



UNIVERSIDAD DE CÓRDOBA

ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA

DEPARTAMENTO DE
INFORMÁTICA Y ANÁLISIS NUMÉRICO



PROGRAMACIÓN DECLARATIVA

INGENIERÍA INFORMÁTICA

CUARTO CURSO

PRIMER CUATRIMESTRE

Tema 6.- Abstracción de Datos



Primera
parte:
Scheme

Tema 1.- Introducción al lenguaje Scheme

Tema 2.- Expresiones y funciones

Tema 3.- Predicados y sentencias condicionales

Tema 4.- Iteración y recursión

Tema 5.- Tipos de datos compuestos

Tema 6.- **Abstracción de datos**

Tema 7.- Lectura y escritura

Segunda
parte: Prolog

Tema 8.- Introducción al lenguaje Prolog

Tema 9.- Elementos básicos de Prolog

Tema 10.- Listas

Tema 11.- Reevaluación y el “corte”

Tema 12.- Entrada y salida

Primera parte: Scheme

Tema 1.- Introducción al lenguaje Scheme

Tema 2.- Expresiones y funciones

Tema 3.- Predicados y sentencias condicionales

Tema 4.- Iteración y recursión

Tema 5.- Tipos de datos compuestos

Tema 6.- **Abstracción de datos**

Tema 7.- Lectura y escritura

Índice

1. Definición de tipo abstracto de datos
2. Representación de tipos abstractos mediante vectores
3. Representación de tipos abstractos mediante listas
4. Representación de tipos abstractos mediante listas de asociación
5. Creación de tipos abstractos de datos a partir de otros previamente definidos

ANEXO: Tipo abstracto “árbol”

Índice

1. Definición de tipo abstracto de datos
 2. Representación de tipos abstractos mediante vectores
 3. Representación de tipos abstractos mediante listas
 4. Representación de tipos abstractos mediante listas de asociación
 5. Creación de tipos abstractos de datos a partir de otros previamente definidos
- ANEXO: Tipo abstracto “árbol”

1. Definición de tipo abstracto de datos

- Un **tipo abstracto de datos** está compuesto por:
 - **Información:** atributos o campos
 - **Funciones** de manejo de la información:
 - Funciones de **creación**
 - Funciones de **acceso**
 - Funciones de **modificación**

1. Definición de tipo abstracto de datos

- **Abstracción de datos:**
 - **Observación:**
 - Las funciones del tipo abstracto de datos son utilizadas por todas las demás funciones.

1. Definición de tipo abstracto de datos

- **Abstracción de datos:**
 - Proceso que **oculta** los detalles de representación de la información de los datos
 - **Nunca** se accede **directamente** a la información
 - **Siempre** se utilizan las **funciones del tipo abstracto de datos** para manejar la información.

1. Definición de tipo abstracto de datos

- Ejemplo de tipo abstracto de datos: **libro**
 - Información:
 - Título
 - Autor
 - Tema: novela, poesía, teatro, etc.

1. Definición de tipo abstracto de datos

- **Ejemplo de tipo abstracto de datos: libro**
 - **Funciones del tipo abstracto de datos:**
 - **Creación:** Crear un libro
 - **Acceso:**
 - Consultar el título
 - Consultar el autor
 - Consultar el tema
 - **Modificación:**
 - Cambiar el título
 - Cambiar el autor
 - Cambiar el tema



TÍTULO	AUTOR	TEMA
LIBRO		

**Crear
Libro**

TÍTULO

AUTOR

TEMA

LIBRO

**Crear
Libro**

**Consultar
Título**

**Consultar
Autor**

**Consultar
Tema**

TÍTULO

AUTOR

TEMA

LIBRO

Función exterior

**Crear
Libro**

**Consultar
Título**

**Consultar
Autor**

**Consultar
Tema**

TÍTULO

AUTOR

TEMA

LIBRO

Función exterior

**Crear
Libro**

**Consultar
Título**

**Consultar
Autor**

**Consultar
Tema**

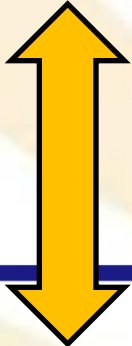
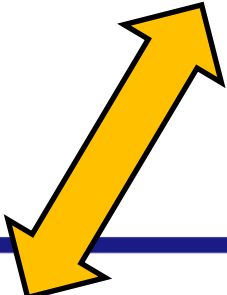
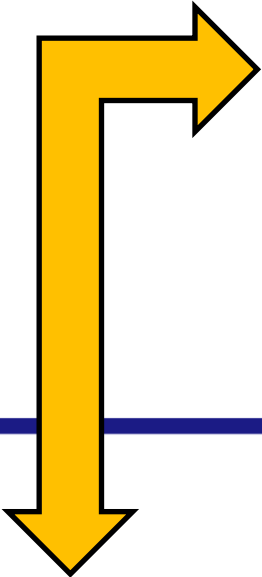
TÍTULO

AUTOR

TEMA

LIBRO

Función exterior



**Consultar
Título**

**Consultar
Autor**

**Consultar
Tema**

**Crear
Libro**

TÍTULO

AUTOR

TEMA

LIBRO

1. Definición de tipo abstracto de datos

- **Implementación de un tipo abstracto de datos en Scheme**
 - Vectores
 - Listas
 - Listas de asociación
- **Ventajas**
 - Elegancia
 - Sencillez
 - Eficiencia

Índice

1. Definición de tipo abstracto de datos
2. Representación de tipos abstractos mediante vectores
3. Representación de tipos abstractos mediante listas
4. Representación de tipos abstractos mediante listas de asociación
5. Creación de tipos abstractos de datos a partir de otros previamente definidos

ANEXO: Tipo abstracto “árbol”

2. Representación de un tipo abstracto de datos mediante “vectores”

- Tipo abstracto de datos: **libro**

- Un **libro** es **definido** mediante un **vector** compuesto por tres elementos:

