La hora de la pastilla

Estructuras de Datos
Facultad de Informática - UCM

Por culpa de una infección bacteriana el médico me ha recetado antibióticos. Por suerte, no voy a estar mucho tiempo así, ya que el tratamiento dura menos de 24 horas. No obstante, el médico me ha dicho que sea muy cuidadoso con el tiempo entre pastilla y pastilla, ya que todas las tomas deben estar igualmente espaciadas. Es decir, el tiempo que pasa entre una pastilla y la siguiente ha de ser siempre el mismo.

Supongamos que tenemos una clase llamada `Hora` que representa una determinada hora del día con una precisión a nivel de segundos.

```
class Hora {
public:
  Hora(int horas, int minutos, int segundos);
  // ...
private:
  // ...
};
```

Quiero implementar una función que me diga la hora a la que me toca tomar una determinada pastilla:

```
Hora hora_pastilla(const Hora &primera, const Hora &ultima,
                   int num_pastillas, int i)
```

donde `primera` y `ultima` son las horas a las que tengo que tomar la primera y última pastilla del tratamiento, respectivamente. El parámetro `num_pastillas` indica el número total de pastillas de las que costa el tratamiento, mientras que `i` es un entero entre 1 y `num_pastillas`. La función debe devolver la hora en la que tengo que tomar la `i`-ésima pastilla del tratamiento.

Ten en cuenta que `primera` puede ser mayor que `última`. Esto pasa cuando el tratamiento comienza en la hora `primera` y se extiende hasta la hora `última` del día siguiente.

Por ejemplo, supongamos que `primera = 00:00:00`, `ultima = 01:00:00` y `num_pastillas = 5`. Entonces:

- Si el parámetro `i` vale `1`, la función debe devolver la hora `00:00:00`.
- Si el parámetro `i` vale `2`, la función debe devolver la hora `00:15:00`.

- Si el parámetro `i` vale `3`, la función debe devolver la hora `00:30:00`.
- Si el parámetro `i` vale `4`, la función debe devolver la hora `00:45:00`.
- Si el parámetro `i` vale `5`, la función debe devolver la hora `01:00:00`.

Otro ejemplo: si `primera = 23:00:00`, `ultima = 02:00:30`, y `num_pastillas = 3`, entonces:

- Si el parámetro `i` vale `1`, la función debe devolver `23:00:00`.
- Si el parámetro `i` vale `2`, la función debe devolver `00:30:15`.
- Si el parámetro `i` vale `3`, la función debe devolver `02:00:30`.

1. Identifica las operaciones elementales que necesitaría el TAD `Hora` para poder implementar la función `hora_pastilla`. Puedes suponer que ya existe un TAD llamado `Duracion` que representa el lapso de tiempo (en segundos) que transcurre entre dos horas determinadas.

   ```
   using Duracion = int;
   ```

2. A continuación se muestran dos posibles representaciones del TAD `Hora`:

   ```
   // Representacion A
   class Hora {
   public:
     // ...
   private:
     int horas;
     int minutos;
     int segundos;
   };
   ```

   ```
   // Representacion B
   class Hora {
   public:
     // ...
   private:
     // Numero de segundos
      transcurridos
     // desde la hora 00:00:00
     int num_segundos;
   };
   ```

   Selecciona la representación que consideres más adecuada y completa la definición de la clase `Hora`, teniendo en cuenta las operaciones del apartado anterior.

3. Indica el coste en tiempo de cada una de las operaciones implementadas en la clase `Hora`, incluyendo su constructor.

4. Implementa la función `hora_pastilla` e indica su coste en tiempo.

5. (Opcional) Supongamos que el modelo del TAD `Hora` es una tupla $(h, m, s) \in \mathbb{N}^3$ cuyas componentes indican, respectivamente, las horas, minutos y segundos de un determinado instante en el tiempo. Para cada una de las representaciones A y B mostradas en el apartado 2, determina:

a) El invariante de representación, que indica qué instancias de la clase `Hora` son válidas.

b) La función de abstracción, que devuelve, para cada instancia `x` de la clase `Hora`, el elemento de $\mathbb{N}^3$ que representa.

