UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

Práctica 2: Abstracción y documentación

Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Estructuras de Datos

Grado en Ingeniería Informática
Doble Grado en Ingeniería Informática y Matemáticas
Doble Grado en Ingeniería Informática y ADE

1. Introducción

Los objetivos de este guión de prácticas son los siguientes:

1. Asimilar los conceptos fundamentales de abstracción, aplicado al desarrollo de programas.
2. Documentar un Tipo de Dato Abstracto (TDA)
3. Practicar con el uso de la herramienta doxygen para generar documentación.
4. Profundizar en los conceptos relacionados con la especificación de un TDA, representación de un TDA, función de abstracción e invariante de la representación.

Los requisitos para poder realizar esta práctica son:

1. Haber estudiado el Tema 1: Introducción a la eficiencia de los algoritmos.
2. Haber estudiado el Tema 2: Abstracción de datos.

2. Tipos de datos abstractos

Los tipos de datos abstractos son nuevos tipos de datos con un grupo de operaciones que proporcionan la única manera de manejarlos. De esta forma, debemos conocer las operaciones que se pueden usar, pero no necesitamos saber la forma en que se almacenan los datos ni cómo se implementan las operaciones.

2.1 Selección de operaciones

Una tarea fundamental en el desarrollo de un TDA es la selección del conjunto de operaciones que se usarán para manejar el nuevo tipo de dato. Para ello, el diseñador deberá considerar los problemas que quiere resolver en base a este tipo y ofrecer el conjunto de operaciones que considere más adecuado. Las operaciones seleccionadas deben atender a las siguiente exigencias:

1. Debe existir un conjunto mínimo de operaciones para garantizar la abstracción. Este conjunto mínimo permite resolver cualquier problema en el que se necesite el TDA.
2. Las operaciones deben ser usadas con bastante frecuencia.
3. Debemos considerar que el tipo de dato sufra en el futuro modificaciones y conlleve también la modificación de las operaciones. Por lo tanto, un número muy alto de operaciones puede conllevar un gran esfuerzo en la modificación.

Por otro lado las operaciones seleccionadas pueden clasificarse en dos conjuntos:

1. Fundamentales. Son aquellas necesarias para garantizar la abstracción. No es posible prescindir de ellas ya que habría problemas que no se podrían resolver sin acceder a la parte interna del tipo de dato. A estas funciones también se le denominan operaciones **primitivas.**
2. No fundamentales. Corresponden a las operaciones prescindibles ya que el usuario podría construirlas en base al resto de operaciones.

3. Documentación

El objetivo fundamental de un programador es que los TDA que programe sean reutilizados en el futuro por él u otros programadores. Para que esta tarea pueda llevarse a cabo, los módulos donde se materializa un TDA deben de estar bien documentados. Para llevar a cabo una buena documentación de TDA se deben crear dos documentos bien diferenciados:

1. **Especificación**. Es el documento donde se presentan las características sintácticas y semánticas que describen la parte pública (parte del TDA visible a otros módulos). Con este documento cualquier otro módulo podría usar el módulo desarrollado y, además, es totalmente independiente de los detalles internos de construcción del mismo.
2. **Implementación.** Corresponden al documento que presenta las características internas del módulo. Para facilitar futuras mejoras o modificaciones de los detalles internos del módulo es necesario que la parte de implementación sea bien documentada.

3.1 Especificación del TDA

En este caso vamos a especificar un tipo de dato junto con un conjunto de operaciones. Por lo tanto en la especificación de TDA aparecerán dos partes:

1. **Definición.** En esta parte se define el nuevo tipo de dato abstracto, así como todos los términos relacionados que sean necesarios para comprender el resto de la especificación.
2. **Operaciones**. En esta parte se especifican las operaciones, tanto sintáctica como semánticamente.

3.1.1 Definición

Se dará una definición del TDA en lenguaje natural. Para ello el TDA se distinguirá como una nueva clase de objetos en la que cualquier instancia de esta nueva clase tomará valores en un conjunto establecido (dominio). Por ejemplo, si queremos definir un *Racional* diremos:

> *Una instancia f del tipo de dato abstracto Racional es un objeto del conjunto de los números racionales, compuesto por dos valores enteros que representan, respectivamente, numerador y denominador. Lo representamos como num/den.*

3.1.2 Operaciones

En esta parte se realiza una especificación de las operaciones que se usarán sobre el tipo de dato abstracto que se está construyendo. Cada una de estas operaciones representará una función y, por lo tanto, se hará la especificación con los siguientes ítems:

1. Breve descripción: Qué es lo que hace la función. Usando doxygen la sentencia es `@brief`.
2. Se especifican cada uno de los parámetros de la función. Por cada parámetro se especificará si el parámetro se modifica o no tras la ejecución de la función. Usando doxygen el comando es `@param`.
3. Las condiciones previas a la ejecución de la función (precondiciones) que deben cumplirse para un buen funcionamiento de la función. Usando doxygen la sentencia es `@pre`.
4. Qué devuelve la función. En doxygen usaremos el comando `@return`.