- Título

- Autor

- Tema

- **Ejemplo**

#(“Misericordia” “Pérez Galdós” “Novela”)

2. Representación de un tipo abstracto de datos mediante “vectores”

- Tipo abstracto de datos: **libro**

- Función de creación

```
(define (crear-libro título autor tema)
  (vector título autor tema)
)
```

- Ejemplo

```
(define ejemplar
  (crear-libro “Misericordia” “Pérez Galdós” “Novela”)
)
```

ejemplar

→ #3(“Misericordia” “Pérez Galdós” “Novela”)

2. Representación de un tipo abstracto de datos mediante “vectores”

- Tipo abstracto de datos: **libro**
 - Funciones de acceso

```
(define (ver-título libro)  
  (vector-ref libro 0)  
)
```

```
(define (ver-autor libro)  
  (vector-ref libro 1)  
)
```

```
(define (ver-tema libro)  
  (vector-ref libro 2)  
)
```

2. Representación de un tipo abstracto de datos mediante “vectores”

- Tipo abstracto de datos: **libro**
 - Funciones de acceso: ejemplos

(*ver-título* ejemplar)

→ “*Misericordia*”

(*ver-autor* ejemplar)

→ “*Pérez Galdós*”

(*ver-tema* ejemplar)

→ “*Novela*”

2. Representación de un tipo abstracto de datos mediante “vectores”

- Tipo abstracto de datos: **libro**
 - Uso de las funciones de acceso para definir otras funciones

```
(define (consultar-libro libro)
  (display "Título: ")
  (display (ver-titulo libro))
  (newline)

  (display "Autor: ")
  (display (ver-autor libro))
  (newline)

  (display "Tema: ")
  (display (ver-tema libro))
  (newline)
)
```

2. Representación de un tipo abstracto de datos mediante “vectores”

- Tipo abstracto de datos: **libro**
 - Uso de las funciones de acceso para definir otras funciones

(consultar-libro ejemplar)

→ *Título: Misericordia
Autor: Pérez Galdós
Tema: Novela*

2. Representación de un tipo abstracto de datos mediante “vectores”

- Tipo abstracto de datos: **libro**
 - Funciones de modificación:

```
(define (cambiar-título! libro título-nuevo)
  (vector-set! libro 0 título-nuevo)
)
```

```
(define (cambiar-autor! libro autor-nuevo)
  (vector-set! libro 1 autor-nuevo)
)
```

```
(define (cambiar-tema! libro tema-nuevo)
  (vector-set! libro 2 tema-nuevo)
)
```

2. Representación de un tipo abstracto de datos mediante “vectores”

- Tipo abstracto de datos: **libro**
 - Funciones de modificación: *Ejemplos*

(*cambiar-título-libro!* ejemplar “Electra”)
ejemplar

→ #3(“Electra” “Pérez Galdós” “Novela”)

(*cambiar-autor-libro!* ejemplar “Benito Pérez Galdós”)
ejemplar

→ #3(“Electra” “Benito Pérez Galdós” “Novela”)

(*cambiar-tema-libro!* ejemplar “Teatro”)
ejemplar

→ #3(“Electra” “Benito Pérez Galdós” “Teatro”)

Índice

1. Definición de tipo abstracto de datos
2. Representación de tipos abstractos mediante vectores
3. Representación de tipos abstractos mediante listas
4. Representación de tipos abstractos mediante listas de asociación
5. Creación de tipos abstractos de datos a partir de otros previamente definidos

ANEXO: Tipo abstracto “árbol”

3. Representación de un tipo abstracto de datos mediante “listas”

- Tipo abstracto de datos: **libro**
 - Un **libro** es **definido** mediante un **lista** compuesta por tres elementos:
 - Título
 - Autor
 - Tema
 - **Ejemplo**

(“Misericordia” “Pérez Galdós” “Novela”)

3. Representación de un tipo abstracto de datos mediante “listas”

- Tipo abstracto de datos: **libro**

- Función de creación

```
(define (crear-libro título autor tema)  
  (list título autor tema)  
)
```

- Ejemplo

```
(define ejemplar  
  (crear-libro “Misericordia” “Pérez Galdós” “Novela”)  
)
```

ejemplar → (“Misericordia” “Pérez Galdós” “Novela”)

3. Representación de un tipo abstracto de datos mediante “listas”

- Tipo abstracto de datos: **libro**
 - Función de creación

```
(define (crear-libro título autor tema)  
  (list título autor tema)  
)
```

- Ejemplo

```
(define (crear-libro título autor tema)  
  (list título autor tema)  
)
```

```
ejemplar → (“Misericordia” “Pérez Galdós” “Miseria”) )
```

Observación: la función tiene el mismo nombre que en la versión con “vectores”

3. Representación de un tipo abstracto de datos mediante “listas”

- Tipo abstracto de datos: **libro**
 - Funciones de acceso

```
(define (ver-título libro)  
  (car libro)  
)
```

```
(define (ver-autor libro)  
  (cadr libro)  
)
```

```
(define (ver-tema libro)  
  (caddr libro)  
)
```

3. Representación de un tipo abstracto de datos mediante “listas”

- Tipos

- Funciones

```
(define (ver-título libro)
  (car libro)
)
```

```
(define (ver-autor libro)
  (cadr libro)
)
```

```
(define (ver-tema libro)
  (caddr libro)
)
```

Observación: las funciones tienen los mismos nombres que en la versión con “vectores”

3. Representación de un tipo abstracto de datos mediante “listas”

- Tipo abstracto de datos: **libro**
 - Funciones de acceso: ejemplos

(*ver-título* ejemplar)

→ “*Misericordia*”

(*ver-autor* ejemplar)

→ “*Pérez Galdós*”

(*ver-tema* ejemplar)

→ “*Novela*”

