UNIVERSIDAD
DE GRANADA

Estructuras de datos

2º Grado en Ingeniería Informática

Práctica 2. Abstracción

DECSAI

Departamento de Ciencias de la Computación e Inteligencia Artificial

Índice de contenidos

1 Introducción

Los objetivos que se pretenden alcanzar con la realización de este guión de prácticas son los siguientes:

- Asimilar los conceptos fundamentales de abstracción, aplicándolos al desarrollo de programas.
- Documentar un tipo de dato abstracto (en adelante, TDA).
- Practicar con el uso de `doxygen` para generar la documentación del TDA.
- Profundizar en los conceptos relacionados de especificación del TDA, representación del TDA, función de abstracción e invariante de la representación.

Para realizar correctamente este guión de prácticas se deben haber estudiado los siguientes temas:

- Tema 1: Introducción a la eficiencia de los algoritmos.
- Tema 2: Abstracción de datos.

2 Tipos de datos abstractos (TDA)

Los TDA son nuevos tipos de datos que tienen asociado un grupo de operaciones que proporcionan la única manera de utilizarlos. De esta forma, tan solo es necesario conocer las operaciones que se pueden usar sobre ellos, pero no necesitamos saber cómo se almacenan los datos ni cómo se implementan las operaciones asociadas.

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 siguientes exigencias:

- Debe existir un conjunto mínimo de operaciones para garantizar la abstracción. Este conjunto mínimo debe permitir resolver cualquier problema en el que se necesite el TDA.
- Las operaciones deben usarse con bastante frecuencia.
- Se debe considerar que el tipo de dato pueda sufrir en el futuro modificaciones que conlleven también la modificación de las operaciones. Por tanto, un número muy alto de operaciones puede conllevar un gran esfuerzo en la modificación.

Por otro lado, las operaciones seleccionadas para el nuevo tipo de dato pueden clasificarse en dos conjuntos:

- **Fundamentales**. Son aquellas operaciones 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 les denomina operaciones *primitivas*.
- **No fundamentales**. Corresponden a las operaciones prescindibles, ya que se podrían construir 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 y otros programadores. Para que esta tarea pueda llevarse a cabo, los módulos donde se materializa el TDA deben estar bien documentados. Para realizar una buena documentación de un TDA se deben crear dos documentos bien diferenciados:

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

3.1 Especificación del TDA

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

- **Definición**. En esta parte debemos definir el nuevo TDA, así como todos los términos relacionados que sean necesarios para comprender el resto de la especificación.
- **Operaciones**. En esta parte debemos especificar 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 (dominio) en un conjunto establecido. Por ejemplo, si queremos definir un racional diremos:

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

3.1.2 Operaciones

En esta parte se realiza una especificación de las operaciones que se usarán sobre el TDA que se está desarrollando. Cada una de estas operaciones representará una función y, por lo tanto, se hará la especificación con la siguiente información:

- **Breve descripción**. ¿Qué es lo que hace la función? Para ello, en `doxygen` se usa la sentencia `@brief`.
- **Especificación de los parámetros de la función**. Además, por cada parámetro se especificará si el parámetro se modifica o no tras la ejecución de la función. Para ello, en `doxygen` se usa la sentencia `@param`.
- **Precondiciones**. Condiciones previas a la ejecución de la función que deben cumplirse para un buen funcionamiento de la función. Para ello, en `doxygen` se usa la sentencia `@pre`.
- **Valor devuelto por la función**. Para ello, en `doxygen` se usa la sentencia `@return`.
- **Postcondiciones**. Condiciones que deben cumplirse tras la ejecución de la función. Para ello, en `doxygen` se usa la sentencia `@post`.

Supongamos que queremos especificar en nuestro TDA `Racional` la función `comparar`. Esta función compara un racional con el racional al que apunta `this`. Una posible especificación de esta función se puede ver en el Listado 3.1.

```
/**
 * @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);
```

Listado 3.1. Especificación de la función `comparar`.

Un ejemplo donde se usa una precondición es en la función `asignar`, que asigna al racional apuntado por `this` unos valores concretos para el numerador y el denominador (ver Listado 3.2). 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 del TDA `Racional`.

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

Listado 3.2. Especificación de la función `asignar`.

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 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 implementemos 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 nuestro *tipo rep* podrían ser por ejemplo:

- Un vector de enteros formado por dos posiciones para almacenar el numerador y el denominador (ver Listado 3.3).

```
class Racional {

 private:

  int r[2];

  ...

}
```

Listado 3.3. TDA `Racional` con *tipo rep* representado por un vector de enteros.

- Dos enteros utilizados para representar el numerador y el denominador, respectivamente (ver Listado 3.4).

```
class Racional {

 private:

  int numerador;
  int denominador;

  ...

}
```

Listado 3.4. TDA `Racional` con *tipo rep* representado por dos enteros.

