

Prácticas de Programación

PEC1 - 20191

Estudiante

Apellidos: Giménez Gorrís

Nombre: Álvaro

Enunciado

Ejercicio 1: Definición de tipo de datos [20 %]

Una tienda dedicada a vender instrumentos musicales quiere informatizar su sistema de pedidos. A continuación, se enumeran los detalles del sistema que se quiere crear:

- El catálogo de instrumentos contiene un número desconocido de modelos y no está limitado. Cada modelo de instrumento se identifica por un código entero positivo diferente de cero y también nos interesa almacenar una pequeña descripción, su precio en euros y su disponibilidad. Dependiendo del número de unidades que hay del modelo en el stock, la disponibilidad del modelo puede ser: alta (*high*) cuando hay más de 15 unidades; media (*medium*) cuando hay entre 6 y 15 unidades; poca (*low*) cuando hay entre 1 y 5 unidades; o no hay (*none*) cuando no hay ninguna unidad.
- La tienda dispone de un stock de instrumentos. Queremos guardar qué modelos de instrumentos hay en el almacén y de cuántas unidades se dispone de cada uno de ellos. Dadas las dimensiones del almacén, el stock está limitado a 100 modelos de instrumentos.
- Los pedidos contienen el identificador numérico del cliente que hace la compra, la fecha del encargo y el listado de instrumentos musicales que el cliente quiere comprar. Como máximo, un pedido puede contener 10 modelos de instrumentos. Por cada uno de ellos, se indica el número de unidades que el cliente quiere adquirir. El número de pedidos no es conocido ni limitado.

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

a) Definir un tipo de datos **tInstrument** que almacene la información de un modelo de instrumento musical.

b) Definir un tipo de datos **tInstrumentSet** que guarde un código de modelo de instrumento y el número de unidades de este modelo. Este tipo te será útil para el stock y los pedidos.

c) Definir un tipo de datos **tOrder** que represente un pedido de un cliente.

d) Finalmente, definir un tipo de datos **tMusicStore** que contenga el catálogo de modelos de instrumentos de la tienda, el stock de instrumentos y el conjunto de todos los pedidos de los clientes.

Nota: Podéis definir los tipos adicionales que consideráis oportunos.

const

MAX_MODELS : **integer** := 100;

MAX_MODELS_ORDER : **integer** := 10;

end const

type

tAvailability = {NONE, LOW, MEDIUM, HIGH};

tInstrument = **record**

id : **integer**;

description : **string**;

price : **integer**;

availability : tAvailability;

end record

tinstrumentSet = **record**

idModel : **integer**;

stockModel : **integer**;

end record

tDate = **record**

day : **integer**;

month : **integer**;

year : **integer**;

end record

tOrder = **record**

idClient : **integer**;

date : tDate;

shoppingList : **vector**[MAX_MODELS_ORDER] **of** tInstrumentSet;

end record

tMusicStore = **record**

catalogue : **pointer to** tInstrument;

numInstruments : **integer**;

stock : **vector**[MAX_MODELS] **of** tInstrumentSet;

numStock : **integer**;

orders : **pointer to** tOrder;

end record

end type

Ejercicio 2: Manipulación de tablas [40 %]

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

a) init_music_store [10%]: Dada una estructura del tipo **tMusicStore**, la inicializa correctamente dejando los campos internos vacíos.

b) new_instrument [10%]: Dada una estructura del tipo **tMusicStore**, añade en el catálogo un nuevo modelo de instrumento con un código, una descripción y su precio. Inicialmente, el stock no tiene ninguna unidad del nuevo modelo. Si ya existe el código del modelo en el catálogo, entonces no se añade y se muestra un mensaje de error.

c) add_stock [10%]: Dada una estructura del tipo **tMusicStore**, un código de modelo de instrumento, y un número de unidades, añade el número de unidades del modelo al stock. Si el código no se encuentra en el catálogo, devuelve un mensaje de error. Si el stock ya dispone de alguna unidad de este modelo, entonces simplemente se incrementa el número de unidades; en caso contrario, añade el modelo con las unidades al stock. Si no hay espacio en el stock, entonces devuelve un mensaje de error.

d) print_stock [10%]: Dada una estructura del tipo **tMusicStore**, muestra todos los modelos de instrumentos que se encuentran en el catálogo de la tienda. Por cada modelo se muestra su código, su descripción, su precio (con dos decimales) y su disponibilidad.

