Ejemplo de fechas: Especificación no formal

...

año: fecha f → entero

{Dada una fecha f, se obtiene el entero que corresponde al año en la fecha f}

iguales: fecha f1, fecha f2 → booleano

{Dadas dos fechas f1 y f2, se obtiene un booleano con valor verdad si y solo si la fecha f1 es igual que la fecha f2, es decir, corresponden al mismo día, mes y año.}

anterior: fecha f1, fecha f2 → booleano

{Dadas dos fechas f1 y f2, se obtiene un booleano con valor verdad si y solo si la fecha f1 es cronológicamente anterior a la fecha f2.}

posterior: fecha f1, fecha f2 → booleano

{Dadas dos fechas f1 y f2, se obtiene un booleano con valor verdad si y solo si la fecha f1 es cronológicamente posterior a la fecha f2.}

fespec

Paso de especificación a Implementación

- Lo que el módulo exporta constituye su parte visible o **interfaz**
- La parte pública debe ir acompañada de la **documentación** necesaria
- El código a partir de la palabra *implementación* queda oculto, no es visible desde el exterior

módulo fechas

exporta

INTERFAZ

tipo fecha

procedimiento crear(**ent** d,m,a:entero; **sal** f:fecha; **sal** error:booleano)

función día(f:fecha) **devuelve** entero

función mes(f:fecha) **devuelve** entero

función año(f:fecha) **devuelve** entero

función iguales(f1,f2:fecha) **devuelve** booleano

función anterior(f1,f2:fecha) **devuelve** booleano

función posterior(f1,f2:fecha) **devuelve** booleano

↑ tratamiento de caso de error

procedimiento día (ent f:fecha; sal d:entero)

implementación

tipo fecha = **registro**

eldía,elmes,elaño:entero

freg

Parte OCULTA (privada):

- Representación interna de los datos

- Implementación de las operaciones

...

Paso de especificación a Implementación

...

```
procedimiento crear(ent d,m,a:entero; sal f:fecha; sal error:booleano)
principio
    si d<1 or d>31 or m<1 or m>12 or a<1583 or (d=31 and (m=2 or m=4 or m=6 or m=9 or m=11)) or  (m=2 and d=30) entonces
     {1582=año del inicio de la adopción del calendario gregoriano}
       error:=verdad
    sino
       si m=2 and d=29 and ((a mod 4/=0) or  (a mod 4=0 and a mod 100=0 and a mod 400/=0))  entonces
          error:=verdad
       sino
          f.eldía:=d; f.elmes:=m; f.elaño:=a; error:=falso
       fsi
    fsi
fin

función día(f:fecha) devuelve entero
principio
  devuelve f.eldía
fin

función mes(f:fecha) devuelve entero
principio
  devuelve f.elmes
fin

función año(f:fecha) devuelve entero
principio
  devuelve f.elaño
fin
```

← *tratamiento de caso de error*

Parte OCULTA (privada):

- Representación interna de los datos

- Implementación de las operaciones

...

Paso de especificación a Implementación

...

```
función iguales(f1,f2:fecha) devuelve booleano
principio
    - - devuelve f1=f2
  devuelve ((f1.elaño=f2.elaño) and (f1.elmes=f2.elmes) and (f1.eldia=f2.eldia))
fin

función anterior(f1,f2:fecha) devuelve booleano
principio
  devuelve (f1.elaño<f2.elaño) or
              ((f1.elaño=f2.elaño) and (f1.elmes<f2.elmes)) or
              ((f1.elaño=f2.elaño) and (f1.elmes=f2.elmes) and (f1.eldia<f2.eldia))
fin

función posterior(f1,f2:fecha) devuelve booleano
principio
  devuelve not( iguales(f1,f2) or anterior(f1,f2) )
fin
```

