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

Algoritmos y Estructura de Datos I

Primer cuatrimestre de 2012

29 de Octubre 2012

TPI - Hoteles y Pasajeros v1.0

1. Introducción

En el TPE, hemos especificado el comportamiento de una serie de problemas relacionados con Hoteles. 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. El archivo mainHoteles.cpp, que también proveemos, contiene la función main que llama a dicho menú.

2. Demostraciones

Deben implementar la función `huespedesConPalabra` dentro del archivo mainHoteles.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 huespedesConPalabra` (this: Hotel) = `result` : [DNI] {
  `asegura` $mismos(result, sacarRepetidos(buscarHuespedesDePalabra(this)))$;

  `aux buscarHuespedesDePalabra` (h: Hotel) : [DNI] = $[\, dniCheckOut(o) \mid i \leftarrow ingresos(h),$
    $o \leftarrow salidasDe(h, dniCheckIn(prm(i))), noHaySalidaEnElMedio(prm(i), o, h),$
    $existeReserva(h, dniCheckIn(prm(i)), fechaCheckIn(prm(i)), fechaCheckOut(o), tipo(sgd(i)), true)\,]$;
  `aux existeReserva` (h: Hotel, d: DNI, fd: Fecha, fh: Fecha, t: TipoHabitacion, c: Bool) : Bool =
    $(\exists r \leftarrow reservas(h))documento(r) == d \wedge fechaDesde(r) == fd \wedge tipo(r) == t \wedge fechaHasta(r) == fh \wedge confirmada(r) == c$;
}

Observación: De ser necesario, tomar la especificación de las funciones auxiliares del documento provisto por la materia.

3. Implementación de tipos

El tipo `Reserva` mantiene en los atributos `_documento`, `_fechaDesde`, `_fechaHasta`, `_tipo` y `_confirmada` los observadores del tipo que se llaman de manera similar. Siempre debe cumplir que la fechaHasta es posterior a la fechaDesde.

El tipo `Habitacion` posee los atributos `_numero`, `_tipo` y `_accesorios` que mantienen la información correspondiente a los observadores de mismo nombre. En el atributo `_accesorios`, se guardan los accesorios de manera ordenada. Dicha lista **DEBE MANTENERSE SIEMPRE ORDENADA** utilizando como criterio la comparación por defecto de estos tipos y NO puede tener accesorios repetidos.

El tipo `Hotel` almacena sus observadores en los atributos `_nombre`, `_cadena`, `_habitaciones`, `_tarifaXDia`, `_precioAccesorio`, `_ingresos`, `_salidas` y `_reservas` que corresponden a los observadores con nombre similar. En todo momento el hotel debe satisfacer todos los invariantes del tipo.

El tipo `Ministerio` almacena sus observadores en los atributos `_provincias` y `_registro`. El `_registro` es una lista de tuplas de provincia con un hotel. Cada elemento (P, H) representa la registración del hotel H en la provincia P.

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 "cargar" 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 `|`.

Reserva: Se debe guardar el `ENCABEZADO_ARCHIVO`, el documento del pasajero, la fecha de ingreso, la fecha de egreso y entre barras (barrita vertical `"|"`) el tipo de habitación y si esta confirmada. Por ejemplo, la reserva sin confirmar del huesped con DNI 31459265, que entra el día 3 y se va el día 5 de una habitación simple, debería guardarse de la siguiente manera:

```
R 31459265 3 5 |Simple| |False|
```

Habitación: Se debe guardar el `ENCABEZADO_ARCHIVO`, el número de habitación, entre barras el tipo, y la lista de los accesorios que pertencencen a la habitación. Por ejemplo la habitación número 50 de tipo cuadruple con Jacuzzi, DVD y PS3:

```
A 50 |Cuadruple| [(|Jacuzzi|), (|PS3|), (|DVD|)]
```

Aclaración 1: en este tipo, y en todos los que involucran listas, no está permitido usar 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 entre paréntesis y separado por comas.
En el caso de las listas donde los elementos sean tuplas, no es necesario agregar paréntesis adicionales.
En el caso de las listas donde los elementos sean números enteros, no es necesario agregar paréntesis.
Aclaración 2: Recordar que en el tipo Habitación es importante respetar el orden de los accesorios.

Hotel: Se debe guardar el `ENCABEZADO_ARCHIVO`, entre barras el nombre del hotel, la cadena, seguido de la listas de habitaciones, ingresos, salidas, reservas, tarifas por dia y precios de accesorios. Por ejemplo:

```
H |Royal Efecen| |Efecen| [(A 21 |Simple| [(|LCD|)]), (A 22 |Simple| [(|Pelotero|), (|Inflable|)])]
  [((1223, 3),21), ((1224, 3), 22)] [(1223, 5), (1224, 7)]
  [(R 1223 3 10 |Simple| |True|), (R 1224 3 11 |Simple| |True|)]
  [(|Triple|,31),(|Simple|,14)] [(|LCD|,15),(|Pelotero|,92),(|Inflable|,65)]

H |Efecen Inn| |Efecen| [(A 3 |Simple| [(|LCD|)])] [] [] [(R 3 14 15 |Simple| |False|)]
  [(|Simple|,92)] [(|LCD|,65)]
```

Aclaración 3: 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. Por ejemplo:

Ministerio: Se debe guardar el `ENCABEZADO_ARCHIVO`, y una lista que contenga el registro, donde cada elemento de la lista es una dupla de provincia y una lista de hoteles. Por ejemplo:

```
M [(|Buenos Aires|,
   [(H |Royal Efecen| |Efecen|
     [(A 21 |Simple| [(|LCD|)]), (A 22 |Simple| [(|Pelotero|), (|Inflable|)])]
     [((1223, 3),21), ((1224, 3), 22)]
     [(1223, 5), (1224, 7)] [(R 1223 3 10 |Simple| |True|), (R 1224 3 11 |Simple| |True|)])]),
  (|Catamarca|,
   [(H |Efecen Inn| |Efecen|
   [(A 21 |Simple| [(|LCD|)]), (A 22 |Doble| [])] [] []
   [(R 1224 3 11 |Simple| |False|)])]
  )]
```