Representación de los reales en la computadora, estructuras de datos básicas y estructuras de datos abstractos (ADTs)

mat-151

Representación de números **reales** (de punto fijo)

Representación de números **reales** (de punto fijo)

- es la representación para números que tienen un número fijo de dígitos después del punto.

Representación de números **reales** (de punto fijo)

- es la representación para números que tienen un número fijo de dígitos después del punto.

- se usa cuando se quiere trabajar con una precisión absoluta dada.

Representación de números **reales** (de punto fijo)

- es la representación para números que tienen un número fijo de dígitos después del punto.
- se usa cuando se quiere trabajar con una precisión absoluta dada.
- puede escribirse como ***M.F***, donde ***M*** representa su magnitud (la parte entera) y ***F*** representa la parte fraccionaria.

Representación de números **reales** (de punto fijo)

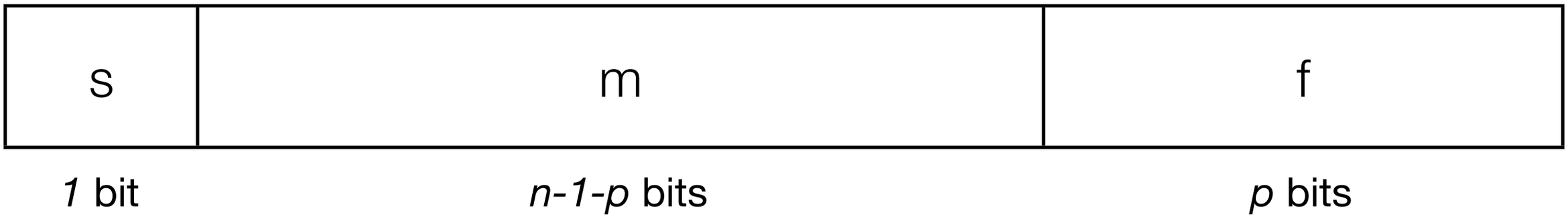
- es la representación para números que tienen un número fijo de dígitos después del punto.

- se usa cuando se quiere trabajar con una precisión absoluta dada.

- puede escribirse como ***M.F***, donde ***M*** representa su magnitud (la parte entera) y ***F*** representa la parte fraccionaria.

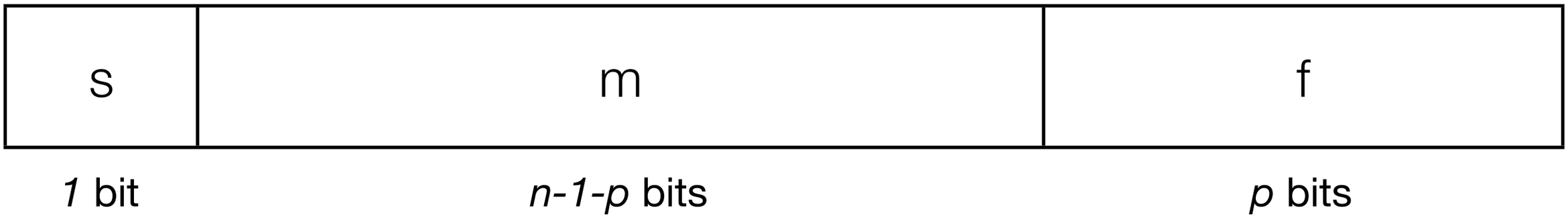
- para números **binarios** de punto fijo, cada **bit** de la **magnitud** representa una potencia de dos y cada **bit** de la **fracción** representa una potencia **inversa** de dos.

Representación de números **reales** (de punto fijo)



$$b = s\,(m + f\,2^{-p})$$

Representación de números **reales** (de punto fijo)



$$b = s\,(m + f\,2^{-p})$$

- Ejemplo: ¿Cómo representar 50.7 con n=16 y p=5?

Representación de números **reales** (de punto fijo)

	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
0	0	0	0	0	1	1	0	0	1	0	1	0	1	1	0

$$= 50.6875$$

Representación de números **reales** (de punto fijo)

	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
0	0	0	0	0	1	1	0	0	1	0	1	0	1	1	0

$$= 50.6875$$

- C y C++ no incluyen soporte para representación de números reales con punto fijo.

Representación de números **reales** (de punto flotante)

Representación de números **reales** (de punto flotante)

- Representación definida por el standard IEEE 754.

Representación de números **reales** (de punto flotante)

- Representación definida por el standard IEEE 754.
- El punto puede colocarse en cualquier lugar, relativamente a los dígitos significativos del número.

Representación de números **reales** (de punto flotante)

- Representación definida por el standard IEEE 754.
- El punto puede colocarse en cualquier lugar, relativamente a los dígitos significativos del número.
- Se puede ver como la implementación de la notación científica.

Representación de números **reales** (de punto flotante)

- Representación definida por el standard IEEE 754.
- El punto puede colocarse en cualquier lugar, relativamente a los dígitos significativos del número.
- Se puede ver como la implementación de la notación científica.
- Puede representar un rango mucho mas grande de números.

Representación de números **reales** (de punto flotante)

- Representación definida por el standard IEEE 754.
- El punto puede colocarse en cualquier lugar, relativamente a los dígitos significativos del número.
- Se puede ver como la implementación de la notación científica.
- Puede representar un rango mucho mas grande de números.
    - por ejemplo, con punto fijo que tiene 8 dígitos, con el punto decimal después del sexto dígito podemos representar los números 123456.78, 8765.43, 123.00, etc.

