PEC3
Prácticas de Programación
PEC3 - 20221

Fecha límite de entrega: **20 / 11 / 2022**

Formato y fecha de entrega

La PEC debe entregarse antes del día **20 de noviembre 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. El documento debe indicar en su primera página el **nombre y apellidos** del estudiante que hace la entrega.

Es necesario hacer la entrega en el apartado de entregas de EC del aula de teoría antes de la fecha de entrega. **Únicamente el último envío** dentro del período establecido será evaluado.

Objetivos

- Saber interpretar y seguir el código de terceras personas.
- Adquirir los conceptos teóricos explicados sobre la recursividad.
- Diseñar funciones recursivas, identificando los casos base y recursivos, siendo capaces de simular la secuencia de llamadas dada una entrada.

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 avanza en el proyecto presentado en la PEC1.

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**.

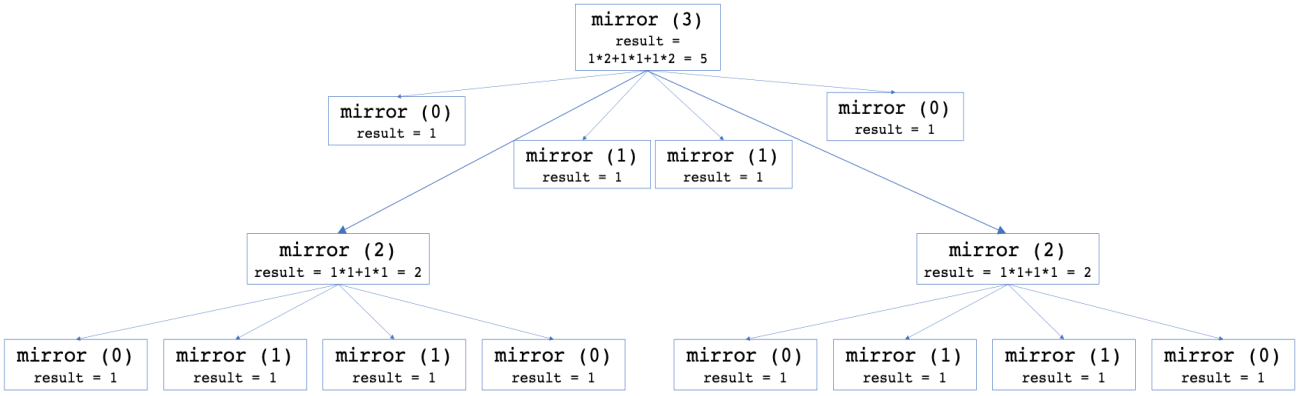
Ejercicio 1: Conceptos básicos de recursividad [40%]

Dada la función recursiva **mirror**, calcula qué valor devuelve la llamada **mirror(3)** y completa el **modelo de las copias** o **grafo de invocaciones** para ver cómo has llegado al resultado:

```
function mirror (a : integer) : integer
var
  result, i : integer;
end var
  if a ≤ 1 then
    result := 1;
  else
      result := 0;
      for i := 0 to a - 1 do
         result := result + mirror(i) * mirror(a - i - 1);
      end for
  end if
  return result;
end function
```

Solución:

El resultado de mirror (3) es 5, que resulta de ejecutar la secuencia de invocaciones representadas en el modelo de copias de la siguiente figura:



Ejercicio 2: Diseño de algoritmos recursivos [60%]

Para facilitar la gestión de las campañas, se han redefinido las estructuras introducidas en la PEC2, agrupando datos por días y eliminando la restricción en el número de elementos. Los nuevos tipos de datos para guardar las campañas asociadas a una ong se facilitan a continuación:
will

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

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

      tCampaign = record
            project : pointer to tProject;
            city : string;
            cost : real;
            numPeople : integer;
            staff : pointer to tStaff;
      end record

      tCampaignNode = record
            elem : tCampaign;
            next : pointer to tCampaignNode;
      end record

      tCampaignDaily = record
            day : tDate;
            first : pointer to tCampaignNode;
            next : pointer to tCampaignDaily;
            count : integer;
      end record

