UNIVERSIDADE DA CORUÑA

FACULTADE DE INFORMÁTICA
Programación II – Curso 2016/17

Práctica 2: Especificación de requisitos

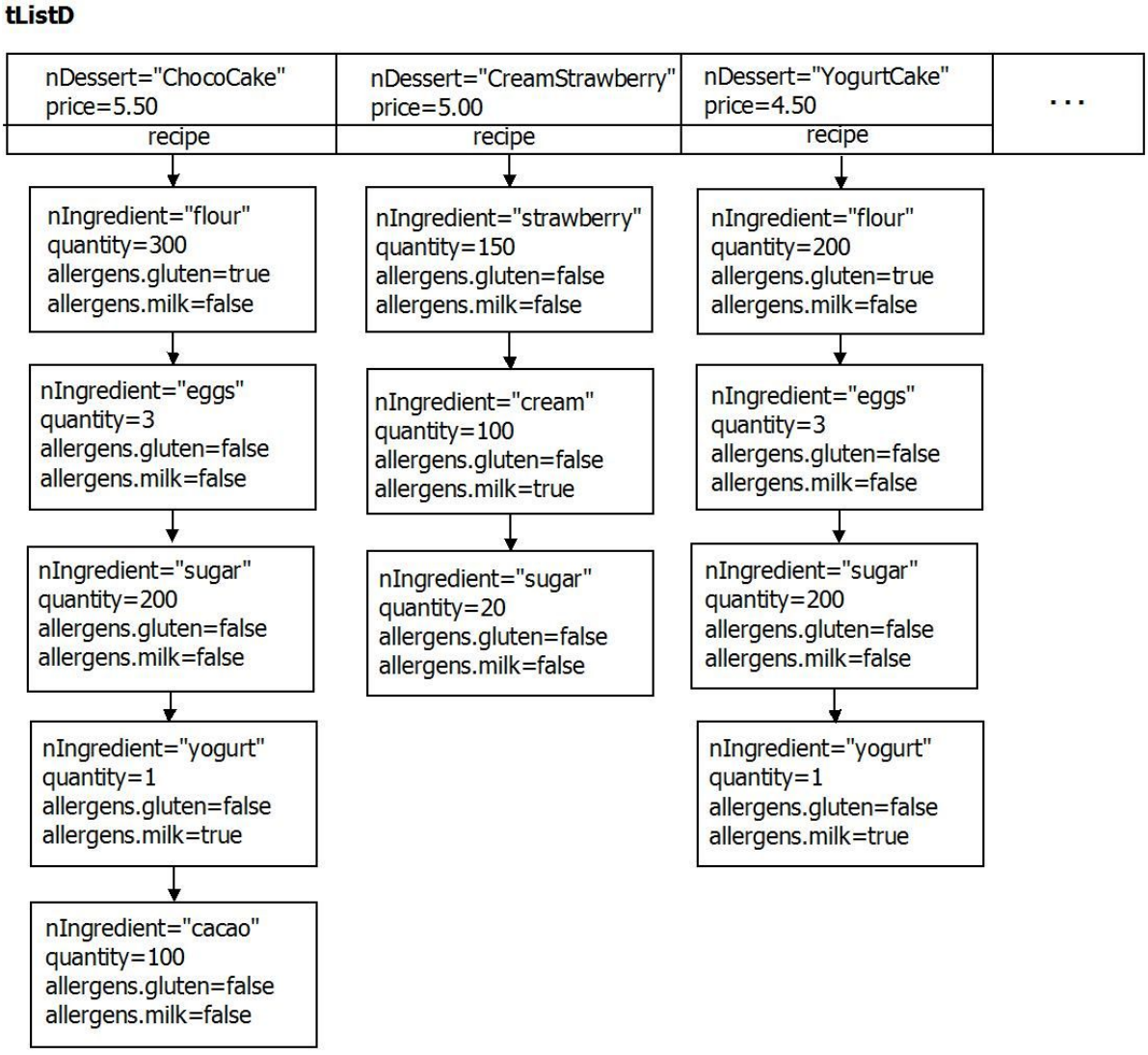
1. El problema

Mientras que la Práctica 1 se centró en gestionar la *despensa* del local de hostelería DESSERTS, en la Práctica 2 se incluirá también la gestión de la *carta* de postres. Así, el programa mantendrá actualizada tanto la *carta* de postres e ingredientes para elaborarlos como el stock de ingredientes disponibles en la *despensa*. Cuando se pida un postre a cocina, el sistema deberá comprobar que se disponga de cada uno de sus ingredientes en cantidad suficiente para su elaboración. Como consecuencia la *despensa* se actualizará, eliminando aquellos ingredientes que se hayan agotado. Paralelamente, la *carta* se modificará eliminando aquellos postres para los cuales no exista cantidad suficiente de ingredientes. Esto permite a los camareros informar a los clientes de que ya no es posible servir dichos postres.