Representación de números **reales** (de punto flotante)

- Representación definida por el standard IEEE 754.
- El punto puede colocarse en cualquier lugar, relativamente a los dígitos significativos del número.
- Se puede ver como la implementación de la notación científica.
- Puede representar un rango mucho mas grande de números.
  - por ejemplo, con punto fijo que tiene 8 dígitos, con el punto decimal después del sexto dígito podemos representar los números 123456.78, 8765.43, 123.00, etc.
  - con punto flotante de 8 dígitos podemos representar 1.2345678, 1234567.8, 0.000012345678, 12345678000000, etc.

Representación de números **reales** (de punto flotante)

- Representación definida por el standard IEEE 754.
- El punto puede colocarse en cualquier lugar, relativamente a los dígitos significativos del número.
- Se puede ver como la implementación de la notación científica.
- Puede representar un rango mucho mas grande de números.
    - por ejemplo, con punto fijo que tiene 8 dígitos, con el punto decimal después del sexto dígito podemos representar los números 123456.78, 8765.43, 123.00, etc.
    - con punto flotante de 8 dígitos podemos representar 1.2345678, 1234567.8, 0.000012345678, 12345678000000, etc.
- Necesita más espacio de almacenamiento, porque hay que codificar la posición del punto.

Representación de números **reales** (de punto flotante)

- La velocidad de las operaciones con punto flotante se han convertido en medida del desempeño de las computadoras en muchos dominios. Esta unidad de medida se conoce como "***FLOPS***" (floating-point operations per second).

La base es: ***b***=2

Por ejemplo, para float, *e = k-127, por lo tanto k = e+127*

exponentes de -127 (k es todos 0s) y +128 (k es todos 1s) estan reservados para números especiales

Representación de números **reales** (de punto flotante)

- La velocidad de las operaciones con punto flotante se han convertido en medida del desempeño de las computadoras en muchos dominios. Esta unidad de medida se conoce como "***FLOPS***" (floating-point operations per second).

La base es: ***b***=2

Por ejemplo, para float, *e = k-127, por lo tanto k = e+127*

exponentes de -127 (k es todos 0s) y +128 (k es todos 1s) estan reservados para números especiales

Representación de números **reales** (de punto flotante)

- La velocidad de las operaciones con punto flotante se han convertido en medida del desempeño de las computadoras en muchos dominios. Esta unidad de medida se conoce como "***FLOPS***" (floating-point operations per second).

elemento	signo	exponente	mantisa
float	1	8	23
double	1	11	52

La base es: ***b***=2

Por ejemplo, para float, *e = k-127, por lo tanto k = e+127*

exponentes de -127 (k es todos 0s) y +128 (k es todos 1s) estan reservados para números especiales

Representación de números **reales** (de punto flotante)

- La velocidad de las operaciones con punto flotante se han convertido en medida del desempeño de las computadoras en muchos dominios. Esta unidad de medida se conoce como "***FLOPS***" (floating-point operations per second).

elemento	signo	exponente	mantisa
float	1	8	23
double	1	11	52

$$f = smb^e$$

La base es: ***b***=2

Por ejemplo, para float, *e = k-127, por lo tanto k = e+127*

exponentes de -127 (k es todos 0s) y +128 (k es todos 1s) estan reservados para números especiales

Representación de números **reales** (de punto flotante)

- La velocidad de las operaciones con punto flotante se han convertido en medida del desempeño de las computadoras en muchos dominios. Esta unidad de medida se conoce como "***FLOPS***" (floating-point operations per second).

elemento	signo	exponente	mantisa
float	1	8	23
double	1	11	52

$$f = smb^e$$

La base es: ***b***=2

$$e = k - (2^{q-1} - 1);$$

Por ejemplo, para float, *e = k-127, por lo tanto k = e+127*

exponentes de -127 (k es todos 0s) y +128 (k es todos 1s) estan reservados para números especiales

Representación de números **reales** (de punto flotante). Un ejemplo:

Codifiquemos el número decimal **-118.625 en float**.

Necesitamos obtener el signo, el exponente y la fracción.

Dado que es un número negativo, el signo es "1".

Escribimos el número (sin signo) usando notación binaria. El resultado es **1110110.101.**

Movamos el punto decimal a la izquierda, dejando sólo un 1 a su izquierda.
$1110110.101 = 1.110110101 * 2^6$
Esto es un número en punto flotante normalizado.

La mantisa es **11011010100000000000000**
la parte a la derecha del punto decimal, rellenada con ceros a la derecha hasta que obtengamos todos los 23 bits.

El exponente es 6, así es que 6 + 127 = 133. En binario, esto se escribe como **10000101.**

```
 1     8                23             <-- tamaño en bits
+-+--------+-----------------------+
|S|  Exp   |  Mantisa              |
|1|10000101|11011010100000000000000|
+-+--------+-----------------------+
31 30    23 22                    0   <-- índice del bit (0 a la derecha)
   desplazado +127
```

Representación de números **reales** (de punto flotante). Tipos:

Clase	Exp	Fracción
Ceros	0	0
Números desnormalizados	0	distinto de 0
Números normalizados	1-254	cualquiera
Infinitos	255	0
NaN (Not a Number)	255	distinto de 0

NOTAS:

- Hay dos ceros. +0 (S es 0) y –0 (S es 1)
- Hay dos infinitos +∞ (S es 0) y –∞ (S es 1)
- Los NaNs pueden tener un signo y un significando (una explicación),
- Los NaNs y los infinitos tienen todos los bits a 1 en el campo Exp.