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 TDA. Las propiedades de esta función son:

- Parcial. Todos los valores de los objetos del *tipo rep* no se corresponden con un objeto del TDA. Por ejemplo, valores del numerador iguales a 1 y del denominador iguales a 0 no son valores válidos para un objeto del TDA `Racional`.
- Todos los elementos del TDA 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 tanto, en la documentación debemos incluir esta aplicación para indicar el significado de la representación. Esta aplicación tiene dos objetivos:

- 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:

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

- 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 TDA se garantiza que la representación cumple dicha condición y cuando se devuelve el control de la llamada debemos asegurarnos de que se sigue cumpliendo. En nuestro ejemplo del TDA `Racional`, en la documentación de la implementación incluiríamos lo mostrado en el Listado 3.5.

```
class Racional {

 private:
/**
 * @page repConjunto Rep del TDA Racional
 *
 * @section invConjunto Invariante de la representacion
 *
 * El invariante es \e rep.den!=0
 *
 * @section faConjunto Funcion de abstraccion
 *
 * Un objeto valido @e rep del TDA Racional representa al valor
 *
 * (rep.num,rep.den)
 *
 */

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

 public:
```

Listado 3.5. Invariante de la representación y función de abstracción del TDA `Racional`.

4 Ejercicio

Queremos desarrollar una aplicación que nos permita obtener diferentes definiciones asociadas a una palabra dada y viceversa, obtener las palabras que contengan en su definición una palabra clave dada. Con este fin, desarrollaremos varios TDA:

- `Término`. Se compone de una palabra y una o más definiciones asociadas a esa palabra. Cada una de las definiciones puede contener más de una palabra.

- `Diccionario`. Es una colección de términos ordenados alfabéticamente.

Se pide desarrollar el TDA `Término` y el TDA `Diccionario`. En particular, para cada uno de estos TDA se pide:

- Dar la especificación. Establecer una definición y el conjunto de operaciones básicas.
- Determinar diferentes estructuras de datos para *tipo rep*.
- Escoger una de las estructuras de datos para representar el *tipo rep*.
- Para la estructura de datos del *tipo rep*, establecer cuál es el invariante de la representación y la función de abstracción.
- Fijado el *tipo rep*, realizar la implementación de las operaciones.
- Documentar los TDA desarrollados.
- Haciendo uso de `pruebadiccionario.cpp`, probar los TDA desarrollados.

4.1 Fichero de datos

Para probar el programa se usará un fichero compuesto por una serie de líneas. Cada una de ellas se corresponde con una palabra y su definición. El formato del fichero es el siguiente (ver Listado 4.1):

```
palabra;definición
```

```
cockney;the nonstandard dialect of natives of the east end of London
cockney;a native of the east end of London
cocteau;French writer and film maker who worked in many artistic media (1889−1963)
cody;United States showman famous for his Wild West Show (1846−1917)
cognac;high quality grape brandy distilled in the Cognac district of France
cohan;United States songwriter and playwright famous for his patriotic songs (1878−1942)
coke;carbon fuel produced by distillation of coal
coke;Coca Cola is a trademarked cola
coke;street names for cocaine
col;a pass between mountain peaks
colbert;butter creamed with parsley and tarragon and beef extract
cole;coarse curly−leafed cabbage
cole;a hardy cabbage with coarse curly leaves that do not form a head
coleridge;English romantic poet (1772−1834)
colette;French writer of novels about women (1873−1954)
colleen;an Irish girl
collier;someone who works in a coal mine
collins;tall  iced drink of liquor (usually gin) with fruit juice
collins;English writer noted for early detective novels (1824−1889)
cologne;a perfumed liquid made of essential oils and alcohol
```

Listado 4.1. Parte del fichero de datos `diccionario.txt`.

Como se puede observar, una palabra puede tener tantas entradas como definiciones posibles. En el directorio **datos** tenéis un diccionario de palabras en inglés (`diccionario.txt`) para probar los TDA desarrollados.

4.2 Módulos a desarollar

Se desarrollarán al menos dos módulos (aunque es posible añadir más módulos en caso de ser necesarios):

- El módulo asociado al TDA `Término` (`Termino.cpp` y `Termino.h`).
- El módulo asociado al TDA `Diccionario` (`Diccionario.cpp` y `Diccionario.h`).

Todos los módulos tienen que tener la documentación suficiente para que cualquier usuario pueda usarlos sin problemas. Para documentar los módulos se usará el programa `doxygen`.

4.2.1 Módulo `Término`

Este módulo está dedicado a la especificación e implementación del TDA `Término`. Este TDA está formado por una palabra y un conjunto de definiciones asociadas a esa palabra. Las operaciones más relevantes de este TDA son:

- Constructor por defecto.
- Constructor por parámetros.
- Constructor por copia.
- Operadores de consulta:
  - Obtener la palabra.
  - Obtener las definiciones asociadas a la palabra.
  - Obtener el número de definiciones asociadas a la palabra.
- Operadores de modificación:
  - Establecer la palabra.
  - Añadir nuevas definiciones asociadas a la palabra.

Además, se podrían sobrecargar los operadores de escritura y lectura en flujos, así como añadir cualquier otra operación que se considere necesaria.

