ESTRUCTURAS DE DATOS

INTRODUCCIÓN A LOS TIPOS ABSTRACTOS DE DATOS

Modelo vs. representación

Manuel Montenegro Montes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática – Universidad Complutense de Madrid

Ya hemos visto que los TADS sirven para pensar en un modelo conceptual en lugar de pensar si es modelo está implementado mediante un arrya una matriz...Que haya un modelo independiente al tipo de datos.

Modelo vs. representación

Veremos el puente entre el modelo y las representaciones concretas es la función de abstracción

TAD ConjuntoChar (representación 1)

Modelo

Conjuntos de letras mayúsculas

$\mathcal{P}(\{A..Z\})$

$\{A, D, Z\}$

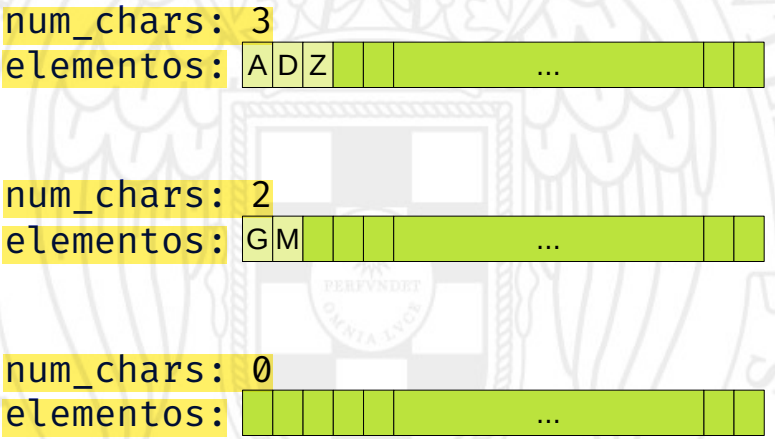
Modelo conceptual piensa en trabajar con conjuntos.

$\{G, M\}$

$\varnothing$

Representación

```
class ConjuntoChar {
   ...
private:
  int num_chars;
  char elementos[MAX_CHARS];
};
```



TAD ConjuntoChar (representación 2)

Modelo

Conjuntos de letras mayúsculas

$\mathcal{P}(\{A..Z\})$

El modelo es el mismo, cambia la representación.

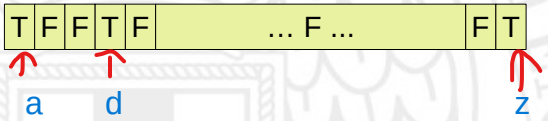
Representación

```
class ConjuntoChar {
   ...
private:
  bool esta[MAX_CHARS];
};
```

$\{A, D, Z\}$

Pensamos en los conjuntos

esta:



$\varnothing$

esta:

Todos a false

TAD de números `int`

Modelo

Elementos de $\mathbb{Z}$

enteros

Representación

32 bits en complemento a 2

25

`000…00011001`

pensamos en términos de números enteros.

-7

`111…11111001`

TAD de números `float`

Modelo	Representación
Elementos de $\mathbb{Q}$	IEEE 754
1.3423121	`00111111101010111101000011100010`
-0.5	`10111111000000000000000000000000`

No conocemos la representación pero nosotros pensamos en términos del modelo.

¡Cuidado! La representación es relevante

¿Por qué nos interesa conocer la representación?

- **Eficiencia de las operaciones**
  - El coste en tiempo puede depender de la representación.
- **Coste en memoria de la representación**
  - Algunas representaciones necesitan más memoria.
- **Limitaciones de algunas representaciones**

Algunas pueden estar más limitadas que otras.

A pesar de que él en los vídeos dice que lo más importante es el modelo, no nos podemos olvidar de la representación , la representación también es importante.