Casos especiales

- **NAN**
  - Las divisiones indeterminadas o por infinito (0/0, ∞/∞, ∞/-∞, -∞/∞, -∞/-∞)
  - Las multiplicaciones de 0 por infinito (0×∞, 0×-∞)
  - Las sumas y restas de valores infinitos (∞ + (-∞), (-∞) + ∞)
  - Aplicando funciones que excedan el dominio de la misma. Por ejemplo, la raíz cuadrada de un número negativo, logaritmo de cualquier número menor o igual que 0, o la inversa de un coseno que sea menor que -1 o mayor que +1.

Representación de números **reales** (de punto flotante)

- Se definieron, Overflows, underflows, Redondeos

- No todos los reales pueden representarse correctamente. Para escribir 0.1 en binario necesitaríamos una infinidad de dígitos.

Convertir 0.3125

0.3125*2 = 0.625 => 0

0.625*2 = 1.25 => 1

0.25*2 = 0.5 => 0

0.5*2 = 1 => 1

En orden: .0101

Converting	Result
0.1	0.
0.1 × 2 = **0.2** < 1	0.0
0.2 × 2 = **0.4** < 1	0.00
0.4 × 2 = **0.8** < 1	0.000
0.8 × 2 = **1.6** ≥ 1	0.0001
0.6 × 2 = **1.2** ≥ 1	0.00011
0.2 × 2 = **0.4** < 1	0.000110
0.4 × 2 = **0.8** < 1	0.0001100
0.8 × 2 = **1.6** ≥ 1	0.00011001
0.6 × 2 = **1.2** ≥ 1	0.000110011
0.2 × 2 = **0.4** < 1	0.0001100110

Ejemplo: Ariane 5

Ejemplo: Ariane 5

- La orientación del sistema de lanzamiento y de sus movimientos en el espacio son medidos por un Sistema de Referencia Inercial (SRI)

Ejemplo: Ariane 5

- La orientación del sistema de lanzamiento y de sus movimientos en el espacio son medidos por un Sistema de Referencia Inercial (SRI)
- Con el objeto de mejorar la fiabilidad, hay una redundancia considerable en los equipos. Hay dos SRI operando en paralelo con idénticos hardware y software.

Ejemplo: Ariane 5

- La orientación del sistema de lanzamiento y de sus movimientos en el espacio son medidos por un Sistema de Referencia Inercial (SRI)
- Con el objeto de mejorar la fiabilidad, hay una redundancia considerable en los equipos. Hay dos SRI operando en paralelo con idénticos hardware y software.
- Un SRI está activo y el otro está en stand-by, y si la computadora a bordo detecta que el SRI activo ha fallado inmediatamente conmuta al otro.

Ejemplo: Ariane 5

- La orientación del sistema de lanzamiento y de sus movimientos en el espacio son medidos por un Sistema de Referencia Inercial (SRI)
- Con el objeto de mejorar la fiabilidad, hay una redundancia considerable en los equipos. Hay dos SRI operando en paralelo con idénticos hardware y software.
- Un SRI está activo y el otro está en stand-by, y si la computadora a bordo detecta que el SRI activo ha fallado inmediatamente conmuta al otro.
- También hay dos computadoras a bordo.

Ejemplo: Ariane 5

Ejemplo: Ariane 5

- El primer Ariane 5 explotó en 1996, y con él sus 500 millones de dólares porque el SRI dejo de enviar datos. Esto fue a causa de una **excepción no procesada.**

Ejemplo: Ariane 5

- El primer Ariane 5 explotó en 1996, y con él sus 500 millones de dólares porque el SRI dejo de enviar datos. Esto fue a causa de una **excepción no procesada.**
- Una excepción es una condición que cambia el curso normal de comportamiento.

Ejemplo: Ariane 5

- El primer Ariane 5 explotó en 1996, y con él sus 500 millones de dólares porque el SRI dejo de enviar datos. Esto fue a causa de una **excepción no procesada.**
- Una excepción es una condición que cambia el curso normal de comportamiento.
- Esta excepción fue generada por un ***overflow***: una velocidad estaba guardada como **double** (64 bits) y se convertía en un **short** en una porción del programa.

Ejemplo: Ariane 5

- El primer Ariane 5 explotó en 1996, y con él sus 500 millones de dólares porque el SRI dejo de enviar datos. Esto fue a causa de una **excepción no procesada.**
- Una excepción es una condición que cambia el curso normal de comportamiento.
- Esta excepción fue generada por un ***overflow***: una velocidad estaba guardada como **double** (64 bits) y se convertía en un **short** en una porción del programa.
- Esta función solo funcionaba al momento del despegue.

Ejemplo: Ariane 5

- El primer Ariane 5 explotó en 1996, y con él sus 500 millones de dólares porque el SRI dejo de enviar datos. Esto fue a causa de una **excepción no procesada.**
- Una excepción es una condición que cambia el curso normal de comportamiento.
- Esta excepción fue generada por un ***overflow***: una velocidad estaba guardada como **double** (64 bits) y se convertía en un **short** en una porción del programa.
- Esta función solo funcionaba al momento del despegue.
- Este bug está clasificado entre los 10 más grandes de la historia.

