# Fundamentos de Programación
# PEC4 - 20221

## Fecha límite de entrega: **02/11/2022 a las 23:59**

Apellidos: PEÑALVER FERNÁNDEZ
Nombre: ADRIÁN

---

# Entrega

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 en el Registro de Evaluación Contínua (REC) 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 (ficheros .c, .h y .workspace) del ejercicio 2, tal y como se explica en el apartado correspondiente de la *xWiki*.

**No respetar el formato de entrega puede comportar que no se pueda corregir la actividad.** En cualquier caso, penalizará su evaluación.

# 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 manera correcta.
  - TD8 - Ser capaz de mostrar datos por el canal estándar de manera correcta.
- Resto de indicadores de tratamiento de datos 25 %:
  - TD1 - Ser capaz de elegir los tipos básicos de datos de manera correcta.
  - TD2 - Ser capaz de definir y usar los tipos enumerativos de manera 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 manera 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 manera 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%**
- **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 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%

## 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 *UOCCinema,* entidad propietaria de una cadena de cines, nos ha encargado el desarrollo de una aplicación para gestionar sus espacios, salas, sesiones y películas exhibidas.

Para dar respuesta a esta petición, a lo largo de algunas de las PECs iremos desarrollando una pequeña parte de la aplicación final que se encargará de la gestión de la cadena de cines mediante la definición del modelo de datos, y la implementación de distintas funcionalidades y algoritmos.

**Aplicación a desarrollar en esta PEC**: de un grupo de cines, obtener cuál de estos tiene mejor 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

Revisad el algoritmo desarrollado parcialmente que se expone a continuación. Este algoritmo os servirá de base para generar vuestra solución. **No es necesario realizar ninguna entrega relacionada con este apartado**.

```
const
    MIN_CINEMAS: integer = 1;           {Min. number of cinemas}
    MAX_CINEMAS: integer = 10;          {Max. number of cinemas}

    SCREENS_3: integer = 3;             {3 screens}
    SCREENS_6: integer = 6;             {6 screens}

    POINTS_LE_3_SCREENS: real = 2.0;    {Points less/equals 3 screens}
    POINTS_LE_6_SCREENS: real = 3.0;    {Points less/equals 6 screens}
    POINTS_GT_6_SCREENS: real = 4.0;    {Points greater than 6 screens}

    PARKING_POINTS_INCREASE: real = 1.5; {Points increase if has parking}
end const

type
    tCinemaType = {PREMIERE, RERELEASE, INDIE, OTHER}
end type
```

```
algorithm UOCCinema
    {Variable definitions}
    var
        cinemaIdVector: vector[MAX_CINEMAS] of integer;
        cinemaTypeVector: vector[MAX_CINEMAS] of tCinemaType;
        numScreensVector: vector[MAX_CINEMAS] of integer;
        hasParkingVector: vector[MAX_CINEMAS] of boolean;
        totalPointsVector: vector[MAX_CINEMAS] of real;

        numCinemas: integer;
        bestCinema: integer;

        {...}
    end var

    {Exercise 2.1}
    {Data input}
    writeString("INPUT DATA");
    writeString("NUMBER OF CINEMAS (1-10)? >>");
    numCinemas:= readInteger();

    {Data validation}
    {...}

    {Exercise 2.2}
    {...}
    for i:= 1 to numCinemas do

        {Data input}
        {...}

        {Data processing}
        {....}

    end for

    {Data output}
    {Exercise 2.3}
    writeString("RESULT");
    writeString("BEST CINEMA [ID]:");
    {...}

end algorithm
```

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

# 2. Diseño algorítmico

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

**2.1. Lectura de datos.** El algoritmo ha de leer por el canal estándar de entrada, la siguiente información necesaria para la aplicación:

- El número de cines que se darán de alta.

Para la lectura y validación de esta información, se aplicará la siguiente regla:

- El número de cines debe **estar comprendido entre 1 y 10.** 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.*

**2.2. Lectura y procesamiento de datos.** El algoritmo también tiene que leer por el canal estándar de entrada, los siguientes datos **para cada uno de los cines a registrar**, y almacenarlos en el **vector** correspondiente.

| Variable | Descripción | Tipo / validación |
|---|---|---|
| cinemaIdVector | Conjunto de datos que guarda los códigos únicos de identificación de cada cine. | Vector de valores de tipo entero. |
| cinemaTypeVector | Conjunto de datos que guarda el tipo de cada cine. | Vector de valores de tipo tCinemaType. |
| numScreensVector | Conjunto de datos que guarda el número de las salas de proyección de cada cine. | Vector de valores de tipo entero, comprendido entre 1 y 10. |
| hasParkingVector | Conjunto de datos que indica si el cine posee o no parking para sus clientes. | Vector de valores de tipo booleano. |
| totalPointsVector | Conjunto de datos que guarda los puntos obtenidos por cada cine. | Vector de valores de tipo real. |

