# PEC1

## Formato y fecha de entrega

La PEC debe entregarse antes del **lunes 20 de marzo de 2017 a las 23:59**. Para la entrega es necesario que entreguéis un fichero en formato **ZIP**, que contenga:

- Fichero con las respuestas a los diferentes ejercicios que se plantean.
- Los ficheros .h y .c pedidos. El **main** de este proyecto ha de contener las llamadas a las pruebas que se solicitan, de forma que se muestre por pantalla el resultado de cada prueba.

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

## Presentación

En esta PEC trabajaremos los conceptos básicos de programación que se dan por alcanzados en la asignatura previa, y que son la base de esta asignatura. A partir de este punto inicial, agregaremos la especificación formal de los algoritmos y su traducción al lenguaje C. Finalmente, trabajaremos el concepto de diseño descendente. A diferencia de Fundamentos de Programación, en que los problemas se daban muy pautados, se espera que en esta asignatura seáis capaces de resolver los problemas a partir de su descripción en lenguaje natural.

## Competencias

***Transversales***

- Capacidad de comunicación en lengua extranjera.

***Específicas***

- Capacidad de diseñar y construir aplicaciones informáticas mediante técnicas de desarrollo, integración y reutilización.
- Conocimientos básicos sobre el uso y la programación de los ordenadores, sistemas operativos, bases de datos y programas informáticos con aplicación a la ingeniería.

## Objetivos

- 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 para garantizar que las pre condiciones se cumplan.
- Saber reducir un problema dado en problemas más pequeños mediante el diseño descendente.

## Recursos

Para realizar esta actividad tenéis a vuestra disposición los recursos:

**Básicos**

- Materiales en formato web de la asignatura de Fundamentos de Programación.
- Materiales en formato web de la asignatura de las 3 primeras semanas.
- Laboratorio de C.

**Complementarios**

- Internet: La forma más efectiva de encontrar información sobre C es la búsqueda a través de un buscador.

## Criterios de valoración

Cada ejercicio lleva asociado la puntuación sobre el total de la actividad. Se valorará tanto la corrección de las respuestas como su completitud.

- En ejercicios donde se pide lenguaje algorítmico, es necesario respetar el formato.
- En el caso de ejercicios en lenguaje C, estos han de compilar para ser evaluados. En tal caso, se valorará:
  - Que funcionen
  - Que es respeten los criterios de estilo
  - Que el código esté comentado
  - Que las estructuras utilizadas sean las correctas

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

Una cooperativa de granjas productoras de quesos de cabra desea informatizar su producción y pedidos.

La cooperativa se especializa en la fabricación de un conjunto de quesos. Cada queso tiene asociado un identificador numérico entero, el nombre comercial identificado por una cadena de caracteres, el número de días de curación y el precio por kilo en euros. Para cada queso se tiene también la información acerca de qué ingredientes lo componen (la proporción de cada uno es secreta y no estará disponible en el sistema).

Los quesos se ofrecen en diferentes formatos: med3kg, mini1kg, wedge.

En cada pedido se debe especificar el queso y el formato solicitados. Cada pedido tiene asociado un identificador numérico de cliente y el número de unidades solicitadas.

A partir de este enunciado, se pide (**en lenguaje algorítmico**):

- **a)** Define un tipo de datos **tCheese** que pueda almacenar la información de un queso con sus ingredientes. Cada queso tiene como máximo 32 ingredientes.
- **b)** Define un tipo de datos **tCatalogue** que pueda almacenar la variedad de quesos que la cooperativa es capaz de fabricar. El número de quesos diferentes no es conocido ni limitado.
- **c)** Define un tipo de datos **tCheeseStock** que pueda almacenar toda información de un queso concreto fabricado (identificador y formato), así como el número de unidades existentes. También define una estructura de tipo **tStock** que pueda almacenar la información de todos los quesos disponibles en el almacén.
- **d)** Define un tipo de datos **tOrder** que represente un pedido de quesos. Un pedido contiene el identificador del cliente, el identificador del queso, el formato y el número de unidades que el cliente quiere comprar de ese queso. Se debe crear un pedido por cada combinación diferente de cliente, queso y formato.
- **e)** Los pedidos que llegan de los clientes se procesan según el orden de llegada y se despachan según la disponibilidad de los productos solicitados. Define un tipo de datos **tOrders**, que represente un conjunto de pedidos. El número de pedidos que pueden estar en espera no es conocido ni limitado.