Estructuras de datos básicas

Estructuras de datos básicas

- Manera de **organizar** datos en una computadora de tal manera que puedan ser utilizados **eficientemente**.

Estructuras de datos básicas

- Manera de **organizar** datos en una computadora de tal manera que puedan ser utilizados **eficientemente**.

- Una estructura correctamente utilizada nos permitirá usar el **algoritmo más eficiente.**

Estructuras de datos básicas

- Manera de **organizar** datos en una computadora de tal manera que puedan ser utilizados **eficientemente**.

- Una estructura correctamente utilizada nos permitirá usar el **algoritmo más eficiente.**

- La elección de la estructura correcta empieza frecuentemente con la elección de un **tipo de datos abstracto.**

Tipos de Datos Abstractos (**ADT**s)

Tipos de Datos Abstractos (**ADT**s)

- Especificación de un conjunto de datos y de un conjunto de operaciones para realizar a los datos.

Tipos de Datos Abstractos (**ADT**s)

- Especificación de un conjunto de datos y de un conjunto de operaciones para realizar a los datos.

- Es abstracta en el sentido que es independiente de sus varias **implementaciones concretas**.

Tipos de Datos Abstractos (**ADT**s)

- Especificación de un conjunto de datos y de un conjunto de operaciones para realizar a los datos.

- Es abstracta en el sentido que es independiente de sus varias **implementaciones concretas**.

- Enfocarse en lo **que** hace y no en **como** lo hace.Ejemplo: pilas, colas, etc.

Tipos de Datos Abstractos (**ADT**s)

- Especificación de un conjunto de datos y de un conjunto de operaciones para realizar a los datos.

- Es abstracta en el sentido que es independiente de sus varias **implementaciones concretas**.

- Enfocarse en lo **que** hace y no en **como** lo hace.Ejemplo: pilas, colas, etc.

- Los tipos de datos **concretos** sirven para implementar los ADTs. Son la implementación directa de un concepto simple. Ejemplo: arreglos, listas ligadas, etc.

Estructuras de datos concretas: **struct**

Estructuras de datos concretas: **struct**

- son tipos adicionales que utilizamos para definir colecciones de datos, muchas veces **heterogéneos.**

Estructuras de datos concretas: **struct**

- son tipos adicionales que utilizamos para definir colecciones de datos, muchas veces **heterogéneos.**
- esto nos permite manipular la colección como unidad, pero al mismo tiempo nos da acceso a los componentes individuales.

Estructuras de datos concretas: **struct**

- son tipos adicionales que utilizamos para definir colecciones de datos, muchas veces **heterogéneos.**

- esto nos permite manipular la colección como unidad, pero al mismo tiempo nos da acceso a los componentes individuales.

- ***Ejemplo***: al procesar datos geométricos es útil tener la noción de **puntos en un plano.** Para esto podemos escribir:

Estructuras de datos concretas: **struct**

- son tipos adicionales que utilizamos para definir colecciones de datos, muchas veces **heterogéneos.**

- esto nos permite manipular la colección como unidad, pero al mismo tiempo nos da acceso a los componentes individuales.

- ***Ejemplo***: al procesar datos geométricos es útil tener la noción de **puntos en un plano.** Para esto podemos escribir:

    - `struct point { float x; float y; };`

Estructuras de datos concretas: **struct**

- son tipos adicionales que utilizamos para definir colecciones de datos, muchas veces **heterogéneos.**

- esto nos permite manipular la colección como unidad, pero al mismo tiempo nos da acceso a los componentes individuales.

- ***Ejemplo***: al procesar datos geométricos es útil tener la noción de **puntos en un plano.** Para esto podemos escribir:

    - `struct point { float x; float y; };`

- Para utilizar el tipo ***point*** en operaciones dentro de nuestra aplicación podemos declaralas como dos variables:

Estructuras de datos concretas: **struct**

- son tipos adicionales que utilizamos para definir colecciones de datos, muchas veces **heterogéneos.**
- esto nos permite manipular la colección como unidad, pero al mismo tiempo nos da acceso a los componentes individuales.
- ***Ejemplo***: al procesar datos geométricos es útil tener la noción de **puntos en un plano.** Para esto podemos escribir:
    - `struct point { float x; float y; };`
- Para utilizar el tipo ***point*** en operaciones dentro de nuestra aplicación podemos declaralas como dos variables:
    - `struct point a, b;`

Estructuras de datos concretas: **struct**

Estructuras de datos concretas: **struct**

- Para referirnos a los miembros individuales de una estructura por su nombre podemos escribir:

Estructuras de datos concretas: **struct**

- Para referirnos a los miembros individuales de una estructura por su nombre podemos escribir:

- `a.x=1.0; a.y=1.0; b.x=4.0, b.y=5.0;`

Estructuras de datos concretas: **struct**

- Para referirnos a los miembros individuales de una estructura por su nombre podemos escribir:

- `a.x=1.0; a.y=1.0; b.x=4.0, b.y=5.0;`

- para representar los puntos con coordenadas *a(1,1)* y *b(4,5)*

Estructuras de datos concretas: **struct**

- Para referirnos a los miembros individuales de una estructura por su nombre podemos escribir:

