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:

o 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):

1. Define un tipo de datos tCheese que pueda almacenar la información de un queso con sus ingredientes. Cada queso tiene como máximo 32 ingredientes.

type

tCheese: record

id: integer;

name: string;

curingDays: integer;

priceKg: real;

ingredients: array[32] of string;

count: integer;

frecord

ftype

1. 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.

type

tCatalogue: record

elements: pointer to tCheese;

count: integer;

frecord

ftype

1. 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.

type

tCheeseFormat = {med3kg, mini1kg, wedge};

tCheeseStock: record

cheeseID: integer;

format: tCheeseFormat;

quantity: integer;

frecord

tStock: record

elements: pointer to tCheeseStock;

count: integer;

frecord

ftype

1. 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.

type

tOrder: record

clientID: integer;

cheeseID: integer;

format: tCheeseFormat;

quantity: integer;

frecord

ftype

1. 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.

type

tOrders: record

elements: pointer to tOrder;

count: integer;

frecord

ftype

Así si presentáramos todo en un solo tipo podríamos estructurarlo de la siguiente manera, aunque en el programa en C que presento lo he subdividido en los apartados del ejercicio:

Type

tCheeseFormat = {med3kg, mini1kg, wedge};

tCheese: record

id: integer;

name: string;

curingDays: real;

priceKg: real;

ingredients: array[32] of string;

count: integer;

frecord

tCatalogue: record

elements: pointer to tCheese;

count: integer;

frecord

tCheeseStock: record

cheeseID: integer;

format: tCheeseFormat;

quantity: integer;

frecord

tStock: record

elements: pointer to tCheeseStock;

count: integer;

frecord

tOrder: record

clientID: integer;

cheeseID: integer;

format: tCheeseFormat;

quantity: string;

frecord

tOrders: record

elements: pointer to tOrder;

count: integer;

frecord

ftype

Nota: Podéis definir los tipos adicionales que consideréis oportunos.

# [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):

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.

* 1. 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: wedge Units:8

Ingredients(5): raw milk sheep. salt. rennet. e-252. milk enzymes.

Fichero cheese.c

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <assert.h>

#include "cheese.h"

void init (tCatalogue \*catalogue, tStock \*stock, tOrders \*orders){

//Comprobamos precondiciones

assert(catalogue!=NULL);

assert(stock!=NULL);

assert(orders!=NULL);

//Iniciamos tCatalogue

catalogue->elements=NULL;

catalogue->count=0;

//Iniciamos tStock

stock->elements=NULL;

stock->count=0;

//Iniciamos tOrders

orders->elements=NULL;

orders->count=0;

}

tCheese create\_cheese(int id, char name[120], int curingDays, float priceKg){

tCheese c1;

c1.id = id;

strncpy(c1.name, name, 120);

c1.curingDays = curingDays;

c1.priceKg = priceKg;

return c1;

}

void add\_ingredient(char name[120], tCheese \*cheese){

//Comprobamos precondiciones

assert(cheese!=NULL);

//Comprobamos si existe un ingrediente

int i=0;

for (i=0; i<cheese->count; i++){

if (strncmp(cheese->ingredients[i],name,120)==0){

//If exists then show error

printf("ERROR. El ingrediente ya existe\n");

return;

}

}

//añadimos el ingrediente sino existe al queso

strncpy(cheese->ingredients[cheese->count], name, 120);

cheese->count++;

}

void add\_cheese(tCatalogue \*catalogue, tCheese cheese){

//Comprobamos precondiciones

assert(catalogue!=NULL);

assert(cheese.count>0);

//Comprobamos si el queso existe en el catalogo

int i=0;

for (i=0; i<catalogue->count; i++){

if (cheese.id==catalogue->elements[i].id){

//Mostramos error porque existe el queso

printf("ERROR. Queso ya presente en el catalogo");

return;

}

}

//Si no existe, se añade

catalogue->elements = (tCheese\*)realloc(catalogue->elements, (catalogue->count+1)\*sizeof(tCheese));

catalogue->elements[catalogue->count].id=cheese.id;

catalogue->elements[catalogue->count].curingDays=cheese.curingDays;

strncpy(catalogue->elements[catalogue->count].name,cheese.name,120);

catalogue->elements[catalogue->count].priceKg=cheese.priceKg;

catalogue->elements[catalogue->count].count=cheese.count;

int j=0;

for (j=0; j<cheese.count; j++){

strncpy(catalogue->elements[catalogue->count].ingredients[j],cheese.ingredients[j],120);

}

catalogue->count++;

}

void add\_cheese\_stock(tCatalogue catalogue, tStock \*stock, int id, char format[120], int quantity){

//Comprobamos si el ID existe

int i=0;

for (i=0; i<catalogue.count; i++){

if (catalogue.elements[i].id == id){

//comprobamos si el queso existe

int j=0;

for (j=0; j<stock->count; j++){

if (stock->elements[j].cheeseID==id && strncmp(stock->elements[j].format,format,120)==0){

//Actualizamos stock

stock->elements[j].quantity+=quantity;

return;

}

}

//Si no existe creamos una nueva entrada

stock->elements = (tCheeseStock\*)realloc(stock->elements, (stock->count+1)\*sizeof(tCheeseStock));

stock->elements[stock->count].cheeseID=id;

strncpy(stock->elements[stock->count].format, format, 120);

stock->elements[stock->count].quantity=quantity;

stock->count++;

return;