2. Estructuras de datos

Para resolver el problema se implementarán varios tipos abstractos de datos (TADs). Dos de ellos emplearán ESTRUCTURAS DINÁMICAS: una **cola** para almacenar las distintas operaciones (**TAD RequestQueue**) y una **lista no ordenada** para almacenar los ingredientes de la despensa (**TAD IngredientList)**. Por último, se implementará una **multilista** (**TAD DessertsList**) (véase la figura de la siguiente página) que empleará tanto una ESTRUCTURA ESTÁTICA como ESTRUCTURAS DINÁMICAS, así la multilista estará compuesta por una **lista ordenada** implementada estáticamente donde se almacenarán los postres, y por **varias listas no ordenadas** implementadas dinámicamente que, para cada postre, mantiene los ingredientes que lo forman. Por simplicidad, se ha mantenido la lista de ingredientes, ya desarrollada en la práctica 1, como no ordenada, si bien, dado que en esta lista se realizan numerosas búsquedas sería más eficiente haber empleado una lista ordenada.

Fuera de sus correspondientes units, cada TAD se manejará exclusivamente a través de las operaciones que ofrece su interfaz.



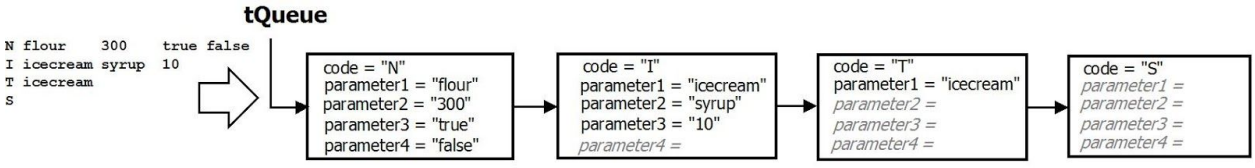
2.1. TAD RequestQueue

Para gestionar las peticiones recibidas por el sistema (descritas en el Aptdo. 3 de este documento), emplearemos un TAD Cola (de) Peticiones o **RequestQueue** (unit `RequestQueue.pas`). Se trata de **una cola** en la que se irán almacenando, por orden de llegada, las peticiones recibidas de los usuarios del sistema para su posterior procesamiento. La implementación de dicho TAD deberá ser **dinámica**.

2.1.1. Tipos de datos incluidos en el TAD RequestQueue

`tQueue`	Representa una cola de peticiones
`tCode`	Tipo de petición recibida (`char`); véase Aptdo. 3.1 para más detalles: [N]ew, [M]odify, [R]emove, [A]llergens, [S]tock, new [D]essert, add [I]ngredient, [T]ake off, [V]isualize, [O]rder
`tItemQ`	Datos de un elemento de la cola (correspondiente a una petición); está compuesto por los siguientes campos: • `code`: de tipo `tCode`; representa la clase de la petición • `parameter1`: de tipo cadena; según el tipo de petición representará el nombre del ingrediente (`nIngredient`), una cantidad (`minQuantity`), el *nombre del postre* (`nDessert`) o si un ingrediente contiene gluten o no (`GG`) • `parameter2`: de tipo cadena; según la petición representará el nombre del ingrediente (`nIngredient`), una cantidad (`quantity`) o si un ingrediente contiene lácteos (`MM`). • `paramater3`: de tipo cadena; según la petición representará si un ingrediente contiene gluten (`GG`) o una cantidad (`quantity`). • `paramater4`: de tipo cadena; utilizado en una única operación para indicar si un ingrediente contiene lácteos (`MM`).
`tPosQ`	Posición de un elemento de la cola
`NULLQ`	Constante para representar posiciones nulas en la cola

A modo de ejemplo, tal como se muestra gráficamente a continuación, el contenido del fichero de texto de entrada (parte izquierda de la figura) se volcará en la estructura, obteniendo una cola como la presentada en la parte derecha:



2.1.2. Operaciones incluidas en el TAD RequestQueue

A mayores de las especificaciones abajo indicadas, una precondición común para todas las operaciones (salvo `createEmptyQueue`) es que la cola debe estar previamente inicializada:

- `createEmptyQueue (tQueue) → tQueue`
  Crea una cola vacía.
  PostCD: La cola queda inicializada y vacía.