- `a.x=1.0; a.y=1.0; b.x=4.0, b.y=5.0;`

- para representar los puntos con coordenadas *a(1,1)* y *b(4,5)*

- Podemos también pasar estructuras como argumentos de funciones:

Estructuras de datos concretas: **struct**

- Para referirnos a los miembros individuales de una estructura por su nombre podemos escribir:

- `a.x=1.0; a.y=1.0; b.x=4.0, b.y=5.0;`

- para representar los puntos con coordenadas *a(1,1)* y *b(4,5)*

- Podemos también pasar estructuras como argumentos de funciones:

```
float distance ( point a, point b )
```

Estructuras de datos concretas: **struct**

- Para referirnos a los miembros individuales de una estructura por su nombre podemos escribir:

- `a.x=1.0; a.y=1.0; b.x=4.0, b.y=5.0;`

- para representar los puntos con coordenadas *a(1,1)* y *b(4,5)*

- Podemos también pasar estructuras como argumentos de funciones:

```
float distance ( point a, point b )

{
```

Estructuras de datos concretas: **struct**

- Para referirnos a los miembros individuales de una estructura por su nombre podemos escribir:

- `a.x=1.0; a.y=1.0; b.x=4.0, b.y=5.0;`

- para representar los puntos con coordenadas *a(1,1)* y *b(4,5)*

- Podemos también pasar estructuras como argumentos de funciones:

```
float distance ( point a, point b )
{
   float dx=a.x-b.x, dy=a.y-b.y;
```

Estructuras de datos concretas: **struct**

- Para referirnos a los miembros individuales de una estructura por su nombre podemos escribir:

- `a.x=1.0; a.y=1.0; b.x=4.0, b.y=5.0;`

- para representar los puntos con coordenadas *a(1,1)* y *b(4,5)*

- Podemos también pasar estructuras como argumentos de funciones:

```
float distance ( point a, point b )
{
   float dx=a.x-b.x, dy=a.y-b.y;
   return sqrt(dx*dx + dy*dy );
```

Estructuras de datos concretas: **struct**

- Para referirnos a los miembros individuales de una estructura por su nombre podemos escribir:

- `a.x=1.0; a.y=1.0; b.x=4.0, b.y=5.0;`

- para representar los puntos con coordenadas *a(1,1)* y *b(4,5)*

- Podemos también pasar estructuras como argumentos de funciones:

```
float distance ( point a, point b )
{
   float dx=a.x-b.x, dy=a.y-b.y;
   return sqrt(dx*dx + dy*dy );
}
```

Estructuras de datos concretas: **arreglos**

Estructuras de datos concretas: **arreglos**

- Es la estructura de datos fundamental.

Estructuras de datos concretas: **arreglos**

- Es la estructura de datos fundamental.
- Colección de datos del mismo tipo, almacenados de manera contigua y que se pueden acceder por un índice. Están presentes en la mayoría de los lenguajes modernos de programación.

Estructuras de datos concretas: **arreglos**

- Es la estructura de datos fundamental.
- Colección de datos del mismo tipo, almacenados de manera contigua y que se pueden acceder por un índice. Están presentes en la mayoría de los lenguajes modernos de programación.
- Tienen correspondencia directa con los sistemas de memoria de virtualmente todas las computadoras.

Estructuras de datos concretas: **arreglos**

- Es la estructura de datos fundamental.
- Colección de datos del mismo tipo, almacenados de manera contigua y que se pueden acceder por un índice. Están presentes en la mayoría de los lenguajes modernos de programación.
- Tienen correspondencia directa con los sistemas de memoria de virtualmente todas las computadoras.
- Acceso **inmediato** a los datos vía el indice.

Estructuras de datos concretas: **arreglos**

- Es la estructura de datos fundamental.
- Colección de datos del mismo tipo, almacenados de manera contigua y que se pueden acceder por un índice. Están presentes en la mayoría de los lenguajes modernos de programación.
- Tienen correspondencia directa con los sistemas de memoria de virtualmente todas las computadoras.
- Acceso **inmediato** a los datos vía el indice.
- Acceso **secuencial** práctico (ciclos `for`).

Estructuras de datos concretas: **arreglos**

- Es la estructura de datos fundamental.
- Colección de datos del mismo tipo, almacenados de manera contigua y que se pueden acceder por un índice. Están presentes en la mayoría de los lenguajes modernos de programación.
- Tienen correspondencia directa con los sistemas de memoria de virtualmente todas las computadoras.
- Acceso **inmediato** a los datos vía el indice.
- Acceso **secuencial** práctico (ciclos `for`).
- Implican hacer pruebas por índice.

Estucturas de datos concretas: **arreglos**

Estucturas de datos concretas: **arreglos**

- Ejemplo del uso eficiente de un arreglo: **criba de Eratóstenes**
- algoritmo que permite hallar todos los **números primos** menores que un **número natural dado** N.

Estucturas de datos concretas: **arreglos**

- Objetivo Final: Poner a[i] a **1** si i es primo y **0** si no lo es.

