Prácticas de Programación
PEC1 - 20222

Fecha límite de entrega: **19 / 3 / 2023**

Formato y fecha de entrega

La PEC debe entregarse antes del día **19 de marzo de 2023** a las 23:59.

Se entregará un archivo en formato **ZIP** , que contenga:

- Un documento en formato **pdf** con las respuestas de los **ejercicios 1 y 3**. El documento debe indicar en su primera página el **nombre y apellidos** del estudiante que hace la entrega.
- Los archivos **donation.h** y **donation.c** solicitados en el **ejercicio 2** .

La entrega debe realizarse en el apartado de entregas de EC del aula de teoría antes de la fecha de entrega.

Objetivos

- Repasar los conceptos básicos de programación.
- Saber interpretar y seguir el código de terceras personas.
- Saber realizar pequeños programas para solucionar problemas a partir de una descripción general del problema.
- Saber especificar un algoritmo mediante las pre y post condiciones.
- Saber incluir controles en los programas a fin de garantizar que las pre-condiciones se cumplan.

Criterios de corrección

Cada ejercicio lleva asociado la puntuación sobre el total de la actividad. Se valorará tanto que las respuestas sean correctas como que también sean completas.

- No seguir el **formato de entrega**, tanto por lo que se refiere al **tipo y nombre de los archivos** como al contenido solicitado, comportará una **penalización importante** o la calificación con una **D de la actividad**.
- En los ejercicios en que se pide lenguaje algorítmico, se debe respetar el **formato** .
- En el caso de ejercicios en lenguaje C, éstos **deben compilar para ser evaluados** . Si compilan, se valorará:
  - Que **funcionen** tal y como se describe en el enunciado.
  - Que se respeten los **criterios de estilo** y que el código esté **debidamente comentado**. Se valorará especialmente el uso de comentarios en inglés.
  - Que las **estructuras** utilizadas sean las correctas.

Aviso

Aprovechamos para recordar que **está totalmente prohibido copiar en las PECs** de la asignatura. Se entiende que puede haber un trabajo o comunicación entre los estudiantes durante la realización de la actividad, pero la entrega de ésta debe ser individual y diferenciada del resto. Las entregas serán analizadas con **herramientas de detección de plagio**.

Así pues, las entregas que contengan alguna parte idéntica respecto a entregas de otros estudiantes serán consideradas copias y todos los implicados (sin que sea relevante el vínculo existente entre ellos) suspenderán la actividad entregada.

Guía citación: https://biblioteca.uoc.edu/ca/continguts/Com-citar/index.html

Monográfico sobre plagio: http://biblioteca.uoc.edu/ca/biblioguies/biblioguia/Plagi-academic/

Observaciones

Esta PEC presenta el proyecto que se desarrollará durante las diferentes actividades del semestre, que se ha simplificado y adaptado a las necesidades académicas.

En este documento se utilizan los siguientes símbolos para hacer referencia a los bloques de diseño y programación:

 Indica que el código mostrado está en **lenguaje algorítmico**.

 Indica que el código mostrado está en **lenguaje C**.

 Muestra la ejecución de un programa en **lenguaje C**.

PELP

PeLP (*Programming eLearning Platform*) es una herramienta que se encuentra en el espacio de recursos del aula de teoría y laboratorio. Los objetivos principales de la herramienta son:

- Facilitar un entorno común para la evaluación de los ejercicios de codificación.
- Facilitar un sistema para compartir vuestros ejercicios de código con los profesores, evitando enviar el código en archivos adjuntos al correo.
- Facilitar el uso de herramientas de análisis del uso y la gestión de la memoria.
- Garantizar que se respete el formato de entrega de las actividades.

Os aconsejamos realizar envíos periódicos a la herramienta de los diferentes ejercicios de código, ya que os permitirá detectar posibles errores antes de la entrega final. **Tened presente que es la herramienta utilizada como base para corregir vuestros códigos, y que no se corregirá ningún código en otro entorno o máquina.** Así pues, si vuestro código no funciona en la herramienta PeLP, se considerará que no funciona, aunque lo haga en vuestro ordenador.

Aunque la herramienta está especialmente preparada para las prácticas, para asegurar que no surgen problemas de acceso a la herramienta, y que puedas familiarizarte con ella, la ponemos a vuestra disposición para el ejercicio de código de la PEC1.

En todo caso, **hay que tener presente que las entregas finales deben seguir haciéndose en el apartado correspondiente del aula, tal como indica el enunciado**. Esta herramienta es una ayuda adicional que ponemos a vuestra disposición, y en ningún caso es obligatorio su utilización.