- `isEmptyQueue (tQueue) → Boolean`
  Determina si la cola está vacía.

- `enqueue (tQueue, tItemQ) → tQueue, Boolean`
  Inserta un nuevo elemento (`tItemQ`) en la cola. Devuelve `false` si no hay memoria suficiente para realizar la operación.

- `front (tQueue) → tItemQ`
  Devuelve el contenido (`tItemQ`) del frente de la cola (i.e. el elemento más antiguo).
  PreCD: la cola no está vacía.

- `dequeue (tQueue) → tQueue`
  Elimina el elemento que está en el frente de la cola.
  PreCD: la cola no está vacía.

2.2. TAD IngredientList

Como ya se hizo en la Práctica 1, a la hora de representar la *despensa* del local el sistema utilizará una lista, en este caso un TAD Lista (de) Ingredientes o **IngredientList (**unit `IngredientList.pas`). En este caso, la implementación del TAD corresponderá a una **lista dinámica no ordenada**. En lo que respecta a los tipos de datos y operaciones asociados a este TAD, **son básicamente los mismos que para el TAD Lista empleado Práctica 1, si bien algunos han sido renombrados** para evitar confundirlos con los del TAD Lista (de) Postres (**DessertList**) descrito en el siguiente apartado. Indicamos a continuación la correspondencia entre los nombres empleados en la Práctica 1 para el TAD Lista y los empleados en la presente práctica para el TAD Lista (de) Ingredientes (**IngredientList**), aquellos nombres que no figuren en estas tablas **no cambian.**

Tipo Práctica 1	**Tipo Práctica 2**
tList	tListI
tPosL	tPosI
tItem	tItemI
NULL	NULLI

Operación Práctica 1	**Operación Práctica 2**
createEmptyList (tList) → tList	createEmptyListI (tListI) → tListI
isEmptyList (tList) → Boolean	isEmptyListI (tListI) → Boolean
first (tList) → tPosL	firstI (tListI) → tPosI
last (tList) → tPosL	lastI (tListI) → tPosI
next (tPosL, tList) → tPosL	nextI (tPosI, tListI) → tPosI
previous (tPosL, tList) → tPosL	previousI (tPosI, tListI) → tPosI
insertItem (tItem, tPosL, tList) → tList, Boolean	insertItemI (tItemI, tPosI, tListI) → tListI, Boolean
deleteAtPosition (tPosL, tList) → tList	deleteAtPositionI (tPosI, tListI) → tListI
getItem (tPosL, tList) → tItem	getItemI (tPosI, tListI) → tItemI
updateItem (tList, tPosL, tItem) → tList	updateItemI (tListI, tPosI, tItemI) → tListI
findItem (tnIngredient, tList) → tPosL	findItemI (tnIngredient, tListI) → tPosI

2.3. TAD DessertList

Finalmente, se implementará un TAD Lista (de) Postres o **DessertList** (unit `DessertList.pas`) en la cual se almacenarán los postres que ofrece el local, así como los ingredientes necesarios para su elaboración. Dicho TAD se implementará como una **lista estática <u>ordenada por el nombre del postre</u>**.

2.3.1. Tipos de datos incluidos en el TAD DessertList

`tListD`	Representa una lista de postres **ordenada por el nombre** del postre
`tnDessert`	Nombre del postre (cadena)
`tPrice`	Precio de un postre (real)
`tItemD`	Datos de un elemento de la lista (correspondiente a un postre); está compuesto por los siguientes campos: • `nDessert`: *nombre del postre*, de tipo `tnDessert` • `price`: *precio del postre*, de tipo `tPrice` • `recipe`: lista con los ingredientes necesarios para elaborar el postre, de tipo tListI
`tPosD`	posición de un elemento en la lista

`NULLD`	constante utilizada para representar posiciones nulas

2.3.2. Operaciones incluídas en el TAD DessertList

A mayores de las especificaciones abajo indicadas, una precondición común para todas estas operaciones (salvo `createEmptyListD`) es que la lista debe estar previamente inicializada:

- `createEmptyListD (tListD) → tListD`
  Crea una lista vacía.
  PostCD: La lista queda inicializada y vacía.

- `isEmptyListD (tListD) → Boolean`
  Determina si la lista está vacía.

- `firstD (tListD) → tPosD`
  Devuelve la posición del primer elemento de la lista.
  PreCD: La lista no está vacía.

- `lastD (tListD) → tPosD`
  Devuelve la posición del último elemento de la lista.
  PreCD: La lista no está vacía.

