Fundamentos de Programación
PEC6 - 20221

Fecha límite de entrega: **16/11/2022 a las 23:59**

Apellidos: Peñalver Fernández
Nombre: Adrián

Entrega

La entrega debe realizarse en el apartado de **entregas de REC** 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 el tablón 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 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 PEC 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: **dados tres cines, determinar cuál de ellos tiene mayor aforo.**

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

```
const
    NUM_CINEMAS: integer = 3;  {Number of cinemas}
    MIN_SCREENS: integer = 1;  {Min. cinema screens}
    MAX_SCREENS: integer = 5;  {Max. cinema screens}
end const

type
    tScreen = record
        screenId: integer;
        capacity: integer;
    end record


    tCinema = record
        name: string;
        screenVector: vector[MAX_SCREENS] of tScreen;
```

```
        nScreens: integer;
    end record


end type

{Exercise 2.1}
{...}

{Exercise 2.2}
{...}

{Exercise 2.3}
{...}


algorithm UOCCinema
    {Variable definitions}
    var
        cinemaVector: vector[NUM_CINEMAS] of tCinema;
        bestCinema: tCinema;
        {...}
    end var

    {Exercise 2.4}
    {Data input}
    writeString("INPUT DATA");
    for i:= 1 to NUM_CINEMAS do
        {...}
    enf for

    {Data processing}
    {...}
    for i:= 1 to NUM_CINEMAS do
        {...}
    enf for

    {Data output}
    writeString("BEST CINEMA\n");
    {...}
    writeString("MAX CAPACITY\n");
    {...}
end algorithm
```

Definición de tipos de datos.

- El tipo de datos estructurado *tScreen* representa una pantalla de proyección de películas en un cine. Los campos son los siguientes:

Campo	Descripción	Tipo / validación
screenId	Identificador único de la sala de proyección.	Valor de tipo *integer*
capacity	Aforo máximo de la sala de proyección.	Valor de tipo *integer*

- El tipo de datos estructurado *tCinema* representa un cine perteneciente a la compañía *UOCCinema* . Los campos son los siguientes:

Campo	Descripción	Tipo / validación
name	Nombre del cine, sin espacios en blanco.	Valor de tipo *string*
screenVector	Salas de proyección disponibles en el cine, la cantidad de estas irá determinada por el campo nScreen.	Valor de tipo *vector de tScreen*
nScreens	Número de salas de proyección, determina la cantidad de elementos que posee el vector screenVector.	Valor de tipo *integer*

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

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

2.1. Desarrollo de funciones/acciones. Desarrollar la acción *readCinema* que reciba como parámetro una variable de nombre *cinema*, de tipo *tCinema* sin inicializar, y devuelva la misma variable con los campos informados con los datos leídos por el canal estándar de entrada.

Deberá introducirse la cantidad de salas de proyección disponibles en el cine, y deberá estar comprendida entre 1 y 5, 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.

Seguidamente se introducirán los campos correspondientes a cada sala de proyección, donde la capacidad será un valor comprendido entre 25 y 50. En este caso 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**.

A continuación se da la descripción del parámetro.

Parámetro	Tipo	Clase de parámetro	Descripción
cinema	*tCinema*	Salida	Salida - los campos de la variable *cinema han* inicializado con los datos leídos por el canal estándar de entrada.

El parámetro cinema representa los datos básicos de un cine, definida mediante el tipo estructurado tCinema. Por ello, para leer los datos del cine, hay que leer todos los campos que forman parte de la variable.

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. Desarrollo de funciones/acciones. Desarrollar la acción *writeCinema.* Esta recibe como parámetro de entrada una variable de nombre *cinema,* de tipo *tCinema* ya inicializada, y muestra por el canal estándar de salida la siguiente información:

- Nombre del cine.
- Número de pantallas de proyección.
- Campos correspondientes a cada pantalla de proyección.

A continuación se da la descripción del parámetro.

Parámetro	Tipo	Clase	Descripción
cinema	tCinema	Entrada	Parámetro *cinema* inicializado a mostrar por el canal de salida estándar.

El parámetro cinema representa los datos de un cine mediante el tipo estructurado tCinema. Por ello, para mostrar los datos del cine, hay que mostrar todos los campos que forman parte de la variable.

2.3. Desarrollo de funciones/acciones. Desarrollar la función *getTotalCapacity*. Mediante esta función se obtendrá el aforo total de espectadores que puede albergar un cine determinado.

A continuación se da la descripción del parámetro y el valor retornado.

Parámetro	Tipo	Clase	Descripción

cinema	tCinema	Entrada	Cine.
	integer	Valor de retorno de la función	Aforo total de espectadores.

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

- Leer por el canal estándar de entrada los datos de tres cines.
- Recuperar el cine que tiene mayor capacidad de espectadores. En caso de empate en la capacidad, prevalecerá el primero recuperado.
- Mostrar finalmente todos los datos correspondientes al cine con mayor aforo y, a continuación, el aforo total de este.