Enunciado

En el contexto de las múltiples necesidades humanitarias que generan las diferentes crisis económicas y guerras que afectan al mundo globalizado en el que vivimos, se nos ha pedido colaborar en la aplicación que debe permitir gestionar toda la información relacionada con las ONGs (*Organizaciones no gubernamentales*) o NGOs (*Non-governmental organizations*) que intentan paliar los efectos negativos mediante, entre otros, el voluntariado y las donaciones. En concreto, a nosotros se nos ha pedido que nos encarguemos **de la gestión de las donaciones y los aspectos económicos básicos.**

Nota de nomenclatura: en el enunciado nos referiremos a una Organización no gubernamental con su acrónimo español ONG, escrito en mayúsculas. No obstante en el código usaremos el acrónimo inglés NGO, escrito en mayúsculas o minúsculas según criterios de estilo de código asumidos en esta asignatura.

Nuestra aplicación se conectará con el sistema final, y por tanto los datos de entrada y salida tendrán que seguir el formato pactado. Para facilitar esta conexión, se ha acordado que todas las aplicaciones recibirán un archivo en formato CSV (*Comma Separated Values*) para los datos de entrada y generarán la salida en el mismo formato. En este tipo de archivos, cada línea corresponde a un registro, y los distintos campos de cada registro se separan con un carácter de separación conocido. En concreto, en nuestro caso utilizaremos como carácter de separación el punto-y-coma (;), y los formatos serán:

Datos de entrada

La entrada a nuestra aplicación serán los **datos de las personas** dadas de alta como donantes de la ONG, que vendrán en el siguiente formato:

> “document;name;surname;phone;email;address;cp;birthday”

- **document** : Se refiere al documento de identidad con el que se ha dado de alta la persona en el sistema. Este documento identifica a la persona de forma única.
- **name** : Se refiere al nombre.
- **surname** : Hace referencia a los apellidos.

- **phone** : Hace referencia al teléfono. Se aceptan dígitos y el carácter “+” en la primera posición. Por ejemplo: +34699999999.
- **email** : Hace referencia al correo electrónico.
- **address** : Hace referencia a la dirección postal.
- **cp**: Hace referencia al código postal donde reside. El formato son 5 dígitos, pudiendo ser el primero un 0. Por ejemplo: 08001
- **birthday** : Hace referencia a la fecha de nacimiento en formato dd/mm/yyyy (dd: día con dos dígitos, mm: mes con dos dígitos, yyyy: año con cuatro dígitos)

Un ejemplo de registro sería:

“12345678N;John;Smith;+34699999999;john.smith@example.com;My street, 25;08001;30/12/1980”

De forma puntual recibiremos también la **información sobre las donaciones** que la ONG tiene previsto facturar, en el formato:

“date;document;ngo;projectCode;amount”

- **date** : Se refiere a la fecha en que se tiene previsto facturar el importe de la donación, en formato dd/mm/yyyy (dd: día con dos dígitos, mm: mes con dos dígitos, yyyy: año con cuatro dígitos).
- **document** : Se refiere al documento de identidad de la persona que hace la donación.
- **ngo :** Se refiere al código de la ONG que recibe la donación en formato CCC (CCC: código de 3 letras que identifica la ONG).
- **projectCode** : Se refiere al código del proyecto y tiene el formato CCCNNNN (CCC: código de la ONG, NNNN: código numérico de 4 cifras que identifica unívocamente el proyecto dentro de la ONG). Este código indica a qué proyecto o causa específica se realiza la donación, por ejemplo, para paliar el hambre en Somalia. Si la donación es genérica, no vinculada a una causa en particular, pero se destina a la ONG, entonces se usará el código de proyecto 0000 y tendremos el código CCC0000.
- **amount :** Se refiere a cantidad (en euros) aportada en la donación y que está previsto facturar, empleando siempre dos cifras para los céntimos (por ejemplo, podemos donar 10.00 o 45.99).

Algunos ejemplos de registro serían:

“15/04/2023;12345678N;ACN;ACN8455;15.50”

“25/12/2023;87654321B;MSF;MSF0000;100.00”

***Nota**:* Los campos que contienen un texto no tienen saltos de línea ni el carácter “;”. Tampoco contienen acentos o diéresis.

