

Introducción a la Computación (Matemática)

Primer Cuatrimestre de 2018

Tipos Abstractos de Datos

Tipo de Datos

- **Tipo de datos:** conjunto de valores que sirven de dominio a ciertas operaciones.

Tipo de Datos

- **Tipo de datos:** conjunto de valores que sirven de dominio a ciertas operaciones.
- **Limitaciones:** Los tipos de datos de los usuarios no tienen más restricciones que la que imponen los tipos básicos.

Tipo de Datos

- **Tipo de datos:** conjunto de valores que sirven de dominio a ciertas operaciones.
- **Limitaciones:** Los tipos de datos de los usuarios no tienen más restricciones que la que imponen los tipos básicos.
- **TAD's:** A mediados de los '70 se introduce el concepto de tipos abstractos de datos, que caracteriza un tipo de datos que no sólo tiene un conjunto de valores y las operaciones que se pueden aplicar sino también un conjunto de propiedades que determinan unívocamente su comportamiento.

TAD's

- ¿Por que abstracto?

TAD's

- ¿Por que abstracto?

TAD's

- ¿Por que abstracto?
Encapsulamiento=Especificación+Implementación.
- Especificación: Propiedades del TAD. Precisas, general, y legible.

TAD's

- ¿Por que abstracto?
Encapsulamiento=Especificación+Implementación.
- Especificación: Propiedades del TAD. Precisas, general, y legible.
- Implementación: Representación de los valores y codificación de sus operaciones. Estructurada, eficiente y entendible.

Ejemplo: Agenda

Queremos programar una agenda de contactos.

De cada persona nos interesa guardar:

- Nombre
- Teléfono
- Dirección

¿Cómo representamos los datos de las personas?

Ejemplo: Agenda

Queremos programar una agenda de contactos.

De cada persona nos interesa guardar:

- Nombre
- Teléfono
- Dirección

¿Cómo representamos los datos de las personas?

- Nombres: lista de strings.
- Teléfonos: lista de strings.
- Direcciones: lista de strings.

Tales que la i -ésima posición de los 3 listas correspondan a la misma persona.

Ejemplo: Agenda

Queremos programar una agenda de contactos.

De cada persona nos interesa guardar:

- Nombre
- Teléfono
- Dirección

¿Cómo representamos los datos de las personas?

- Nombres: lista de strings.
- Teléfonos: lista de strings.
- Direcciones: lista de strings.

Tales que la i -ésima posición de los 3 listas correspondan a la misma persona. Esta representación *funciona* (cumple el objetivo), pero tiene serios problemas...

Ejemplo: Agenda

Mucho mejor sería contar con un **tipo Persona**, que **encapsule** todos los datos relevantes a una persona para nuestra agenda.

Ejemplo: Agenda

Mucho mejor sería contar con un **tipo Persona**, que **encapsule** todos los datos relevantes a una persona para nuestra agenda.

Tipo Abstracto de Datos (TAD) Persona

Ejemplo: Agenda

Mucho mejor sería contar con un **tipo Persona**, que **encapsule** todos los datos relevantes a una persona para nuestra agenda.

Tipo Abstracto de Datos (TAD) Persona

Operaciones:

- **CrearPersona**(*nom, tel, dir*) → **Persona**: Crea una persona nueva con el nombre, teléfono y dirección especificados.

Ejemplo: Agenda

Mucho mejor sería contar con un **tipo Persona**, que **encapsule** todos los datos relevantes a una persona para nuestra agenda.

Tipo Abstracto de Datos (TAD) Persona

Operaciones:

- $\text{CrearPersona}(\text{nom}, \text{tel}, \text{dir}) \rightarrow \text{Persona}$: Crea una persona nueva con el nombre, teléfono y dirección especificados.
- $\text{Nombre}(p) \rightarrow \text{String}$: Devuelve el nombre de la persona p .
- $\text{Teléfono}(p) \rightarrow \text{String}$: Devuelve el teléfono de la persona p .
- $\text{Dirección}(p) \rightarrow \text{String}$: Devuelve la dirección de la persona p .

Así, podemos representar la agenda con una lista de Personas.

Ejemplo: Agenda

Mucho mejor sería contar con un **tipo Persona**, que **encapsule** todos los datos relevantes a una persona para nuestra agenda.

Tipo Abstracto de Datos (TAD) Persona

Operaciones:

- $\text{CrearPersona}(\text{nom}, \text{tel}, \text{dir}) \rightarrow \text{Persona}$: Crea una persona nueva con el nombre, teléfono y dirección especificados.
- $\text{Nombre}(p) \rightarrow \text{String}$: Devuelve el nombre de la persona p .
- $\text{Teléfono}(p) \rightarrow \text{String}$: Devuelve el teléfono de la persona p .
- $\text{Dirección}(p) \rightarrow \text{String}$: Devuelve la dirección de la persona p .

Así, podemos representar la agenda con una lista de Personas.

¿Cómo se implementa el TAD?

Ejemplo: Agenda

Mucho mejor sería contar con un **tipo Persona**, que **encapsule** todos los datos relevantes a una persona para nuestra agenda.

Tipo Abstracto de Datos (TAD) Persona

Operaciones:

- $\text{CrearPersona}(\text{nom}, \text{tel}, \text{dir}) \rightarrow \text{Persona}$: Crea una persona nueva con el nombre, teléfono y dirección especificados.
- $\text{Nombre}(p) \rightarrow \text{String}$: Devuelve el nombre de la persona p .
- $\text{Teléfono}(p) \rightarrow \text{String}$: Devuelve el teléfono de la persona p .
- $\text{Dirección}(p) \rightarrow \text{String}$: Devuelve la dirección de la persona p .

Así, podemos representar la agenda con una lista de Personas.

¿Cómo se implementa el TAD? Como **usuarios**, **no nos importa**.

Otro ejemplo: Contar días entre 2 fechas

Otro ejemplo: Contar días entre 2 fechas

```
# Dice si el año a es bisiesto.
def bisiesto(a):
    return a%4==0 and (a%100!=0 or a%400==0)

# Devuelve la cantidad de dias del mes m, año a.
def diasEnMes(m, a):
    r=0
    if m==1 or m==3 or m==5 or m==7 or \
        m==8 or m==10 or m==12:
        r = 31
    if m==4 or m==6 or m==9 or m==11:
        r = 30
    if m==2:
        if bisiesto(a):
            r = 29
        else:
            r = 28
    return r

# Cuenta los dias entre los meses m1 y m2
# inclusive, en el año a.
def diasEntreMeses(m1, m2, a):
    r = 0
    m = m1
    while m <= m2:
        r = r + diasEnMes(m, a)
        m = m + 1
    return r
```

```
# Cuenta los dias entre el d1/m1/a1 y el d2/m2/a2.
def contarDias(d1, m1, a1, d2, m2, a2):
    r = 0

    if a1 == a2 and m1 == m2:
        r = d2 - d1

    if a1 == a2 and m1 < m2:
        r = diasEnMes(m1, a1) - d1
        r = r + diasEntreMeses(m1+1, m2-1, a1)
        r = r + d2

    if a1 < a2:
        r = diasEnMes(m1, a1) - d1
        r = r + diasEntreMeses(m1+1, 12, a1)
        a = a1 + 1
        while a < a2:
            r = r + 365
            if bisiesto(a):
                r = r + 1
            a = a + 1
        r = r + diasEntreMeses(1, m2-1, a2)
        r = r + d2

    return r
```

Otro ejemplo: Contar días entre 2 fechas

Imaginemos que contamos con un **tipo Fecha** que nos ofrece estas operaciones (entre otras):

- $\text{CrearFecha}(d, m, a) \rightarrow \text{Fecha}$
- $\text{FechaSiguiente}(f) \rightarrow \text{Fecha}$: Devuelve la fecha siguiente a la fecha dada. (Ej: al 31/12/1999 le sigue el 1/1/2000.)
- $\text{Menor}(f_1, f_2) \rightarrow \mathbb{B}$: Devuelve TRUE si la fecha f_1 es anterior que la fecha f_2 , y FALSE en caso contrario.