Tenéis que utilizar la rutina principal que se os facilita en el fichero **main.c** **sin ninguna modificación**. Esta rutina realiza las siguientes tareas:

a. Inicializa la tienda con el catálogo y almacén vacíos, y sin pedidos.

b. Añade los siguientes modelos de instrumentos musicales en el catálogo:

- «Electric Guitar Gibson SG Tributo Walnut» con código 107 y un precio de 875€.
- «Drums Set Tama Imperialstar IE52KH6W Black» con código 351 y un precio de 671.43€.
- «Digital Piano Roland FP-30 with Black Stand» con código 437 y un precio de 719.99€.
- «Violin acoustic Primavera 200 Natural Walnut» con código 405 y un precio de 116.80€.
- «Microphone Shure SM 58 LCE dynamic Silver» con código 351 y un precio de 83.20€. *Producirá un error por código repetido.*
- «Saxophone Yamaha YAS-280 Eb-Alto Gold Lacquer» con código 229 y un precio de 834.08€.

c. Añade las siguientes unidades de instrumentos al stock:

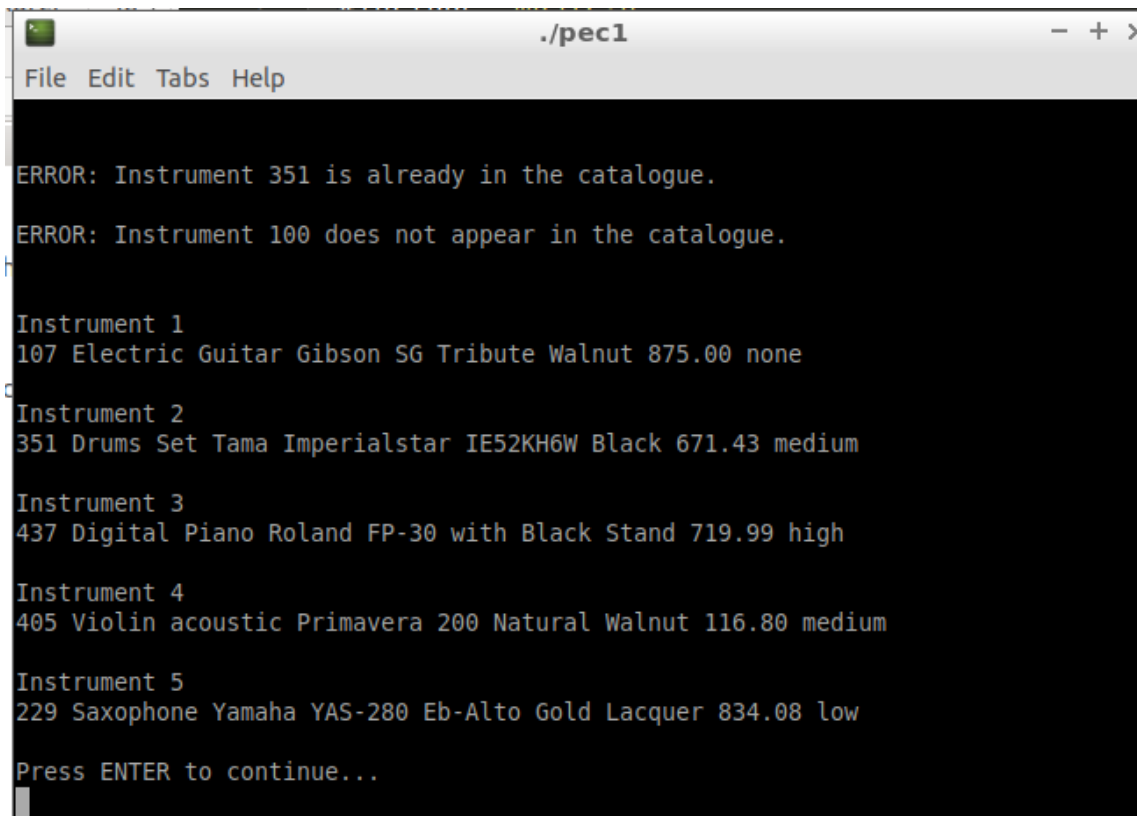
- 3 unidades del modelo con código 351.
- 1 unidad del modelo con código 405.

- 3 unidades del modelo con código 229.
- 10 unidades del modelo con código 100. *Producirá un error por código inexistente en el catálogo.*
- 7 unidades del modelo con código 351.
- 20 unidades del modelo con código 437.
- 7 unidades del modelo con código 405.

d. Muestra los modelos de instrumentos que hay en el catálogo. Por cada modelo se muestra su código, descripción, precio y disponibilidad.