**Nota:** Podéis definir los tipos adicionales que consideréis oportunos.

```
const
        MAX_ING : integer := 32;
end const
type
        tIngrList = record
                        ingredients : vector[MAX_ING] of string;
                        numIng : integer;
        end record
        tCheese = record
                        id : integer;
                        name : string;
                        ingrList : tIngrList;
                        ripening : integer;
                        price : real;
        end record
        tFormat = { med3kg, mini1kg, wedge }
        tCatalogue = record
                        cheeses : pointer to tCheese;
                        number : integer;
        end record
        tCheeseStock = record
                        idCheese : integer;
                        format : tFormat;
                        numUnits : integer;
        end record
        tStock = record
                        cheeses : pointer to tCheeseStock;
                        numItems : integer;
        end record
        tOrder = record
                        idClient : integer;
                        idCheese : integer;
```

```
                format : tFormat;
                numUnits : integer;
        end record
        tOrders = record
                orders : pointer to tOrder;
                count : integer;
        end record
end type
```

## [40%] Ejercicio 2: Manipulación de tablas

A partir de las estructuras de datos definidas en el ejercicio 1, define los siguientes métodos (acciones o funciones) **en lenguaje C** (crea los ficheros **cheese.h** y **cheese.c** para declarar e implementar los métodos):

**a)** **init:** Dada una estructura de tipo **tCatalogue,** una de tipo **tStock** y una de tipo **tOrders,** las inicializa con un catálogo de quesos vacío, un stock vacío en el almacén y un conjunto de pedidos vacío, respectivamente.

**b)** **create_cheese**: Dado un identificador, un nombre comercial, un número de días de curación y un precio por kg, crea un queso de tipo **tCheese** sin ingredientes.

**c)** **add_ingredient**: Dado un ingrediente de tipo **string**, lo agrega al conjunto de ingredientes de un queso de tipo **tCheese** dado. Si el ingrediente ya existe se muestra un mensaje de error.

**d)** **add_cheese**: Dada una estructura de tipo **tCatalogue** y un queso de tipo **tCheese**, inserta el queso en el catálogo de quesos. Si el queso ya existe, se muestra un mensaje de error.

**e)** **add_cheese_stock:** Dadas las estructuras de tipo **tCatalogue** y **tStock,** añade un número de unidades de un queso en el stock, a partir de su identificador y formato. Si el identificador del queso no se encuentra en el catálogo, el método debe mostrar un mensaje de error. Si no existe un queso idéntico en el stock, entonces se tiene que crear una entrada para él con las unidades indicadas.

**f)** **print_stock**: Dada una estructura de tipo **tCatalogue** y otra de tipo **tStock**, muestra todos los quesos que se encuentran en el stock. Por cada queso se muestra su identificador, formato, número de unidades y la lista de ingredientes.

**g)** En el fichero **main.c**, escribe un código que:

1. Inicializa un catálogo de quesos, un almacén y un conjunto de pedidos. Todos ellos vacíos.
2. Genera el queso con el nombre “raw sheep”, identificador 1000, con lista de ingredientes: {“raw milk sheep”, “salt”, “rennet”, “e-252”, “milk enzymes” }, 180 días de curación y precio 12,5 euros por kg. Inserta este queso en el catálogo de quesos.
3. Inserta en el stock, si es posible, el siguiente conjunto de quesos fabricados:

- Identificador 1000, formato mini1kg, 8 unidades
- Identificador 1000, formato wedge, 6 unidades
- Identificador 2000, formato wedge, 2 unidades
- Identificador 1000, formato mini1kg, 4 unidades
- Identificador 1000, formato wedge, 2 unidades

**4.** Muestra todos los quesos almacenados en el stock. El resultado por pantalla debería ser:

```
ERROR: Cheese 2000 is not stored in the catalogue.

Id:1000      Format: mini1kg     Units:12
Ingredients(5): raw milk sheep. salt. rennet. e-252. milk
enzymes.

Id:1000      Format: wedgeUnits:8
Ingredients(5): raw milk sheep. salt. rennet. e-252. milk
enzymes.
```

## [15%] Ejercicio 3: Especificación formal

Define las declaraciones (**no las implementaciones**) de los métodos **init**, **add_ingredient** y **add_cheese** del ejercicio anterior en lenguaje algorítmico.