Otro ejemplo: Contar días entre 2 fechas

Imaginemos que contamos con un **tipo Fecha** que nos ofrece estas operaciones (entre otras):

- $\text{CrearFecha}(d, m, a) \rightarrow \text{Fecha}$
- $\text{FechaSiguiente}(f) \rightarrow \text{Fecha}$: Devuelve la fecha siguiente a la fecha dada. (Ej: al 31/12/1999 le sigue el 1/1/2000.)
- $\text{Menor}(f_1, f_2) \rightarrow \mathbb{B}$: Devuelve TRUE si la fecha f_1 es anterior que la fecha f_2 , y FALSE en caso contrario.

Usando esto, un algoritmo para contar días podría ser:

$\text{ContarDías}(f_1, f_2) \rightarrow \mathbb{Z}$:

Otro ejemplo: Contar días entre 2 fechas

Imaginemos que contamos con un **tipo Fecha** que nos ofrece estas operaciones (entre otras):

- $\text{CrearFecha}(d, m, a) \rightarrow \text{Fecha}$
- $\text{FechaSiguiente}(f) \rightarrow \text{Fecha}$: Devuelve la fecha siguiente a la fecha dada. (Ej: al 31/12/1999 le sigue el 1/1/2000.)
- $\text{Menor}(f_1, f_2) \rightarrow \mathbb{B}$: Devuelve TRUE si la fecha f_1 es anterior que la fecha f_2 , y FALSE en caso contrario.

Usando esto, un algoritmo para contar días podría ser:

$\text{ContarDías}(f_1, f_2) \rightarrow \mathbb{Z}$:

```
RV  $\leftarrow$  0
while (Menor( $f_1, f_2$ )) {
    RV  $\leftarrow$  RV + 1
     $f_1 \leftarrow \text{FechaSiguiente}(f_1)$ 
}
```

Modularidad y encapsulamiento

La clave está en **encapsular** los datos y sus operaciones.

Modularidad y encapsulamiento

La clave está en **encapsular** los datos y sus operaciones.

Al programar, definimos funciones (ej: `Primo(n)`) para generar código más simple y claro (código *modular*).

Encapsulamos un algoritmo para poder reusarlo muchas veces.

Modularidad y encapsulamiento

La clave está en **encapsular** los datos y sus operaciones.

Al programar, definimos funciones (ej: `Primo(n)`) para generar código más simple y claro (código *modular*).

Encapsulamos un algoritmo para poder reusarlo muchas veces.

Ahora generalizamos este concepto, y encapsulamos datos (ej: personas, fechas) y sus operaciones (ej: `FechaSiguiente(f)`) en **Tipos Abstractos de Datos (TAD)**.

Modularidad y encapsulamiento

La clave está en **encapsular** los datos y sus operaciones.

Al programar, definimos funciones (ej: `Primo(n)`) para generar código más simple y claro (código *modular*).

Encapsulamos un algoritmo para poder reusarlo muchas veces.

Ahora generalizamos este concepto, y encapsulamos datos (ej: personas, fechas) y sus operaciones (ej: `FechaSiguiente(f)`) en **Tipos Abstractos de Datos (TAD)**.

Para **usar** un TAD, el programador sólo necesita conocer el nombre del TAD y la especificación de sus operaciones (y quizá sus órdenes de complejidad).

Un TAD puede estar implementado de muchas formas distintas. Esto debe ser **transparente** para el usuario.

Partes de un Tipo Abstracto de Datos

- Parte **pública**: Disponible para el usuario externo.
 - Nombre y tipos paramétricos (ej: Fecha, Lista(ELEM)).
 - Operaciones, sus especificaciones y posiblemente sus órdenes de complejidad temporal.

Partes de un Tipo Abstracto de Datos

- Parte **pública**: Disponible para el usuario externo.
 - Nombre y tipos paramétricos (ej: Fecha, Lista(ELEM)).
 - Operaciones, sus especificaciones y posiblemente sus órdenes de complejidad temporal.
- Parte **privada**: Sólo accesible desde dentro del TAD. ¡El usuario externo **nunca** debe ver ni meterse en esto!
 - Próxima clase...

TAD Fecha

Operaciones públicas (para $f, f_1, f_2 : \text{Fecha}$; $d, m, a : \mathbb{Z}$):

- $\text{CrearFecha}(d, m, a) \rightarrow \text{Fecha}$
- $\text{Día}(f) \rightarrow \mathbb{Z}$
- $\text{Mes}(f) \rightarrow \mathbb{Z}$
- $\text{Año}(f) \rightarrow \mathbb{Z}$
- $\text{Menor}(f_1, f_2) \rightarrow \mathbb{B}$
- $\text{FechaSiguiente}(f) \rightarrow \text{Fecha}$

TAD Fecha

Operaciones públicas (para $f, f_1, f_2 : \text{Fecha}$; $d, m, a : \mathbb{Z}$):

- $\text{CrearFecha}(d, m, a) \rightarrow \text{Fecha}$
- $\text{Día}(f) \rightarrow \mathbb{Z}$
- $\text{Mes}(f) \rightarrow \mathbb{Z}$
- $\text{Año}(f) \rightarrow \mathbb{Z}$
- $\text{Menor}(f_1, f_2) \rightarrow \mathbb{B}$
- $\text{FechaSiguiente}(f) \rightarrow \text{Fecha}$

Como usuarios del TAD, podemos programar algo como:

$\text{ContarDías}(f_1, f_2) \rightarrow \mathbb{Z}$:

```
RV  $\leftarrow$  0
while (Menor( $f_1, f_2$ )) {
    RV  $\leftarrow$  RV + 1
     $f_1 \leftarrow \text{FechaSiguiente}(f_1)$ 
}
```

¿Cómo se implementa?

TAD Fecha

Operaciones públicas (para $f, f_1, f_2 : \text{Fecha}$; $d, m, a : \mathbb{Z}$):

- $\text{CrearFecha}(d, m, a) \rightarrow \text{Fecha}$
- $\text{Día}(f) \rightarrow \mathbb{Z}$
- $\text{Mes}(f) \rightarrow \mathbb{Z}$
- $\text{Año}(f) \rightarrow \mathbb{Z}$
- $\text{Menor}(f_1, f_2) \rightarrow \mathbb{B}$
- $\text{FechaSiguiente}(f) \rightarrow \text{Fecha}$

Como usuarios del TAD, podemos programar algo como:

$\text{ContarDías}(f_1, f_2) \rightarrow \mathbb{Z}$:

```
RV  $\leftarrow$  0
while (Menor( $f_1, f_2$ )) {
    RV  $\leftarrow$  RV + 1
     $f_1 \leftarrow$  FechaSiguiente( $f_1$ )
}
```

¿Cómo se implementa? Como usuarios, no nos importa.

Arreglos y Listas

En C++ teníamos el tipo array.

- Ejemplo:

```
array<int, 2> a;  
a[0] = 10;  
a[1] = a[0] + 1;  
cout << a.size();
```


Arreglos y Listas

En C++ teníamos el tipo array.

- Ejemplo:

```
array<int, 2> a;  
a[0] = 10;  
a[1] = a[0] + 1;  
cout << a.size();
```

- Los arreglos son muy eficientes.
- Pero proveen muy poca funcionalidad, y no pueden cambiar de tamaño.

Arreglos y Listas

En C++ teníamos el tipo array.

- Ejemplo:

```
array<int, 2> a;  
a[0] = 10;  
a[1] = a[0] + 1;  
cout << a.size();
```

- Los **arreglos** son muy eficientes.
- Pero proveen muy poca funcionalidad, y no pueden cambiar de tamaño.

