ESTRUCTURAS DE DATOS

INTRODUCCIÓN A LOS TIPOS ABSTRACTOS DE DATOS

TADs: definición

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

¿Qué hemos hecho mal?

La lógica del juego utiliza detalles relativos a la implementación de los conjuntos de caracteres

```
int main() {
  int jugador_actual = 1;
  ConjuntoChar letras_nombradas;
  letras_nombradas.num_chars = 0;

  char letra_actual = preguntar_letra(jugador_actual);

  while (!esta_en_conjunto(letra_actual, letras_nombradas)) {
    letras_nombradas.elementos[letras_nombradas.num_chars] = letra_actual;
    letras_nombradas.num_chars++;

    jugador_actual = cambio_jugador(jugador_actual);
    letra_actual = preguntar_letra(jugador_actual);
  }

  std::cout << "Jugador " << jugador_actual << " ha perdido!" << std::endl;
  std::cout << "La letra repetida ha sido: " << letra_actual << std::endl;
  return 0;
}
```

¿Qué hemos hecho mal?

Sin embargo, aquí sí lo hemos hecho bien...

```
int main() {
  int jugador_actual = 1;
  ConjuntoChar letras_nombradas;
  letras_nombradas.num_chars = 0;

  char letra_actual = preguntar_letra(jugador_actual);

  while (!esta_en_conjunto(letra_actual, letras_nombradas)) {
    letras_nombradas.elementos[letras_nombradas.num_chars] = letra_actual;
    letras_nombradas.num_chars++;

    jugador_actual = cambio_jugador(jugador_actual);
    letra_actual = preguntar_letra(jugador_actual);
  }

  std::cout << "Jugador " << jugador_actual << " ha perdido!" << std::endl;
  std::cout << "La letra repetida ha sido: " << letra_actual << std::endl;
  return 0;
}
```

Abstrayendo los detalles

N

S

O

T

E

T

V

Jugador 1

Jugadora 2

- El sitio en el que se guardan las letras nombradas hasta el momento se corresponde con la definición matemática de **conjunto**.

$$LetrasNombradas = \{\text{'N'}, \text{'E'}, \text{'S'}, \text{'O'}, \text{'T'}, \text{'V'}\}$$

En un lenguaje ideal...

```
int main() {
  int jugador_actual = 1;
  LetrasNombradas = ∅;

  char letra_actual = preguntar_letra(jugador_actual);

  while (letra_actual ∉ LetrasNombradas) {
    LetrasNombradas = LetrasNombradas ∪ {letra_actual}

    jugador_actual = cambio_jugador(jugador_actual);
    letra_actual = preguntar_letra(jugador_actual);
  }

  std::cout << "Jugador " << jugador_actual << " ha perdido!" << std::endl;
  std::cout << "La letra repetida ha sido: " << letra_actual << std::endl;
  return 0;
}
```

¿Qué necesitamos de un conjunto?

- Obtener un conjunto vacío.

  $LetrasNombradas = \varnothing;$

- Saber si una letra pertenece (o no) a un conjunto.

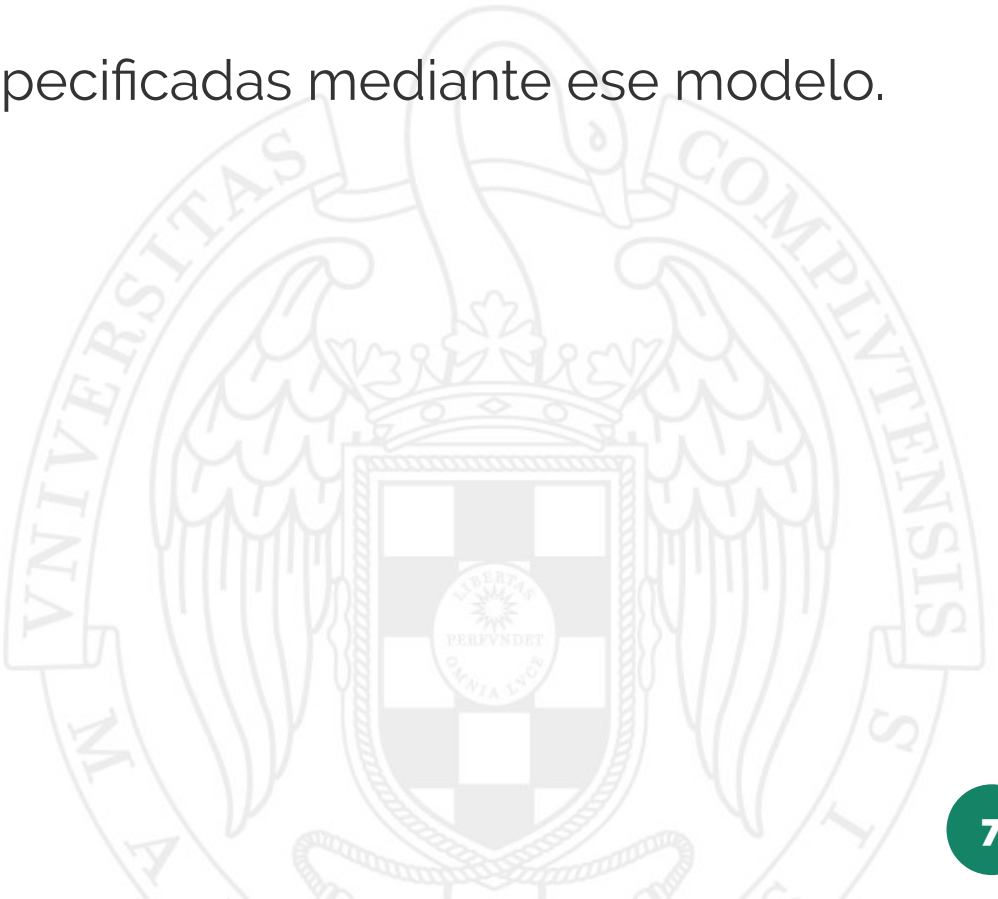
  `while (`$letra\_actual \notin LetrasNombradas$`) { … }`