3. Representación de un tipo abstracto de datos mediante “listas”

- Tipo abstracto de datos: **libro**
 - Funciones de modificación: **primera** versión

```
(define (cambiar-título! libro título-nuevo)
  (set-car! libro título-nuevo)
)
```

```
(define (cambiar-autor! libro autor-nuevo)
  (set-car! (cdr libro) autor-nuevo)
)
```

```
(define (cambiar-tema! libro tema-nuevo)
  (set-car! (cddr libro) tema-nuevo)
)
```

3. Representación de un tipo abstracto de datos mediante

Observación: las funciones tienen los mismos nombres que en la versión con “vectores”

```
(define (cambiar-título! libro título-nuevo)
  (set-car! libro título-nuevo)
)
```

```
(define (cambiar-autor! libro autor-nuevo)
  (set-car! (cdr libro) autor-nuevo)
)
```

```
(define (cambiar-tema! libro tema-nuevo)
  (set-car! (cddr libro) tema-nuevo)
)
```

3. Representación de un tipo abstracto de datos mediante “listas”

- Tipo abstracto de datos: **libro**
 - Funciones de modificación: *ejemplos*

(*cambiar-título-libro!* ejemplar “Electra”)
ejemplar

→ (“Electra” “Pérez Galdós” “Novela”)

(*cambiar-autor-libro!* ejemplar “Benito Pérez Galdós”)
ejemplar

→ (“Electra” “Benito Pérez Galdós” “Novela”)

(*cambiar-tema-libro!* ejemplar “Teatro”)
ejemplar

→ (“Electra” “Benito Pérez Galdós” “Teatro”)

3. Representación de un tipo abstracto de datos mediante “listas”

- Tipo abstracto de datos: **libro**
 - Funciones de modificación: **segunda** versión

```
(define (cambiar-título libro nuevo)
  (cons nuevo (cdr libro))
)
```

```
(define (cambiar-autor libro nuevo)
  (cons (car libro)
        (cons nuevo (cddr libro)))
)
```

```
(define (cambiar-tema libro nuevo)
  (reverse
   (cons nuevo (cdr (reverse libro))))
)
```

3. Representación de un tipo abstracto de datos mediante “listas”

- Tipo abstracto de datos: **libro**
 - Funciones de modificación: ejemplos

(**set!** ejemplar (**cambiar-título-libro** ejemplar “Electra”))
ejemplar

→ (“Electra” “Pérez Galdós” “Novela”)

(**set!** ejemplar (**cambiar-autor-libro** ejemplar “Benito
Pérez Galdós”))

ejemplar

→ (“Electra” “Benito Pérez Galdós” “Novela”)

(**set!** ejemplar (**cambiar-tema-libro** ejemplar “Teatro”))
ejemplar

→ (“Electra” “Benito Pérez Galdós” “Teatro”)

3. Representación de un tipo abstracto de datos mediante

- “li
- **Observación:** las funciones crean un nuevo “libro” que se asigna al “libro” original.

(**set!** ejemplar (**cambiar-título-libro** ejemplar “Electra”) ejemplar

→ (“Electra” “Pérez Galdós” “Novela”))

(**set!** ejemplar (**cambiar-autor-libro** ejemplar “Benito Pérez Galdós”) ejemplar

ejemplar

→ (“Electra” “Benito Pérez Galdós” “Novela”))

(**set!** ejemplar (**cambiar-tema-libro** ejemplar “Teatro”) ejemplar

→ (“Electra” “Benito Pérez Galdós” “Teatro”))

Índice

1. Definición de tipo abstracto de datos
2. Representación de tipos abstractos mediante vectores
3. Representación de tipos abstractos mediante listas
4. Representación de tipos abstractos mediante listas de asociación
5. Creación de tipos abstractos de datos a partir de otros previamente definidos

ANEXO: Tipo abstracto “árbol”

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Inconveniente del uso de “vectores” o “listas”
- Definición
- Creación
- Búsqueda
- Modificación
- Tipo abstracto de datos: libro

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Inconveniente del uso de “vectores” o “listas”
- Definición
- Creación
- Búsqueda
- Modificación
- Tipo abstracto de datos: libro

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- **Inconveniente** del uso de “vectores” o “listas”
 - **Rigidez de la representación**
 - Se debe conocer el orden de los atributos
 - No se puede modificar el orden de los atributos
 - No facilita la inclusión o eliminación de atributos en el tipo abstracto de datos

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- **Inconveniente** del uso de “vectores” o “listas”
 - **Rigidez de la representación**
 - Se debe **conocer** el orden de los atributos
 - El **orden de los atributos** en la estructura de datos debe ser **conocido** previamente para poder codificar las funciones de acceso y modificación.

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- **Inconveniente** del uso de “vectores” o “listas”
 - **Rigidez de la representación**
 - **No se puede modificar el orden de los atributos**
 - Si se **cambia** la distribución interna de los campos entonces se deben **revisar** las funciones de acceso y modificación.

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- **Inconveniente** del uso de “vectores” o “listas”
 - **Rigidez de la representación**
 - **No facilita la inclusión o eliminación de atributos en el tipo abstracto de datos**
 - **La inclusión o eliminación de atributos provoca que las funciones de acceso y modificación deban ser revisadas.**

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- **Inconveniente** del uso de “vectores” o “listas”

- **Rigidez de la representación**

- **No facilita la inclusión o eliminación de atributos en el tipo abstracto de datos**

- **Por ejemplo:**

Si se introduce el campo **editorial** al principio de la tipo abstracto de datos “libro” entonces todas las funciones de acceso y modificación han de ser reformadas.