Las **listas** (p.ej., en Python) son bastante más útiles. Proveen las operaciones de arreglos, y muchas otras.

TAD Lista(ELEM)

TAD Lista(ELEM)

Operaciones:

- `CrearLista()` \rightarrow `Lista(ELEM)`: Crea una lista vacía.

TAD Lista(ELEM)

Operaciones:

- $\text{CrearLista}() \rightarrow \text{Lista}(\text{ELEM})$: Crea una lista vacía.
- $L.\text{Agregar}(x)$: Inserta el elemento x al final de L .

donde $L : \text{Lista}(\text{ELEM})$; $i : \mathbb{Z}$; $x : \text{ELEM}$ (entero, char, etc.).

TAD Lista(ELEM)

Operaciones:

- $\text{CrearLista}() \rightarrow \text{Lista}(\text{ELEM})$: Crea una lista vacía.
- $L.\text{Agregar}(x)$: Inserta el elemento x al final de L .
- $L.\text{Longitud}() \rightarrow \mathbb{Z}$: Devuelve la cantidad de elementos de L .

donde $L : \text{Lista}(\text{ELEM})$; $i : \mathbb{Z}$; $x : \text{ELEM}$ (entero, char, etc.).

TAD Lista(ELEM)

Operaciones:

- $\text{CrearLista}() \rightarrow \text{Lista}(\text{ELEM})$: Crea una lista vacía.
- $L.\text{Agregar}(x)$: Inserta el elemento x al final de L .
- $L.\text{Longitud}() \rightarrow \mathbb{Z}$: Devuelve la cantidad de elementos de L .
- $L.\text{lésimo}(i) \rightarrow \text{ELEM}$: Devuelve el i -ésimo elemento de L .
Precondición: $0 \leq i < L.\text{Longitud}()$.

donde $L : \text{Lista}(\text{ELEM})$; $i : \mathbb{Z}$; $x : \text{ELEM}$ (entero, char, etc.).

TAD Lista(ELEM)

Operaciones:

- $\text{CrearLista}() \rightarrow \text{Lista}(\text{ELEM})$: Crea una lista vacía.
- $L.\text{Agregar}(x)$: Inserta el elemento x al final de L .
- $L.\text{Longitud}() \rightarrow \mathbb{Z}$: Devuelve la cantidad de elementos de L .
- $L.\text{lésimo}(i) \rightarrow \text{ELEM}$: Devuelve el i -ésimo elemento de L .
Precondición: $0 \leq i < L.\text{Longitud}()$.
- $L.\text{Cantidad}(x) \rightarrow \mathbb{Z}$: Devuelve la cantidad de veces que aparece el elemento x en L .

donde $L : \text{Lista}(\text{ELEM})$; $i : \mathbb{Z}$; $x : \text{ELEM}$ (entero, char, etc.).

TAD Lista(ELEM)

Operaciones (cont.):

- $L.\text{Insertar}(i, x)$: Inserta el elemento x en la posición i de L , pasando los elementos de la posición i y siguientes a la posición inmediata superior. Pre: $0 \leq i \leq L.\text{Longitud}()$.
(Si $L = a_0, \dots, a_{n-1}$, se convierte en $L = a_0, \dots, a_{i-1}, x, a_i, \dots, a_{n-1}$.)

donde $L : \text{Lista}(\text{ELEM})$; $i : \mathbb{Z}$; $x : \text{ELEM}$ (entero, char, etc.).

TAD Lista(ELEM)

Operaciones (cont.):

- $L.\text{Insertar}(i, x)$: Inserta el elemento x en la posición i de L , pasando los elementos de la posición i y siguientes a la posición inmediata superior. Pre: $0 \leq i \leq L.\text{Longitud}()$.
(Si $L = a_0, \dots, a_{n-1}$, se convierte en $L = a_0, \dots, a_{i-1}, x, a_i, \dots, a_{n-1}$.)
- $L.\text{Indice}(x) \rightarrow \mathbb{Z}$: Devuelve el índice ($0 \dots L.\text{Longitud}() - 1$) de la primera aparición de x en L . Pre: $L.\text{Cantidad}(x) > 0$.

donde $L : \text{Lista}(\text{ELEM})$; $i : \mathbb{Z}$; $x : \text{ELEM}$ (entero, char, etc.).

TAD Lista(ELEM)

Operaciones (cont.):

- $L.\text{Insertar}(i, x)$: Inserta el elemento x en la posición i de L , pasando los elementos de la posición i y siguientes a la posición inmediata superior. Pre: $0 \leq i \leq L.\text{Longitud}()$.
(Si $L = a_0, \dots, a_{n-1}$, se convierte en $L = a_0, \dots, a_{i-1}, x, a_i, \dots, a_{n-1}$.)
- $L.\text{Indice}(x) \rightarrow \mathbb{Z}$: Devuelve el índice ($0 \dots L.\text{Longitud}() - 1$) de la primera aparición de x en L . Pre: $L.\text{Cantidad}(x) > 0$.
- $L.\text{Borrarlésimo}(i)$: Borra el i -ésimo elemento de L
Precondición: $0 \leq i < L.\text{Longitud}()$.
(Si $L = a_0, \dots, a_{n-1}$, se convierte en $L = a_0, \dots, a_{i-1}, a_{i+1}, \dots, a_{n-1}$.)
- ...

donde $L : \text{Lista}(\text{ELEM})$; $i : \mathbb{Z}$; $x : \text{ELEM}$ (entero, char, etc.).

TAD Lista(ELEM) - Ejemplo de uso

Encabezado: $Máximo : L \in Lista(\mathbb{Z}) \rightarrow \mathbb{Z}$

Precondición: $\{L = L_0 \wedge L.Longitud() > 0\}$

Poscondición: $\{(\forall i) 0 \leq i < L_0.Longitud() \Rightarrow L_0.Iésimo(i) \leq RV$
 $\wedge L_0.Cantidad(RV) > 0\}$

TAD Lista(ELEM) - Ejemplo de uso

Encabezado: $Máximo : L \in Lista(\mathbb{Z}) \rightarrow \mathbb{Z}$

Precondición: $\{L = L_0 \wedge L.Longitud() > 0\}$

Poscondición: $\{(\forall i) 0 \leq i < L_0.Longitud() \Rightarrow L_0.Iésimo(i) \leq RV$
 $\wedge L_0.Cantidad(RV) > 0\}$

$m \leftarrow L.Iésimo(0)$

$i \leftarrow 1$

while $(i < L.Longitud())$ {
 if $(L.Iésimo(i) > m)$ {
 $m \leftarrow L.Iésimo(i)$
 }
 $i \leftarrow i + 1$

}
 $RV \leftarrow m$

TAD Lista(ELEM) - Implementación

La clase que viene vamos a ver formas de implementar Listas y otros TADs.

En esta clase nos enfocamos en el **uso** de los TADs, que podemos haber creado **nosotros o alguien más**.

Desde el punto de vista del usuario, los detalles de implementación (la **parte privada**) son irrelevantes.



TAD Pila(ELEM)

Operaciones:

- `CrearPila()` \rightarrow `Pila(ELEM)`: Crea una pila vacía.

TAD Pila(ELEM)

Operaciones:

- $\text{CrearPila}() \rightarrow \text{Pila}(\text{ELEM})$: Crea una pila vacía.
- $P.\text{Apilar}(x)$: Inserta el elem. x sobre el tope de la pila P .

TAD Pila(ELEM)

Operaciones:

- $\text{CrearPila}() \rightarrow \text{Pila}(\text{ELEM})$: Crea una pila vacía.
- $P.\text{Apilar}(x)$: Inserta el elem. x sobre el tope de la pila P .
- $P.\text{Vacía?}(P) \rightarrow \mathbb{B}$: Dice si la pila P está vacía.