Datos de salida

La salida de la aplicación serán los mensajes a enviar. El envío de estos mensajes lo hará otra aplicación, por lo que habrá que generar un archivo CSV con la información necesaria. Como esta aplicación de envío será utilizada con distintos propósitos, los mensajes tienen un formato genérico y los datos se pasan en campos predefinidos. Cada línea será un mensaje en el siguiente formato:

“email;when;who;what”

- **email** : Se refiere al correo electrónico del destinatario del mensaje.
- **when** : Fecha prevista de facturación. El formato será “dd/mm/yyyy” (dd: día con dos dígitos, mm: mes con dos dígitos, yyyy: año con cuatro dígitos).
- **who** : Código del proyecto asociado a la donación, en el formato anteriormente especificado de CCCNNNN.
- **what** : importe de la donación a facturar, en euros y empleando 2 dígitos para los céntimos.

Un ejemplo de registro sería:

“john.smith@example.com;15/04/2023;ACN8455;15.50”

Dado que los datos de las personas y los mensajes de salida se utilizan en diversas aplicaciones, como parte del proyecto nos han facilitado la definición de los siguientes tipos (se deberán traducir al lenguaje C aquellos que necesitéis para resolver los ejercicios):

```
const
      MAX_PEOPLE: integer = 100;
end const

type
      tDate = record
            day: integer;
            month: integer;
            year: integer;
      end record

      tPerson = record
            document: string;
            name: string;
            surname: string;
            phone: string;
            email: string;
            address: string;
            cp: string;
            birthday: tDate;
      end record

      tPeople = record
            elems: vector [MAX_PEOPLE] of tPerson;
            count: integer;
      end record

      tMessage = record
            email: string;
            when: tDate;
            who: string;
            what: real;
      end record
end type
```

Ejercicio 1: Definición de tipos de datos [25%]

A partir de la descripción de los datos de entrada del enunciado, define **en lenguaje algorítmico** :

- Un tipo **tDonation** que pueda guardar la información de una donación a una ONG (fecha de la donación, ONG, …) que está previsto realizar en una fecha concreta.

- Un tipo **tDonationData** que pueda guardar todos los datos de las donaciones que vayamos recibiendo **ordenados por el código del proyecto**. Podéis asumir que guardaremos una tabla con un máximo de 120 donaciones.

Nota: Podéis definir los tipos adicionales que consideréis oportunos. También podéis utilizar los tipos de datos del enunciado.

SOLUCIÓN

```
const
      MAX_DONATIONS: integer = 120;
end const

type
      tDonation = record
            date: tDate;
            personDocument: string;
            ngo: string;
            projectCode: string;
            amount: real;
      end record

      tDonationData = record
            elems: vector [MAX_DONATIONS] of tDonation;
            count: integer;
      end record

end type
```

Ejercicio 2: Manipulación de tablas [50%]

A partir de las estructuras de datos definidas en el ejercicio 1 y las estructuras y métodos detallados al final de este enunciado, implementa los siguientes métodos (acciones o funciones) en lenguaje C (utiliza sólo los archivos **donation.h** y **donation.c** para declarar e implementar los métodos):

a) **donationData_init:** Dada una estructura de tipo **tDonationData,** la inicializa correctamente obteniendo una estructura vacía.

b) **donationData_len:** Dada una estructura de tipo **tDonationData,** devuelve el número de donaciones guardadas en esta estructura.

c) **donation_parse:** Dada una estructura de tipo **tDonation** y una entrada de archivo CSV (**tCSVEntry**) con los datos de una donación, inicializa la estructura con estos datos.

d) **donationData_add**: Dada una estructura de tipo **tDonationData** y una donación (**tDonation**), añade esta donación a la lista de donaciones (**tDonationData**) manteniendo la ordenación por el código del proyecto. Si ya existe una donación de la misma persona para la misma fecha y proyecto, la donación se ignorará y no se añadirá. Se recuerda que la persona se identifica con su documento de identidad y que los proyectos tienen un código único que los identifica.

e) **donationData_get:** Dada una estructura de tipo **tDonationData**, una posición y una cadena de caracteres de salida, escribe en la cadena de caracteres la información de la donación en la posición dada de **tDonationData**. La cadena de caracteres seguirá el mismo formato que los datos de entrada. Un ejemplo de salida sería (mira el método de C **sprintf**):

15/04/2023;12345678N;ACN;ACN8455;15.50

f) **donationData_del:** Dada una estructura de tipo **tDonationData**, la fecha de una donación, el documento de una persona y el código de un proyecto, elimina esta donación de la lista. Si no existe ninguna donación con esas características, entonces el método no hace nada.

Debéis utilizar la rutina principal que se os facilita en el archivo **main.c** sin ninguna modificación. Esta rutina ejecuta las siguientes tareas (y muestra los datos almacenados después de cada paso):

1. Añade 3 donaciones.
2. Añade 2 donaciones nuevas.
3. Añade una donación ya existente.
4. Elimina 2 donaciones existentes.
5. Elimina una donación que no existe.