1. Poner a **1** todos los elementos del arreglo, para indicar que ninguno se conoce como no-primo.

2. Poner a **0** los elementos del arreglo que correspondan a los indices conocidos como no-primos (los múltiplos de los primos conocidos)

3. Si a[i] es **1** después de que los múltiplos o primos mas pequeños han sido puestos a **0** se dice que es un primo.

4. El proceso termina cuando el cuadrado del mayor número confirmado como primo es mayor que N.

Estucturas de datos concretas: **arreglos**

```
#include <iostream>
static const int N=1000;
int main() {
   int i, a[N];
   for (i=2; i<N; i++) a[i]=1;
   for (i=2; i<N; i++)
     if(a[i])
        for (int j=i; j*i<N; j++) a[i*j] = 0;
   for (i=2; i<N; i++)
     if(a[i]) cout << " " << i;
   cout << endl;
   }
```

Estucturas de datos concretas: **arreglos**

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Prime numbers

Estucturas de datos concretas: **arreglos**

```
int main( int argc, char *argv[])
{
   int i, N=atoi(argv[1]);
   int *a = new int[N];
   if (a == 0)
     { cout << “out of memory” << endl; return 0;}

     ...

     delete [] a;
}
```

- Es dificil re-arreglar datos (insertar)
- se pueden utilizar como entrada de funciones, hacer arreglos de arreglos y arreglos de estructuras.

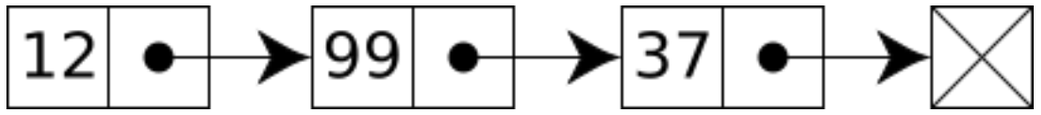
Estucturas de datos concretas: **listas ligadas**

- útiles cuando queremos recorrer una colección de elementos secuencialmente, uno a uno.
- cada elemento contiene información de cómo llegar al siguiente elemento.
- permite re-arreglar los datos eficientemente.
- para acceder a un dato random de la lista hay que recorrerla desde el inicio.
- estructura auto-referente.
- acceso secuencial tal cómo para un arreglo.

•Una **lista ligada** es un conjunto de elementos donde cada elemento es parte de un **nodo** que también contiene un **link** a un nodo.

Estucturas de datos concretas: **listas ligadas**

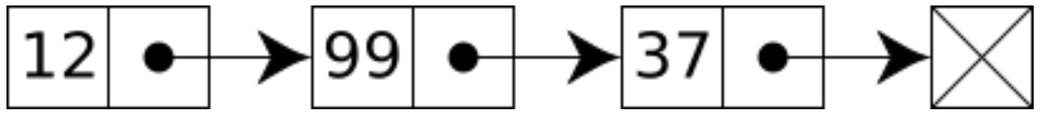
- los elementos pueden no estar arreglados secuencialmente en la memoria.



- convenciones para definir el último nodo:
  - es un ***link nulo*** que no apunta a otro nodo.
  - se refiere a un ***nodo falso (dummy)*** que no contiene elemento.
  - se refiere al primer elemento de la lista, definiendo una ***lista circular***.

Estucturas de datos concretas: **listas ligadas**

- los elementos pueden no estar arreglados secuencialmente en la memoria.



- convenciones para definir el último nodo:
  - es un ***link nulo*** que no apunta a otro nodo.
  - se refiere a un ***nodo falso (dummy)*** que no contiene elemento.
  - se refiere al primer elemento de la lista, definiendo una ***lista circular***.

```
typedef int Item;  // solo un ejemplo


struct node {
 Item item;
 node *next;
};
typedef node* link;
```

Estucturas de datos concretas: **listas ligadas**

- Normalmente, en lenguajes como C++ y Java, no solo **reservamos memoria** para un elemento, sino también **inicializamos** los datos dentro. Para esto se difine un ***constructor*** en cada ***struct*** que definamos.
- Un **constructor** es una función que se define dentro de una estructura, y que lleva el mismo nombre de ésta.
- El propósito es dar valores iniciales a la estructura y evitar problemas de variables no-inicializadas.

```
struct node {
 Item item; node *next;
 node ( Item x; node *t){
      item = x;
      next = t;
    }
};
typedef node* link;
```

```
link s = NULL;
link t = new node(x,s)
```

Estucturas de datos concretas: **listas ligadas**

Estucturas de datos concretas: **listas ligadas**

- Una vez que la lista es creada, ¿cómo nos referimos a la información que contiene?

Estucturas de datos concretas: **listas ligadas**

- Una vez que la lista es creada, ¿cómo nos referimos a la información que contiene?

- El ***item*** lo podemos acceder por medio de la instrucción **(*x).item**.

Estucturas de datos concretas: **listas ligadas**

- Una vez que la lista es creada, ¿cómo nos referimos a la información que contiene?

- El ***item*** lo podemos acceder por medio de la instrucción **(*x).item**.

- El ***link*** lo podemos acceder por medio de la instrucción **(*x).link**.

Estucturas de datos concretas: **listas ligadas**

- Una vez que la lista es creada, ¿cómo nos referimos a la información que contiene?

- El ***item*** lo podemos acceder por medio de la instrucción **(*x).item**.

- El ***link*** lo podemos acceder por medio de la instrucción **(*x).link**.