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Inconveniente del uso de “vectores” o “listas”
- Definición
- Creación
- Búsqueda
- Modificación
- Tipo abstracto de datos: libro

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Definición

- Una **lista de asociación** es una lista compuesta por sublistas de **dos** elementos

(
 (atributo₁ valor₁)
 (atributo₂ valor₂)
 ...
 (atributo_n valor_n)
)

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Definición

- Una **lista de asociación** es una lista compuesta por sublistas de **dos** elementos

- Ejemplos

((a 1) (b 2) (c 3))

(

(nombre “Ana Luque Parra”)

(localidad “Córdoba”)

(edad 12)

)

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Inconveniente del uso de “vectores” o “listas”
- Definición
- Creación
- Búsqueda
- Modificación
- Tipo abstracto de datos: libro

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Creación

- Ejemplos

```
(define lista '((a 1) (b 2) (a 3)))
```

lista

→ *((a 1) (b 2) (a 3))*

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Creación

- Ejemplos

(define persona

(list

(list 'nombre “Ana Luque Parra”)

(list 'localidad “Córdoba”)

)

)

persona

→ ((nombre “Ana Luque Parra”)

(localidad “Córdoba”)

)

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Inconveniente del uso de “vectores” o “listas”
- Definición
- Creación
- **Búsqueda**
- Modificación
- Tipo abstracto de datos: libro

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- **Búsqueda**

- *assoc*

(*assoc* objeto lista-asociación)

- Devuelve la primera sublista de la lista de asociación cuyo campo *car* es el *objeto* buscado.
- Si no existe una sublista cuyo campo *car* sea el objeto buscado entonces devuelve *#f*

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Búsqueda

- *assoc*

- Ejemplos

```
(define lista '((a 1) (b 2) (a 3)))
```

```
lista
```

```
→ ((a 1) (b 2) (a 3))
```

```
(assoc 'a lista)
```

```
→ (a 1)
```

```
(assoc 'z lista)
```

```
→ #f
```


4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- **Búsqueda**

- Acceso al valor de un atributo buscado

*(**cadr** (**assoc** atributo lista-asociación))*

- **Ejemplos**

lista → ((a 1) (b 2) (c 3))

(cadr (assoc 'b lista))

→ 2

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Inconveniente del uso de “vectores” o “listas”
- Definición
- Creación
- Búsqueda
- **Modificación**
- Tipo abstracto de datos: libro

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- **Modificación**

- **Modificación del valor de un atributo buscado**

(set-cdr!

(assoc 'atributo lista-asociación)

(list nuevo)

)

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- **Modificación**

- **Modificación del valor de un atributo buscado**

- **Ejemplo**

```
(define lista '((a 1) (b 2) (a 3)))
```

```
lista → ((a 1) (b 2) (a 3))
```

```
(set-cdr!
```

```
  (assoc 'b lista)
```

```
  (list 9)
```

```
)
```

```
lista → ((a 1) (b 9) (a 3))
```

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Inconveniente del uso de “vectores” o “listas”
- Definición
- Creación
- Búsqueda
- Modificación
- Tipo abstracto de datos: libro

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Tipo abstracto de datos: **libro**

- Representación mediante “listas de asociación”

```
(  
  (título “Misericordia”)  
  (autor “Pérez Galdós”)  
  (tema “Novela”)  
)
```

- Los nombres de los atributos o campos:

- *título*
- *autor*
- *tema*

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Tipo abstracto de datos: **libro**
 - **Ventaja:** independencia del orden de los atributos
 - La **inclusión de los nombres de los atributos o campos** permite que la representación sea **independiente del orden de los atributos**
 - Las siguientes representaciones son **equivalentes**

```
(  
  (título “Misericordia”)  
  (autor “Pérez Galdós”)  
  (tema “Novela”)  
)
```

```
(  
  (autor “Pérez Galdós”)  
  (tema “Novela”)  
  (título “Misericordia”)  
)
```

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Tipo abstracto de datos: **libro**
 - Función de creación

```
(define (crear-libro título autor tema)
  (list
    (list 'título título)
    (list 'autor autor)
    (list 'tema tema)
  )
)
```


4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Tipo abstracto de datos: **libro**
 - Función de creación

```
(define (crear-libro título autor tema)
  (list
    (list 'título título)
    (list 'autor autor)
    (list 'tema tema)
  )
)
```

Observación: la función tiene el mismo nombre que en las versiones con “vectores” o “listas”

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Tipo abstracto de datos: **libro**
 - *Función de creación: ejemplo*

(define ejemplar
 (crear-libro “Misericordia” “Pérez Galdós” “Novela”)
)

ejemplar

→ (
 (título “Misericordia”)
 (autor “Pérez Galdós”)
 (tema “Novela”)
)

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Tipo abstracto de datos: **libro**
 - Función de creación

```
(define (crear-libro autor tema título)
  (list
    (list 'autor autor)
    (list 'tema tema)
    (list 'título título)
  )
)
```

Observación: se puede definir **otra** función de creación con **otro orden** de los atributos

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Tipo abstracto de datos: **libro**
 - Funciones de acceso

```
(define (ver-título libro)
  (cadr (assoc 'título libro))
)
(define (ver-autor libro)
  (cadr (assoc 'autor libro))
)
(define (ver-tema libro)
  (cadr (assoc 'tema libro))
)
```

4. Representar un tipo abstracto de datos mediante “listas de consorcio”

- Tipo abstracto de datos
 - Funciones de acceso

```
(define (ver-título libro)
  (cadr (assoc 'título libro))
)
(define (ver-autor libro)
  (cadr (assoc 'autor libro))
)
(define (ver-tema libro)
  (cadr (assoc 'tema libro))
)
```

Observación: las tres funciones tienen la misma “estructura”

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Tipo abstracto de datos: **libro**
 - Funciones de acceso: ejemplos

(*ver-título* ejemplar) → “Misericordia”

(*ver-autor* ejemplar) → “Pérez Galdós”

(*ver-tema* ejemplar) → “Novela”

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Tipo abstracto de datos: **libro**
 - Funciones de acceso: ejemplos