TAD Pila(ELEM)

Operaciones:

- $\text{CrearPila}() \rightarrow \text{Pila}(\text{ELEM})$: Crea una pila vacía.
- $P.\text{Apilar}(x)$: Inserta el elem. x sobre el tope de la pila P .
- $P.\text{Vacía?}(P) \rightarrow \mathbb{B}$: Dice si la pila P está vacía.
- $P.\text{Tope}() \rightarrow \text{ELEM}$: Devuelve el elemento del tope de P .
Precondición: $\neg P.\text{Vacía?}()$.

TAD Pila(ELEM)

Operaciones:

- $\text{CrearPila}() \rightarrow \text{Pila}(\text{ELEM})$: Crea una pila vacía.
- $P.\text{Apilar}(x)$: Inserta el elem. x sobre el tope de la pila P .
- $P.\text{Vacía?}(P) \rightarrow \mathbb{B}$: Dice si la pila P está vacía.
- $P.\text{Tope}() \rightarrow \text{ELEM}$: Devuelve el elemento del tope de P .
Precondición: $\neg P.\text{Vacía?}()$.
- $P.\text{Desapilar}()$: Borra el elemento del tope de P .
Precondición: $\neg P.\text{Vacía?}()$.

donde $P : \text{Pila}(\text{ELEM})$, $x : \text{ELEM}$ (entero, char, etc.).

Las pilas tienen estrategia **LIFO** (*last in, first out*).

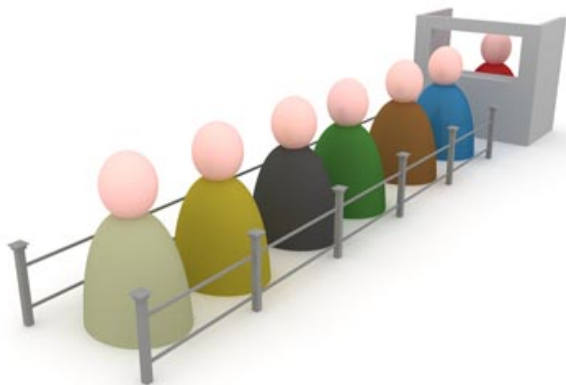
Problema: Paréntesis balanceados

Dado un string, determinar si los caracteres { }, [], () están balanceados correctamente.

Ejemplos:

- “{a(b)x[()] }” está balanceado.
- “}”, “a(b))” y “[()]” no están balanceados.

¿Se les ocurre una solución? Sugerencia: usar el tipo Pila.



TAD Cola(ELEM)

Operaciones:

- $\text{CrearCola}() \rightarrow \text{Cola}(\text{ELEM})$: Crea una cola vacía.
- $C.\text{Encolar}(x)$: Agrega el elemento x al final de la cola C .
- $C.\text{Vacía}() \rightarrow \mathbb{B}$: Dice si la cola C está vacía.
- $C.\text{Primero}() \rightarrow \text{ELEM}$: Devuelve el primer elemento de C .
Precondición: $\neg C.\text{Vacía}()$.
- $C.\text{Desencolar}()$: Borra el primer elemento de C .
Precondición: $\neg C.\text{Vacía}()$.

donde $C : \text{Cola}(\text{ELEM})$, $x : \text{ELEM}$ (entero, char, etc.).

Las colas tienen estrategia **FIFO** (*first in, first out*).

Otro ejemplo

Queremos representar conjuntos de especies animales.

Podemos hacerlo con listas, por ejemplo:

- Felinos = [león, gato, tigre, guepardo, pantera, puma]
- Cánidos = [lobo, coyote, chacal, dingo, zorro]
- Cetáceos = [delfín, ballena, orca, narval, cachalote]

Otro ejemplo

Queremos representar conjuntos de especies animales.

Podemos hacerlo con listas, por ejemplo:

- Felinos = [león, gato, tigre, guepardo, pantera, puma]
- Cánidos = [lobo, coyote, chacal, dingo, zorro]
- Cetáceos = [delfín, ballena, orca, narval, cachalote]

¿Las listas son una buena forma de representar conjuntos?

Otro ejemplo

Queremos representar conjuntos de especies animales.

Podemos hacerlo con listas, por ejemplo:

- Felinos = [león, gato, tigre, guepardo, pantera, puma]
- Cánidos = [lobo, coyote, chacal, dingo, zorro]
- Cetáceos = [delfín, ballena, orca, narval, cachalote]

¿Las listas son una buena forma de representar conjuntos?

- Orden: [lobo, coyote] = [coyote, lobo] ?
- Repetidos: [delfín, delfín] ?

Otro ejemplo

Queremos representar conjuntos de especies animales.

Podemos hacerlo con listas, por ejemplo:

- Felinos = [león, gato, tigre, guepardo, pantera, puma]
- Cánidos = [lobo, coyote, chacal, dingo, zorro]
- Cetáceos = [delfín, ballena, orca, narval, cachalote]

¿Las listas son una buena forma de representar conjuntos?

- Orden: [lobo, coyote] = [coyote, lobo] ?
- Repetidos: [delfín, delfín] ?

Mejor usar un TAD Conjunto que nos evite estos problemas.

TAD Conjunto(ELEM)

Operaciones:

TAD Conjunto(ELEM)

Operaciones:

- `CrearConjunto()` \rightarrow `Conjunto(ELEM)`: Crea un conjunto vacío.

TAD Conjunto(ELEM)

Operaciones:

- $\text{CrearConjunto}() \rightarrow \text{Conjunto}(\text{ELEM})$: Crea un conjunto vacío.
- $C.\text{Agregar}(x)$: Agrega el elemento x al conjunto C .

TAD Conjunto(ELEM)

Operaciones:

- $\text{CrearConjunto}() \rightarrow \text{Conjunto}(\text{ELEM})$: Crea un conjunto vacío.
- $C.\text{Agregar}(x)$: Agrega el elemento x al conjunto C .
- $C.\text{Pertenece?}(x) \rightarrow \mathbb{B}$: Dice si el elemento x está en C .

TAD Conjunto(ELEM)

Operaciones:

- $\text{CrearConjunto}() \rightarrow \text{Conjunto}(\text{ELEM})$: Crea un conjunto vacío.
- $C.\text{Agregar}(x)$: Agrega el elemento x al conjunto C .
- $C.\text{Pertenece?}(x) \rightarrow \mathbb{B}$: Dice si el elemento x está en C .
- $C.\text{Eliminar}(x)$: Elimina el elemento x de C .

TAD Conjunto(ELEM)

Operaciones:

- $\text{CrearConjunto}() \rightarrow \text{Conjunto}(\text{ELEM})$: Crea un conjunto vacío.
- $C.\text{Agregar}(x)$: Agrega el elemento x al conjunto C .
- $C.\text{Pertenece?}(x) \rightarrow \mathbb{B}$: Dice si el elemento x está en C .
- $C.\text{Eliminar}(x)$: Elimina el elemento x de C .
- $C.\text{Tamaño}() \rightarrow \mathbb{Z}$: Devuelve la cantidad de elementos de C .

TAD Conjunto(ELEM)

Operaciones:

- $\text{CrearConjunto}() \rightarrow \text{Conjunto}(\text{ELEM})$: Crea un conjunto vacío.
- $C.\text{Agregar}(x)$: Agrega el elemento x al conjunto C .
- $C.\text{Pertenece?}(x) \rightarrow \mathbb{B}$: Dice si el elemento x está en C .
- $C.\text{Eliminar}(x)$: Elimina el elemento x de C .
- $C.\text{Tamaño}() \rightarrow \mathbb{Z}$: Devuelve la cantidad de elementos de C .
- $C_1.\text{Igual?}(C_2) \rightarrow \mathbb{B}$: Dice si los dos conjuntos son iguales.

donde $C, C_1, C_2 : \text{Conjunto}(\text{ELEM}), x : \text{ELEM}$.