- Estas operaciones son tan utilizadas que en C++ se proporciona una abreviatura, que se presenta como: **x->item** y **x->link.**

arreglos

arreglos

- Probablemente la forma mas fácil de mantener una secuencia de elementos (lista).

arreglos

- Probablemente la forma mas fácil de mantener una secuencia de elementos (lista).
- Nos permite responder preguntas del tipo:

arreglos

- Probablemente la forma mas fácil de mantener una secuencia de elementos (lista).
- Nos permite responder preguntas del tipo:
  - ¿Cuál es el i-ésimo elemento de la lista? en tiempo ***O(1)***, con un acceso directo al valor de A[i].

arreglos

- Probablemente la forma mas fácil de mantener una secuencia de elementos (lista).
- Nos permite responder preguntas del tipo:
  - ¿Cuál es el i-ésimo elemento de la lista? en tiempo ***O(1)***, con un acceso directo al valor de A[i].
  - ¿El elemento *e* **pertenece** a la lista? (si es igual a *A[i]* para algún *i*), necesitamos verificar los elementos uno a uno en tiempo ***O(n)*** **suponiendo** que no sabemos el orden de los elementos de ***A.***

arreglos

- Probablemente la forma mas fácil de mantener una secuencia de elementos (lista).
- Nos permite responder preguntas del tipo:
  - ¿Cuál es el i-ésimo elemento de la lista? en tiempo ***O(1)***, con un acceso directo al valor de A[i].
  - ¿El elemento *e* **pertenece** a la lista? (si es igual a *A[i]* para algún *i*), necesitamos verificar los elementos uno a uno en tiempo ***O(n)*** **suponiendo** que no sabemos el orden de los elementos de ***A.***
  - Si los elementos están ordenados podemos determinar si el elemento **pertenece** o no al arreglo en un tiempo ***O(log n)*** con una búsqueda binaria.

arreglos vs listas

arreglos vs listas

- Los arreglos no son adecuados para mantener una lista de elementos que cambia con el tiempo.

arreglos vs listas

- Los arreglos no son adecuados para mantener una lista de elementos que cambia con el tiempo.
- Una forma más adecuada para mantener un **conjunto dinámico** es una **lista ligada.**

arreglos vs listas

- Los arreglos no son adecuados para mantener una lista de elementos que cambia con el tiempo.
- Una forma más adecuada para mantener un **conjunto dinámico** es una **lista ligada.**
- Una **lista ligada** es una estructura de datos en la que los objetos están arreglados en orden lineal.

arreglos vs listas

- Los arreglos no son adecuados para mantener una lista de elementos que cambia con el tiempo.
- Una forma más adecuada para mantener un **conjunto dinámico** es una **lista ligada.**
- Una **lista ligada** es una estructura de datos en la que los objetos están arreglados en orden lineal.
- Está formada de **nodos**. Cada nodo consta de dos componentes: un **elemento** y un **apuntador** a otro nodo.

arreglos vs. listas ligadas

arreglos vs. listas ligadas

- En un **arreglo** el orden esta determinado por sus **índices**, en una **lista ligada** el orden se determina por un apuntador a cada objeto.

arreglos vs. listas ligadas

- En un **arreglo** el orden esta determinado por sus **índices**, en una **lista ligada** el orden se determina por un apuntador a cada objeto.
- Una **lista ligada** nos permite re-ordenar los datos de forma eficiente.

arreglos vs. listas ligadas

- En un **arreglo** el orden esta determinado por sus **índices**, en una **lista ligada** el orden se determina por un apuntador a cada objeto.
- Una **lista ligada** nos permite re-ordenar los datos de forma eficiente.
- Un **arreglo** nos permite acceder a uno de sus elementos por medio del índice.

arreglos vs. listas ligadas

- En un **arreglo** el orden esta determinado por sus **índices**, en una **lista ligada** el orden se determina por un apuntador a cada objeto.
- Una **lista ligada** nos permite re-ordenar los datos de forma eficiente.
- Un **arreglo** nos permite acceder a uno de sus elementos por medio del índice.
- se pueden **insertar elementos** a la lista **indefinidamente** mientras que un arreglo al llenarse debe ser redimensionado.

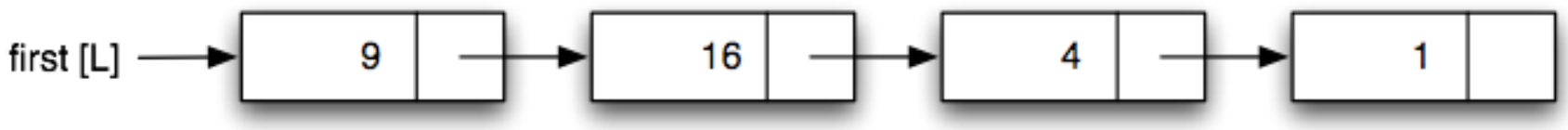
arreglos vs. listas ligadas

- En un **arreglo** el orden esta determinado por sus **índices**, en una **lista ligada** el orden se determina por un apuntador a cada objeto.
- Una **lista ligada** nos permite re-ordenar los datos de forma eficiente.
- Un **arreglo** nos permite acceder a uno de sus elementos por medio del índice.
- se pueden **insertar elementos** a la lista **indefinidamente** mientras que un arreglo al llenarse debe ser redimensionado.
- un **arreglo** permite **acceso aleatorio**, mientras que una **lista ligada** permite solo un acceso secuencial a los elementos.