Fin *- - modulo Fechas*

→ Ver implementación en lenguaje Ada en el material de clase

Implementación de un TAD

- Cuando se implementa un TAD, se está construyendo una interpretación de la especificación:
  - → **La implementación de un TAD debe corresponderse con la especificación, manteniendo sus propiedades**, si es posible *sin introducir "basura" ni "confusión"*
    - ***Basura**: la representación de datos elegida para el TAD hace que sean representables más valores, que los definidos por la especificación del TAD (llamados "basura")*
    - ***Confusión**: la representación de datos elegida para el TAD hace que varios de los valores definidos por la especificación del TAD tengan una misma representación (se confunden)*
  - → Si no es posible evitar introducir alguna ***limitación***, "***confusión***" o "***basura***" en la implementación del TAD, **deberán estar documentadas en la interfaz**, para que quien use la implementación del TAD sepa exactamente qué se le está ofreciendo

Implementación de un TAD

Ejemplos de "basura" y "confusión" posibles en un TAD Fechas:

- *Basura: que un dato fecha pueda tomar valores de fechas no válidas*

  ***31-2**-2011 2-**15**-2011 …*

- *Confusión: que varios valores válidos se representen exactamente igual y por tanto sean indistinguibles*

  *31-1-19**20** → 31-1-**20** ← 31-1-20**20***

Ejemplos de limitaciones:

- Rango no infinito de valores posibles (ejemplo típico: números enteros)
- Capacidad de almacenamiento limitada, por la cantidad de memoria disponible en el ordenador o por la capacidad máxima de la estructura de memoria utilizada en la implementación
  - Ej.: implementación que limita el tamaño al de un vector

➢ **…deberán estar documentados en la interfaz del TAD**

Implementación de un TAD

Más **decisiones** a tomar:

- A menudo ocurre que, en el dominio de una operación aparece un parámetro del mismo género que su resultado

  Ejemplo, operación para sumar dos enteros:

  sumar: entero e1, entero e2→ entero

➢ Podríamos decidir implementarla como:

función sumar (e1,e2: entero) **devuelve** entero

procedimiento sumar (**ent** e1,e2: entero; **sal** r: entero)

procedimiento sumar (**e/s** e1:entero; **ent** e2: entero)

…

- combinando el uso de cualquiera de esas posibles implementaciones con el uso de una **operación *copiar* o *duplicar***, siempre se podrán generar nuevas copias separadas de los datos (valores previo y posterior a la modificación), o reemplazar el valor original con el del resultado

Implementación de un TAD

- Al implementar la operación, a menudo se puede **decidir que el resultado sea**:

  a) la **actualización** de uno de los parámetros del dominio:
  - el parámetro del dominio y el resultado se convertirán en un único parámetro de entrada y salida (**ent/sal**) del procedimiento que implementa la operación
  - se evita ocupar nueva memoria para los datos resultado y el tiempo de copiar todo lo que no resultaba modificado (**más eficiente**, en tiempo y memoria)
  - combinando su uso con el de una **operación *copiar* o *duplicar***, siempre se podrán generar nuevas copias separadas de los datos (valores previo y posterior a la modificación)

  b) una **copia** distinta, en memoria, del parámetro del dominio:
  - el parámetro del dominio será de entrada y el resultado será de salida (o el valor devuelto por la función)
  - se ocupa memoria adicional, independiente, para el valor de entrada y para el resultado (**menos eficiente**, en tiempo y memoria)
  - combinando su uso con el de una **operación *copiar* (o *duplicar)*,** siempre se podrá hacer que el nuevo valor sustituya al original (en memoria)

Tablas de frecuencia: Especificación no formal

espec tablas ***{Vista en la lección 2}***

usa naturales, enteros

género tabla

{DESCRIPCION: Los valores del TAD tablas de frecuencia representan colecciones de números enteros tales que:

- no contiene enteros repetidos, pero si se registra cuántas veces se ha introducido cada entero (su frecuencia)