4.2.2 Módulo `Diccionario`

Este módulo está dedicado a la especificación e implementación del TDA `Diccionario`. Este TDA está formado por un conjunto de términos ordenados alfabéticamente (de acuerdo a la palabra asociada a cada término). Las operaciones más relevantes de este TDA son:

- Constructor por defecto.
- Constructor por parámetros.
- Constructor por copia.
- Operadores de consulta:
  - Obtener las definiciones asociadas a la palabra de un término.

  - Obtener todos los términos del diccionario.
  - Obtener el número de términos del diccionario.
- Operadores de modificación:
  - Añadir un nuevo término.
  - Eliminar un término.

Además, se deben implementar las siguientes operaciones:

- Sobrecarga de los operadores de escritura y lectura en flujos.
- Filtrado por intervalo. Dado un diccionario, el objetivo es obtener un subconjunto de este diccionario que incluya únicamente los términos cuya palabra asociada esté en el intervalo especificado por `[carácter_inicio, carácter_fin]`.
- Filtrado por palabra clave. Dado un diccionario, el objetivo es obtener un subconjunto de este diccionario que incluya únicamente las palabras en cuya definición aparezca la palabra clave. Si una palabra tiene varias definiciones, solo se devolverían como resultado del filtrado por palabra clave aquellas definiciones relacionadas con la palabra clave.
- Recuento de definiciones. Dado un diccionario, el objetivo es obtener el número total de definiciones, el máximo de definiciones asociadas a una única palabra y el promedio de definiciones por palabra.

4.3 Fichero de prueba

En el directorio **src** se encuentra el fichero `pruebadiccionario.cpp` (ver Listado 4.2). Este código debe funcionar con los TDA desarrollados.

```
#include "Diccionario.h"
#include <fstream>
#include <iostream>

using namespace std;

int main(int argc, char * argv[]){

 if (argc!=2){
    cout<<"Dime el nombre del fichero con el diccionario"<<endl;
    return 0;
  }

  ifstream f (argv[1]);
  if (!f){
   cout<<"No puedo abrir el fichero "<<argv[1]<<endl;
   return 0;
  }

  Diccionario mi_diccionario;
  f>>mi_diccionario; //Cargamos en memoria el diccionario
```

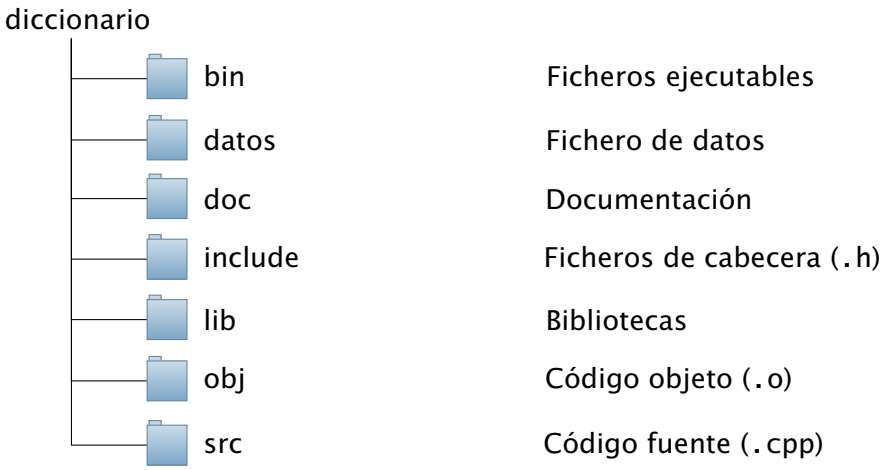
```
  /* Exhibir aqui la funcionalidad programada para el TDA Diccionario / TDA
     Termino
   Algunas sugerencias:
   - Obtener las definiciones asociadas a una palabra
   - Obtener el (sub)diccionario de términos comprendidos en
       [caracter_inicial, caracter_final]
   - Obtener el (sub)diccionario de términos asociados a una palabra clave.
       Ejemplo: el diccionario de terminos asociados a "color"
   - Obtener el numero total de definiciones, el maximo de definiciones
       asociadas a una unica palabra y el promedio de definiciones por palabra
   - Cualquier otra funcionalidad que considereis de interes
  */

}
```

Listado 4.2. `pruebadiccionario.cpp`.

5 Entrega

Se deberán empaquetar todos los ficheros relacionados con la práctica en un fichero con nombre `diccionario.tar`. No se incluirán ficheros objeto ni ficheros ejecutables. Por tanto, se deben eliminar los ficheros temporales y aquellos que se generen a partir de los fuentes.

Se deberá incluir el fichero `Makefile` para realizar la compilación. Los ficheros deben estar distribuidos en directorios:



Para realizar la entrega, se deben eliminar los ficheros que no se incluirán en ella. Estando en la carpeta superior (en el mismo nivel de la carpeta **diccionario**) se ejecutará la siguiente instrucción:

```
tar -cvf diccionario.tar diccionario
```

tras lo cual se dispondrá de un nuevo fichero llamado `diccionario.tgz` que contendrá la carpeta `diccionario` y todas las carpetas y ficheros que cuelgan de ella.

Para realizar la práctica **no se podrán usar los contenedores de la Standard Template Library (STL) de C++.**