Facultad de Informática – Universidad Complutense

Fundamentos de la programación – Grupos

Curso 2015–2016

Examen final – 6 de septiembre de 2016

Tiempo disponible: 3 horas

Se pide construir un programa modular que permita gestionar pedidos de una tienda. La información se carga del archivo `tienda.txt`. El programa constará de cuatro módulos: *ItemPedido*, *Envio*, *ListaEnvios* y módulo principal (`main.cpp`).

Módulo *ItemPedido* (1 punto)

Declara un tipo de estructura `tItem` con tres campos, identificador del producto, nombre completo (dos cadenas de caracteres, el nombre posiblemente con espacios) y cantidad (un entero). Implementa, al menos, las siguientes funciones:

- ✓ `cargar()`: Carga un ítem de un flujo. En el archivo cada ítem consta de 3 líneas, una por cada campo (ver ejemplo de archivo al final del enunciado).
- ✓ `mostrar()`: Dado un ítem lo muestra en la salida estándar, como aparece en el ejemplo al final del enunciado.

Módulo *Envio* (3 puntos)

Declara un tipo de estructura `tEnvio` para listas de ítems de un pedido que guarda además el nombre del destinatario y la ciudad del envío (ambos cadenas de caracteres con posibles espacios). La lista no está ordenada y estará implementada con un **array dinámico**. Implementa, al menos, los siguientes subprogramas:

- ✓ `cargar()`: Carga toda la información de un envío de un flujo. En el archivo aparece primero el destinatario en una línea y en la siguiente línea la dirección (ciudad) del envío. En la siguiente línea aparece el número de ítems que aparecerán a continuación, cada uno en tres líneas, como se he indicado arriba (ver el ejemplo de archivo al final del enunciado).
- ✓ `mostrar()`: Dado un envío muestra por pantalla: la dirección de envío, el destinatario y la información de cada ítem del pedido, como aparece en el ejemplo de ejecución al final del enunciado.
- ✓ `destruir()`: Dado un envío, libera la memoria dinámica que utiliza.
- ✓ `cantidad()`: Dado un envío, el identificador de un ítem y una posición, devuelve la cantidad asociada al ítem si el ítem aparece en el envío a partir de la posición dada, y -1 en caso contrario. Debe hacerse de forma **recursiva** (0,5 puntos).

Módulo *ListaEnvios* (3,5 puntos) Máx. 50 envíos

Declara un tipo de estructura `tListaEnvios` para listas de envíos (hasta 50). Esta lista estará implementada con un **array estático de punteros a variables dinámicas** (es decir, un array de punteros a `tEnvio`). No tendrá ningún orden concreto, pudiéndose ordenar por diferentes criterios.

Implementa, al menos, los siguientes subprogramas:

- ✓ `cargar()`: Carga la lista de envíos de un archivo. El archivo comienza con el número de envíos que contiene (en una línea), y a continuación aparece la información de cada uno de ellos.
- ✓ `mostrar()`: Dada una lista de envíos, muestra la información de cada uno de los envíos, como aparece en el ejemplo al final del enunciado.
- ✓ `ordenarPorCliente()`: Dada una lista de envíos la ordena por destinatario, de menor a mayor.
- ✓ `ordenarPorCiudad()`: Dada una lista de envíos la ordena por dirección, de mayor a menor. Debe implementarse una **ordenación por inserción**.
- ✓ `destruir()`: Dada una lista de envíos, libera la memoria dinámica que utiliza.

Módulo principal (2,5 puntos)

El programa principal carga los envíos del archivo `tienda.txt` en una lista de envíos y la muestra ordenada descendentemente por ciudad, y dentro de cada ciudad, los envíos aparecen ordenados ascendentemente por cliente. A continuación solicita al usuario el identificador de un ítem (que se asume que no se repite), y muestra (en el orden en que se encuentren) la ciudad y cliente de los envíos en los que aparece ese ítem, y la cantidad total del ítem en la lista de envíos. Al salir se deberá liberar toda la memoria dinámica utilizada.

Se valorará la legibilidad, así como el uso adecuado de los esquemas de recorrido y búsqueda, de la comunicación entre subprogramas y de la memoria.

Recuerda: El comando para que se muestre la memoria no liberada es

```
_CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
```

Entrega el código del programa (solo los ficheros fuente: .cpp y .h). Añade al inicio del .cpp del módulo principal un comentario con tus datos (nombre completo, DNI, y nº de puesto).

Ejemplo de archivo `tienda.txt`:

```
3
Juan
Madrid
3
mouse
El mejor mouse
2
pc
El mejor pc
1
cable
Los mejores cables
4
Maria
Seattle
2
bici
Gran bicicleta
1
rueda
Ruedas estupendas
2
Luis
Madrid
2
cuchara
Cuchara para comer
6
tenedor
Tenedores para comer
6
```

Ejemplo de ejecución:

```
A Maria de Seattle:
    bici - Gran bicicleta (1)
    rueda - Ruedas estupendas (2)
A Juan de Madrid:
    mouse - El mejor mouse (2)
    pc - El mejor pc (1)
    cable - Los mejores cables (4)
A Luis de Madrid:
    cuchara - Cuchara para comer (6)
    tenedor - Tenedores para comer (6)

Por favor, escriba un identificador: pc
A Juan de Madrid:
    mouse - El mejor mouse (2)
    pc - El mejor pc (1)
    cable - Los mejores cables (4)
Ejecución correcta.
```

Archivo checkML.h

```
#ifdef _DEBUG
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
#ifndef DBG_NEW
#define DBG_NEW new ( _NORMAL_BLOCK , __FILE__ , __LINE__ )
#define new DBG_NEW
#endif
#endif
```

Puedes usar los archivos `tienda.txt` y `checkML.h` que están en el CV.