- las operaciones permiten obtener la información de un entero o su frecuencia según su puesto en el orden decreciente por valores de frecuencia}

operaciones

inicializar: →tabla

{Devuelve una tabla vacía, es decir, que no contiene datos para ningún número entero}

añadir: tabla t , entero e -> tabla

{Dada una tabla t, si e no está en t, devuelve la tabla igual a la resultante de añadir e a t con número de apariciones igual a 1; si e está en t, devuelve la tabla igual a la resultante de incrementar en 1 el número de apariciones de e (su frecuencia) en t}

…

Tablas de frecuencia: Especificación no formal

...

total: tabla t → natural

{Dada una tabla t, devuelve el número total de enteros para los que contiene información}

parcial infoEnt: tabla t, natural n → entero

{Dada una tabla t y un número natural n, devuelve el entero que corresponde al n-ésimo entero en la tabla t según el orden en número de apariciones decreciente.

***Parcial:** la operación no está definida si total(t)< n}*

parcial infoFrec: tabla t, natural n → natural

{Dada una tabla t y un número natural n, devuelve el natural que corresponde al número de apariciones correspondientes al n-ésimo entero en la tabla t según el orden en número de apariciones decreciente.

***Parcial:** la operación no está definida si total(t)< n}*

fespec

Paso de especificación a Implementación

- Lo que el módulo exporta constituye su parte visible o interfaz
- La implementación queda oculta, no es visible desde el exterior

módulo tablas
exporta

INTERFAZ

tipo tabla
{Tabla de frecuencias de enteros definida en la especificación anterior... Implementación limitada a tablas con un tamaño máximo de 1000 números enteros distintos. }

↑ ¡limitación!

procedimiento inicializar(**sal** t:tabla)
{Crea una tabla vacía t de frecuencias}

tratamiento de caso de error consecuencia de la limitación ↓

procedimiento añadir(**e/s** t:tabla; **ent** n:entero; **sal** error:booleano)
{Modifica t incrementando en 1 la frecuencia de n, y si n no estaba en t lo introduce con frecuencia 1. La implementación limitada a contener información de un máximo de 1000 números distintos, por tanto, si n no cabe en la tabla, devuelve error=verdad, y en caso contrario error=falso. }

función total(t:tabla) **devuelve** entero
{Devuelve el nº de enteros distintos en la tabla t.}

(parcial) tratamiento caso error ↓

procedimiento info(**ent** t:tabla; **ent** n:natural; **sal** e:entero; **sal** m:natural; **sal** error:booleano)
{Devuelve en e el entero que ocupa el n-ésimo lugar en la tabla t, en orden de frecuencias decrecientes, en m devuelve su frecuencia, y en error devuelve falso.
(Parcial:) Si no existe ese entero, (total(t)<n), devuelve 0 en ambos y error=verdad.*}*

implementación
… *{Todo lo que no hace falta saber para poder usar el módulo}*
fin

→ ***Ver completo en el material de clase***

Especificación / Implementación

- Dada una especificación de TAD puede haber muchas implementaciones válidas.
- Un cambio de implementación de un TAD es transparente a los programas que lo utilizan.
  - Siempre que no cambie la interfaz (parte pública) de la implementación

Implementación de un TAD

- **Programación con TAD:** Especificación *versus* implementación
  - Dada una *especificación* de un TAD, podremos tener diferentes *implementaciones* del mismo TAD
    - Siempre que respeten la misma *especificación*, las *implementaciones* serán intercambiables de forma transparente para los usuarios (programas) del TAD
  - Dada una *especificación* de un TAD, se podrán *implementar* de forma independiente, simultáneamente y por separado(1):
    a. Los algoritmos que implementan las operaciones del TAD
    b. Los algoritmos que usan el TAD

