Prácticas de Programación
PEC2 - 20221

Fecha límite de entrega: **31 / 10 / 2022**

Formato y fecha de entrega

La PEC debe entregarse antes del día **31 de octubre de 2022** a las 23:59.

Es necesario entregar un fichero 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 ficheros **project.h** y **project.c** solicitados en el **ejercicio 2**.

Es necesario hacer la entrega en el apartado de entregas de EC del aula de teoría antes de la fecha de entrega.

Objetivos

- 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 utilizar el tipo de datos puntero.
- Saber definir tipos de datos usando memoria dinámica.
- Saber reducir un problema dado en problemas más pequeños mediante el diseño descendente.

Criterios de corrección:

Cada ejercicio tiene asociada su 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 ficheros** como al contenido solicitado, comportará una **penalización importante** o la cualificación 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, estos **deben compilar para ser evaluados**. Si compilan, se valorará:
    - Que **funcionen** tal como se describe en el enunciado.
    - Que se respeten los **criterios de estilo y** que el código esté **debidamente comentado**.
    - 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/es/contenidos/Como-citar/index.html
Monográfico sobre plagio:
http://biblioteca.uoc.edu/es/biblioguias/biblioguia/Plagio-academico/

Observaciones

Esta PEC continúa el proyecto que se desarrolla durante las distintas actividades del semestre.

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


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


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


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

Análisis dinámico

En esta actividad empezamos a utilizar memoria dinámica, que requiere que el programador reserve, inicialice y libere la memoria. Para ayudar a detectar memoria que no se ha liberado correctamente, o errores en las operaciones con punteros relacionadas, hay herramientas que ejecutan un análisis dinámico del programa. Una herramienta de código abierto muy empleada es Valgrind (https://valgrind.org/). La utilización de esta herramienta queda fuera del ámbito del curso, pero os facilitamos sus resultados como parte del análisis de la herramienta PeLP. Podéis acceder a los errores detectados por Valgrind en la pestaña de errores:

Para entender el significado de los **códigos de error**, podéis consultar el siguiente enlace, donde encontraréis ejemplos de código que os ayudarán a entender cuando se dan estos errores:

https://bytes.usc.edu/cs104/wiki/valgrind/

Enunciado

Siguiendo con el problema introducido en la PEC1, se ha decidido que dado el volumen de datos que habrá que gestionar, no podemos predecir con exactitud el número de personas que serán dadas de alta en la ONG. Por este motivo, se ha modificado la lista de personas para utilizar memoria dinámica, evitando la limitación de 100 personas inicial:

```
type
      tStaff = record
            elems : pointer to tPerson;
            count : integer;
      end record
end type
```

Además se nos ha facilitado los ficheros **person.h** y **person.c** con la declaración e implementación en lenguaje C de los tipos de datos y métodos básicos para la gestión de personas, ya incorporando el uso de memoria dinámica.

Ejercicio 1: Definición de tipos de datos [10%]

A partir de la descripción de los datos de entrada del enunciado, redefine **en lenguaje algorítmico** el tipo **tProjectData** para que pueda almacenar un número indeterminado de proyectos, eliminando la restricción de máximo 150 posiciones de la PEC1**.**

SOLUCIÓN

```
type
      tProjectData = record
            elems : pointer to tProject;
            count : integer;
      end record

end type
```

Ejercicio 2: Manipulación de tablas con memoria dinámica [40%]

A partir de las estructuras de datos y los métodos definidos e implementados en la PEC1, define o redefine los siguientes métodos (acciones o funciones) en lenguaje C, para que se adapten a la nueva definición de **tProjectData** (del ejercicio 1). Utiliza tus ficheros **project.h** y **project.c** de la PEC1 como base o los de la solución publicada:

a) **projectData_init:** Dada una estructura de tipo **tProjectData,** la inicializa correctamente obteniendo una estructura vacía.
b) **projectData_add**: Dada una estructura de tipo **tProjectData** y un proyecto (**tProject**), añade este proyecto a la lista de proyectos (**tProjectData**). Si ya existe un proyecto con el mismo código, para la misma ciudad y fecha, entonces se sumará el coste y número de personas al proyecto ya existente.
c) **projectData_del:** Dada una estructura de tipo **tProjectData**, un código de proyecto, una ciudad, una fecha, una cantidad de dinero y un número de personas, reduce el número de personas y el coste en el proyecto correspondiente. Si el número de personas o el coste del proyecto queda en cero o en negativo, se eliminará la información del proyecto. En el caso de que no exista ningún proyecto con ese código para la ciudad y momento de tiempo facilitados, entonces no se hará nada.
d) **[Nuevo método] projectData_free:** Dada una estructura de tipo **tProjectData**, elimina todos su contenido, obteniendo una estructura vacía.