- Añadir una letra a un conjunto.

  $LetrasNombradas = LetrasNombradas \cup \{letra\_actual\}$

Tipo Abstracto de Datos: definición

- Un **tipo abstracto de datos** (TAD) es un tipo de datos asociado con:
  - Un **modelo** conceptual.
  - Un conjunto de **operaciones**, especificadas mediante ese modelo.



En nuestro ejemplo

Tipo de datos: `ConjuntoChar`

- **Modelo**: conjuntos de letras, en el sentido matemático del término.
- **Operaciones**:

[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}]

Implementación del TAD

- Nuestro modelo conceptual admite varias representaciones en C++. Hemos propuesto dos:
- **Representación 1**: array de caracteres.

```
struct ConjuntoChar {
  int num_chars;
  char elementos[MAX_CHARS];
};
```

- **Representación 2**: array de booleanos.

```
struct ConjuntoChar {
  bool esta[MAX_CHARS];
};
```

Cada representación implementa de manera distinta las operaciones mostradas anteriormente

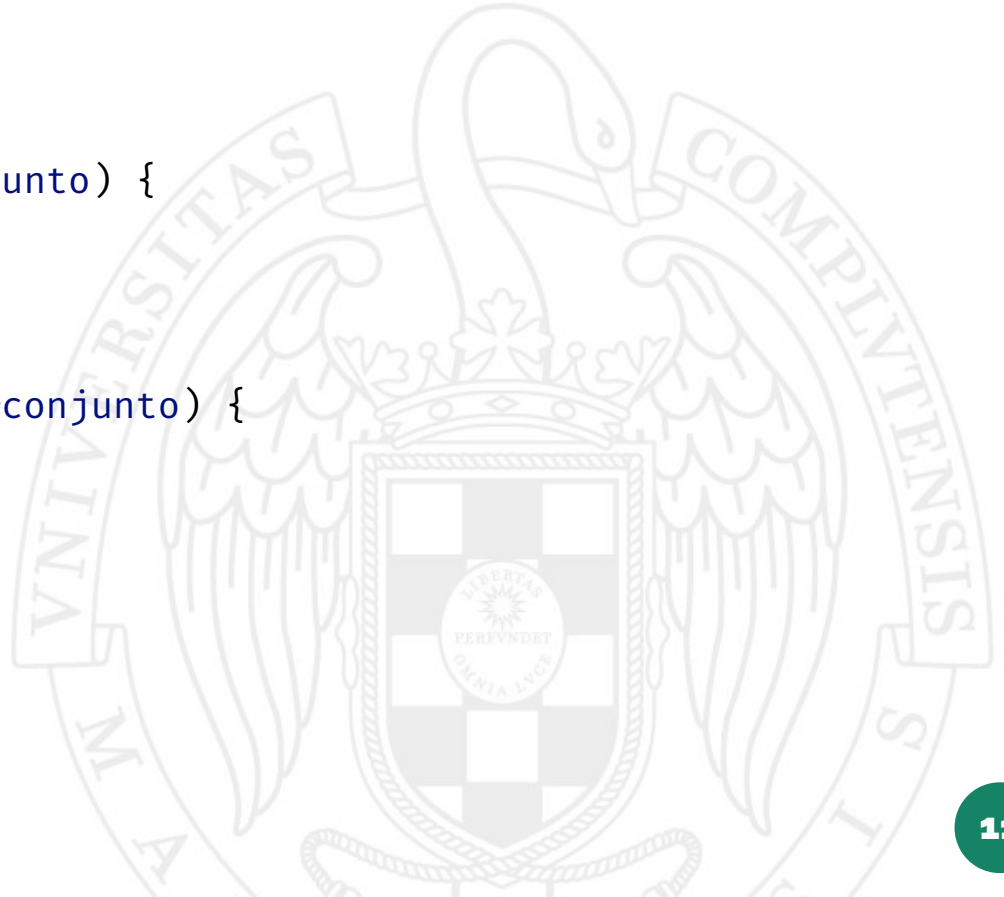
La representación determina la eficiencia de las operaciones implementadas