Limitaciones de algunas representaciones

- Enteros de 32 bits: -2147483648 a 2147483647.
- Coma flotante con `float`:

$$0.7$$

distintos según el modelo pero tienen la misma representación `00111111001100110011001100110011`

$$0.6999999881$$

```
float f = 7.0 / 10;
std::cout << std::setprecision(10) << f << std::endl;
```

precisión de 10 decimales

Limitaciones de algunas representaciones

- Enteros de 32 bits: -2147483648 a 2147483647.
- Coma flotante con `float`:

$$0.7$$

$$0.6999999881$$

```
00111111001100110011001100110011
```

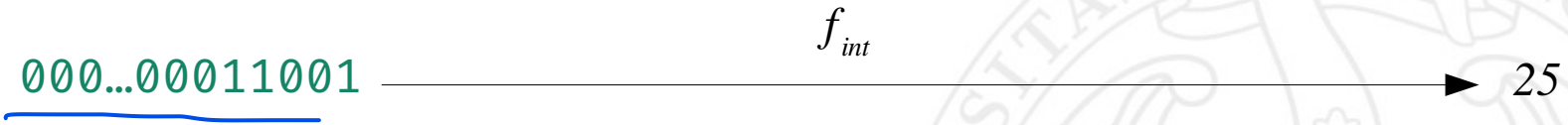
- TAD `ConjuntoChar`: capacidad máxima

Si lo representamos mediante array de caracteres, si no meto duplicados, el tamaño es suficiente, en otro caso, si hubiera duplicados podríamos sobrepasar el tamaño del array.

Función de abstracción

Función de abstracción

- Fijada la representación de un TAD, la función de abstracción asociada a una representación que asocia cada instancia de la representación con el modelo que representa.
- Ejemplo: TAD `int`



Secuencia de 32 bits para el número 25

si el bit de más a la izquierda vale...

$$f_{int}(x_{31}x_{30}\cdots x_0) = \begin{cases} \sum_{i=0}^{30} 2^i * x_i & \text{si } x_{31} = 0 \\ -(1 + \sum_{i=0}^{30} 2^i * \overline{x_i}) & \text{si } x_{31} = 1 \end{cases}$$

Viene dado por esta fórmula

No nos interesan los detalles de la fórmula

Función de abstracción

- Fijada la representación de un TAD, la función de abstracción asociada a una representación que asocia cada instancia de la representación con el modelo que representa.
- Ejemplo: TAD `ConjuntoChar`

```
class ConjuntoChar {
   ...
private:
   int num_chars;
   char elementos[MAX_CHARS];
};
```

$$f_{CC1} : \texttt{ConjuntoChar} \rightarrow \mathcal{P}(\{A..Z\})$$

$$f_{CC1}(x) = \{\ x.elementos[i] \mid 0 \leq i < x.num\_chars\ \}$$

Función de abstracción

- Fijada la representación de un TAD, la función de abstracción asociada a una representación que asocia cada instancia de la representación con el modelo que representa.
- Ejemplo: TAD `ConjuntoChar`

```
class ConjuntoChar {
   ...
private:
  bool esta[MAX_CHARS];
};
```

$$f_{CC2} : \texttt{ConjuntoChar} \rightarrow \mathcal{P}(\{A..Z\})$$

$$f_{CC2}(x) = \{\ c \in \{A..Z\}\ \mid\ x.esta[ord(c) - ord(\text{`A'})] = true\ \}$$

Tipos de operaciones

TAD = Modelo + Operaciones

- Las operaciones en un TAD se especifican en función de los modelos.

[true]

vacio() → (C: ConjuntoChar)

[C = ∅]

[l ∈ {A,...,Z}]

pertenece(l: char, C: ConjuntoChar) → (está: bool)

[está ⇔ l ∈ C]

[l ∈ {A,...,Z}]

añadir(l: char, C: ConjuntoChar)

[C = old(C) ∪ {l}]

Tipos de operaciones

- Funciones **constructoras**
  - Crean una nueva instancia del TAD.
  - Equivalen a los constructores de C++
- Funciones **observadoras**
  - No modifican el TAD sobre el que se aplican.
  - En C++ llevan el modificador `const`.
- Funciones **mutadoras**
  - Modifican el TAD sobre el que se aplican.

NOS QUEDAMOS CON ESTAS 3 CATEGORÍAS.

MUY IMPORTANTE PONER ESTE MODIFICADOR.

Ejemplo

TODO SOBRE EL EJEMPLO DE LA FECHA.