- **a)** Para cada declaración, agrega las pre y post condiciones en lenguaje algorítmico.
- **b)** Agrega al código en lenguaje C del ejercicio anterior, los "asserts" necesarios para asegurar que se cumplen las pre condiciones especificadas.

**a)**

**action** init (**out** c: tCatalogue; **out** s: tStock; **out** o: tOrders)

Pre: {}

Post: { c.number = 0 y s.numItems = 0 y o.count = 0}

**action** add_ingredient (**in**/**out** ch : tCheese ; **in** ingr : string)

Pre:{ ch = CHEESE y CHEESE.ingrList.numIng<MAX_ING y ingr = INGR }

Post: { (∃i único: 0 < i ≤ CHEESE.ingrList.numIng :
ch.ingrList.ingredients[i] = INGR y
ch.ingrList.numIng = CHEESE.ingrList.numIng) o
(ch.ingrList.numIng = CHEESE.ingrList.numIng+1 y
ch.ingrList.ingredients[ch.ingrList.numIng] = INGR) }

**action** add_cheese (**in**/**out** c : tCatalogue ; **in** ch : tCheese)

Pre:{ c = CATALOGUE y ch = CHEESE }

Post: { (∃i único: 0 < i ≤ CATALOGUE.number :
c.cheeses[i].id = CHEESE.id y
c.number = CATALOGUE.number) o
(c.number = CATALOGUE.number+1 y
c.cheeses[c.number] = CHEESE) }

**b)** Asserts agregados al fichero cheese.c adjunto.

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

## [20%] Ejercicio 4: Diseño descendente

Define la función **cheese_items_with_ingredient** que devuelve el número de quesos que se encuentran actualmente en el almacén y que contienen el ingrediente **ing**.

**function** cheese_items_with_ingredient (c: tCatalogue;
s: tStock; ing: **string**) : **integer**

Para implementar esta función se debe descomponer en acciones o funciones más simples (mediante el diseño descendente) que también se deberán implementar.

Como los tipos ya los tenemos definidos, podemos utilizarlos para implementar las acciones / funciones.

**Primer nivel**

```
function cheese_items_with_ingredient (c: tCatalogue; s: tStock; ing: string) : integer
var
    num, i: integer;
fvar
    num := 0;
    for i:=1 to s.numItems do
        num := num +  items_with_ingredient (c, s.cheeses[i], ing);
    end for
    return num;
end function
```

**Segundo nivel**

**function** items_with_ingredient (c: tCatalogue; chs: tCheeseStock; ing: **string**) : **integer**

```
var
        num: integer;
fvar
        if cheese_has_ingredient (c, chs.idCheese, ing) then
                num := chs.numUnits;
        else
                num := 0;
        end if
        return num;
end function
```

**Tercer nivel**

```
function cheese_has_ingredient (c: tCatalogue; id: integer; ing: string) :
boolean
var
        i: integer;
        found, contain: boolean;
fvar
        i := 1;
        found := false;
        contain := false;
        while (i ≤ c.number) and not found do
                if c.cheeses[i].id = id then
                        found := true;
                        contain := contains_ingredient (c.cheeses[i].ingrList, ing);
                else
                        i:= i+1;
                end if
        end while
        return contain;
end function
```

## Cuarto nivel

```
function contains_ingredient (il: tIngrList; ing: string) : boolean
var
        i: integer;
        found: boolean;
fvar
        i := 1;
        found := false;
        while (i ≤ il.numIng) and not found do
                if il.ingredients[i] = ing then
                        found := true;
                else
                        i:= i+1;
                end if
        end while
        return found;
end function
```