TAD Conjunto(ELEM)

Operaciones (cont.):

- $C_1.\text{Unión}(C_2) \rightarrow \text{Conjunto}(\text{ELEM})$: Devuelve un nuevo conjunto con $C_1 \cup C_2$.
- $C_1.\text{Intersección}(C_2) \rightarrow \text{Conjunto}(\text{ELEM})$: Devuelve un nuevo conjunto con $C_1 \cap C_2$.
- $C_1.\text{Diferencia}(C_2) \rightarrow \text{Conjunto}(\text{ELEM})$: Devuelve un nuevo conjunto con $C_1 \setminus C_2$.

TAD Conjunto(ELEM)

Operaciones (cont.):

- $C_1.$ Unión(C_2) \rightarrow Conjunto(ELEM): Devuelve un nuevo conjunto con $C_1 \cup C_2$.
- $C_1.$ Intersección(C_2) \rightarrow Conjunto(ELEM): Devuelve un nuevo conjunto con $C_1 \cap C_2$.
- $C_1.$ Diferencia(C_2) \rightarrow Conjunto(ELEM): Devuelve un nuevo conjunto con $C_1 \setminus C_2$.
- $C.$ ListarElementos() \rightarrow Lista(ELEM): Devuelve una lista de los elementos de C , en cualquier orden.

TAD Conjunto(ELEM)

Operaciones (cont.):

- $C_1.\text{Unión}(C_2) \rightarrow \text{Conjunto}(\text{ELEM})$: Devuelve un nuevo conjunto con $C_1 \cup C_2$.
 - $C_1.\text{Intersección}(C_2) \rightarrow \text{Conjunto}(\text{ELEM})$: Devuelve un nuevo conjunto con $C_1 \cap C_2$.
 - $C_1.\text{Diferencia}(C_2) \rightarrow \text{Conjunto}(\text{ELEM})$: Devuelve un nuevo conjunto con $C_1 \setminus C_2$.
 - $C.\text{ListarElementos}() \rightarrow \text{Lista}(\text{ELEM})$: Devuelve una lista de los elementos de C , en cualquier orden.
 - $C.\text{AgregarTodos}(L)$: Agrega todos los elementos de L en C .
- donde $C, C_1, C_2: \text{Conjunto}(\text{ELEM})$, $x: \text{ELEM}$, $L: \text{Lista}(\text{ELEM})$.

SEGUNDA PARTE Implementación

TAD Fecha

Operaciones públicas (para $f, f_1, f_2 : \text{Fecha}$; $d, m, a : \mathbb{Z}$):

- $\text{CrearFecha}(d, m, a) \rightarrow \text{Fecha}$
- $f.\text{Día}() \rightarrow \mathbb{Z}$
- $f.\text{Mes}() \rightarrow \mathbb{Z}$
- $f.\text{Año}() \rightarrow \mathbb{Z}$
- $f_1.\text{Menor}(f_2) \rightarrow \mathbb{B}$
- $f.\text{FechaSiguiente}() \rightarrow \text{Fecha}$

TAD Fecha

Operaciones públicas (para $f, f_1, f_2 : \text{Fecha}$; $d, m, a : \mathbb{Z}$):

- $\text{CrearFecha}(d, m, a) \rightarrow \text{Fecha}$
- $f.\text{Día}() \rightarrow \mathbb{Z}$
- $f.\text{Mes}() \rightarrow \mathbb{Z}$
- $f.\text{Año}() \rightarrow \mathbb{Z}$
- $f_1.\text{Menor}(f_2) \rightarrow \mathbb{B}$
- $f.\text{FechaSiguiente}() \rightarrow \text{Fecha}$

Con esto, un usuario del TAD puede programar algo como:

$\text{ContarDías}(f_1, f_2) \rightarrow \mathbb{Z}$:

```
RV  $\leftarrow$  0
while ( $f_1.\text{Menor}(f, f_2)$ ) {
    RV  $\leftarrow$  RV + 1
     $f_1 \leftarrow f_1.\text{FechaSiguiente}()$ 
}
```


TAD Fecha - Posible implementación

TAD Fecha - Posible implementación

Primero elegimos una **estructura de representación** para el TAD Fecha (que es **privada**).

TAD Fecha - Posible implementación

Primero elegimos una **estructura de representación** para el TAD Fecha (que es **privada**).

$$\text{Fecha} == \langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$$

$\langle \dots \rangle$ define una **tupla**.

TAD Fecha - Posible implementación

Primero elegimos una **estructura de representación** para el TAD Fecha (que es **privada**).

$$\text{Fecha} == \langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$$

$\langle \dots \rangle$ define una **tupla**.

Después describimos el **invariante de representación** de esta estructura.

TAD Fecha - Posible implementación

Primero elegimos una **estructura de representación** para el TAD Fecha (que es **privada**).

$$\text{Fecha} == \langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$$

$\langle \dots \rangle$ define una **tupla**.

Después describimos el **invariante de representación** de esta estructura.

$$DMAVálidos(\text{día}, \text{mes}, \text{año})$$

donde:

$$\begin{aligned} DMAVálidos(d, m, a) \equiv & (1 \leq d \leq 31 \wedge (m = 1 \vee m = 3 \vee \dots)) \vee \\ & (1 \leq d \leq 30 \wedge (m = 4 \vee m = 6 \vee \dots)) \vee \\ & (1 \leq d \leq 29 \wedge m = 2 \wedge Bisiesto(a)) \vee \\ & (1 \leq d \leq 28 \wedge m = 2 \wedge \neg Bisiesto(a)) \end{aligned}$$

$$Bisiesto(a) \equiv (a \text{ div } 4 = 0 \wedge (a \text{ div } 100 \neq 0 \vee a \text{ div } 400 = 0))$$

TAD Fecha - Posible implementación

Fecha == $\langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$

TAD Fecha - Posible implementación

Fecha == $\langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$

Por último, damos los **algoritmos de las operaciones** del TAD Fecha para la estructura de representación elegida:

TAD Fecha - Posible implementación

Fecha == $\langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$

Por último, damos los **algoritmos de las operaciones** del TAD Fecha para la estructura de representación elegida:

CrearFecha(d, m, a) \rightarrow Fecha: (Pre: $DMAVálidos(d, m, a)$)

TAD Fecha - Posible implementación

Fecha == $\langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$

Por último, damos los **algoritmos de las operaciones** del TAD Fecha para la estructura de representación elegida:

CrearFecha(d, m, a) \rightarrow Fecha: (Pre: $DMAVálidos(d, m, a)$)

$RV.día \leftarrow d$

$RV.mes \leftarrow m$

$RV.año \leftarrow a$

donde $d, m, a : \mathbb{Z}$.

TAD Fecha - Posible implementación

Fecha == $\langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$

Por último, damos los **algoritmos de las operaciones** del TAD Fecha para la estructura de representación elegida:

CrearFecha(d, m, a) \rightarrow Fecha: (Pre: $DMAVálidos(d, m, a)$)

$RV.día \leftarrow d$

$RV.mes \leftarrow m$

$RV.año \leftarrow a$

donde $d, m, a : \mathbb{Z}$.

Operaciones como esta se conocen como **constructores**.
Permiten armar elementos del TAD.