- En el momento de repartir el trabajo de implementación para escribir código *esto* (*Intercambiables / implementables de forma independiente y en paralelo*) **se traduce en**:
  - → conociendo únicamente la ***especificación* del TAD y** habiendo realizado (decisiones tomadas) **la parte pública o *interfaz* de la implementación** *del TAD*
  - → Si la parte pública de la implementación cambia pero respeta la especificación del TAD (¡o no será implementación del TAD!), entonces el cambio no será transparente, pero será muy poco costoso

(1) En los lenguajes de programación modular se incorpora la compilación separada de los módulos

Conociendo la interfaz del TAD…

- Lo que el módulo exporta constituye su parte visible o interfaz
- La implementación queda oculta, no es visible desde el exterior

módulo tablas
exporta
tipo tabla

INTERFAZ

{Tabla de frecuencias de enteros definida en la especificación anterior... Implementación limitada a tablas con un tamaño máximo de 1000 números enteros distintos. }

↑ ¡limitación!

procedimiento inicializar(**sal** t:tabla)
{Crea una tabla vacía t de frecuencias}

tratamiento de caso de error consecuencia de la limitación ↓

procedimiento añadir(**e/s** t:tabla; **ent** n:entero; **sal** error:booleano)
{Modifica t incrementando en 1 la frecuencia de n, y si n no estaba en t lo introduce con frecuencia 1. La implementación limitada a contener información de un máximo de 1000 números distintos, por tanto, si n no cabe en la tabla, devuelve error=verdad, y en caso contrario error=falso. }

función total(t:tabla) **devuelve** entero
{Devuelve el nº de enteros distintos en la tabla t.}

(parcial) tratamiento caso error ↓

procedimiento info(**ent** t:tabla; **ent** n:natural; **sal** e:entero; **sal** m:natural; **sal** error:booleano)
{Devuelve en e el entero que ocupa el n-ésimo lugar en la tabla t, en orden de frecuencias decrecientes, en m devuelve su frecuencia, y en error devuelve falso.
(Parcial:) Si no existe ese entero, (total(t)<n), devuelve 0 en ambos y error=verdad.*}*

implementación
… *{Todo lo que no hace falta saber para poder usar el módulo}*
fin

→ ***Ver completo en el material de clase***

… se puede escribir el programa que usa el TAD

Ejemplo de uso del módulo:

```
programa estadistica
  {Lee una secuencia de enteros de un fichero y escribe en pantalla cada entero
     distinto leído junto con su frecuencia de aparición, en orden de frecuencias
     decrecientes. Si la tabla se llena se muestra un error.}

importa tablas, cadenas, ficheros
variables
  f:fichero de entero;
  nombre:cadena;
  t:tabla;
  dato:entero;
  orden,frec:natural;
  error:booleano:=falso
principio
  escribir("Nombre del fichero: ");   leer(nombre);
  asociar(f,nombre);   iniciarlectura(f);
 …
```

```
...
    inicializar(t);
    mientrasQue not finFichero(f) and not error hacer
      leer(f,dato);
      añadir(t,dato,error)
    fmq;
    disociar(f);

    si error entonces
      escribirLínea("Error por saturación de la capacidad de la tabla utilizada.")
    sino
      para orden:=1 hasta total(t) hacer
        info(t,orden,dato,frec,error);
        escribirLínea("entero: ", dato, " frecuencia: ", frec)
      fpara
    fsi
  fin
```

Implementación en lenguaje C++

- Para implementar TAD utilizaremos registros (***struct***) con las operaciones.
  - otra opción posible, pero no en esta asignatura, sería utilizar Orientación a Objetos y definirlos como *clases* (no permitido, de momento)

- Antes de empezar la definición del *struct*, anunciaremos lo que consideraremos que será la ***interfaz*** de la implementación del TAD:
  - **predeclarando** el/los **tipos** de datos, **y** las **operaciones** **que se ofrecen** (exportan)
  - Todo ello acompañado de la **documentación** para los usuarios de la implementación del TAD:
    - **Debe documentar la implementación,** reflejando la correspondencia con la especificación del TAD (pero no es una mera copia de ella), y **sin mencionarse detalles ocultos de la implementación**

- Al definir el *struct*, la **representación interna** **de los datos** será privada (***private***) y para poder acceder a ella las operaciones del TAD serán funciones amigas (***friend***).