- `nextD (tPosD, tListD) → tPosD`
  Devuelve la posición en la lista del siguiente elemento a la posición indicada (o `NULLD` si la posición no tiene siguiente).
  PreCD: La posición indicada es una posición válida en la lista.

- `previousD (tPosD, tListD) → tPosD`
  Devuelve la posición en la lista del anterior elemento a la posición indicada (o `NULLD` si la posición no tiene anterior).
  PreCD: La posición indicada es una posición válida en la lista.

- `insertItemD (tItemD, tListD) → tListD , Boolean`
  Inserta un elemento en la lista **de forma ordenada** en base al campo `nDessert`. Devuelve un valor `false` si no hay memoria suficiente para realizar la operación.
  PostCD: Las posiciones de los elementos de la lista a continuación del insertado dejan de ser válidas

- `deleteAtPositionD (tPosD, tListD) → tListD`
  Elimina de la lista el elemento que ocupa la posición indicada.
  PreCD: La posición indicada es una posición válida en la lista.
  PostCD: Tanto la posición del elemento eliminado como aquéllas de los elementos de la lista a continuación de él dejan de ser válidas

- `getItemD (tPosD, tListD) → tItemD`
  Devuelve el contenido del elemento de la lista que ocupa la posición indicada.
  PreCD: La posición indicada es una posición válida en la lista.

- `updateItemD (tListD, tPosD, tItemD) → tListD`
  Modifica el contenido del elemento situado en la posición indicada.
  PreCD: La posición indicada es una posición válida en la lista.
  PostCD: El orden de los elementos de la lista no se ve modificado.

- `findItemD (tnDessert, tListD) → tPosD`
  Devuelve la posición **del primer elemento de la lista** cuyo nombre de postre (campo `nDessert`) coincide con el indicado (o `NULLD` si no existe tal elemento).

3. Descripción de la tarea

Al igual que en la Práctica 1, la tarea consiste en implementar un único programa principal (`main.pas`) que gestione el stock almacenado en la *despensa* y la *carta* de postres del local de hostelería DESSERTS. El programa irá procesando las peticiones recibidas por el sistema y, en base a ello, actualizará convenientemente el estado del stock en la *despensa* y el de la *carta*.

3.1. Peticiones a procesar

Dichas peticiones, y el formato empleado en sus respectivas representaciones, son las siguientes:

`N nIngredient quantity GG MM`	***[N]ew***: se añade un nuevo ingrediente a la *despensa*, asignándole el nombre `nIngredient` y asociándole una cantidad `quantity`. El campo booleano `gluten` tomará el valor del parámetro `GG`. De igual forma para el campo booleano `milk` y el parámetro `MM`.
`M nIngredient quantity`	***[M]odify***: modifica la cantidad actualmente disponible en la despensa del ingrediente con nombre `nIngredient` incrementando o decrementando su campo `quantity` en la cantidad (positiva o negativa) indicada por el parámetro `quantity` de la petición.
`R minQuantity`	***[R]emove***: elimina de la despensa todos los ingredientes cuya cantidad actual disponible `quantity` sea inferior a `minQuantity`.

`A GG MM`	***[A]llergens***: muestra todos los ingredientes que contienen algún alérgeno, en función de los parámetros `GG` y `MM`. Si `GG` está a `true` se mostrarán los ingredientes con el campo `gluten` a `true`. Análogamente, si `MM` está a `true` se mostrarán los ingredientes con `milk` a `true`.
`S [minQuantity]`	***[S]tock***: muestra todos los ingredientes disponibles en la *despensa* con la cantidad de cada uno de ellos. Si se incluye el parámetro opcional `minQuantity` se mostrarán entonces sólo aquellos ingredientes cuya cantidad actual sea inferior a `minQuantity`.
`D nDessert price`	***new [D]essert***: añade a la *carta* de postres un nuevo postre de nombre `nDessert` y precio el indicado por `price`.
`I nDessert nIngredient quantity`	***add [I]ngredient***: añade al postre `nDessert` el ingrediente `nIngredient` con la cantidad `quantity`.
`T nDessert`	***[T]ake off***: elimina de la *carta* el postre `nDessert`.
`V`	***[V]isualize:*** muestra todos los postres disponibles en la *carta*.
`O nDessert`	***[O]rder***: corresponde a un pedido a cocina del postre `nDessert` por parte de un cliente.