}

}

//Si el queso no existe en el catalogo mostramos un error

printf("ERROR: Queso %d no existe\n",id);

}

void print\_stock(tCatalogue catalogue, tStock stock){

int i=0;

for (i=0; i<stock.count; i++){

//Imprimimos en pantalla el ID, el formato y la cantidad

printf("Id:%d Format:%s Units:%d\n", stock.elements[i].cheeseID, stock.elements[i].format, stock.elements[i].quantity);

//Mostramos la lista de ingredientes

int j=0;

for (j=0; j<catalogue.count; j++){

if (catalogue.elements[j].id==stock.elements[i].cheeseID){

//Imprimimos la lista de ingredientes

printf("ingredients(%d): ",catalogue.elements[j].count);

int k=0;

for (k=0; k<catalogue.elements[j].count; k++){

printf("%s. ",catalogue.elements[j].ingredients[k]);

}

}

}

printf("\n\n");

}

}

Fichero Cheese.h

ypedef enum {false, true} bool;

//tCheese

typedef struct{

int id;

char name[120];

int curingDays;

float priceKg;

char ingredients[32][120];

int count;

} tCheese;

//tCatalogue

typedef struct{

tCheese \*elements;

int count;

} tCatalogue;

//tCheeseStock

typedef struct{

int cheeseID;

char format[120];

int quantity;

} tCheeseStock;

//tStock

typedef struct{

tCheeseStock \*elements;

int count;

} tStock;

//tOrder

typedef struct{

int clientID;

int cheeseID;

char format[120];

int quantity;

} tOrder;

//tOrders

typedef struct{

tOrder \*elements;

int count;

} tOrders;

//Funciones de implementación

void init (tCatalogue \*catalogue, tStock \*stock, tOrders \*orders);

tCheese create\_cheese(int id, char name[120], int curingDays, float priceKg);

void add\_ingredient(char name[120], tCheese \*cheese);

void add\_cheese(tCatalogue \*catalogue, tCheese cheese);

void add\_cheese\_stock(tCatalogue catalogue, tStock \*stock, int id, char format[120], int quantity);

void print\_stock(tCatalogue catalogue, tStock stock);

# [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.

1. Para cada declaración, agrega las pre y post condiciones en lenguaje algorítmico.

action init (int/out catalogue:tCatalogue, in/out stock:tStock, in/out orders:tOrders)

Pre {catalogue = CATALOGUE y catalogue es del tipo tCatalogue. stock = STOCK y stock es del tipo tStock. orders=ORDERS y orders es del tipo tOrders}

Post {catalogue.elements=NULL y catalogue.count=0. stock.elements=NULL y stock.count=0. orders.elements=NULL y orders.count=0}

action add\_ingredient(in name:string, in/out cheese:tCheese)

Pre {name=NAME, cheese=CHEESE, CHEESE.count>0}

Post{Si no existe el ingrediente con nombre NAME, entonces añadir el ingrediente a la lista CHEESE.ingredients. En caso contrario mostrar error}

action add\_cheese (in/out catalogue:tCatalogue, in cheese:tCheese)

Pre {catalogue=CATALOGUE, cheese=CHEESE}

Post {Si no hay ningún elemento en CATALOGUE.elements con id=CHEESE.id entonces añadir nuevo elemento al catálogo en con: CATALOGUE.elements[i].id=CHEESE.id, CATALOGUE.elements[i].curingDays=CHEESE.curingDays, CATALOGUE.elements[i].name=CHEESE.name, CATALOGUE.elements[i].priceKg=CHEESE.priceKg}

1. Agrega al código en lenguaje C del ejercicio anterior, los “asserts” necesarios para asegurar que se cumplen las pre condiciones especificadas.

Cuando inicializamos:

assert(catalogue!=NULL);

assert(stock!=NULL);

assert(orders!=NULL);

En la función add\_ingredient:

assert(cheese!=NULL);

En la función add\_cheese:

assert(catalogue!=NULL);

assert(cheese.count>0);

Pero se puede comprobar en el código C adjunto.

# [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.

function cheese\_items\_with\_ingredient (c: tCatalogue; s: tStock; ing: string) : integer

var

totalCount: integer;

fvar

totalCount:=0;

for (i=1; i<=s.count; i:=i+1)

cheese := get\_cheese\_from\_catalogue(s.elements[i].cheeseID, c);

if (has\_ingredient(cheese, ing) then

totalCount:=totalCount+1;

fif

ffor

return totalCount;

ffunction

function get\_cheese\_from\_catalogue(id: integer, c: tCatalogue) : tCheese

var

i: integer;

cheese: tCheese;

fvar

for (i=1; i<=c.count; i:=i+1)

if (c.elements[i].id=id) then

cheese := c.elements[i];

fif

ffor

return cheese;

ffunction

function has\_ingredient (cheese: tCheese, ing: string) : boolean

var

i: integer;

found: boolean;

fvar

found:=false;

for (i=1; i<=cheese.count; i:=i+1)

if (cheese.ingredients[i]=ing)

found:=true;

fif

ffor

return found;

ffunction