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

```
ERROR: Instrument 351 is already in the catalogue
ERROR: Instrument 100 does not appear in the catalogue
107. Electric Guitar Gibson SG Tribute Walnut 875.00 none
351. Drums Set Tama Imperialstar IE52KH6W Black 671.43 medium
437. Digital Piano Roland FP-30 with Black Stand 719.99 high
405. Violin acoustic Primavera 200 Natural Walnut 116.80 medium
229. Saxophone Yamaha YAS-280 Eb-Alto Gold Lacquer 834.08 low
```



```
./pec1
File Edit Tabs Help

ERROR: Instrument 351 is already in the catalogue.
ERROR: Instrument 100 does not appear in the catalogue.

Instrument 1
107 Electric Guitar Gibson SG Tribute Walnut 875.00 none

Instrument 2
351 Drums Set Tama Imperialstar IE52KH6W Black 671.43 medium

Instrument 3
437 Digital Piano Roland FP-30 with Black Stand 719.99 high

Instrument 4
405 Violin acoustic Primavera 200 Natural Walnut 116.80 medium

Instrument 5
229 Saxophone Yamaha YAS-280 Eb-Alto Gold Lacquer 834.08 low

Press ENTER to continue...
```

Ejercicio 3: Especificación formal [20 %]

Define las declaraciones (**no las implementaciones**) de los métodos **init_music_store** y **new_instrument** del ejercicio anterior en **lenguaje algorítmico**.

- a) Por cada declaración, añade las pre y post condiciones en lenguaje algorítmico.

action init_musicStore (**in/out** ms : tMusicStore)

Pre: {ms = MS}

No hay que definir las variables porque vienen todas como parámetro.

Post: {ms.numInstruments = 0 y ms.numStock = 0 y ms.stock[i].idModel = 0 y ms.stock[i].stockModel = 0}

action new_instrument (**in/out** ms : tMusicStore; **in** code: **integer**; **in** descript: **string**; **in** price: **real**)

Pre: {ms = MS y code = CODE y CODE > 0 y descript = DESCRIPT y price = PRICE}

found : tBool;

Post: {($\exists i : 0 < i \leq \text{MS.numInstruments}$: ms.numInstruments = MS.numInstruments y ms.stock[i].idModel = CODE) o (ms.numInstruments = MS.numInstruments + 1 y ms.stock[numInstruments].idModel)}

b) Añade al código en lenguaje C del ejercicio anterior, los “asserts” necesarios para asegurar que se cumplen las precondiciones especificadas.

Adjunto pantallazo de los “asserts” introducidos:

```
void init_music_store(tMusicStore *ms){  
    /* Check PRE conditions */  
    assert(ms != NULL);  
}
```

```
void new_instrument(tMusicStore *ms, int code, st  
    //check pre conditions  
  
    assert(ms != NULL);  
    assert(code > 0);  
}
```

Ejercicio 4: Diseño ascendente [20 %]

Define la función **total_from_orders** que calcula el total de dinero que recibirá la tienda si completa todos los pedidos que tiene actualmente.

function total_from_orders (ms:tMusicStore) : **real**

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

Primero modifiko la tupla tOrder para que tenga sentido el enunciado:

type

```
tOrder = record
    idClient : integer;
    date : tDate;
    shoppingList : vector[MAX_MODELS_ORDER] of tInstrumentSet;
    pending : boolean;
    orderPrice : real;
end record
```

end type

A continuación, implemento la función pedida. Para ello, he realizado una función que guarda los pedidos pendientes en una lista y otra que los suma. La elaboración de la lista de pedidos pendientes con todos los datos completos y no con solo los estrictamente necesarios tiene como objetivo que el código se pueda aplicar a posibles futuras funciones.

Nivel 1:

```
function total_from_orders (ms:tMusicStore) : real
    var
        pendingOrderList : tOrder;
        ordersAmount : integer;
        total : real;
    end var

    total := 0;
    ordersAmount := 0;

    find_pending_orders (ms, pendingList, ordersAmount);

    if (ordersAmount > 0) then
        total := sum_order_list (pendingList, ordersAmount);
    end if

    return total;
end function
```

Nivel 2:

```
action find_pending_orders (in ms : tMusicStore, out pendingList : tOrder, out
ordersAmount : integer)
    var
        pending : boolean;
        i : integer;
    end var
    i := 1;
    while ms.stock[i].idClient != 0 do
        pending := search_pending_orders (pendingList);
        if (pending = true) then
```

```

        ordersAmount := ordersAmount + 1;
        save_data(ms.stock[i], pendingList[ordersAmount]);
    end if
    i := i + 1;

end while

end action

function sum_order_list (pendingList : tOrder, ordersAmount : integer) : real
    var
        i : integer;
        num : real;
    end var

    num := 0;

    for i = 1 to ordersAmount do
        num = num + pendingList[i].price;
    end for
    return num;
end function

```

Nivel 3:

```

function search_pending_orders (pendingList : tOrder) : boolean
    var
        pending : boolean;
    end var

    pending := false;

    if (pendingList.pending = true) then
        pending := true;
    end if

    return pending;
end function

action save_data(in ms.stock, out pendingList : tOrder)
    pendingList.idClient := ms.stock.idclient;
    pendingList.date := ms.stock.date;
    pendingList.shoppingList := ms.stock.shoppingList;
    pendingList.pending := ms.stock.pending;
    pendingList.orderPrice := ms.stock.orderPrice;
end action

```