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

Modelo vs. representación

TAD `ConjuntoChar` (representación 1)

Modelo

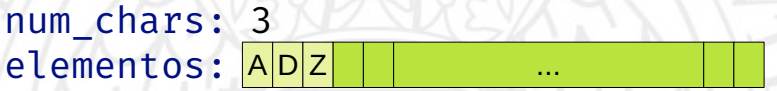
Conjuntos de letras mayúsculas

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

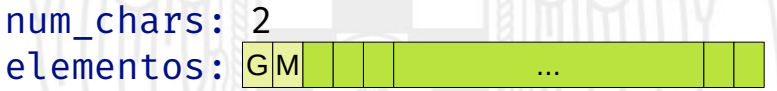
Representación

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

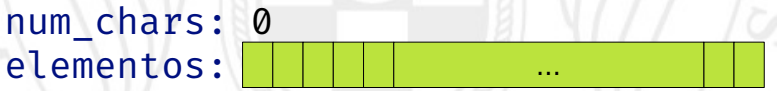
$\{A, D, Z\}$



$\{G, M\}$



$\varnothing$



TAD `ConjuntoChar` (representación 2)

Modelo

Conjuntos de letras mayúsculas

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

Representación

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

$\{A, D, Z\}$

$\varnothing$

TAD de números `int`

Modelo	Representación
Elementos de $\mathbb{Z}$	32 bits en complemento a 2

25	`000…00011001`
-7	`111…11111001`

TAD de números `float`

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

¡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**

Limitaciones de algunas representaciones

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

$$0.7$$

$$0.6999999881$$

```
00111111001100110011001100110011
```

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

Limitaciones de algunas representaciones

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

$$0.7$$

```
00111111001100110011001100110011
```

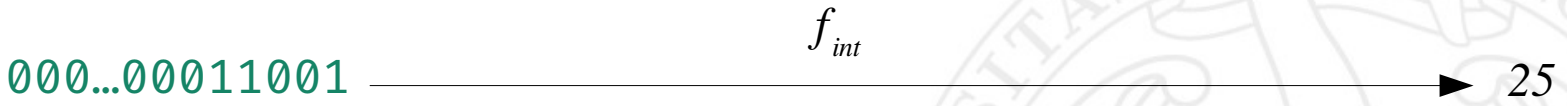
$$0.6999999881$$

- TAD `ConjuntoChar`: capacidad máxima

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`



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

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.

Ejemplo

[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)

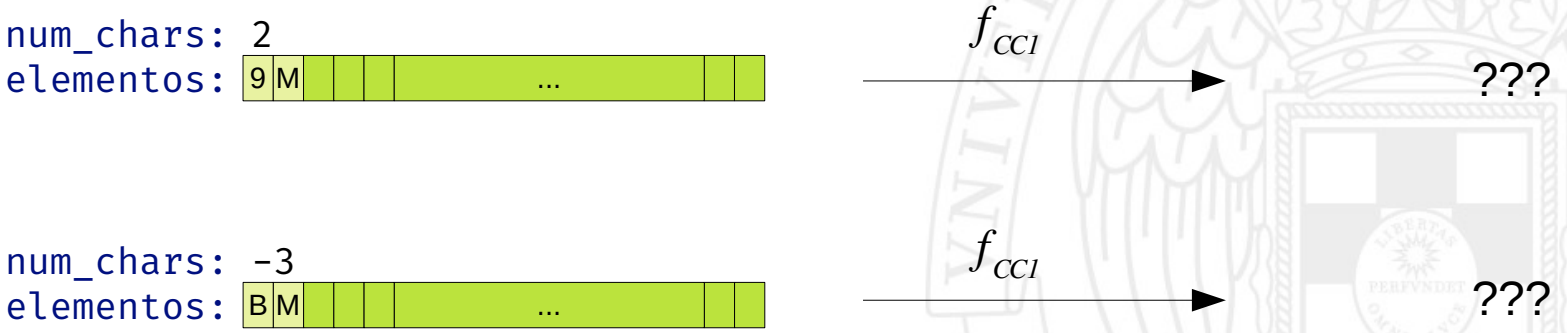
[C = old(C) ∪ {l}]

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];
};
```

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

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

- 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}]