3.2. Ejecución

En el programa principal se implementará un bucle que procese las peticiones recibidas por el servidor. Como en la Práctica 1, el programa deberá **funcionar en modo batch (i.e. sin interaccionar con el usuario)**, tomando para ello como entrada un fichero de texto que contiene la secuencia de peticiones a procesar y cuyo nombre (y ruta, de ser precisa) se le pasará como parámetro a la hora de ejecutarlo. Sin embargo, al contrario que en la Práctica 1, las peticiones no se ejecutarán en el momento de leerlas del fichero, sino que serán previamente cargadas desde el fichero a la **cola de peticiones del sistema (TAD RequestQueue)**, que actuará a modo de *buffer* de entrada, para ser posteriormente procesadas una a una leyéndolas de la cola.

De este modo, podemos diferenciar tres fases en la ejecución del programa, las cuales describimos a continuación.

3.2.1. Fase 1: Inicialización

Durante esta primera fase se llevarán a cabo las siguientes operaciones:

1. Lectura del fichero de entrada que contiene las peticiones a procesar por el servidor e inserción de las mismas en la cola de peticiones del sistema (TAD RequestQueue). De acuerdo a la tabla del Aptdo. 3.1 cada línea del fichero consta, dependiendo del comando, de entre uno a cuatro campos en este orden:
   1.1. (Obligatorio, 1 carácter alfanumérico) Código de la operación.
   1.2. (Opcional, 12 caracteres alfanuméricos). De ser aplicable, según el tipo de petición representará: un nombre de ingrediente (`nIngredient`), una cantidad (`minQuantity`), un *nombre del postre* (`nDessert`) o si un ingrediente contiene gluten o no (`GG`).
   1.3. (Opcional, 12 caracteres alfanuméricos). De ser aplicable, según el tipo de petición representará: un nombre de ingrediente (`nIngredient`), una cantidad (`quantity`) o si un ingrediente contiene lácteos o no (`MM`).
   1.4. (Opcional, 7 caracteres alfanuméricos). De ser aplicable, según el tipo de petición representará si un ingrediente contiene gluten (`GG`) o una cantidad (`quantity`).
   1.5. (Opcional, 5 caracteres alfanuméricos). Empleado únicamente en una operación, indica si un ingrediente contiene lácteos o no (`MM`).

2. Inicialización de la listas correspondientes al stock almacenado en la *despensa* y a la *carta* de postres.

3.2.2. Fase 2: Procesado de peticiones

Mientras queden peticiones en la cola del servidor, el sistema las irá procesando una a una, por orden de llegada. Para cada una de ellas procederemos, a su vez, en dos pasos:

PASO I: Se muestra la operación a realizar. Para ello se imprimirá un mensaje del tipo:

```
*****************************************
Task T: XX YY GG MM
*****************************************
```

que, grosso modo, muestra los datos sobre la petición recibida, de forma que:

- `T` es el tipo de operación a realizar (*new*, *modify*, *remove, allergens*, *stock, newDessert, addIngredient, takeOff, visualize, order*) conforme a los códigos de la tabla anterior (es decir: `N`, `M`, `R`, `A`, `S`, `D`, `I`, `T`, `V` u `O`).
- `XX` puede ser el nombre (`nIngredient`) del ingrediente a procesar (en el caso de las operaciones *new* y *modify*), la cantidad (`minQuantity`) de ingrediente fijada

como umbral (en el caso de las operaciones *remove* y *stock*), o bien el nombre del postre (`nDessert`) a procesar (en las operaciones *newDessert*, *addIngredient*, *takeOff* y *order*).

- `YY` puede ser la cantidad (`quantity`) del ingrediente (si es una operación *new* o *modify*), el nombre del ingrediente (`nIngredient`) a añadir (si es una operación *addIngredient*) o un precio (`price`) de un postre (si es una operación *newDessert*).
- `GG` indicar si un ingrediente nuevo contiene `gluten` o no (en la operación *new*), si se ha de mostrar un ingrediente con `gluten` (en la operación *allergens*), o bien la cantidad (`quantity`) de un ingrediente (en la operación *addIngredient*).
- `MM` indica si un ingrediente nuevo contiene leche (`milk`) o no (en la operación *new*), o si se ha de mostrar un ingrediente con leche (en la operación *allergens*).

Adviértase que sólo se deberán imprimir los parámetros que correspondan a la petición concreta que está siendo procesada en ese momento. Por ejemplo, para una petición "`I IceCream Syrup 10`" (es decir, el usuario desea añadir al postre *IceCream* el ingrediente *Syrup* en una cantidad de *10* unidades), se mostraría únicamente.

```
****************************************
Task I: IceCream  Syrup  10
****************************************
```

PASO II: Se procesa la petición correspondiente:

Las operaciones **[N]ew, [S]tock, [A]llergens, [M]odify** y **[R]emove** se encuentran explicadas en la práctica 1, por brevedad, no las vamos a incluir nuevamente en esta especificación. Deben implementarse tal y como se indicaba en la especificación de la práctica 1. A la hora de desarrollar estas operaciones, se recomienda encarecidamente leer el documento de ***fe de erratas***, así como el documento de ***errores frecuentes***, ambos en la página web (Moodle) de la asignatura.

