UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación

Algoritmos y Estructura de Datos I

Segundo cuatrimestre de 2015

30 de Septiembre de 2015

TPI - Fat Food

1. Introducción

En el TPE, hemos especificado el comportamiento de una serie de problemas relacionados con el local de Fat Food. Nuestro próximo paso será obtener una implementación en un lenguaje imperativo, para lo cual la especificación original ha sido *especialmente* adaptada.

El objetivo de este trabajo es programar, usando los comandos de C++ vistos en clase, pero con una especificación que tiene en cuenta el uso de clases y características propias del paradigma imperativo.

Deberán utilizar, además, el tipo abstracto `vector`, presente en la STL de C++. Publicaremos la interfaz del tipo. Para hacer uso de él, sólo deben incluir la librería `vector`. No hay un .cpp. NO deben mirar la implementación. Sólo deben manejarlo en base a su especificación.

También deberan utilizar fuertemente el tipo `pair` provisto por la `std`.

En la página de la materia estarán disponibles los archivos mencionados arriba, los *headers* de las clases que deberán implementar y la especificación de cada uno de sus métodos. **Importante: Utilizar la especificación diseñada para este TP, no la solución del TPE!**

Pueden agregar los métodos auxiliares que necesiten. Estos deben estar *en todos los casos* en la parte privada de la clase. No se permite modificar la parte pública de las clases ni agregar atributos adicionales, ya sean públicos o privados.

Para que la fiesta sea completa, tendrán disponible un sistema básico de menúes que les permitirá utilizar sus clases en forma interactiva. Este menú estará disponible en la página de la materia (interfaz.cpp e interfaz.h) junto al resto de los archivos. Si desean mejorarlo, no hay inconveniente que lo hagan pero no será evaluado. El archivo fatFood.cpp, que también proveemos, contiene la función main que llama a dicho menú.

2. Demostraciones

Además de realizar la implementación de todo el sistema, el payaso de amplia trayectoria quiere estar seguro que el software que están programando para él va a funcionar correctamente. No les va a pedir que demuestren todo lo implementado, pero le interesa saber que ustedes son capaces de hacer la demostración en caso de ser necesario. Para esto, les pide que escriban Pc, Qc, I, B, $cota$ y fv del ciclo principal de los siguientes problemas.

- candidatosAEmpleadosDelMesL
- elVagonetaL
- agregarComboAlPedidoL

Luego, no deben realizar la demostración entera de estos problemas, pero si deben demostrar los siguientes puntos del teorema del invariante para cada uno de los problemas mencionados:

- $Pc \wedge B \rightarrow I$
- $I \wedge \neg B \rightarrow Qc$
- $I \wedge cota < fv \rightarrow \neg B$

3. Implementación de tipos

Los tipos Combo y Pedido se mantienen, dentro de la clase, en atributos que son similares a sus observadores y por lo tanto es trivial comprender su equivalencia.

En el caso del tipo Local, esto ya no es tan claro. En el caso particular de este tipo, mantendremos en los atributos `_sandwiches` y `_bebidas` el stock actual de los sandwiches y bebidas, respectivamente (el primer valor corresponde a la hamburguesa o bebida, y el segundo a la cantidad que hay actualmente de ese producto). Dichas listas no tienen productos repetidos y sólo cuentan con productos del local. Además, todos los productos del local están en esas listas.

En el atributo `_empleados` mantendremos la lista de empleados y su energía. En el caso de que la energía sea un valor negativo, dicha persona será considerada un desempleado. Nuevamente, la lista no contiene personas repetidas y cabe aclarar que en esa lista están todos los empleados y desempleados del local.

Finalmente, en el atributo `_ventas` se mantiene la lista de los pedidos que corresponden a las ventas realizadas en el local.

4. Entrada/Salida

Todas las clases del proyecto tienen tres métodos relacionados con entrada salida:

mostrar : que se encarga de mostrar todo el contenido de la instancia de la clase en el flujo de salida indicado. El formato es a gusto del consumidor, pero esperamos que sea algo más o menos informativo y legible.

guardar : que se encarga de escribir la información de cada instancia en un formato predeterminado que debe ser decodificable por el método que se detalla a continuación (`cargar`).

cargar : que se encarga de leer e interpretar información generada por el método anterior, modificando el valor del parámetro implícito para que coincida con aquel que generó la información.

En definitiva, `guardar` se usará para "grabar" en un archivo de texto el contenido de una instancia mientras que "cargar" se usará para "recuperar" dicha información. En todos los casos, sus interfaces serán:

- `void mostrar(std::ostream& ) const;`
- `void guardar(std::ostream& ) const;`
- `void cargar(std::istream& );`

El detalle del formato que se debe usar para "guardar" y "leer" en cada clase se indica a continuación. Tener en cuenta que los números se deben guardar como texto. Es decir, si escriben a un archivo y después lo miran con un editor de texto, donde escribieron un número 65 deben ver escrito 65 y no una letra A. Cada vez que aparezca un string pueden suponer que los nombres no contendrán espacios.

Combo: Se debe guardar entre llaves primero una C, indicando que es un combo, luego su bebida, su sandwich y su dificultad. Cada parte debe estar separada por un espacio.

```
{ C FalsaNaranja CukiQueFresco 2 }
```

Pedido: Se debe guardar entre llaves primero una P, indicando que es un pedido, luego su número, el nombre de la persona que lo atendió y su lista de combos. Cada parte debe estar separada por un espacio. La lista debe estar delimitada por corchetes y sus componentes deben estar separadas por un espacio.

```
{ P 123 Ronald [ { C FalsaNaranja CukiQueFresco 2 } { C SeVeNada McPato 3 } ] }
```

Local: Se debe guardar entre llaves primero una L, indicando que es un local, luego la lista de stock de bebidas, la lista de stock de sandwichs, la lista de empleados con sus energias y la lista de ventas. Cada parte debe estar separada por un espacio. Las listas deben estar delimitadas por corchetes y sus componentes deben estar separadas por un espacio. Las tuplas deben estar delimitadas por parentesis y sus componentes separadas por espacios.

```
{ L
[ ( PestiCola 34 ) ( FalsaNaranja 22 ) ( SeVeNada 12 ) ]
[ ( McGyver 5 ) ( CukiQueFresco 7 ) ( McPato 6 ) ( BigMacabra 10 ) ]
[ ( Ronald_Red 67 ) ( Plin_Plin -23 ) ( Pinon_Fijo  75 ) ]
[ { P 123 Ronald_Red [ { C PestiCola McGyver 1 } ] }
  { P 124 Pinon_Fijo [ { C FalsaNaranja CukiQueFresco 2 } { C SeVeNada McPato 3 } ] } ]
}
```

Importante 1: Los saltos de línea han sido dejado por cuestiones de visualización. En el archivo real no deben incorporarse.