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

*En lenguaje algorítmico, los tipos de datos enumerados no tienen una correspondencia numérica, y por este motivo no pueden leerse ni escribirse 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 ? (...)");
cinemaType:= readCinemaType();
{...}
writeString("TYPE (1-PREMIERE, 2-RERELEASE, 3-INDIE, 4-OTHER):");
writeCinemaType(cinemaType);
{...}
```

*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****.*

Finalmente, el algoritmo tiene que calcular mediante la combinación de expresiones y estructuras algorítmicas y basándose en la información introducida por el usuario, el mejor cine en función de su puntuación. Dicha puntuación se calcula de la siguiente forma:

- Si el cine tiene un número igual o inferior a 3 salas de proyección, se le asignará 2 puntos.
- Si el cine tiene un número de salas de proyección superior a 3 e inferior o igual a 6, se le asignará 3 puntos.
- Si el cine tiene un número de salas de proyección superior a 6, se le asignará 4 puntos.
- Si el cine posee parking propio, su puntuación incrementará 1.5 veces su valor.

Una vez se tiene la puntuación, en caso de empate, prevalecerá el primer cine en la posición del vector, por ejemplo, si empatan los cines 3, 6 y 7, el seleccionado será el cine 3.

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

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

- Cine con mayor puntuación, mostrando el valor de todos sus campos y su puntuación obtenida.
  El orden de los datos a mostrar para cada cine será el siguiente:

  1. Identificador del cine.
  2. Tipo de cine.
  3. Número de pantallas.
  4. Si el cine dispone o no de parking.

*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.*

# 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, **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.*

```
{ADRIAN PEÑALVER FERNANDEZ}
{PEC_4}
{01/11/2022}
{ALGORITMO DE WHILE Y FOR}

const
   MIN_CINEMAS: integer = 1; {Min. number of cinemas}
   MAX_CINEMAS: integer = 10; {Max. number of cinemas}
   SCREENS_3: integer = 3; {3 screens}
   SCREENS_6: integer = 6; {6 screens}
   POINTS_LE_3_SCREENS: real = 2.0; {Points less/equals 3 screens}
   POINTS_LE_6_SCREENS: real = 3.0; {Points less/equals 6 screens}
   POINTS_GT_6_SCREENS: real = 4.0; {Points greater than 6 screens}
   PARKING_POINTS_INCREASE: real = 1.5; {Points increase if has parking}
end const

type
   tCinemaType = {PREMIERE, RERELEASE, INDIE, OTHER}
end type



algorithm UOCCinema

{Variable definitions}
   var
      cinemaIdVector: vector[MAX_CINEMAS] of integer;
      cinemaTypeVector: vector[MAX_CINEMAS] of tCinemaType;
      numScreensVector: vector[MAX_CINEMAS] of integer;
      hasParkingVector: vector[MAX_CINEMAS] of boolean;
      totalPointsVector: vector[MAX_CINEMAS] of real;
      numCinemas: integer;
      bestCinema: integer;
      maxPoint: real;

   end var

{Exercise 2.1}
{Data input}

writeString("INPUT DATA");
writeString("NUMBER OF CINEMAS (1-10)? >>");
numCinemas:= readInteger();

{Data validation}

While MIN_CINEMAS > numCinemas or numCinemas > MAX_CINEMAS do

   writeString("INVALID DATA, TRY AGAIN!”);
   writeString("NUMBER OF CINEMAS (1-10)? >>");
   numCinemas:= readInteger();

End while

{Exercise 2.2}

maxPoint:= 0;

for i:= 1 to numCinemas do

{Data input}
       writeString("CINEMA #");
       writeInteger(i+1);

       writeString("ID (AN INTEGER)? >>");
       cinemaIdVector[i]:= readInteger();

       writeString("TYPE (1-PREMIERE, 2-RERELEASE, 3-INDIE, 4-OTHER)? >>");
       cinemaType:= readCinemaType();

       writeString("SCREENS (1-10)? >>");
       numScreensVector[i]:= readInteger();

       writeString("HAS PARKING (0-FALSE, 1-TRUE)? >>");
       hasParkingVector[i]:= readBoolean();
```

```
{Data processing}

        If numScreensVector[i]<= SCREENS_3 then
                totalPointsVector[i]:= POINTS_LE_3_SCREENS;

        else
                if SCREENS_3 < numScreensVector[i] and numScreensVector[i] <= SCREENS_6
        then
                        totalPointsVector[i]:= POINTS_LE_6_SCREENS;

                else
                        totalPointsVector[i]:= POINTS_GT_6_SCREENS;
                end if
        end if

        if hasParkingVector[i] then

        totalPointsVector[i]:= totalPointsVector[i]* PARKING_POINTS_INCREASE;

        end if

        if totalPointsVector[i] > maxPoint then

        maxPoint:= totalPointsVector[i];
        bestcinema:= i;

        end if
end for


{Data output}
{Exercise 2.3}

writeString("RESULTS");
writeString("BEST CINEMA [ID]:");
writeInteger( cinemaIdVector[bestCinema]);

writeString("TOTAL POINTS:");
writeReal( totalPointsVector[bestCinema]);

writeString("SCREENS:");
writeInteger( numScreensVector[bestCinema]);

writeString("TYPE (1-PREMIERE, 2-RELEASE, 3-INDIE, 4-OTHER): ");
WriteCinemaType(cinemaType);

writeString("HAS PARKING (0-FALSE, 1-TRUE): ");
writeBoolean(hasParkingVector[bestCinema]);

end algorithm
```