TAD Fecha - Posible implementación

Fecha == $\langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$

Más **operaciones** del TAD Fecha (para f : Fecha):

TAD Fecha - Posible implementación

Fecha == $\langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$

Más **operaciones** del TAD Fecha (para $f: \text{Fecha}$):

$f.\text{Día}() \rightarrow \mathbb{Z}$:

TAD Fecha - Posible implementación

Fecha == $\langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$

Más **operaciones** del TAD Fecha (para $f: \text{Fecha}$):

$f.\text{Día}() \rightarrow \mathbb{Z}$:

$RV \leftarrow f.\text{día}$

TAD Fecha - Posible implementación

Fecha == $\langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$

Más **operaciones** del TAD Fecha (para f : Fecha):

$f.\text{Día}() \rightarrow \mathbb{Z}$:

$RV \leftarrow f.\text{día}$

$f.\text{Mes}() \rightarrow \mathbb{Z}$:

$RV \leftarrow f.\text{mes}$

$f.\text{Año}() \rightarrow \mathbb{Z}$:

$RV \leftarrow f.\text{año}$

TAD Fecha - Posible implementación

Fecha == $\langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$

Más **operaciones** del TAD Fecha (para f : Fecha):

$f.\text{Día}() \rightarrow \mathbb{Z}$:

$RV \leftarrow f.\text{día}$

$f.\text{Mes}() \rightarrow \mathbb{Z}$:

$RV \leftarrow f.\text{mes}$

$f.\text{Año}() \rightarrow \mathbb{Z}$:

$RV \leftarrow f.\text{año}$

Estas operaciones se conocen como **proyectores**.

Permiten inspeccionar elementos de un TAD.

TAD Fecha - Posible implementación

Fecha == $\langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$

Más **operaciones** del TAD Fecha (para f_1, f_2 : Fecha):

TAD Fecha - Posible implementación

Fecha == $\langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$

Más **operaciones** del TAD Fecha (para f_1, f_2 : Fecha):
 $f_1.\text{Menor}(f_2) \rightarrow \mathbb{B}$:

TAD Fecha - Posible implementación

Fecha == $\langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$

Más **operaciones** del TAD Fecha (para f_1, f_2 : Fecha):

$f_1.\text{Menor}(f_2) \rightarrow \mathbb{B}$:

```
if ( $f_1.\text{Año}() < f_2.\text{Año}()$ ) {  
     $RV \leftarrow \text{TRUE}$   
}  
elseif ( $f_1.\text{Año}() = f_2.\text{Año}() \wedge f_1.\text{Mes}() < f_2.\text{Mes}()$ ) {  
     $RV \leftarrow \text{TRUE}$   
}  
elseif ( $f_1.\text{Año}() = f_2.\text{Año}() \wedge f_1.\text{Mes}() = f_2.\text{Mes}() \wedge f_1.\text{Día}() < f_2.\text{Día}()$ ) {  
     $RV \leftarrow \text{TRUE}$   
}  
else {  
     $RV \leftarrow \text{FALSE}$   
}
```

TAD Fecha - Posible implementación

Fecha == $\langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$

Más **operaciones** del TAD Fecha (para f : Fecha):

TAD Fecha - Posible implementación

Fecha == $\langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$

Más **operaciones** del TAD Fecha (para $f: \text{Fecha}$):

$f.\text{FechaSiguiente()} \rightarrow \text{Fecha}$:

TAD Fecha - Posible implementación

Fecha == $\langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$

Más **operaciones** del TAD Fecha (para f : Fecha):

$f.\text{FechaSiguiente}() \rightarrow \text{Fecha}$:

$d \leftarrow f.\text{Día}() + 1$

$m \leftarrow f.\text{Mes}()$

$a \leftarrow f.\text{Año}()$

if ($d > \text{CantidadDeDías}(m, a)$) {

$d \leftarrow 1$

$m \leftarrow m + 1$

if ($m > 12$) {

$m \leftarrow 1$

$a \leftarrow a + 1$

}

}

$RV \leftarrow \text{CrearFecha}(d, m, a)$

TAD Fecha - Posible implementación

Fecha == $\langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$

Más **operaciones** del TAD Fecha (para $f: \text{Fecha}; m, a : \mathbb{Z}$):

CantidadDeDías(m, a) $\rightarrow \mathbb{Z}$: (Pre: $1 \leq m \leq 12$)

TAD Fecha - Posible implementación

Fecha == $\langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$

Más **operaciones** del TAD Fecha (para $f: \text{Fecha}; m, a: \mathbb{Z}$):

CantidadDeDías(m, a) $\rightarrow \mathbb{Z}$: (Pre: $1 \leq m \leq 12$)

```
if ( $m = 1 \vee m = 3 \vee m = 5 \vee m = 7 \vee m = 8 \vee m = 10 \vee m = 12$ ) {  
     $RV \leftarrow 31$   
}  
} elseif ( $m = 4 \vee m = 6 \vee m = 9 \vee m = 11$ ) {  
     $RV \leftarrow 30$   
}  
} elseif ( $m = 2 \wedge \text{EsBisiesto}(a)$ ) {  
     $RV \leftarrow 29$   
}  
} else {  
     $RV \leftarrow 28$   
}
```

TAD Fecha - Posible implementación

Fecha == $\langle \text{día: } \mathbb{Z}, \text{mes: } \mathbb{Z}, \text{año: } \mathbb{Z} \rangle$

Más **operaciones** del TAD Fecha (para $f: \text{Fecha}; m, a: \mathbb{Z}$):

CantidadDeDías(m, a) $\rightarrow \mathbb{Z}$: (Pre: $1 \leq m \leq 12$)

```
if ( $m = 1 \vee m = 3 \vee m = 5 \vee m = 7 \vee m = 8 \vee m = 10 \vee m = 12$ ) {  
     $RV \leftarrow 31$   
}  
} elsif ( $m = 4 \vee m = 6 \vee m = 9 \vee m = 11$ ) {  
     $RV \leftarrow 30$   
}  
} elsif ( $m = 2 \wedge \text{EsBisiesto}(a)$ ) {  
     $RV \leftarrow 29$   
}  
} else {  
     $RV \leftarrow 28$   
}
```

Podemos elegir que CantidadDeDías y EsBisiesto sean operaciones **privadas**: no accesibles para usuarios del TAD Fecha.

Partes de un Tipo Abstracto de Datos

- Parte **pública**: Disponible para el usuario externo.
 - Nombre y tipos paramétricos (ej: Fecha, Lista(ELEM)).
 - Operaciones, sus especificaciones y posiblemente sus órdenes de complejidad temporal.

Partes de un Tipo Abstracto de Datos

- Parte **pública**: Disponible para el usuario externo.
 - Nombre y tipos paramétricos (ej: Fecha, Lista(ELEM)).
 - Operaciones, sus especificaciones y posiblemente sus órdenes de complejidad temporal.
- Parte **privada**: Sólo accesible desde dentro del TAD. ¡El usuario externo **nunca** debe ver ni meterse en esto!
 - **Estructura de representación** del TAD.
 - **Invariante de representación**: qué propiedades debe cumplir la estructura elegida para que tenga sentido como la representación de una instancia del TAD.
 - **Algoritmos de las operaciones** para la estructura de representación elegida.

TAD Lista(ELEM)

- $\text{CrearLista}() \rightarrow \text{Lista}(\text{ELEM})$
- $L.\text{Agregar}(x)$
- $L.\text{Longitud}() \rightarrow \mathbb{Z}$
- $L.\text{lésimo}(i) \rightarrow \text{ELEM}$
- $L.\text{Cantidad}(x) \rightarrow \mathbb{Z}$
- $L.\text{Insertar}(i, x)$
- $L.\text{Borrarlésimo}(i)$
- $L.\text{Indice}(x) \rightarrow \mathbb{Z}$

donde $L : \text{Lista}(\text{ELEM})$, $i : \mathbb{Z}$, $x : \text{ELEM}$ (entero, char, etc.).

TAD Lista(ELEM) - Posible implementación

TAD Lista(ELEM) - Posible implementación

Vamos a representar una lista como una cadena de nodos.

TAD Lista(ELEM) - Posible implementación

Vamos a representar una lista como una cadena de nodos.
Definamos primero qué es un nodo.

Nodo(ELEM) : $\langle valor : \text{ELEM}, siguiente : \text{Ref}(\text{Nodo}) \rangle$

TAD Lista(ELEM) - Posible implementación

Vamos a representar una lista como una cadena de nodos.
Definamos primero qué es un nodo.