Representación 1

```
void vacio(ConjuntoChar &result) {
  result.num_chars = 0;
}

void anyadir(char letra, ConjuntoChar &conjunto) {
  assert (conjunto.num_chars < MAX_CHARS);
  assert (letra ⩾ 'A' && letra ⩽ 'Z');
  conjunto.elementos[conjunto.num_chars] = letra;
  conjunto.num_chars++;
}

bool pertenece(char letra, const ConjuntoChar &conjunto) {
  assert (letra ⩾ 'A' && letra ⩽ 'Z');
  int i = 0;
  while (i < conjunto.num_chars && conjunto.elementos[i] ≠ letra) {
    i++;
  }
  return conjunto.elementos[i] == letra;
}
```

Representación 2

```
void vacio(ConjuntoChar &result) {
  for (int i = 0; i < MAX_CHARS; i++) {
    result.esta[i] = false;
  }
}

void anyadir(char letra, ConjuntoChar &conjunto) {
  assert (letra ⩾ 'A' && letra ⩽ 'Z');
  conjunto.esta[letra - 'A'] = true;
}

bool pertenece(char c, const ConjuntoChar &conjunto) {
  assert (c ⩾ 'A' && c ⩽ 'Z');
  return conjunto.esta[c - 'A'];
}
```



Nuestro programa ideal...

```
int main() {
  int jugador_actual = 1;
  LetrasNombradas = ∅;

  char letra_actual = preguntar_letra(jugador_actual);

  while (letra_actual ∉ LetrasNombradas) {
    LetrasNombradas = LetrasNombradas ∪ {letra_actual}

    jugador_actual = cambio_jugador(jugador_actual);
    letra_actual = preguntar_letra(jugador_actual);
  }

  std::cout << "Jugador " << jugador_actual << " ha perdido!" << std::endl;
  std::cout << "La letra repetida ha sido: " << letra_actual << std::endl;
  return 0;
}
```

… y nuestro programa real

```
int main() {
  int jugador_actual = 1;
  ConjuntoChar letras_nombradas;
  vacio(letras_nombradas);

  char letra_actual = preguntar_letra(jugador_actual);

  while (!pertenece(letra_actual, letras_nombradas)) {
    anyadir(letra_actual, letras_nombradas);

    jugador_actual = cambio_jugador(jugador_actual);
    letra_actual = preguntar_letra(jugador_actual);
  }

  std::cout << "Jugador " << jugador_actual << " ha perdido!" << std::endl;
  std::cout << "La letra repetida ha sido: " << letra_actual << std::endl;
  return 0;
}
```

¿Qué hemos ganado?

1) Simplificar el desarrollo

No hemos de preocuparnos de cómo está implementado `ConjuntoChar`.

2) Reutilización

`ConjuntoChar` puede utilizarse en otros contextos.

3) Separación de responsabilidades

Podemos reemplazar una implementación de `ConjuntoChar` por otra sin alterar el resto del programa.

Pero hay personas despistadas

```
int main() {
   ...

  std::cout << "Jugador " << jugador_actual << " ha perdido!" << std::endl;
  std::cout << "La letra repetida ha sido: " << letra_actual << std::endl;
  std::cout << "Se han nombrado " << letras_nombradas.num_chars
            << " letras." << std::endl;
  return 0;
}
```

¿Existe algún mecanismo en el compilador de C++ que impida a las personas despistadas acceder a la representación interna de un TAD?

Encapsulación mediante clases

ESTRUCTURAS DE DATOS

INTRODUCCIÓN A LOS TIPOS ABSTRACTOS DE DATOS

TADs: definición

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