



Práctica 2: Documentación del Software

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

Índice de contenido

1.Introducción.....	3
2.Tipos de datos abstractos.....	3
2.1.Selección de operaciones.....	3
3.Documentación.....	3
3.1.Especificación del T.D.A.....	4
3.1.1.Definición.....	4
3.1.2.Operaciones.....	4
3.2.Implementación del T.D.A.....	5
4.Ejercicio	7
4.1.Fichero prueba.....	8
4.2.Módulos a desarrollar.....	8
5.Práctica a entregar.....	8
6.Referencias	9

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 (T.D.A)
3. Practicar con el uso de doxygen.
4. Profundizar en los conceptos relacionados especificación del T.D.A, representación del T.D.A., 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 como se almacena los datos ni
- cómo se implementan las operaciones.

2.1. Selección de operaciones

Una tarea fundamental en el desarrollo de un T.D.A. 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 T.D.A.
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 numero 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 T.D.A. que programe sean reutilizados en el futuro por él u otros programadores. Para que esta tarea puede llevarse a cabo los módulos donde se materializa un T.D.A. deben de estar bien documentados. Para llevar a cabo una buena documentación de T.D.A. 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 (la parte del T.D.A. visible a otros módulos). Con este documento cualquier otro módulo podría usar el módulo desarrollado y además es totalmente independiente de los detalle 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 documentada.

3.1. Especificación del T.D.A

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

1. **Definición.** En esta parte deberemos definir 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 T.D.A en lenguaje natural. Para ello el T.D.A se distinguirá como una nueva clase de objetos en la que cualquier instancia de esta nueva clase tomará valores (dominio) en un conjunto establecido. 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 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 representarán una función y por lo tanto se hará la especificación con los siguientes items:

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

```

/**
 * @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 precondition 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 estará 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 T.D.A.

Para implementar un T.D.A., es necesario en primer lugar, escoger una representación interna adecuada, una forma de estructurar la información de manera que podamos representar todos los objetos de nuestro tipo de dato abstracto 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 éste tipo escogido (la estructura de datos seleccionada), se le denomina **tipo rep.**

Para nuestro T.D.A *Racional* las estructuras de datos posibles para representar nuestro **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 aparecer 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 tipo de dato abstracto. 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 aplicación para indicar el significado de la representación. Esta aplicación tiene dos partes:

1. Indicar exactamente cual 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.

$$f_{inv}: rep \rightarrow \text{booleanos}$$

2. Indicar para cada representación válida cómo se obtiene el tipo abstracto correspondiente, es decir la función de abstracción.

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

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 el T.D.A 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

En una academia de lenguajes queremos obtener una aplicación que nos permita traducir una palabra de un determinado idioma a otra u otras. Es decir una palabra de un idioma puede tener varias traducciones. Con tal fin vamos a desarrollar varios tipos de datos abstractos:

- Palabra: se compone de una palabra en el idioma origen y se transforma en una o más palabras en el idioma destino. Las palabras pueden contener más de un vocablo. P.e abandonar se traduce en inglés como “give up”. Así tanto en el origen como en el destino puede tener más de un vocablo. Los vocablos no contendrán acentos, tildes ni el carácter ñ. Los vocablos si existe más de uno se separa por un espacio.
- Traductor: Es un conjunto de palabras ordenadas por la palabra en el idioma origen.

Se pide desarrollar el T.D.A. Palabra y el T.D.A. Traductor. Para cada uno de estos tipo de datos abstractos:

1. Dar la especificación. Establecer una definición y el conjunto de operaciones básicas.
2. Determinar diferentes estructuras de datos para **tipo rep**.
3. Escoger una de las estructuras de datos para representar el **tipo rep**
4. Para la estructura de datos del **tipo rep** establecer cual es el invariante de la

representación y función de abstracción.

5. Fijado el **tipo rep** realizar la implementación de las operaciones.
6. Haciendo uso de `pruebatraductor.cpp` probar los tipos de datos abstractos desarrollados.

4.1. Fichero prueba

Para poder probar nuestro programa usaremos un fichero compuesto de una serie de líneas. Cada línea se corresponde con una palabra en el idioma origen y a continuación, separados por el carácter ';', las palabras en el idioma destino. Por ejemplo un trozo del fichero inglés-español es el que se muestra a continuación:

```
.....  
warm;caliente;  
warn;avisar;  
warning;aviso;  
warrant;garantia;orden;orden por escrito;  
wash;lavar;  
wash up;fregar;lavar;  
wasp;avispa;  
waste;desechos;detrito;acabar;  
watch;reloj de pulsera;reloj;contemplar;mirar;observar;ver;  
.....
```

En el directorio **doc** tenéis ejemplos de ficheros traductores: `english_spanish` (inglés a español), `spanish_english`, `french_english`, `english_french`.

4.2. Módulos a desarrollar.

Habrà al menos dos módulos que deberéis desarrollar: 1) el módulo asociado a Palabra (`palabra.cpp` y `palabra.h`) y 2) el módulo asociado a Traductor (`traductor.cpp` e `traductor.h`). Es posible añadir más módulos si lo estimáis necesario.

5. Práctica a entregar

El alumno deberá empaquetar todos los archivos relacionados en el proyecto en un archivo con nombre `traductor.tgz` y entregarlo antes de la fecha que se publicará en la página web de la asignatura. Tenga en cuenta que no se incluirán ficheros objeto, ni ejecutables, ni la carpeta `datos`. Es recomendable que haga una “limpieza” para eliminar los archivos temporales o que se puedan generar a partir de los fuentes.

El alumno debe incluir el archivo *Makefile* para realizar la compilación. Tenga en cuenta que los archivos deben estar distribuidos en directorios:

traductor	— include	<i>Ficheros de cabecera (.h)</i>
	— src	<i>Código fuente (.cpp)</i>
	— obj	<i>Código objeto (.o)</i>
	— lib	<i>Bibliotecas</i>
	— doc	<i>Documentación</i>
	— bin	<i>Ficheros ejecutables</i>
	— datos	<i>Fichero de datos.</i>

Para realizar la entrega, en primer lugar, realice la limpieza de archivos que no se incluirán en ella, y sitúese en la carpeta superior (en el mismo nivel de la carpeta “*traductor*”) para ejecutar:

```
prompt% tar zcv traductor.tgz traductor
```

tras lo cual, dispondrá de un nuevo archivo traductor.tgz que contiene la carpeta traductor así como todas las carpetas y archivos que cuelgan de ella.

6. Referencias

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