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

Algoritmos y Estructura de Datos I

Segundo cuatrimestre de 2013

28 de Octubre 2013

TPI - Cine v1.0

1. Introducción

En el TPE, hemos especificado el comportamiento de una serie de problemas relacionados con la administración de Cines de El Pueblo. Después hicimos una primera implementación del trabajo en programación funcional en el TPF. 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, tipos y operaciones análogos a los de los TPs anteriores, pero con una especificación que tiene en cuenta el uso de clases y características propias del paradigma imperativo.

Deberán utilizar, además, el tipo abstracto `Lista`, provisto por la cátedra. Publicaremos la interfaz del tipo y su implementación. Para hacer uso de él, sólo deben incluir el archivo .h correspondiente. 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`. Pueden encontrar una buena referencia para su uso en `http://www.cplusplus.com/reference/std/utility/pair/`

No está permitido utilizar el comando `for`.

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. No se permiten archivos "extra" con funciones auxiliares.

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.

2. Demostraciones

Deben implementar la función `cambiarSala` dentro del archivo mainCine.cpp y demostrar su terminación y correctitud. De utilizar funciones auxiliares para su implementación, si las mismas **NO** se encuentran como problemas en la solución de la cátedra, deberán especificarlas y demostrar terminación y correctitud de ellas también.

`problema cambiarSala` (ts:[Ticket], vieja: Sala, nueva: Sala) = `result` : [Ticket] {
  `asegura mismaCantidad` : $|ts| == |result|$;
  `asegura noCambianLosDeSalaNoVieja` : $(\forall i \leftarrow [0..|ts|), sala(ts[i])! = vieja) ts[i] == result[i]$;
  `asegura losDeSalaViejaCambianSoloLaSala` : $(\forall \leftarrow [0..|ts|), sala(ts[i]) == vieja) pelicula(result[i]) == pelicula(ts[i]) \wedge$
    $usado(ts[i]) == usado(result[i]) \wedge sala(result[i]) == nueva$;
}

3. Implementación de tipos

El tipo `Pelicula` mantiene en los atributos `nombre_` y `es3D_` a los observadores del tipo que se llaman de manera similar. También contiene dos listas con los géneros y de actores. El atributo `generos_` guarda la lista de generos sin repetidos y de manera ordenada. El atributo `actores_` guarda la lista de actores sin repetidos y de manera ordenada. Las listas siempre deberán mantenerse ordenadas.

El tipo `Ticket` mantiene los atributos `pelicula_`, `sala_` y `usado_` correspondientes a los observadores del tipo que se llaman de manera similar.

El tipo `Cine` mantiene los atributos `nombre_, ticketsVendidos_` que se corresponden con los observadores del mismo nombre. El atributo `salasSinUsar_` es una lista con todas las salas sin usar. El atributo `peliculas_` es una lista de tuplas que tiene la pelicula y la sala asociada. Además, el atributo `espectadores_` es una lista de tuplas que tiene para cada sala la cantidad de espectadores de la sala. Las salas que tengan cero espectadores son parte de esta lista. Las peliculas y salas se guardan sin repetidos. Para todos los tipos, en todo momento se deben satisfacer sus invariantes.

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, éste deberá ir guardado entre `|`. Pueden suponer que los nombres no contendrán el caracter `|`.

Película: Se debe guardar

```
P |Nombre| [|Genero1|, ..., |GeneroN|] [|Actor1|, ..., |ActorN|] Es3D
```

Donde Nombre es el nombre de la película, las listas de Generos y Actores empiezan con corchete y cada elemento se representa entre |, Es3D es el caracter V cuando la película es 3D y F cuando no lo es. Por ejemplo, un archivo de película válido sería :

```
P |Los amantes pasajeros| [|Comedia|] [|Penelope Cruz|, |Antonio Banderas|, |Paz Vega|] F
```

Que rerpesenta, la película "Los amantes pasajeros", perteneciente al género de comedia, cuyo elenco está conformado por: Penélope Cruz, Antonio Banderas, y Paz Vega, y no es 3D.

Ticket: Se debe guardar

```
T (Pelicula) Sala Usado
```

Donde Pelicula es una película guardada en el formato de película, Sala es el número y Usado es el caracter V cuando el ticket fue usado y F cuando no. Por ejemplo un ticket para ver los amante pasajero en la sala 1 usado se puede ver así:

```
T
(P |Los amantes pasajeros| [|Comedia|] [|Penelope Cruz|, |Antonio Banderas|, |Paz Vega|] F)
1 V
```

Cine: Se debe guardar

```
C |Nombre| [Sala1, ..., SalaN] [(Sala1,E1), ..., (SalaM,EM)] [(Sala1,T1), ..., (SalaK,TK)]
[(Sala1, (Pelicula1)), ..., (SalaL, (PeliculaL))]
```

Donde Nombre es el nombre del cine, luego sigue una lista entre corchetes con la lista de salas sin usar. Otra lista contiene la cantidad de espectadores por sala en la que cada elemento es una dupla con la sala y la cantidad de espectadores. Luego hay una lista con la cantidad de tickets vendidos sin usar por sala, con el mismo formato que la anterior. Finalmente hay una lista con las peliculas y las salas donde se proyectan. Por ejemplo en el siguiente cine:

```
C |Belgrano Multiplo| [5, 6] [(1,10), (5,0), (6,0)] [(1,10)]
[(1, (P |Los amantes pasajeros| [|Comedia|] [|Penelope Cruz|, |Antonio Banderas|, |Paz Vega|] F))]
```

se proyecta una pelicula en la sala 1, tiene 10 tickets vendidos sin usar para la sala 1, dos salas sin usar, la sala 1 tiene diez espectadores y las demás cero espectadores.

Aclaración 1: en todos los tipos que involucran listas, no está permitido usar la función "mostrar" de Lista para guardar. Sí está permitido usarlo para mostrar.
Las listas deben ir encerradas entre corchetes y cada elemento de la misma debe ir guardado y separado por comas.
Los tipos compuestos deben ir encerrados entre paréntesis.

Aclaración 2: Los saltos de línea están puestos solo por razones de legilibilidad en el enunciado. En la implementación debería ser todo una única linea con los saltos de líneas reemplazados por espacios.