Nota: Para este ejercicio disponéis de los ficheros **csv.h** y **csv.c** de la PEC1, y los ficheros **person.h** y **person.c** como referencia.

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 después de cada paso):

1. Añade 3 proyectos.
2. Añade 3 proyectos nuevos.
3. Elimina parcialmente 2 proyectos.
4. Elimina el coste y las personas restantes de los 2 proyectos anteriores.
5. Elimina un proyecto que no existe.
6. Añade N>150 nuevos proyectos.
7. Elimina todos los datos.

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

```
0;11/10/2022;ACN;ACNUR;Barcelona;ACN0001;12500.00;5
1;11/11/2022;MSF;Medecins Sans Frontieres;Mumbai;MSF1057;8300.50;8
2;11/12/2022;STM;SETEM;Malaga;STM0012;600.20;10

0;11/10/2022;ACN;ACNUR;Barcelona;ACN0001;12500.00;5
1;11/11/2022;MSF;Medecins Sans Frontieres;Mumbai;MSF1057;10000.00;10
2;11/12/2022;STM;SETEM;Malaga;STM0012;600.20;10
3;11/10/2022;ACN;ACNUR;Madrid;ACN0001;22500.00;7
4;11/12/2022;CRE;Cruz Roja España;Oviedo;CRE0333;3500.20;3

0;11/10/2022;ACN;ACNUR;Barcelona;ACN0001;12500.00;5
1;11/11/2022;MSF;Medecins Sans Frontieres;Mumbai;MSF1057;5000.00;5
2;11/12/2022;STM;SETEM;Malaga;STM0012;600.20;10
3;11/10/2022;ACN;ACNUR;Madrid;ACN0001;20000.00;6
4;11/12/2022;CRE;Cruz Roja España;Oviedo;CRE0333;3500.20;3

0;11/10/2022;ACN;ACNUR;Barcelona;ACN0001;12500.00;5
1;11/12/2022;STM;SETEM;Malaga;STM0012;600.20;10
2;11/12/2022;CRE;Cruz Roja España;Oviedo;CRE0333;3500.20;3

0;11/10/2022;ACN;ACNUR;Barcelona;ACN0001;12500.00;5
1;11/12/2022;STM;SETEM;Malaga;STM0012;600.20;10
2;11/12/2022;CRE;Cruz Roja España;Oviedo;CRE0333;3500.20;3

Added new 200 projects

No projects
```

Nota: Los ficheros **person.h y person.c** se facilitan con el enunciado a modo de ejemplo, pero no se precisan para la solución del ejercicio.

Ejercicio 3: Diseño descendiente [50%]

Como parte de la gestión de los proyectos necesitamos saber cuántos proyectos gestiona cada ONG (tONG), los voluntarios y trabajadores de los que dispone (**tStaff**) y su disponibilidad (**tAvailabilityData**) para ayudar o trabajar en el proyecto. Es obvio que para poder realizar un proyecto necesitamos personas. Por este motivo se crea el concepto de campaña (**tCampaign**) que no es más que la aplicación de un proyecto (**tProject**), con unos detalles concretos (**tProjectDetail**) y utilizando un personal específico (**tStaff**). Con esta organización de los proyectos podremos tener una lista de proyectos disponibles en cada ONG y además tener una lista de las campañas concretas que se están llevando a cabo, es decir, el mismo proyecto (tProject) puede ejecutarse varias veces. En otras palabras, en cada campaña (tCampaign) tendremos la información de qué personas desarrollan su trabajo (tStaff) y los detalles del proyecto (tProjectDetail) nos indican datos como el coste y el día.

Para implementar una campaña también es importante conocer qué personas de la ONG se pueden dedicar a dicha campaña. Para ello tenemos la disponibilidad de cada persona (tAvailability), donde tenemos la fecha inicial y la fecha final de los días que tiene disponibles. Una persona puede estar disponible en uno o más períodos de tiempo y sólo podrá participar en aquellas campañas donde esté disponible durante todo el tiempo que ésta dura. Además, una persona no puede estar en dos campañas de forma simultánea. Por tanto, cada campaña tendrá unas personas asignadas de todas las que pertenecen a la ONG. A continuación se muestra la definición de los nuevos tipos de datos, **modificando ligeramente el tipo tProject según el nuevo enfoque**:

```
const
      MAX_CAMPAIGN: integer = 100;
      MAX_AVAILABILITY: integer = 100;
end const

type
      tProject = record
            code : string;
            ongCode : string;
      end record

      tProjectDetail = record
            date : tDate;
            city : string;
            cost : real;
            numPeople : integer;
      end record
```

```
        tAvailability = record
                person : tPerson;
                start : tDate;
                end : tDate;
        end record

        tAvailabilityData = record
                elems : vector [MAX_AVAILABILITY] of tAvailability;
                count : integer;
        end record

        tCampaign = record
                project : tProject;
                detail : tProjectDetail;
                staff: tStaff;
        end record

        tCampaignData = record
                elems : vector [MAX_CAMPAIGN] of tCampaign;
                count : integer;
        end record

        tONG = record
                code: string;
                name: string;
                projects: tProjectData;
                staff: tStaff;
                campaigns: tCampaignData;
                staffAvailability: tAvailabilityData;
        end record
end type
```