Implementación en lenguaje C++

- Las operaciones que en pseudocódigo aparecían como procedimientos se pueden codificar con *funciones void*.

- Las operaciones que en pseudocódigo aparecían como funciones se pueden codificar con funciones que devuelven un dato.

- Los **parámetros** de **entrada** se pueden codificar con
  - Parámetros de entrada (**por valor**) (utilizado si ocupan poca memoria: tipos básicos predefinidos o cadenas)
  - Parámetros **constantes por referencia** (de tipos no básicos, especialmente si ocupan mucha memoria)
- Los parámetros de **salida** o de **entrada/salida** se codifican como
  - Parámetros **por referencia**

Ejemplo de fechas: (1) archivo de cabecera (fecha.hpp) (continúa…)

```
#ifndef _FECHA_HPP
#define _FECHA_HPP
// Interfaz del TAD fecha. Pre-declaraciones:
/* Los valores del TAD fecha representan fechas válidas según las reglas del calendario
 * gregoriano (adoptado en 1583) */
struct Fecha;
/* Dados los tres valores enteros dia, mes y anyo, se devuelve en f la fecha compuesta por ellos, y en error
 * devuelve false.
 * Parcial: se precisa que 1≤dia≤31, 1≤mes≤12, 1583≤anyo, y además que dia, mes y anyo formen una
 * fecha válida según el calendario gregoriano; de lo contrario, en error devuelve el valor true.*/
void crear(int dia, int mes, int anyo, Fecha& f, bool& error);
/* Devuelve el dia de la fecha */
int dia(const Fecha& f);
/* Devuelve el mes de la fecha */
int mes(const Fecha& f);
/* Devuelve el año de la fecha */
int anyo(const Fecha& f);
/* Devuelve verdad si y sólo si f1 y f2 son la misma fecha */
bool iguales(const Fecha& f1, const Fecha& f2);
/* Devuelve verdad si y sólo si la fecha f1 es cronológicamente anterior a la fecha f2 */
bool anterior(const Fecha& f1, const Fecha& f2);
/* Devuelve verdad si y sólo si la fecha f1 es cronológicamente posterior a la fecha f2 */
bool posterior(const Fecha& f1, const Fecha& f2);
// FIN Interfaz del TAD fecha.
```

↑ *tratamiento de caso de error*

//sigue…

Implementación en lenguaje C++

Ejemplo de fechas: (1) archivo de cabecera (*fecha.hpp*) (… fin)

friend : en su implementación podrá hacerse uso de los detalles (private) de implementación del tipo de dato

Aquí listaremos las operaciones del Interfaz (y también aquellas operaciones internas) cuya implementación requiera acceso a los detalles protegidos como private

```
//Declaración:

struct Fecha {

    friend void crear(int dia, int mes, int anyo, Fecha& f, bool& error) ;
    friend int dia(const Fecha& f);
    friend int mes(const Fecha& f);
    friend int anyo(const Fecha& f);
    friend bool iguales(const Fecha& f1, const Fecha& f2) ;
    friend bool anterior(const Fecha& f1, const Fecha& f2);
    friend bool posterior(const Fecha& f1, const Fecha& f2);

  private:
    // Representación interna de los valores del TAD:
    int elDia;
    int elMes;
    int elAnyo;
};

#endif
```

private ➔ ocultación y encapsulación

(Aquí todo lo necesario para definir la representación interna del nuevo tipo)