(*ver-título* ejemplar) → “Misericordia”

(*ver-autor* ejemplar) → “Pérez Galdós”

(*ver-tema* ejemplar) → “Novela”

Observación: las funciones tienen los mismos nombres que en las versiones con “vectores” o “listas”

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Tipo abstracto de datos: **libro**
 - Funciones de modificación: **primera** versión

```
(define (cambiar-título libro nuevo)
  (if (equal? 'título (caar libro))
      ;;
      (cons (list 'título nuevo) (cdr libro))
      ;;
      (cons (car libro)
            (cambiar-título (cdr libro) nuevo)))
)
```


4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Tipo abstracto de datos: **libro**
 - Funciones de modificación: **primera** versión

```
(define (cambiar-autor libro nuevo)
  (if (equal? 'autor (caar libro))
      ;;
      (cons (list 'autor nuevo) (cdr libro))
      ;;
      (cons (car libro)
            (cambiar-autor (cdr libro) nuevo)))
  )
)
```

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Tipo abstracto de datos: **libro**
 - Funciones de modificación: **primera** versión

```
(define (cambiar-tema libro nuevo)
  (if (equal? 'tema (caar libro))
      ;;
      (cons (list 'tema nuevo) (cdr libro))
      ;;
      (cons (car libro)
            (cambiar-tema (cdr libro) nuevo)))
)
```

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Tipo abstracto de datos: **libro**
 - Funciones de modificación: Ejemplos

(set! ejemplar (cambiar-título-libro ejemplar “Electra”))

ejemplar

→ ((título “Electra”) (autor “Pérez Galdós”) (tema “Novela”))

(set! ejemplar (cambiar-autor-libro ejemplar “Benito Pérez Galdós”))

ejemplar

*→ ((título “Electra”) (autor “Benito Pérez Galdós”)
(tema “Novela”))*

(set! ejemplar (cambiar-tema-libro ejemplar “Teatro”))

*ejemplar → ((título “Electra”) (autor “Benito Pérez Galdós”)
(tema “Teatro”))*

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Tipo abstracto de datos: **libro**
 - Funciones de modificación: Ejemplos

(**set!** ejemplar (**cambiar-título-libro** ejemplar “Electra”))
ejemplar
→ ((título “Electra”))

(**set!** ejemplar (**cambiar-título-libro** ejemplar “Electra”))
ejemplar → ((título “Electra”) (autor “Benito Pérez Galdós”)
(tema “Novela”))

(**set!** ejemplar (**cambiar-tema-libro** ejemplar “Teatro”))
ejemplar → ((título “Electra”) (autor “Benito Pérez Galdós”)
(tema “Teatro”))

Observación: las funciones tienen los mismos nombres que en la versión con “listas”

4. Representación de un tipo abstracto de datos mediante “listas de asociación”

- Tipo abstracto de datos: **libro**
 - Funciones de modificación: **segunda** versión
(*define* (*cambiar-título!* libro nuevo)
 (*set-cdr!* (*assoc* 'título libro) (*list* nuevo))
)

(*define* (*cambiar-autor!* libro nuevo)
 (*set-cdr!* (*assoc* 'autor libro) (*list* nuevo))
)

(*define* (*cambiar-tema!* libro nuevo)
 (*set-cdr!* (*assoc* 'tema libro) (*list* nuevo))
)

4. Representación de un tipo abstracto mediante “listas de asociación”

Observación: las tres funciones tienen la misma “estructura”

```
(define (cambiar-título! libro nuevo)
  (set-cdr! (assoc 'título libro) (list nuevo))
)
```

```
(define (cambiar-autor! libro nuevo)
  (set-cdr! (assoc 'autor libro) (list nuevo))
)
```

```
(define (cambiar-tema! libro nuevo)
  (set-cdr! (assoc 'tema libro) (list nuevo))
)
```

4. Representación de un tipo abstracto de datos mediante “listas”

- Tipo abstracto de datos: **libro**
 - Funciones de modificación: ejemplos

(cambiar-título! ejemplar “Electra”)

ejemplar

→ ((título “Electra”) (autor “Pérez Galdós”) (tema “Novela”))

(cambiar-autor! ejemplar “Benito Pérez Galdós”)

ejemplar

→ ((título “Electra”) (autor “Benito Pérez Galdós”) (tema “Novela”))

(cambiar-tema! ejemplar “Teatro”)

ejemplar → ((título “Electra”) (autor “Benito Pérez Galdós”) (tema “Teatro”))

4. Representación de un tipo abstracto de datos mediante “listas”

- Tipo abstracto de datos
 - Funciones de ejemplo

(*cambiar-título!* ejemplar “Electra”) ejemplar

→ ((título “Electra”) (autor “Pérez Galdós”))

(*cambiar-autor!* ejemplar “Benito Pérez Galdós”) ejemplar

→ ((título “Electra”) (autor “Benito Pérez Galdós”) (tema “Novela”))

(*cambiar-tema!* ejemplar “Teatro”)

ejemplar → ((título “Electra”) (autor “Benito Pérez Galdós”) (tema “Teatro”))

Observación: las funciones tienen los mismos nombres que en las versiones con “vectores” o “listas”

4. Representación de un tipo abstracto de datos mediante “listas”

- **Ventajas de la “listas de asociación”**
 - **Independencia** del orden de los atributos
 - Permite la **inclusión o eliminación** de atributos

4. Representación de un tipo abstracto de datos mediante “listas”

- Ventajas de la “listas de asociación”
 - Ejemplo: inclusión de *editorial* y *fecha*

```
(define (incluir-editorial libro nombre-editorial)
  (append
    (list (list 'editorial nombre-editorial))
    libro
  )
)
```

```
(define (incluir-fecha libro dato-fecha)
  (append
    (list (list 'fecha dato-fecha))
    libro
  )
)
```