[true]

vacio() → (C: ConjuntoChar)

OPERACIÓN CONSTRUCTORA

[C = ∅]

[l ∈ {A,...,Z}]

pertenece(l: char, C: ConjuntoChar) → (está: bool)

OPERACIÓN OBSERVADORA

CONST

[está ⇔ l ∈ C]

[l ∈ {A,...,Z}]

añadir(l: char, C: ConjuntoChar)

OPERACIÓN MUTADORA

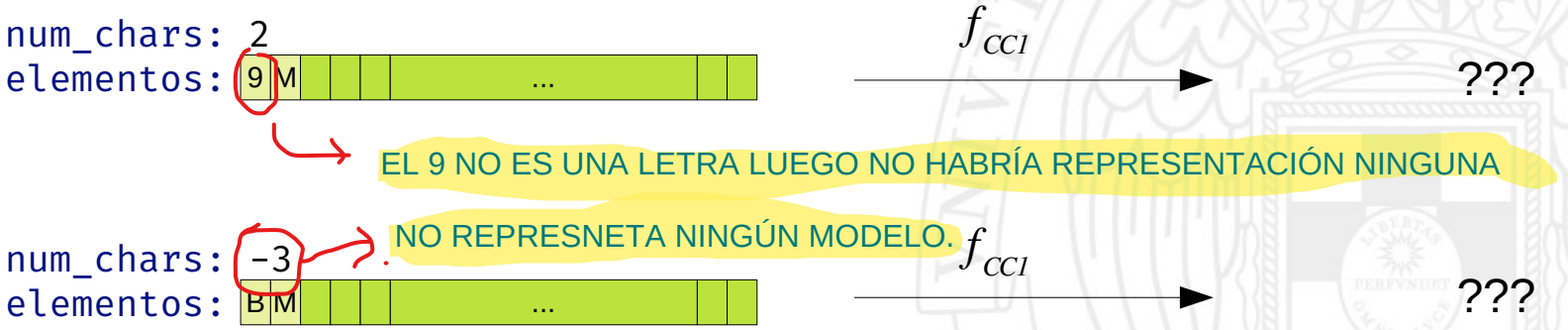
[C = old(C) ∪ {l}]

Invariante de la representación

Instancias no válidas

- No todas las instancias de una representación denotan un modelo.

```
class ConjuntoChar {
   ...
private:
  int num_chars;
  char elementos[MAX_CHARS];
};
```



Invariante de representación

- Un **invariante de representación** es una fórmula lógica que especifica cuándo una instancia es válida.

```
class ConjuntoChar {
   ...
private:
  int num_chars;
  char elementos[MAX_CHARS];
};
```

ESTO COMO EN FAL, SE DEBE DE PRESERVAR, NO SE SI LO UTILIZAREMOS MUCHO EN ED

$$I_{CCI}(x) = \quad \text{INVARIANTE}$$

$$0 \leq x.num\_chars \leq MAX\_CHARS \wedge$$
$$\forall i:\ 0 \leq i < x.num\_chars \Rightarrow x.elementos[i] \in \{A..Z\}$$

LETRAS MAYUSUCULAS EN EL RANGO A....Z

Invariante de representación

- Un **invariante de representación** (o invariante de clase) es una fórmula lógica que especifica cuándo una instancia es válida.

```
class ConjuntoChar {
   ...
private:
  bool esta[MAX_CHARS];
};
```

$$I_{CC2}(x) = true$$

Invariantes y operaciones

TODAS DEBEN PRESERVAR EL INVARIANTE

- Las operaciones constructoras deben producir una instancia que cumpla el invariante.
- Las operaciones consultoras pueden asumir que la instancia cumple el invariante.
- Las operaciones mutadoras pueden asumir que la instancia cumple el invariante, y han de preservarlo al final de su ejecución.

[true]

vacio() → (C: ConjuntoChar)

[C = ∅]

[l ∈ {A,...,Z}]

pertenece(l: char, C: ConjuntoChar) → (está: bool)

[está ⇔ l ∈ C]

[l ∈ {A,...,Z}]

añadir(l: char, C: ConjuntoChar)

[C = old(C) ∪ {l}]