1. Si la operación es añadir un nuevo postre a la carta (**new [D]essert**) se deberá añadir dicho postre a nuestra carta y, en caso de no producirse ningún error, se mostrará por pantalla un mensaje del tipo:

   ```
   **** Adding new dessert to menu:  DD PP.PP euros
   ```

   donde `DD` indica el nombre del postre (`nDessert`) y PP el precio (`price`) con dos cifras decimales.

   En un menú no pueden figurar postres repetidos, por lo que si el nombre ya existe en la carta, se mostrará entonces el siguiente mensaje de error:

   ```
   ++++ ERROR Adding new dessert: dessert DD already exists
   ```

   donde `DD` indica el nombre del postre que se pretendía añadir, abortándose además la operación, con lo que el postre NO se insertará en la carta.

En el caso de que no se pudiese incorporar el nuevo postre a la carta por falta de memoria, se mostrará en lugar del mensaje usual un mensaje tal que así:

```
**** WARNING: Out of memory for adding dessert DD
```

siendo `DD` otra vez el nombre del postre en cuestión.

2. La operación **add [I]ngredient** debe comprobar que el nombre del postre (`nDessert`) existe en la carta y, de no estar, se mostrará el siguiente mensaje de error:

```
++++ ERROR Adding Ingredient to dessert DD: dessert does not exist
```

Si el postre existe, se debe comprobar entonces que el ingrediente en cuestión no esté ya incluido entre los ingredientes del postre; de estarlo, se mostrará el siguiente mensaje de error:

```
++++ ERROR Adding Ingredient to dessert DD: ingredient XX already exists
```

Además, se debe comprobar que la cantidad indicada para el ingrediente sea válida (>0); de no ser así se mostraría el siguiente mensaje de error.

```
++++ ERROR Adding Ingredient to dessert DD: Invalid quantity
```

En caso de que el ingrediente se consiga añadir con éxito, se mostrará el siguiente mensaje:

```
**** Adding new ingredient to dessert DD:  XX YY GG MM
```

donde `GG` y `MM` indican los alérgenos del ingrediente (`true/false`) en función de si éste contiene gluten (`GG`) o lácteos (`MM`). Obsérvese que los parámetros de la petición **add [I]ngredient** no incluyen los alérgenos del ingrediente que está siendo agregado (véase la tabla de la Sección 3.1). De este modo, a la hora de indicarlos habrá que buscar dicho ingrediente en la despensa y tomar esos valores de allí. Por simplicidad, podemos suponer que NO se dará NUNCA el caso de que el ingrediente agregado no se encuentre en la despensa.

En el caso de que no se pudiese incorporar el nuevo ingrediente al postre por falta de memoria, se mostrará en lugar del mensaje anterior un mensaje tal que así:

```
**** WARNING: Out of memory for adding ingredient XX to dessert DD
```

Para todos los casos anteriores, `DD` es el nombre del postre (`nDessert`), `XX` es el nombre del ingrediente (`nIngredient`) e `YY` es la cantidad requerida (`quantity`) del mismo.

3. En la operación **[T]ake off** el postre indicado se eliminará de la carta de postres. Para ello hay que comprobar primero que exista tal postre; en caso contrario se mostrará el siguiente mensaje:

```
++++ ERROR Taking off dessert DD: dessert does not exist
```

En caso de que sí esté en la carta, el postre se eliminará, junto con su correspondiente lista de ingredientes, y se mostrará además el siguiente mensaje:

```
**** Removing Dessert DD from the menu
```

En ambos casos, `DD` indica el nombre del postre (`nDessert`) a eliminar.

4. La operación **[V]isualize** permite mostrar la carta de postres de Desserts. En caso de que no haya postres en la lista, se mostrará el siguiente mensaje:

```
**** Menu not available
```

El local quiere también ofrecer en la carta información sobre los alérgenos, de modo que si un postre NO incluye un tipo de alérgeno, (es decir, ninguno de sus ingredientes lo tiene), se mostrará entonces un mensaje que lo indique. Así, la carta aparecerá con el siguiente formato:

```
**** Menu ****

* DD1: PP.PP1.
  Contains: YY11 YY12 YY13
  Gluten Free. Milk free.
* DD2: PP.PP2.
  Contains: YY21 YY22
* DD3: PP.PP3.
  Recipe not included.
       ...
* DDn: PP.PPn.
  Contains: YYn1 YYn2..YYnm
  Gluten Free.
```

donde `DDi` indica el nombre de los distintos postres (`nDessert`) en carta, listados por orden alfabético de la A a la Z, siendo `PP.PPi` el precio de los mismos y `YYi1...YYim` los `m` ingredientes que forman el `i`-ésimo postre. En el caso de que el postre `DDi` no tenga ningún ingrediente se deberá mostrar el mensaje "`Recipe not included`" tal y como se aprecia para el postre `DD3` en el ejemplo anterior.