4. Representación de un tipo abstracto de datos mediante “listas”

- Ventajas de la “listas de asociación”
 - Ejemplo: inclusión de *editorial* y *fecha*

(*define ejemplar*
 (*crear-libro* “Nazarín” “Benito Pérez Galdós” “Novela”)
)

(*set!* *ejemplar* (*incluir-editorial* *ejemplar* “Arco iris”))
(*set!* *ejemplar* (*incluir-fecha* *ejemplar* 1992))

ejemplar → ((*fecha* 1992)
 (*editorial* “Arco iris”)
 (*título* “Nazarín”)
 (*autor* “Benito Pérez Galdós”)
 (*tema* “Novela”)
)

4. Representación de un tipo abstracto de datos mediante “listas”

- Ventajas de la “listas de asociación”
 - Nuevas funciones de acceso a *editorial* y *fecha*

```
(define (ver-editorial libro)
  (cadr (assoc 'editorial libro))
)
```

```
(define (ver-fecha libro)
  (cadr (assoc 'fecha libro))
)
```

4. Representación de un tipo abstracto de datos mediante “listas”

- **Ventajas de la “listas de asociación”**
 - Nueva función de consulta del libro

```
(define (consultar-libro-nuevo libro)
  (display "Título: ")
  (display (ver-titulo libro))
  (newline)
  (display "Autor: ")
  (display (ver-autor libro))
  (newline)
  (display "Tema: ")
  (display (ver-tema libro))
  (newline)
  (display "Editorial: ")
  (display (ver-editorial libro))
  (newline)
  (display "Fecha: ")
  (display (ver-fecha libro))
  (newline)
)
```

4. Representación de un tipo abstracto de datos mediante “listas”

- Ventajas de la “listas de asociación”
 - Nueva función de consulta del libro

(consultar-libro-nuevo ejemplar) →

Título: Nazarín

Autor: Benito Pérez Galdós

Tema: Novela

Editorial: Arco iris

Fecha: 1992

4. Representación de un tipo abstracto de datos mediante “listas”

- Ventajas de la “listas de asociación”
 - Nueva función para consultar cualquier atributo del libro

```
(define (consultar-atributo nombre libro)  
  (cadr (assoc nombre libro))  
)
```

- Ejemplo

```
(consultar-atributo 'fecha ejemplar) → 1992
```

Índice

1. Definición de tipo abstracto de datos
2. Representación de tipos abstractos mediante vectores
3. Representación de tipos abstractos mediante listas
4. Representación de tipos abstractos mediante listas de asociación
5. Creación de tipos abstractos de datos a partir de otros previamente definidos

ANEXO: Tipo abstracto “árbol”

5. Creación de tipos abstractos de datos a partir de otros previamente definidos

- **Ejemplo: biblioteca**

- Se va a utilizar el tipo abstracto “libro” representado con “**listas de asociación**”.
- También se pueden utilizar las representaciones basadas en “vectores” o en “listas”.

5. Creación de tipos abstractos de datos a partir de otros previamente definidos

- Ejemplo: **biblioteca**

(
((título “Misericordia”) (autor “Pérez Galdós”) (tema “Novela”))
((título “El sí de las niñas”) (autor “Fernández Moratín”) (tema Teatro”))
((título “Pepita Jiménez”) (autor “Valera”) (tema “Novela”))
((título “Don Juan Tenorio”) (autor “Zorrilla”) (tema “Teatro”))
)

5. Creación de tipos abstractos de datos a partir de otros previamente definidos

- Ejemplo: **biblioteca**

- Funciones adicionales:

- Crear-biblioteca-vacía
- Biblioteca-vacía?
- Insertar-libro
- Primer-libro
- Resto-de-libros
- Lista-autores (o cualquier otro atributo)
- Buscar-libros-por-tema (o cualquier otro atributo)
- Etc.

5. Creación de tipos abstractos de datos a partir de otros previamente definidos

- Ejemplo: **biblioteca**

- Funciones adicionales

- Crear una biblioteca vacía

```
(define (crear-biblioteca-vacia)  
  (list)  
)
```

- Ejemplo

```
(define Biblioteca-PD (crear-biblioteca-vacia))
```

5. Creación de tipos abstractos de datos a partir de otros previamente definidos

- Ejemplo: **biblioteca**

- Funciones adicionales

- Predicado que comprueba si una **biblioteca** está vacía

```
(define (biblioteca-vacia? biblioteca)  
  (equal? biblioteca '() )  
)
```

5. Creación de tipos abstractos de datos a partir de otros previamente definidos

- Ejemplo: **biblioteca**

- Funciones adicionales

- Insertar un libro en la biblioteca

```
(define (insertar-libro biblioteca libro)  
  (cons libro biblioteca)  
)
```

5. Creación de tipos abstractos de datos a partir de otros previamente definidos

- Ejemplo: **biblioteca**
 - Funciones adicionales: **insertar un libro**
 - Ejemplos: primera parte

```
(set! Biblioteca-PD
  (insertar-libro Biblioteca-PD
    (crear-libro "Nazarín" "Benito Pérez Galdós" "Novela")
  )
)
```

```
(set! Biblioteca-PD
  (insertar-libro Biblioteca-PD
    (crear-libro "Pepita Jiménez" "Juan Valera" "Novela")
  )
)
```

5. Creación de tipos abstractos de datos a partir de otros previamente definidos

- Ejemplo: **biblioteca**
 - Funciones adicionales: insertar un libro
 - Ejemplos: segunda parte

```
(set! Biblioteca-PD
  (insertar-libro Biblioteca-PD
    (crear-libro "Juan Tenorio" "José Zorrilla" "Teatro")
  )
)
```

```
(set! Biblioteca-PD
  (insertar-libro Biblioteca-PD
    (crear-libro "El sí de las niñas" "Leandro Fernández Moratín"
      "Teatro")
  )
)
```