Se pide implementar en **lenguaje algorítmico** la función:

```
function isAvailable(availabilities: tAvailabilityData, campaigns: tCampaignData, person: tPerson, date: tDate): boolean
```

que dada una lista de disponibilidades (**tAvailabilityData**), una lista de campañas (**tCampaignData**), una persona (**tPerson**) y una fecha (**tDate**), compruebe si dicha persona está disponible en la fecha indicada. Hay que tener en cuenta:

- Se considera que una persona está disponible un día cuando se cumple que:
  - El día se encuentra dentro del rango de fechas de alguno de los **tAvailability** de dicha persona.
  - La persona no está dentro del staff de una campaña cuyo proyecto tiene lugar dicho día.

- Podéis asumir que todas las personas siempre tienen indicada al menos una disponibilidad tAvailability en tAvailabilityData.
- Podéis asumir que las personas están ordenadas ascendentemente por documento dentro de **tStaff**.
- Podéis asumir que las campañas están ordenadas ascendentemente por fecha del proyecto dentro de tCampaignData.

Para realizar este ejercicio podéis asumir que tenéis ya implementado el método:

- **date_cmp**: Compara dos fechas y devuelve:
  - -1 si la primera fecha es menor a la segunda.
  - 0 si ambas fechas son iguales.
  - +1 si la primera es mayor a la segunda.

Descompón esta función en acciones o funciones más simples utilizando diseño descendente e implementa **en lenguaje algorítmico** las acciones y funciones resultantes.

Solució

Nivel 1:

```
function isAvailable(availabilities: tAvailabilityData, campaigns: tCampaignData, person: tPerson, date: tDate) : boolean

var
   i, resCmp: integer;
   stop: boolean;
end var

stop := false;
res := false;
{ If the person is not available that day }
if not checkAvailability(availabilities, person, date) then
    stop := true;
end if

{ Loop through campaign list }
i := 1;
while i <= campaigns.count and stop = false do
    { If the project date coincides }
    resCmp := date_cmp(campaigns.elems[i].detail.date, date);
    if resCmp = 0 then
       { Check if the person is within the project staff }
        if person_inStaff(person, campaigns.elems[i].staff) then
```

```
            stop := true;
        end if
    else if resCmp = 1 then
        stop := true;
        res := true;
    end if
    i := i + 1;
end while

return res;

end function
```

Nivel 2:

```
function checkAvailavility(availabilities: tAvailabilityData, person:
tPerson, date: tDate) : boolean

var
   i: integer;
   found : boolean;
end var

i := 1;
found := false;
{ Search over availabilities }
while i <= availabilities.count and not found do
    if person_cmp(availabilities.elems[i].person, person) = 0 and
date_inRange(date, availabilities.elems[i].start,
availabilities.elems[i].end) then
            found := true;
    else
       i := i + 1;
    end if
end while

return found;

end function


function person_inStaff(person: tPerson, staff: tStaff) : boolean

var
   i, resCmp: integer;
   stop, found: boolean;
end var

i := 1;
stop := false;
```

```
found := false;
{ Search over campaign staff }
while i <= staff.count and stop = false do
    resCmp := person_cmp(staff.elems[i], person);
    if resCmp = 0 then
        found := true;
        stop := true;
    else if resCmp = 1 then
        stop:= true;
    else
        i := i + 1;
    end if
end while

return found;

end function
```

Nivel 3:

```
function date_inRange(date: tDate, start: tDate, end: tDate): boolean
    { Check if date is between two dates }
    return not (date_cmp(date, start) < 0 or date_cmp(date, end) > 0);
end function

function person_cmp(person1: tPerson, person2: tPerson): integer
var
   sol : integer;
end var
    { Check if a person document is greater than other }
    if person1.document < person2.document then
        sol := -1;
    else if person1.document > person2.document then
        sol := 1;
    else
        sol := 0;
    end if

    return sol;

end function
```

Nota: Aquí os proponemos una posible solución, pero hay otras válidas.