5. Por último, la operación **[O]rder** gestiona el pedido de un cliente en el local. Para ello, se busca en la carta el postre indicado; si no existe se debe imprimir un mensaje del tipo:

```
++++ ERROR Order Not attended. Unknown dessert DD
```

Si el postre existe, se deberán mostrar los ingredientes necesarios en el orden que se han insertado en la receta **(i.e., de más antiguo a más reciente)**, de acuerdo con el siguiente formato:

```
**** Dessert DD
* Ingredient XX1 QQ1 (available QS1)
* Ingredient XX2 QQ2 (available QS2)
* Ingredient XX3 QQ3 (available QS3)
      ...
* Ingredient XXn QQn (available QSn)
```

donde `DD` es el nombre del postre, `XXi` es el nombre del ingrediente y `QQi` es la cantidad necesaria para la elaboración del mismo y `QSi` es la cantidad disponible en el stock y, al igual que en la operación *add [I]ngredient*, por cuestiones de simplicidad, se supone que el ingrediente se encuentra siempre en la despensa. Por el mismo motivo, supondremos que todo postre siempre va a tener ingredientes.

Para elaborar dicho postre, habrá que determinar la disponibilidad en la despensa de cada uno de dichos ingredientes en las cantidades necesarias. En caso de que no haya cantidad suficiente de algún ingrediente, se procederá entonces a eliminar automáticamente dicho postre de la carta, imprimiendo además un mensaje del tipo:

```
**** Order not attended. Not enough ingredients.
**** Removing dessert DD from the menu
```

Por el contrario, si existen ingredientes suficientes, se procede a la elaboración del postre, siguiendo los siguientes pasos:

i. Imprimir un mensaje que indique que se va a preparar el postre:

   ```
   **** Order attended. Preparing dessert DD.
   ```

   donde `DD` es el nombre del postre.

ii. Actualizar los ingredientes en la despensa, reduciendo las cantidades empleadas en la elaboración del postre. Dicha reducción puede provocar que se agoten uno o varios ingredientes en la despensa.

iii. Si se agotan ingredientes, se procederá a eliminar aquel(los) ingrediente(s) de la despensa agotados, mostrando asimismo un siguiente mensaje como el siguiente:

   ```
   **** Removing ingredient XXi from stock
   ```

   Además, en este caso, inmediatamente a continuación deberán eliminarse también de la carta todos aquellos postres que incluyan dicho ingrediente en su

preparación. Así, en caso de que exista al menos un postre con este ingrediente se mostrará el siguiente mensaje:

```
**** Removing desserts that contains XXi
* Removing DD1i
* Removing DD2i
      ...
* Removing DDni
```

donde `XXi` indica el nombre del ingrediente y `DDji` indica el nombre uno de esos postres eliminados. En caso de que no hubiese ningún otro postre afectado por la eliminación de dicho ingrediente, mostraríamos entonces en su lugar el siguiente mensaje:

```
**** No more desserts affected
```

iv. Si no se agotan ingredientes, se mostrará el siguiente mensaje:

```
**** Stock updated. No ingredients removed.
```

3.2.3. Fase 3: Liberación de recursos

Para terminar, una vez procesadas todas las peticiones recibidas, se procederá a liberar la memoria del sistema vaciando tanto la despensa como la lista de postres, teniendo especial cuidado de vaciar también, para cada postre, sus ingredientes.

