
- DA7 - Ser capaz de utilizar la notación algorítmica de forma adecuada y aplicar las buenas prácticas en cuanto a la nomenclatura y el uso de comentario. 10%

Codificación 15%

- 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%

Herramientas y entorno 15%

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

Enunciado

Siguiendo con la ayuda que proporcionamos a la compañía UOCoworking, nos han pedido nuestra colaboración para crear un programa que les ayude a gestionar los datos de sus trabajadores y centros de trabajo colaborativo.

En esta PEC continuaremos trabajando la modularidad, creando acciones con parámetros de salida y dividiendo el código en acciones y funciones que nos permita reutilizarlo. En este sentido, el ejercicio de codificación no se limita a traducir a C el ejercicio de lenguaje algorítmico. Se pide, además, estructurar el proyecto de Codelite en carpetas y ficheros.

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, mediante la definición del modelo de datos y la implementación de distintas funcionalidades y algoritmos.

En las prácticas PR1 y PR2, se completará la aplicación con la gestión de los usuarios.

Aplicación a desarrollar en esta PEC: **Generar una puntuación de un centro de trabajo en función de ciertas características, y comparar dos centros para obtener cuál de estos es mejor según su puntuación.**

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}


    POINTS_FACILITIES:  real = 5.0;     {Points for facilities}
    OPTIMAL_WORKSPACES: integer = 3;    {Optimal amount of workspaces}
```

```
end const

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

    tCoworkingCenter = record
        name: string;
        city: string;
        category: integer;
        centerType: tCoworkingType;
        numSpaces: integer;
        price: real;
        distanceFromCityCenter: real;
        hasMeetingRooms: boolean;
        hasAuditorium: boolean;
    end record

end type

{Exercise 2.1}
{…}

{Exercise 2.2}
{…}

algorithm UOCoworking
    var
        center1: tCoworkingCenter;
        center2: tCoworkingCenter;
   {…}
    end var

   {Exercise 2.3}
   {Data input & processing}
   {…}

   {Data output}
   writeString("RESULT");
   writeString("THE BEST CHOICE IS:");

   {…}
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.

Nota: En lenguaje algorítmico podéis utilizar las acciones *readCenter(…)* y *writeCenter(…)* ya diseñadas en la anterior PEC6, por lo que no es necesario que vuelvan ser implementadas. Recordad que sus cabeceras son:

action *readCenter(**out** center: tCoworkingCenter);*

action *writeCenter(**in** center: tCoworkingCenter);*

Para el desarrollo, se deberá optar entre una acción o función. Así mismo, deberán definirse el número, tipo y clase de parámetros (entrada, salida, entrada/salida) que se consideren necesarios y, en su caso, el tipo de datos de retorno. Finalmente, 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.

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

2.1. Desarrollo de funciones/acciones. Desarrollar la función/acción *pointsCenter(...)* que dados los datos de precio (*price*) y distancia *(distance)* aceptables aplicados a un determinado centro, devuelva una puntuación alcanzada y calculada a partir de la suma de los siguientes conceptos:

- 5 puntos si tiene sala de reuniones o auditorio.
- 5 puntos si tiene una cantidad óptima de al menos 3 salas de trabajo.
- El resultado de la operación: 100 * (*price* - center.price) / *price*; a menor precio que el aceptado, más puntuación para el centro.
- El resultado de la operación: 100 * (*distance* - center.distanceFromCityCenter) / *distance*; a menor distancia del centro de la ciudad que la aceptada, más puntuación para el centro.

2.2. Desarrollo de funciones/acciones. Desarrollar la función/acción *cmpPointsCenter(...)* que comparará la puntuación de dos centros, haciendo uso de la acción/función *pointsCenter(...)* implementada en el anterior apartado. La acción/función deberá realizar lo siguiente:

- Leer del canal estándar de entrada el precio y la distancia que nos parecen aceptables.
- Calcular el número de puntos de los dos centros a partir de los parámetros de entrada y retornar como salida una copia del centro que tiene el número superior de puntos. En caso de empate, devuelve el primer centro.

2.3. Procesamiento y salida de datos. Completar el algoritmo para que:

- Lea del canal estándar de entrada la información de dos centros. Es obligatorio utilizar la acción *readCenter(…)*.
- Muestre por el canal estándar de salida los campos del centro (usando la acción *writeCenter(...*) que tiene más puntos, haciendo uso de las acciones y funciones que se han diseñado previamente.

3. Codificación en C

Codificar en C el algoritmo diseñado anteriormente y, además, estructurar el código en carpetas. Concretamente hay que realizar lo siguiente:

1. Crear un nuevo proyecto en Codelite llamado *UOCoworking,* creando una carpeta *include* y una carpeta *src* en este proyecto siguiendo las explicaciones que se pueden encontrar en la unidad de la xWiki *Modularidad en Codelite* de la asignatura.
2. Dentro de la carpeta *include*, crea un nuevo archivo llamado ***coworkingcenter.h*** que contenga la declaración de los tipos estructurados *tCoworkingType*, y *tCoworkingCenter,* así como de las constantes necesarias.
3. Como ya hemos visto en lenguaje C no se puede utilizar el operador = para asignar el contenido de una tupla a otra, es necesario hacer la asignación de valores campo a campo. A continuación os proporcionamos el código C de la acción ***copyCenter(...)*** necesaria para asignar el contenido de una tupla de tipo *tCoworkingCenter* a otra.

```
void copyCenter (tCoworkingCenter centerSrc, tCoworkingCenter *centerDst)
{
    strcpy(centerDst->name, centerSrc.name);
    strcpy(centerDst->city, centerSrc.city);
    centerDst->category = centerSrc.category;
    centerDst->centerType = centerSrc.centerType;
    centerDst->numSpaces = centerSrc.numSpaces;
    centerDst->price = centerSrc.price;
    centerDst->distanceFromCityCenter = centerSrc.distanceFromCityCenter;
    centerDst->hasMeetingRooms = centerSrc.hasMeetingRooms;
    centerDst->hasAuditorium = centerSrc.hasAuditorium;
}
```

4. Copiar las cabeceras de todas las acciones/funciones que hay que usar (***readCenter(...), writeCenter(...), copyCenter(...), pointsCenter(...)*** y ***cmpPointsCenter(...)***) en el archivo ***coworkingcenter.h***.
5. Dentro de la carpeta *src,* crea un nuevo archivo ***coworkingcenter.c*** donde hay que incluir el código de las acciones/funciones a implementar (***readCenter(...), writeCenter(...), copyCenter(...), pointsCenter(...)*** y ***cmpPointsCenter(...)***). Tened en cuenta las acciones ***readCenter(...), writeCenter(...)*** ya diseñadas e implementadas en la PEC6, **copiad el código en este nuevo proyecto para**

reutilizarlas al igual que la acción *copyCenter(...)* proporcionada en el punto 3 de este apartado

6. Codificar el algoritmo anterior dentro de la función principal ***main.c*** ubicada en la carpeta *src*.
7. Comprobar que compila y funciona correctamente.

El programa en C debe cumplir con las siguientes particularidades:

- Los campos *name* y *city* del tipo de datos estructurado deben tener una longitud máxima de 15 caracteres.
- 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, **podéis recuperar dichos textos entrando dentro de la herramienta**, están disponibles en el enunciado asociado a la PEC.*

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.