Nodo(ELEM) : $\langle valor : \text{ELEM}, siguiente : \text{Ref}(\text{Nodo}) \rangle$

$\langle \dots \rangle$ define una **tupla**.

TAD Lista(ELEM) - Posible implementación

Vamos a representar una lista como una cadena de nodos.
Definamos primero qué es un nodo.

Nodo(ELEM) : $\langle valor : \text{ELEM}, siguiente : \text{Ref}(\text{Nodo}) \rangle$

$\langle \dots \rangle$ define una **tupla**.

Ref(Nodo) es una **referencia** a una instancia de tipo Nodo.
Puede tomar el valor especial *None* (“referencia a nada”).

TAD Lista(ELEM) - Posible implementación

Vamos a representar una lista como una cadena de nodos.
Definamos primero qué es un nodo.

Nodo(ELEM) : $\langle valor : \text{ELEM}, siguiente : \text{Ref(Nodo)} \rangle$

$\langle \dots \rangle$ define una **tupla**.

Ref(Nodo) es una **referencia** a una instancia de tipo Nodo.
Puede tomar el valor especial *None* (“referencia a nada”).

Construcción de un nuevo Nodo:

- $n \leftarrow \text{Nodo}(x, r)$

TAD Lista(ELEM) - Posible implementación

Vamos a representar una lista como una cadena de nodos.
Definamos primero qué es un nodo.

Nodo(ELEM) : $\langle \text{valor} : \text{ELEM}, \text{siguiente} : \text{Ref}(\text{Nodo}) \rangle$

$\langle \dots \rangle$ define una **tupla**.

Ref(Nodo) es una **referencia** a una instancia de tipo Nodo.
Puede tomar el valor especial *None* (“referencia a nada”).

Construcción de un nuevo Nodo:

- $n \leftarrow \text{Nodo}(x, r)$

Acceso a los campos de un nodo n :

- $n.\text{valor}$ devuelve el campo *valor*.
- $n.\text{siguiente}$ devuelve el campo *siguiente*.

TAD Lista(ELEM) - Posible implementación

Primero elegimos una **estructura de representación** del TAD Lista(ELEM) (que es **privada**).

TAD Lista(ELEM) - Posible implementación

Primero elegimos una **estructura de representación** del TAD Lista(ELEM) (que es **privada**).

$$\text{Lista(ELEM)} == \langle \text{cabeza} : \text{Ref(Nodo)}, \text{longitud} : \mathbb{Z} \rangle$$

TAD Lista(ELEM) - Posible implementación

Primero elegimos una **estructura de representación** del TAD Lista(ELEM) (que es **privada**).

$$\text{Lista(ELEM)} == \langle \text{cabeza} : \text{Ref(Nodo)}, \text{longitud} : \mathbb{Z} \rangle$$

Después describimos el **invariante de representación** de esta estructura.

TAD Lista(ELEM) - Posible implementación

Primero elegimos una **estructura de representación** del TAD Lista(ELEM) (que es **privada**).

$$\text{Lista(ELEM)} == \langle \text{cabeza} : \text{Ref(Nodo)}, \text{longitud} : \mathbb{Z} \rangle$$

Después describimos el **invariante de representación** de esta estructura. En este caso *longitud* siempre debe ser igual a la cantidad de nodos encadenados a partir de *cabeza*, y en la cadena de nodos no deben formarse ciclos.

TAD Lista(ELEM) - Posible implementación

Primero elegimos una **estructura de representación** del TAD Lista(ELEM) (que es **privada**).

$$\text{Lista(ELEM)} == \langle \text{cabeza} : \text{Ref(Nodo)}, \text{longitud} : \mathbb{Z} \rangle$$

Después describimos el **invariante de representación** de esta estructura. En este caso *longitud* siempre debe ser igual a la cantidad de nodos encadenados a partir de *cabeza*, y en la cadena de nodos no deben formarse ciclos.

Por último, damos los **algoritmos de las operaciones** del TAD Lista(ELEM) para la estructura de representación elegida:

TAD Lista(ELEM) - Posible implementación

Primero elegimos una **estructura de representación** del TAD Lista(ELEM) (que es **privada**).

$$\text{Lista(ELEM)} == \langle \text{cabeza} : \text{Ref(Nodo)}, \text{longitud} : \mathbb{Z} \rangle$$

Después describimos el **invariante de representación** de esta estructura. En este caso *longitud* siempre debe ser igual a la cantidad de nodos encadenados a partir de *cabeza*, y en la cadena de nodos no deben formarse ciclos.

Por último, damos los **algoritmos de las operaciones** del TAD Lista(ELEM) para la estructura de representación elegida:

CrearLista() \rightarrow Lista(ELEM):

TAD Lista(ELEM) - Posible implementación

Primero elegimos una **estructura de representación** del TAD Lista(ELEM) (que es **privada**).

$$\text{Lista}(\text{ELEM}) == \langle \text{cabeza} : \text{Ref}(\text{Nodo}), \text{longitud} : \mathbb{Z} \rangle$$

Después describimos el **invariante de representación** de esta estructura. En este caso *longitud* siempre debe ser igual a la cantidad de nodos encadenados a partir de *cabeza*, y en la cadena de nodos no deben formarse ciclos.

Por último, damos los **algoritmos de las operaciones** del TAD Lista(ELEM) para la estructura de representación elegida:

CrearLista() \rightarrow Lista(ELEM):

RV.cabeza \leftarrow None

RV.longitud \leftarrow 0

TAD Lista(ELEM) - Posible implementación

L.Agregar(x):

TAD Lista(ELEM) - Posible implementación

L.Agregar(x):

nuevo \leftarrow *Nodo(x, None)*

if (*L.cabeza* = *None*) {

L.cabeza \leftarrow *nuevo*

}

else {

aux \leftarrow *L.cabeza*

 while (*aux.siguiente* \neq *None*) {

aux \leftarrow *aux.siguiente*

 }

aux.siguiente \leftarrow *nuevo*

}

L.longitud \leftarrow *L.longitud* + 1

TAD Lista(ELEM) - Posible implementación

$L.\text{Longitud}() \rightarrow \mathbb{Z}$:

TAD Lista(ELEM) - Posible implementación

$L.\text{Longitud}() \rightarrow \mathbb{Z}$:

$RV \leftarrow L.\text{longitud}$

$L.\text{lésimo}(i) \rightarrow ELEM$: (Pre: $0 \leq i < L.\text{Longitud}()$)

TAD Lista(ELEM) - Posible implementación

$L.Longitud() \rightarrow \mathbb{Z}$:

$RV \leftarrow L.longitud$

$L.lésimo(i) \rightarrow ELEM$: (Pre: $0 \leq i < L.Longitud()$)

$aux \leftarrow L.cabeza$

while ($i > 0$) {

$aux \leftarrow aux.siguiete$

$i \leftarrow i - 1$

}

$RV \leftarrow aux.valor$

Y así con las otras operaciones del TAD Lista(ELEM):
Cantidad, Insertar, Índice y Borrarlésimo

TAD Lista(ELEM) - Posible implementación

Complejidad temporal de los algoritmos vistos para esta estructura de representación.

- $\text{CrearLista}() \rightarrow \text{Lista}(\text{ELEM})$
- $L.\text{Agregar}(x)$
- $L.\text{Longitud}() \rightarrow \mathbb{Z}$
- $L.\text{lésimo}(i) \rightarrow \text{ELEM}$
- $L.\text{Cantidad}(x) \rightarrow \mathbb{Z}$
- $L.\text{Insertar}(i, x)$
- $L.\text{Borrarlésimo}(i)$
- $L.\text{Indice}(x) \rightarrow \mathbb{Z}$

donde $L : \text{Lista}(\text{ELEM})$, $i : \mathbb{Z}$, $x : \text{ELEM}$ (entero, char, etc.).

TAD Lista(ELEM) - Posible implementación

Complejidad temporal de los algoritmos vistos para esta estructura de representación.