Este apartado se encuentra desarrollado parcialmente en el algoritmo del enunciado. ***Tened en cuenta las variables ya declaradas para contener los cines a comparar, y el cine recuperado con mayor aforo.***

3. Codificación en C

Codificar en C el algoritmo diseñado anteriormente.

El programa en C debe cumplir con las siguientes particularidades:

- Como ya hemos visto en lenguaje C, no se puede utilizar el operador = para asignar el contenido de una tupla a otra, ya que es necesario hacer la asignación de valores campo a campo. A continuación os proporcionamos el código C de las acciones ***cinemaCpy(...)*** y ***screenCpy(...)*** necesarias para asignar el contenido de una tupla de tipo *tCinema* a otra. Observar que la primera acción hace uso de la segunda para la copia de los datos de cada pantalla de proyección.

```
void cinemaCpy(tCinema *cinemaDst, tCinema cinemaSrc)
{
    int i;
    strcpy(cinemaDst->name, cinemaSrc.name);
    cinemaDst->nScreens = cinemaSrc.nScreens;
    for (i = 0; i < cinemaSrc.nScreens; i++) {
        screenCpy(&cinemaDst->screenVector[i], cinemaSrc.screenVector[i]);
    }
}

void screenCpy(tScreen *screenDst, tScreen screenSrc)
{
    screenDst->screenId = screenSrc.screenId;
    screenDst->capacity = screenSrc.capacity;
}
```

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 dentro de la herramienta ACME**, 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_6}
{15/11/2022}
{ACCIONES Y FUNCIONES}

const
    NUM_CINEMAS: integer = 3;  {Number of cinemas}
    MIN_SCREENS: integer = 1;  {Min. cinema screens}
    MAX_SCREENS: integer = 5;  {Max. cinema screens}
    MAX_LETRAS: integer = 25;  {Max. Numero letras nombre}
    MIN_VECTOR: integer = 0;  {Pos 0 del vector}
end const

type
    tScreen = record
        screenId: integer;
        capacity: integer;
    end record


    tCinema = record
        name: string;
        screenVector: vector[MAX_SCREENS] of tScreen;
        nScreens: integer;
    end record
end type

{Exercise 2.1 READ_CINEMA}

action readCinema( inout cinema:  tCinema ): tCinema

        writeString("NAME (25 CHAR MAX, NO SPACES)? >>");
        cinema.name:= readString();
        writeString("NUMBER OF SCREENS (1-5)? >>");
        cinema.nScreens:= readInteger();

        while cinema.nScreens >MAX_SCREENS or cinema.nScreens <MIN_SCREENS do
                writeString("INVALID DATA, TRY AGAIN!");
                writeString("NUMBER OF SCREENS (1-5)? >>");
                cinema.nScreens:= readInteger();
        end while

        for i := 1 to cinema.nScreens [step 1] do
                writeString("ID (AN INTEGER)? >>");
                cinema.screenVector[i].screenId:= readInteger();
                writeString("CAPACITY (25-50)? >>");
                cinema.screenVector[i].capacity:= readInteger();
        end for

        return cinema;
end action

{Exercise 2.2 WRITE_CINEMA}

action writeCinema( in cinema:  tCinema ): tCinema

        writeString("NAME: ");
        writeString(cinema.name);
        writeString("SCREENS: ");
        writeInteger(cinema.nScreens);

        for i := 1 to cinema.nScreens [step 1] do
                writeString("ID: ");
                writeInteger(cinema.screenVector[i].screenId);
                writeString("CAPACITY: ");
                writeInteger(cinema.screenVector[i].capacity);
```

```
        end for

end action

{Exercise 2.3 GET_TOTAL_CAPACITY}

function getTotalCapacity( cinema:  tCinema ): tCinema

        var
                sum: integer;
        end var

        sum:=0;

        for i := 1 to cinema.nScreens [step 1] do
                sum:= sum + cinema.screenVector[i].capacity;
        end for

        return sum;

end function

algorithm UOCCinema

        {Variable definitions}
        var
                cinemaVector: vector[NUM_CINEMAS] of tCinema;
                bestCinema: tCinema;
        end var

        {Exercise 2.4}
        {Data input}

        writeString("INPUT DATA");

        for i:= 1 to NUM_CINEMAS [step 1] do
                readCinema(cinemaVector[i]);
        end for

        bestCinema:= cinemaVector[MIN_VECTOR]

        for i:= 1 to NUM_CINEMAS [step 1] do
                if getTotalCapacity( bestcinema ) < getTotalCapacity(cinemaVector[i])then
                        bestCinema:= cinemaVector[i]
                end if
        end for

        {Data output}
        writeString("BEST CINEMA\n");
        writeCinema(bestCinema);
        writeString("MAX CAPACITY: ");
        getTotalCapacity(bestCinema);

end algorithm
```