Para considerar que el programa funciona correctamente, el resultado final por pantalla deberá ser el siguiente:

```
0;15/04/2023;12345678N;ACN;ACN8455;15.50
1;30/12/2022;55755010C;CRJ;CRJ0333;3500.60
2;11/12/2022;76203765G;MSF;MSF0012;600.20

0;15/04/2023;12345678N;ACN;ACN8455;15.50
1;30/12/2022;55755010C;CRJ;CRJ0333;3500.60
2;12/05/2022;65239334J;GRP;GRP0020;23.00
3;25/08/2022;87654321B;MSF;MSF0000;100.00
4;11/12/2022;76203765G;MSF;MSF0012;600.20

0;15/04/2023;12345678N;ACN;ACN8455;15.50
1;30/12/2022;55755010C;CRJ;CRJ0333;3500.60
2;12/05/2022;65239334J;GRP;GRP0020;23.00
3;25/08/2022;87654321B;MSF;MSF0000;100.00
4;11/12/2022;76203765G;MSF;MSF0012;600.20

0;15/04/2023;12345678N;ACN;ACN8455;15.50
1;30/12/2022;55755010C;CRJ;CRJ0333;3500.60
2;25/08/2022;87654321B;MSF;MSF0000;100.00

0;15/04/2023;12345678N;ACN;ACN8455;15.50
1;30/12/2022;55755010C;CRJ;CRJ0333;3500.60
2;25/08/2022;87654321B;MSF;MSF0000;100.00
```

Nota: Para la realización de este ejercicio, en los archivos **csv.h, csv.c, donation.h** y **donation.c** encontraréis la declaración e implementación de los siguientes métodos:

tCSVEntry	Tipo de datos que guarda una línea de un archivo CSV.
csv_numFields	Dada una estructura tCSVEntry, devuelve el número de campos que contiene.

csv_getAsInteger	Dada una estructura tCSVEntry y la posición de un campo (0 indica la primera posición), devuelve su valor como entero.
csv_getAsString	Dada una estructura tCSVEntry y la posición de un campo (0 indica la primera posición), devuelve su valor como string.
csv_getAsReal	Dada una estructura tCSVEntry y la posición de un campo (0 indica la primera posición), devuelve su valor como real.
tDate	Estructura que permite guardar información de una fecha
date_parse	Dada una estructura de tipo tDate y un string con una fecha (en el formato de los datos de entrada), inicializa la estructura tDate con la fecha.
date_equals	Compara dos estructuras tipo tDate e indica si contienen el mismo valor o no.

Ejercicio 3: Especificación formal [25%]

Define la declaración (**no la implementación**) de los métodos **donationData_init** y **donationData_del** del ejercicio anterior en **lenguaje algorítmico**.

a) Añade las pre y post condiciones a la declaración de estos métodos en lenguaje algorítmico.

b) Añade al código en lenguaje C del ejercicio anterior, los "asserts" necesarios para asegurar que se cumplen las pre-condiciones que acabas de escribir.

Solución

a)

```
action donationData_init (out data: tDonationData)
Pre:{}
No hace falta definir variables dado que todas vienen como parámetros
Post: { data.count = 0 }

action donationData_del (inout data: tDonationData, in date: tDate,
                         in personDoc: string, in projectCode: string)
Pre:{data = DATA and date = DATE and personDoc = PERSONDOC and
     projectCode = PROJECTCODE and∀i,j: 0 < i < j ≤ DATA.count:
     ((DATA.elems[i].date ≠ DATA.elems[j].date or
     DATA.elems[i].personDocument ≠ DATA.elems[j].personDocument or
     DATA.elems[i].projectCode ≠ DATA.elems[j].projectCode) and
     DATA.elems[i].projectCode ≤ DATA.elems[j].projectCode)}
No hace falta definir variables dado que todas vienen como parámetros
Post: { ¬∃i : 0 < i ≤ data.count : data.elems[i].date = DATE and
        data.elems[i].personDocument = PERSONDOC and
        data.elems[i].projectCode = PROJECTCODE}
```

b)

Asserts añadidos al archivo donation.c adjunto

NOTA: En este ejercicio no es necesario haber definido exactamente las mismas PRE y POST condiciones que se proponen, pero sí haber detectado las condiciones iniciales necesarias y el estado final. En el caso de la traducción al lenguaje C, es importante haber definido los correspondientes asserts para verificar que los punteros no son NULL cuando se trata de variables de entrada/salida.