5. Creación de tipos abstractos de datos a partir de otros previamente definidos

- Ejemplo: **biblioteca**

- Funciones adicionales

- Primer libro y resto de libros

```
(define (primer-libro biblioteca)
```

```
  (car biblioteca)
```

```
)
```

```
(define (resto-de-libros biblioteca)
```

```
  (cdr biblioteca)
```

```
)
```

5. Creación de tipos abstractos de datos a partir de otros previamente definidos

- Ejemplo: **biblioteca**

- Funciones adicionales:

- **Consultar** los libros de la biblioteca

(define (consultar-libros-biblioteca biblioteca)

(cond

*((biblioteca-vacia? biblioteca) (**newline**))*

(else

(consultar-libro (primer-libro biblioteca))

(consultar-libros-biblioteca (resto-de-libros biblioteca))

)

)

)

5. Creación de tipos abstractos de datos a partir de otros previamente definidos

- Ejemplo: **biblioteca**

- Funciones adicionales:

- **Consultar** los libros de la biblioteca
(*consultar-libros-biblioteca Biblioteca-PD*)

➔ *Título: El sí de las niñas*

Autor: Leandro Fernández Moratín

Tema: Teatro

...

Título: Nazarín

Autor: Benito Pérez Galdós

Tema: Novela

5. Creación de tipos abstractos de datos a partir de otros previamente definidos

- Ejemplo: **biblioteca**

- Funciones adicionales:

- Lista de autores de los libros de la biblioteca

```
(define (lista-autores biblioteca)
```

```
(if (biblioteca-vacia? biblioteca)()
```

```
(cons (ver-autor (primer-libro biblioteca))
```

```
(lista-autores (resto-de-libros biblioteca))
```

```
)
```

```
)
```

```
)
```

5. Creación de tipos abstractos de datos a partir de otros previamente definidos

- Ejemplo: **biblioteca**

- Funciones adicionales:

- **Lista de autores de los libros de la biblioteca**
(*lista-autores Biblioteca-PD*)

→ (

"Leandro Fernández Moratín"

"José Zorrilla"

"Juan Valera"

"Benito Pérez Galdós"

)

5. Creación de tipos abstractos de datos a partir de otros previamente definidos

- Ejemplo: **biblioteca**

- Funciones adicionales:

- Escribir la lista de autores de los libros

```
(define (escribir-lista-autores biblioteca)
```

```
  (cond ((biblioteca-vacia? biblioteca) (newline))
```

```
        (else (display (ver-autor (primer-libro biblioteca))
```

```
              (newline))
```

```
        (escribir-lista-autores (resto-de-libros biblioteca))
```

```
  )
```

```
)
```

```
)
```

5. Creación de tipos abstractos de datos a partir de otros previamente definidos

- Ejemplo: **biblioteca**

- Funciones adicionales:

- **Escribir la lista de autores de los libros**
(*escribir-lista-autores Biblioteca-PD*)

→ *Leandro Fernández Moratín*

José Zorrilla

Juan Valera

Benito Pérez Galdós

5. Creación de tipos abstractos de datos a partir de otros previamente definidos

- Ejemplo: **biblioteca**

- Funciones adicionales: **buscar libros por tema**

(*define* (*buscar-libros-por-tema* biblioteca clase)

(*cond*

((*biblioteca-vacia?* biblioteca) '())

((*equal?* (*ver-tema* (*primer-libro* biblioteca)) clase)

(*cons* (*primer-libro* biblioteca)

(*buscar-libros-por-tema* (*resto-de-libros* biblioteca) clase)

)

)

(*else* (*buscar-libros-por-tema* (*resto-de-libros* biblioteca) clase))

)

)

5. Creación de tipos abstractos de datos a partir de otros previamente definidos

- Ejemplo: **biblioteca**

- Funciones adicionales

- Buscar libros por tema: ejemplo

(consultar (*buscar-libros-por-tema* Biblioteca-PD “Teatro”))

→ *Título: El sí de las niñas*

Autor: Leandro Fernández Moratín

Tema: Teatro

Título: Juan Tenorio

Autor: José Zorrilla

Tema: Teatro

5. Creación de tipos abstractos de datos a partir de otros previamente definidos

- Ejemplo: **biblioteca**

- Las demás funciones de búsqueda de autores, título, etc., son semejantes.
- Otras funciones
 - Funciones que cuenten el número de libros que posean alguna característica.
 - Funciones que muestren información
 - Etc.

Índice

1. Definición de tipo abstracto de datos
2. Representación de tipos abstractos mediante vectores
3. Representación de tipos abstractos mediante listas
4. Representación de tipos abstractos mediante listas de asociación
5. Creación de tipos abstractos de datos a partir de otros previamente definidos

ANEXO: Tipo abstracto “árbol”

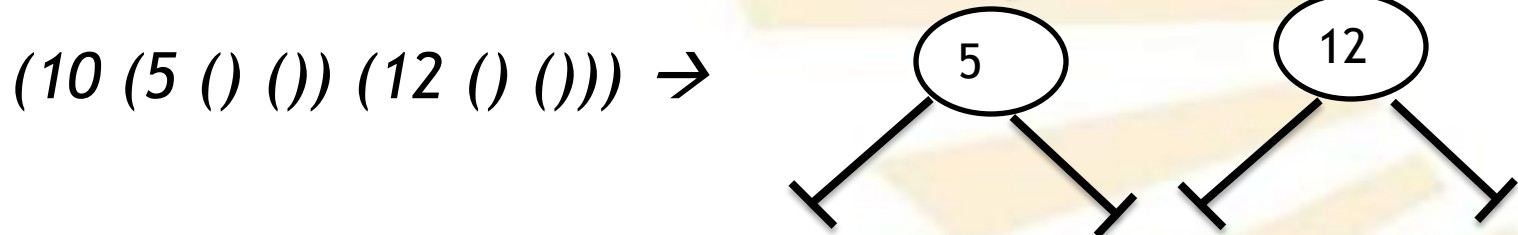
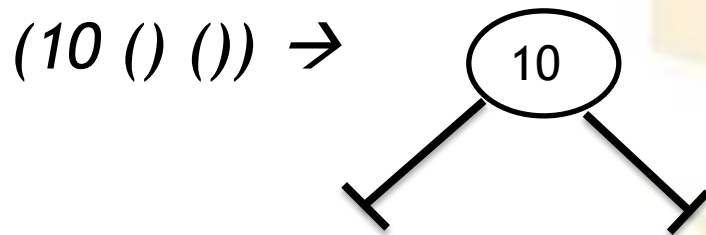
ANEXO: Tipo abstracto “árbol”