      tCampaignData = record
            first : pointer to tCampaignDaily;
            count : integer;
      end record
end type
```

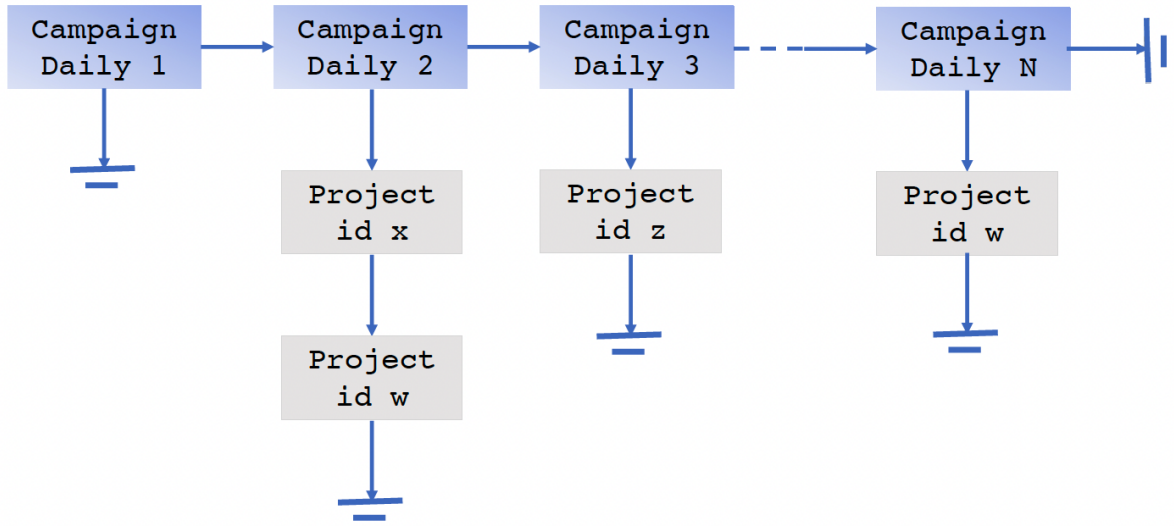
Las campañas de una ONG se guardarán en una estructura ***tCampaignData***, que apuntará al primer elemento de una lista encadenada (*first*) en la que cada elemento ***tCampaignDaily*** representa las campañas previstas para un día determinado. Para facilitar su gestión, los elementos ***tCampaignDaily*** se guardan ordenados por fecha de menor a mayor, guardando el número de elementos de este tipo para cada día (count).

Dentro de las campañas de un día ***tCampaignDaily***, las campañas se guardan como una lista encadenada de nodos tipo ***tCampaignNode***, que a su vez se guardan ordenados alfabéticamente por el código del proyecto. Además del día al

que corresponde la campaña (*day*), se guarda el puntero al primer elemento de la lista de campañas (*first*) y el número total de elementos (*count*) de esta lista.

Se ha decidido no replicar la información de los proyectos y de los trabajadores, utilizando en el tipo **tCampaign** un puntero al elemento correspondiente de la lista de proyectos (*project*) y un puntero a la lista de trabajadores/voluntarios (*staff*). Este tipo de datos también guarda otra información (*city* y *cost*).

A continuación se puede ver una representación gráfica simplificada de las campañas de una ONG (por claridad no se muestra la lista de proyectos, y en vez del puntero hacia ésta, se ha utilizado directamente el código del proyecto. Tampoco se muestra la lista de trabajadores):



Donde los nodos en azul representan las campañas de un día concreto ***tCampaignDaily*** y los nodos en gris los datos de cada campaña (***tCampaignNode*** y ***tCampaign***) de ese día.

Se pide definir en **lenguaje algorítmico** las siguientes funciones **recursivas**:

a. La función **costDayProject** que dado el puntero al primer nodo de la lista de campañas para un determinado día **tCampaignNode**, y el puntero a la definición de un proyecto (**tProject**), nos devuelve el coste de la campaña asociada al proyecto en ese día. En otras palabras, si ese día está previsto realizar una campaña del proyecto, se devolverá el coste de la campaña. De lo contrario el coste será 0. La cabecera de la función es la siguiente:

> **function** costDayProject (dailyCampaigns: **pointer to** tCampaingNode, project: **pointer to** tProject) : **real**

b. La función **costProject** que dado el puntero a la lista de campañas de una ONG **tCampaignData** y el puntero a la definición de un proyecto (**tProject**), nos devuelva el coste total que tendrá el proyecto después de ejecutar todas sus campañas. La cabecera de la función es la siguiente:

function costProject(campaigns: **pointer to** tCampaignData, project: **pointer to** tProject) : **real**

Nota: Recordad que en los nodos terminales de una lista, los punteros "next" tienen un valor NULL. De la misma manera, una lista vacía será un puntero con un valor a NULL.

Solución

```
function costDayProject (dailyCampaigns: pointer to tCampaignNode,
project: pointer to tProject) : real
var
  result : real;
end var

 if dailyCampaigns = NULL then
    { Trivial case: empty list }
    result := 0.0;
 else if dailyCampaigns^.elem.project^.code = project^.code then
    { Trivial case: current node is the value we are looking for }
    result := dailyCampaigns^.elem.cost;
 else if dailyCampaigns^.elem.project^.code > project^.code then
    { Trivial case: no project with the given code}
    result := 0.0;
 else
    { Recursive call }
    result := costDayProject (dailyCampaigns^.next, project);
 end if

 return result;

end function


function costProject (campaigns: pointer to tCampaignData, project:
pointer to tProject) : real
var
  result : real;
end var

 if  campaigns = NULL then
    { Trivial case: empty list }
    result := 0.0;
```

```
 else
    { starts the recursive calls }
    result := costProjectPerDay(campaigns^.first, project);
 end if

 return result;

end function

function costProjectPerDay (campaigns: pointer to tCampaignDaily,
project: pointer to tProject) : real
var
  result : real;
end var

 if  campaigns = NULL then
    { Trivial case: empty list }
    result := 0.0;
 else
    { Recursive call: Traverse nodes in campaigns list }
    result := costDayProject(campaigns^.first, project) +
              costProjectPerDay (campaigns^.next, project);
 end if

 return result;

end function
```