5. Las condiciones que deben cumplirse tras la ejecución de la función (postcondiciones). Usando doxygen el comando es `@post`.

Supongamos que queremos especificar en nuestra clase *Racional* la función *comparar* que compara un *Racional* con el *Racional* que apunta `this`. Una posible especificación de esta función sería la siguiente:

```
/**
 * @brief Compara dos racionales
 * @param r racional a comparar
 * @return Devuelve 0 si este objeto es igual a r,
 *              <0 si este objeto es menor que r,
 *              >0 si este objeto es mayor que r
 */
 bool comparar(Racional r);
```

Un ejemplo donde se usa una precondición es en la función *asignar,* que se le asigna al *Racional* apuntado por `this` unos valores concretos para el numerador y denominador. En esta especificación cabe resaltar que si el nuevo denominador es 0 se estarán violando las propiedades que deben mantenerse para una correcta instanciación de un objeto de tipo *Racional*.

```
/**
 * @brief Asignación de un racional
 * @param n numerador del racional a asignar
 * @param d denominador del racional a asignar
 * @return Asigna al objeto implícito el numero racional n/d
 * @pre d debe ser distinto de cero
 */
 void asignar(int n, int d);
```

3.2 Implementación del TDA

Para implementar un TDA es necesario, en primer lugar, escoger una representación interna adecuada, es decir, una forma de estructurar la información de manera que podamos representar todos los objetos de nuestro TDA de una manera eficaz. Por lo tanto, debemos seleccionar una estructura de datos adecuada para la implementación, es decir, un tipo de dato que corresponda a esta representación interna y sobre el que implementamos las operaciones. A este tipo escogido (la estructura de datos seleccionada) se le denomina ***tipo rep*.**

Para nuestro TDA *Racional* las estructuras de datos posibles para representar el ***tipo rep*** podrían ser por ejemplo:

- Un vector de dos posiciones para almacenar el numerado y denominador:

```
Class Racional{
  private:
  int r[2];
  ….
}
```

- Dos enteros que representen el numerador y denominador respectivamente:

```
Class Racional{
  private:
  int numerador;
  int denominador;
  ….
}
```

De entre las representaciones posibles, en el documento debe aparece la estructura de datos escogida para el ***tipo rep***. Esta elección debe formalizarse mediante la especificación de la función de abstracción. Esta función relaciona los objetos que se pueden representar con el ***tipo rep*** y los objetos del TDA. Las propiedades de esta función son:

- Parcial, todos los valores de los objetos del ***tipo rep*** no se corresponden con un objeto del tipo abstracto. Por ejemplo, valores de numerador=1 y denominador =0 no son valores válidos para un objeto del tipo *Racional*.
- Todos los elementos del tipo abstracto tienen que tener una representación.
- Varios valores de la representación podrían representar a un mismo valor abstracto. Por ejemplo {4,8} y {1,2} representan al mismo racional.

Por lo tanto, en la documentación podemos incluir esta función para indicar el significado de la representación. Consta de dos partes:

- Indicar exactamente cuál es el conjunto de valores de representación que son válidos, es decir, que representen a un tipo abstracto. Por tanto, será necesario establecer una condición sobre el conjunto de valores del ***tipo rep*** que nos indique si corresponden a un objeto válido. Esta condición se denomina invariante de la representación.

$$f_{inv}: rep \dashrightarrow booleanos$$

- Para cada representación válida, indicar cómo se obtiene el tipo abstracto correspondiente, es decir la función de abstracción.

$$f_{abs}: rep \dashrightarrow A$$

Un invariante de la representación es “invariante” porque siempre es cierto para la representación de cualquier objeto abstracto. Por tanto, cuando se llama a una función del tipo de dato se garantiza que la representación cumple dicha condición y cuando se devuelve el control de la llamada debemos asegurarnos que se sigue cumpliendo. En nuestro ejemplo del TDA *Racional*, en la documentación de la implementación incluiríamos lo siguiente:

```
class Racional {

 private:
/**
  * @page repConjunto Rep del TDA Racional
  *
  * @section invConjunto Invariante de la representación
  *
  * El invariante es \e rep.den!=0
  *
```

```
 * @section faConjunto Función de abstracción
 *
 * Un objeto válido @e rep del TDA Racional representa al valor
 *
 * (rep.num,rep.den)
 *
 */

 int num; /**< numerador */
 int den; /**< denominador */

public:
```

4. Ejercicio

Completar el TDA 3 en raya, cuyo código se proporciona a través de la plataforma de docencia virtual (carpeta llamada *3raya*):

1. Dar la especificación completa, estableciendo una definición y el conjunto de operaciones básicas.
2. Analizar y escoger la estructura de datos apropiada para ***tipo rep.***
3. Para la estructura de datos del ***tipo rep*** establecer cuál es el invariante de la representación y función de abstracción.
4. Fijado el ***tipo rep*** realizar la implementación de todas las operaciones (ya suministradas).
5. Probar el tipo de dato abstracto racional desarrollado.

5. Referencias

- [GAR06b] Garrido, A. Fdez-Valdivia, J. "*Abstracción y estructuras de datos en C++*". Delta publicaciones, 2006.