- Estructura de la información

(raíz árbol-hijo-izquierdo árbol-hijo-derecho)

- Ejemplos

$() \rightarrow$ árbol vacío



ANEXO: Tipo abstracto “árbol”

- **Función de creación**
 - *Crear-árbol*
- **Funciones de acceso**
 - *Raíz-árbol*
 - *Hijo-izquierdo-árbol*
 - *Hijo-derecho-árbol*
- **Funciones de modificación**
 - *Insertar*

ANEXO: Tipo abstracto “árbol”

- Otras funciones de acceso
 - *Preorden*
 - *Postorden*
 - *En-orden*
 - *Pertenece?*

ANEXO: Tipo abstracto “árbol”

- Función de creación

- *Crear-árbol*

(define (crear-árbol raíz hijo-izquierdo hijo-derecho)
(list raíz hijo-izquierdo hijo-derecho)
)

- Ejemplo:

(define x (crear-árbol 10 '() '()))

ANEXO: Tipo abstracto “árbol”

- Funciones de acceso

- *Árbol vacío*

(*define* (*árbol-vacío?* árbol)

(*null?* árbol)

)

ANEXO: Tipo abstracto “árbol”

- Funciones de acceso
 - *Raíz del árbol*

(define (raíz-árbol árbol)

(if (árbol-vacío? árbol)

'()

(car árbol)

)

)

ANEXO: Tipo abstracto “árbol”

- Funciones de acceso

- Hijo izquierdo e hijo derecho

```
(define (hijo-izquierdo-árbol árbol)
```

```
(if (árbol-vacío? árbol)()
```

```
(cadr árbol)
```

```
)
```

```
)
```

```
(define (hijo-derecho-árbol árbol)
```

```
(if (árbol-vacío? árbol)
```

```
'()
```

```
(caddr árbol)
```

```
)
```

```
)
```

ANEXO: Tipo abstracto “árbol”

- **Función de modificación**
 - Insertar

(define (insertar x árbol)

(if (árbol-vacío? árbol) (crear-árbol x () ())

(let ((r (raiz-árbol árbol)))

(cond

((> r x) (crear-árbol r

(insertar x (hijo-izquierdo-árbol árbol))

(hijo-derecho-árbol árbol)

)

)

((< r x) (crear-árbol r

(hijo-izquierdo-árbol árbol)

(insertar x (hijo-derecho-árbol árbol))

)

)

(else árbol)

)

)

)

)

ANEXO: Tipo abstracto “árbol”

- Función de modificación

- Insertar

- Ejemplos

```
(define árbol (crear-árbol 10 '() '()))
```

```
(set! árbol (insertar 9 árbol))
```

árbol

```
→ (10 (9 () ()) ())
```

```
(set! x (insertar 19 x))
```

árbol

```
→ (10 (9 () ()) (19 () ()))
```

ANEXO: Tipo abstracto “árbol”

- Otras funciones de acceso

- Recorrido en “preorden”

- (*define* (*preorden* arbol)

- (*cond*

- ((*not* (*arbol-vacio?* arbol))

- (*display* (*raiz-arbol* arbol))

- (*display* " ")

- (*preorden* (*hijo-izquierdo-arbol* arbol))

- (*preorden* (*hijo-derecho-arbol* arbol))

-)

-)

-)

ANEXO: Tipo abstracto “árbol”

- Otras funciones de acceso

- Recorrido en “postorden”

- (*define* (*postorden* arbol)

- (*cond*

- ((*not* (*arbol-vacio?* arbol))

- (*postorden* (*hijo-izquierdo-arbol* arbol))

- (*postorden* (*hijo-derecho-arbol* arbol))

- (*display* (*raiz-arbol* arbol))

- (*display* " ")

-)

-)

-)

ANEXO: Tipo abstracto “árbol”

- Otras funciones de acceso

- Recorrido en “en-orden”

- (*define* (*en-orden* arbol)

- (*cond*

- ((*not* (*arbol-vacio?* arbol))

- (*en-orden* (*hijo-izquierdo-arbol* arbol))

- (*display* (*raiz-arbol* arbol))

- (*display* " ")

- (*en-orden* (*hijo-derecho-arbol* arbol))

-)

-)

-)

ANEXO: Tipo abstracto “árbol”

- Otras funciones de acceso

- Pertenencia a un árbol

(*define* (*pertenece?* x árbol)

(*if* (*árbol-vacío?* árbol) **#f**

(*let* ((r (*raiz-árbol* árbol)))

(*cond*

((> r x) (*pertenece?* x (*hijo-izquierdo-árbol* árbol)))

((< r x) (*pertenece?* x (*hijo-derecho-árbol* árbol)))

(*else* **#t**)

)

)

)

)



UNIVERSIDAD DE CÓRDOBA

ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA

DEPARTAMENTO DE
INFORMÁTICA Y ANÁLISIS NUMÉRICO



PROGRAMACIÓN DECLARATIVA

INGENIERÍA INFORMÁTICA

CUARTO CURSO

PRIMER CUATRIMESTRE

Tema 6.- Abstracción de Datos