Solución

1. Tenemos que calcular los segundos que transcurren entre `primera` y `ultima`, dividirlos entre el número de pastillas menos uno, obteniendo así el intervalo en segundos entre pastillas. Tenemos que multplicar ese intervalo por $(n - 1)$ para obtener el tiempo que transcurre entre la pastilla inicial y la pastilla `n`-ésima, para así sumárselo a la hora de la pastilla inicial. Por tanto, necesito dos operaciones:

   - Devolver la diferencia entre dos horas, como una `Duracion` expresada en segundos.
   - Obtener la hora que se obtendría al sumar una determinada cantidad de segundos (`Duracion`) a otra dada.

   Por tanto, esta es la interfaz de nuestra clase:

   ```
   class Hora {
     public:
       Hora(int h, int m, int s);  // constructor
       Duracion diferencia_hasta(const Hora &h) const;
       Hora suma(Duracion d) const;
     private:
       // ...
   }
   ```

2. Escogemos la segunda representación, ya que las operaciones del apartado anterior se traducen, simplemente, a sumas y restas sobre números enteros.

   ```
   class Hora {
   public:
     Hora(int horas, int minutos, int segundos)
       : num_segundos(horas * 3600 + minutos * 60 + segundos) {
       // Comprobamos precondiciones (aunque no se pide en el ejercicio)
       assert (0 <= horas && horas < 24);
       assert (0 <= minutos && minutos < 60);
       assert (0 <= segundos && segundos < 60);
     }

     Duracion diferencia_hasta(const Hora &h) const {
       if (this->num_segundos < h.num_segundos) {
         // Si la hora 'this' ocurre antes que 'h' dentro del mismo dia,
         // solo hay que obtener la diferencia en segundos de cada hora
         return h.num_segundos - this->num_segundos;
   ```

```
    } else {
      // En caso contrario, consideramos que 'h' es una hora del dia
      // siguiente. Por tanto, calculamos la diferencia que hay entre
      // el final del dia (hora 24:00:00) y this, y le sumamos la
      // cantidad de segundos de 'h'
      return 24 * 3600 - this->num_segundos + h.num_segundos;
    }
  }

  Hora suma(Duracion d) const {
    Hora result(0, 0, 0);
    result.num_segundos = (this->num_segundos + d) % (24 * 3600);
    return result;
  }

private:
  int num_segundos;
};
```

3. Todas las operaciones tienen coste constante, ya que solo realizan operaciones aritméticas elementales.

4. ```
   Hora hora_pastilla(const Hora &primera, const Hora &ultima, int
       num_pastillas, int i) {
     Duracion num_segundos_dif = primera.diferencia_hasta(ultima);
     Duracion segundos_entre_pastillas = num_segundos_dif / (num_pastillas -
       1);
     Duracion segundos_desde_primera = segundos_entre_pastillas * (i - 1);

     return primera.suma(segundos_desde_primera);
   }
   ```

   También tiene coste en tiempo constante ($\mathcal{O}(1)$), ya que solamente hace operaciones aritméticas elementales y llamadas a métodos `diferencia_hasta` y `suma`, los cuales también tienen coste en tiempo constante.

5. Para la representación A tenemos la siguiente función de abstracción f e invariante I. Suponemos que `x` es una instancia de la clase `Hora`.

$$f(\mathtt{x}) = (\mathtt{x.horas}, \mathtt{x.minutos}, \mathtt{x.segundos})$$

$$I(\mathtt{x}) = 0 \leq \mathtt{x.horas} < 24 \land 0 \leq \mathtt{x.minutos} < 60 \land 0 \leq \mathtt{x.segundos} < 60$$

   Para la representación B tenemos lo siguiente:

$$f(\mathtt{x}) = (\mathtt{x.num\_segundos}/3600, \mathtt{mod}(\mathtt{x.num\_segundos}, 3600)/60, \mathtt{mod}(\mathtt{x.num\_segundos}, 60))$$
   ```

$$I(x) = 0 \leq x < 86400$$

donde $86400 = 24 * 60 * 60$ es el número de segundos que tiene un día.