- $\text{CrearLista}() \rightarrow \text{Lista}(\text{ELEM}) \quad O(1)$
- $L.\text{Agregar}(x)$
- $L.\text{Longitud}() \rightarrow \mathbb{Z}$
- $L.\text{lésimo}(i) \rightarrow \text{ELEM}$
- $L.\text{Cantidad}(x) \rightarrow \mathbb{Z}$
- $L.\text{Insertar}(i, x)$
- $L.\text{Borrarlésimo}(i)$
- $L.\text{Indice}(x) \rightarrow \mathbb{Z}$

donde $L : \text{Lista}(\text{ELEM})$, $i : \mathbb{Z}$, $x : \text{ELEM}$ (entero, char, etc.).

TAD Lista(ELEM) - Posible implementación

Complejidad temporal de los algoritmos vistos para esta estructura de representación.

- $\text{CrearLista}() \rightarrow \text{Lista}(\text{ELEM}) \quad O(1)$
- $L.\text{Agregar}(x) \quad O(n)$
- $L.\text{Longitud}() \rightarrow \mathbb{Z}$
- $L.\text{lésimo}(i) \rightarrow \text{ELEM}$
- $L.\text{Cantidad}(x) \rightarrow \mathbb{Z}$
- $L.\text{Insertar}(i, x)$
- $L.\text{Borrarlésimo}(i)$
- $L.\text{Indice}(x) \rightarrow \mathbb{Z}$

donde $L : \text{Lista}(\text{ELEM})$, $i : \mathbb{Z}$, $x : \text{ELEM}$ (entero, char, etc.).

TAD Lista(ELEM) - Posible implementación

Complejidad temporal de los algoritmos vistos para esta estructura de representación.

- $\text{CrearLista}() \rightarrow \text{Lista}(\text{ELEM}) \quad O(1)$
- $L.\text{Agregar}(x) \quad O(n)$
- $L.\text{Longitud}() \rightarrow \mathbb{Z} \quad O(1)$
- $L.\text{lésimo}(i) \rightarrow \text{ELEM}$
- $L.\text{Cantidad}(x) \rightarrow \mathbb{Z}$
- $L.\text{Insertar}(i, x)$
- $L.\text{Borrarlésimo}(i)$
- $L.\text{Indice}(x) \rightarrow \mathbb{Z}$

donde $L : \text{Lista}(\text{ELEM})$, $i : \mathbb{Z}$, $x : \text{ELEM}$ (entero, char, etc.).

TAD Lista(ELEM) - Posible implementación

Complejidad temporal de los algoritmos vistos para esta estructura de representación.

- $\text{CrearLista}() \rightarrow \text{Lista}(\text{ELEM}) \quad O(1)$
- $L.\text{Agregar}(x) \quad O(n)$
- $L.\text{Longitud}() \rightarrow \mathbb{Z} \quad O(1)$
- $L.\text{lésimo}(i) \rightarrow \text{ELEM} \quad O(n)$
- $L.\text{Cantidad}(x) \rightarrow \mathbb{Z}$
- $L.\text{Insertar}(i, x)$
- $L.\text{Borrarlésimo}(i)$
- $L.\text{Indice}(x) \rightarrow \mathbb{Z}$

donde $L : \text{Lista}(\text{ELEM})$, $i : \mathbb{Z}$, $x : \text{ELEM}$ (entero, char, etc.).

TAD Lista(ELEM) - Posible implementación

Complejidad temporal de los algoritmos vistos para esta estructura de representación.

- $\text{CrearLista}() \rightarrow \text{Lista}(\text{ELEM}) \quad O(1)$
- $L.\text{Agregar}(x) \quad O(n)$
- $L.\text{Longitud}() \rightarrow \mathbb{Z} \quad O(1)$
- $L.\text{lésimo}(i) \rightarrow \text{ELEM} \quad O(n)$
- $L.\text{Cantidad}(x) \rightarrow \mathbb{Z} \quad O(n)$
- $L.\text{Insertar}(i, x)$
- $L.\text{Borrarlésimo}(i)$
- $L.\text{Indice}(x) \rightarrow \mathbb{Z}$

donde $L : \text{Lista}(\text{ELEM})$, $i : \mathbb{Z}$, $x : \text{ELEM}$ (entero, char, etc.).

TAD Lista(ELEM) - Posible implementación

Complejidad temporal de los algoritmos vistos para esta estructura de representación.

- $\text{CrearLista}() \rightarrow \text{Lista}(\text{ELEM}) \quad O(1)$
- $L.\text{Agregar}(x) \quad O(n)$
- $L.\text{Longitud}() \rightarrow \mathbb{Z} \quad O(1)$
- $L.\text{lésimo}(i) \rightarrow \text{ELEM} \quad O(n)$
- $L.\text{Cantidad}(x) \rightarrow \mathbb{Z} \quad O(n)$
- $L.\text{Insertar}(i, x) \quad O(n)$
- $L.\text{Borrarlésimo}(i)$
- $L.\text{Indice}(x) \rightarrow \mathbb{Z}$

donde $L : \text{Lista}(\text{ELEM})$, $i : \mathbb{Z}$, $x : \text{ELEM}$ (entero, char, etc.).

TAD Lista(ELEM) - Posible implementación

Complejidad temporal de los algoritmos vistos para esta estructura de representación.

- $\text{CrearLista}() \rightarrow \text{Lista}(\text{ELEM}) \quad O(1)$
- $L.\text{Agregar}(x) \quad O(n)$
- $L.\text{Longitud}() \rightarrow \mathbb{Z} \quad O(1)$
- $L.\text{lésimo}(i) \rightarrow \text{ELEM} \quad O(n)$
- $L.\text{Cantidad}(x) \rightarrow \mathbb{Z} \quad O(n)$
- $L.\text{Insertar}(i, x) \quad O(n)$
- $L.\text{Borrarlésimo}(i) \quad O(n)$
- $L.\text{Indice}(x) \rightarrow \mathbb{Z}$

donde $L : \text{Lista}(\text{ELEM})$, $i : \mathbb{Z}$, $x : \text{ELEM}$ (entero, char, etc.).

TAD Lista(ELEM) - Posible implementación

Complejidad temporal de los algoritmos vistos para esta estructura de representación.

- $\text{CrearLista}() \rightarrow \text{Lista}(\text{ELEM}) \quad O(1)$
- $L.\text{Agregar}(x) \quad O(n)$
- $L.\text{Longitud}() \rightarrow \mathbb{Z} \quad O(1)$
- $L.\text{lésimo}(i) \rightarrow \text{ELEM} \quad O(n)$
- $L.\text{Cantidad}(x) \rightarrow \mathbb{Z} \quad O(n)$
- $L.\text{Insertar}(i, x) \quad O(n)$
- $L.\text{Borrarlésimo}(i) \quad O(n)$
- $L.\text{Indice}(x) \rightarrow \mathbb{Z} \quad O(n)$

donde $L : \text{Lista}(\text{ELEM})$, $i : \mathbb{Z}$, $x : \text{ELEM}$ (entero, char, etc.).

Partes de un Tipo Abstracto de Datos

- Parte **pública**: Disponible para el usuario externo.
 - Nombre y tipos paramétricos (ej: Fecha, Lista(ELEM)).
 - Operaciones, sus especificaciones y posiblemente sus órdenes de complejidad temporal.
- Parte **privada**: Sólo accesible desde dentro del TAD. ¡El usuario externo **nunca** debe ver ni meterse en esto!
 - **Estructura de representación** del TAD.
 - **Invariante de representación**: qué propiedades debe cumplir la estructura elegida para que tenga sentido como la representación de una instancia del TAD.
 - **Algoritmos de las operaciones** para la estructura de representación elegida.

Próxima clase: Implementaciones de Pila(ELEM), Cola(ELEM) y Conjunto(ELEM).