4. Notas de implementación

Apuntes breves acerca de diversas cuestiones de implementación:

- Al ser *IngredientList* una lista NO ORDENADA, a la hora de insertar un nuevo ingrediente en la despensa lo haremos **en la última posición de la lista (más reciente)**, como ya hacíamos en la Práctica 1.
- Por simplicidad, a la hora de implementar la lista de postres supondremos que el **número máximo de postres** que puede contener **es 20**.
- A la hora de convertir los parámetros de la cola (de tipo `string`) a otros tipos de datos (`boolean`, `integer` y `real`), se recomienda usar, respectivamente, las funciones `StrToBool`, `StrToInt` y `StrToFloat`. Con respecto a esta última, los ficheros de prueba proporcionados usan el punto como separador decimal y se procesan adecuadamente en los equipos de referencia. No obstante, otros sistemas operativos pueden emplear otro separador decimal (por ejemplo, la coma) lo que provocaría errores de ejecución, para solventarlos es necesario indicar el separador decimal, por ejemplo como se detalla en el siguiente código:

```
Var
        format: TFormatSettings;
        r: real;
        param1:string;

 format.DecimalSeparator:='.';
 r:=StrToFloat(param1, format);
```

5. Lectura de ficheros

El fichero `read.pas` facilita la lectura de los ficheros de prueba. Se proporcionan varios ficheros para procesar las diferentes peticiones (`new.txt`, `modify.txt`, `allergens.txt`, `remove.txt`, `dessert.txt`, `ingredient.txt`, `order.txt`, `takeoff.txt`). Todos ellos constituyen el conjunto de operaciones mínimas exigidas y se pueden probar de manera conjunta con el script proporcionado (`scriptP2.sh`), descrito en el Aptdo 6.2.

6. Normas de presentación

Además de ejecutar correctamente todas las operaciones de la forma descrita anteriormente, la práctica debe ajustarse a unas normas estrictas de presentación que, de no cumplirse, podrán conllevar una penalización en la calificación de la misma.

6.1. Estructura que deberá tener cada programa/unit desarrollados

- Los nombres de los ficheros del programa y de las *units* deberán ajustarse a los especificados en este documento. En particular, es importante destacar que el programa principal debe llamarse `main.pas`. Recuérdese que GNU/Linux, el sistema sobre el que corregiremos la práctica, **diferencia entre mayúsculas y minúsculas**.

- En el encabezamiento del programa/*units* deberá constar, entre comentarios, la siguiente información:

```
TITLE: PROGRAMMING II LABS
SUBTITLE: Practical 2
AUTHOR 1: **************************** LOGIN 1: **********
AUTHOR 2: **************************** LOGIN 2: **********
GROUP: *.*
DATE: **/**/****
```

- Con respecto al cuerpo del programa/*units*:
    - El código deberá estar convenientemente comentado, incluyendo las variables empleadas. Los comentarios han de ser concisos pero explicativos.

- Después de la cabecera de cada procedimiento o función, se incluirá la siguiente información correspondiente a su *especificación*, tal y como se explicó en el TGR correspondiente:
  - *Objetivo* del procedimiento/función.
  - *Entradas* (identificador y breve descripción, una por línea).
  - *Salidas* (identificador y breve descripción, una por línea).
  - *Precondiciones* (condiciones que han de cumplir las entradas para el correcto funcionamiento de la subrutina).
  - *Postcondiciones* (otras consecuencias de la ejecución de la subrutina que no quedan reflejadas en la descripción del objetivo o de las salidas).

6.2. Criterios de valoración de la práctica

A la hora de corregir la práctica, además del empleo de buenas prácticas de programación, la corrección de la implementación realizada y el cumplimiento de las condiciones especificadas en el enunciado, se tendrán especialmente en cuenta los siguientes aspectos:

- *Eficacia:* que se cumplan las especificaciones, implementando correctamente todas las funcionalidades requeridas. A este respecto debemos llamar de nuevo la atención sobre el hecho de que la práctica entregada por el estudiante DEBERÁ FUNCIONAR PERFECTAMENTE con el script proporcionado con este enunciado (`scriptP2.sh`), del que ya se habló en el Aptdo. 5. De no ser así, la práctica será calificada automáticamente con un NO APTO. Obsérvese, sin embargo, que el adecuado funcionamiento con este script tampoco garantiza la superación de la práctica si no se cumplen adecuadamente los requisitos restantes, teniéndose de nuevo en cuenta la *eficacia* de la solución.
- *Eficiencia:* deberá evitarse la ejecución de operaciones y procesos innecesarios o claramente ineficientes.
- *Control de errores:* control intensivo de todos los errores de ejecución.
- *Claridad:* que el programa se pueda entender con facilidad, que contenga comentarios oportunos, indentación adecuada, empleo de identificadores significativos, etc.
- *Modularidad:* los módulos deben ser intercambiables y reutilizables. Se valorará la correcta estructuración del programa principal en procedimientos y funciones.

6.3. Normas, plazos, entrega y evaluación

Las normas relativas a estos aspectos se detallan en el documento anexo *"Práctica 2: Enunciado"*, que es importante consultar junto con el presente.