Implementación en lenguaje C++

Ejemplo de fechas: (2) archivo fuente (.cpp) (continúa…)

```
#include "fecha.hpp"
// Implementación de las operaciones del TAD fecha

// Operaciones auxiliares sobre enteros.

// Devuelve verdad si y sólo si el año a es bisiesto.
bool esBisiesto(int a) {
    return (a % 4 == 0  &&  !(a % 100 == 0 && a % 400 != 0));
}

// Devuelve verdad si y sólo si (d,m,a) representan una fecha válida.
bool esFechaValida(int d, int m, int a) {
   …
}
```

Operaciones NO ofrecidas en el Interfaz (pero en este caso tampoco necesitan acceso a los detalles privados)

```
// Operaciones ofrecidas por el TAD:
void crear(int dia, int mes, int anyo, Fecha& f, bool& error) {
   …
};
…
```

→ *Ver completo en el material de clase*

Implementación en lenguaje C++

Ejemplo de fechas: (2) archivo fuente (.cpp) (… fin)

```
int dia(const Fecha& f) {
    return f.elDia;
};

int mes(const Fecha& f) {
    return f.elMes;
};

int anyo(const Fecha& f) {
    return f.elAnyo;
};

bool iguales(const Fecha& f1, const Fecha& f2) {
    return f1.elDia == f2.elDia && f1.elMes == f2.elMes && f1.elAnyo == f2.elAnyo;
};

bool anterior(const Fecha& f1, const Fecha& f2) {
    …
};

bool posterior(const Fecha& f1, const Fecha& f2) {
    …
};
```

→ *Ver completo en el material de clase*

Implementación en lenguaje C++

Ejemplo de Tablas: (1) archivo de cabecera (tabla_frec.*hpp*) (continúa…)

```
#ifndef _TABLA_FREC_HPP
#define _TABLA_FREC_HPP

// Interfaz del TAD tabla de frecuencias. Pre-declaraciones:

// Constantes y tipos previos
const   int  MAX_NUM_DATOS = 1000;
struct Tabla;

void inicializar(Tabla& t);
bool anyadir(Tabla& t, int n);
int total(const Tabla& t);
int infoEnt(const Tabla& t, int n);
int infoFrec(const Tabla& t, int n);
// Fin de la Interfaz
…
```

Faltaría incluir la documentación….

Queda como ejercicio introducirla, tras analizar la especificación y el código de la implementación

Implementación no robusta: no informa de uso en los casos de error?

void info(const Tabla& t, int n, int& entero, int& frecuencia, **bool& error**);

Implementación en lenguaje C++

Ejemplo de Tablas: (1) archivo de cabecera (tabla_frec.*hpp*) (… fin)

```
...
// Declaración:
struct Tabla {
    friend void inicializar(Tabla& t);
    friend bool anyadir(Tabla& t, int n);
    friend int total(const Tabla& t);
    friend int infoEnt(const Tabla& t, int n);
    friend int infoFrec(const Tabla& t, int n);

  private: // Representación interna de los valores del TAD:

      struct Frecuencia {
          int numero;
          int frec;
      };

      Frecuencia elementos [MAX_NUM_DATOS];
      int numElementos;
};
#endif
```

*Tienen que ser **cabeceras idénticas** a lo que ha aparecido **en** la parte de **Interfaz, salvo por la palabra friend***

private ➔ *ocultación y encapsulación*

(Aquí todo lo necesario para definir la representación interna del nuevo tipo)

Implementación en lenguaje C++

Ejemplo de Tablas: (2) archivo fuente (*.cpp*)

```
#include "tabla_frec.hpp“
// Implementacion de las operaciones del TAD
void inicializar(Tabla& t){
   t.numElementos = 0;
};

bool anyadir(Tabla& t, int n){
   …
};

int total(const Tabla& t) {
   return t.numElementos;
};

int infoEnt(const Tabla& t, int n) {
   …
};

int infoFrec(const Tabla& t, int n) {
   …
};
```

→ *Ver completo en el material de clase*