arreglos vs. listas ligadas

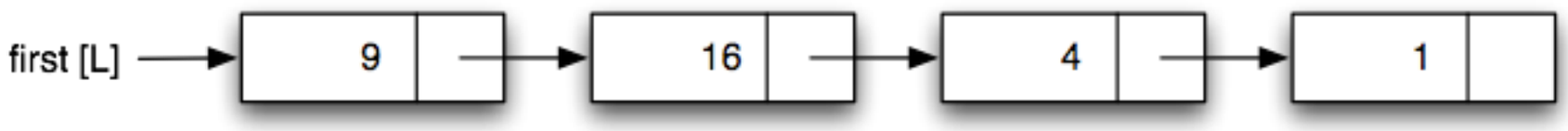
- En un **arreglo** el orden esta determinado por sus **índices**, en una **lista ligada** el orden se determina por un apuntador a cada objeto.
- Una **lista ligada** nos permite re-ordenar los datos de forma eficiente.
- Un **arreglo** nos permite acceder a uno de sus elementos por medio del índice.
- se pueden **insertar elementos** a la lista **indefinidamente** mientras que un arreglo al llenarse debe ser redimensionado.
- un **arreglo** permite **acceso aleatorio**, mientras que una **lista ligada** permite solo un acceso secuencial a los elementos.
- las **listas ligadas** necesitan memoria adicional para almacenar las **referencias**. Esto las hace poco recomendables para char o bool.

listas ligadas



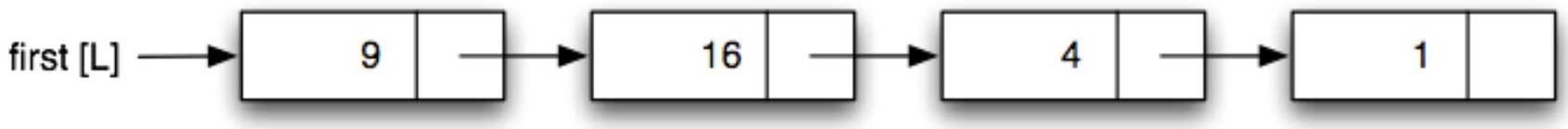
listas ligadas

- Realiza una lista de elementos, por ejemplo, (9,16,4,1)



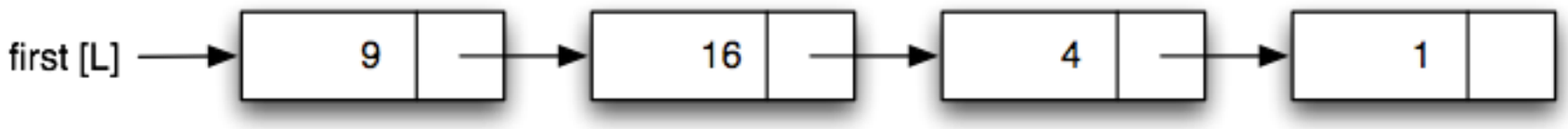
listas ligadas

- Realiza una lista de elementos, por ejemplo, (9,16,4,1)
- **Insertar** y **remover** elementos al inicio de la lista requieren un tiempo ***O(1)***.



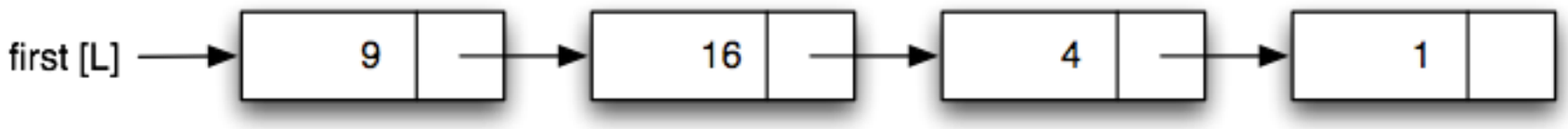
listas ligadas

- Realiza una lista de elementos, por ejemplo, (9,16,4,1)
- **Insertar** y **remover** elementos al inicio de la lista requieren un tiempo ***O(1)***.
- Una **búsqueda** requiere iterar a lo largo de la lista: tiempo ***O(n)***.



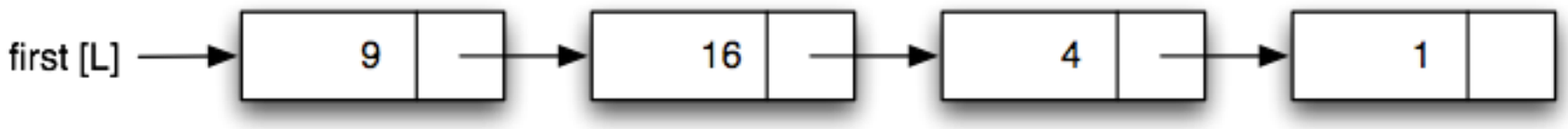
listas ligadas

- Realiza una lista de elementos, por ejemplo, (9,16,4,1)
- **Insertar** y **remover** elementos al inicio de la lista requieren un tiempo ***O(1)***.
- Una **búsqueda** requiere iterar a lo largo de la lista: tiempo ***O(n)***.
- La lista ligada puede ser iterada del frente hacia atrás.



listas ligadas

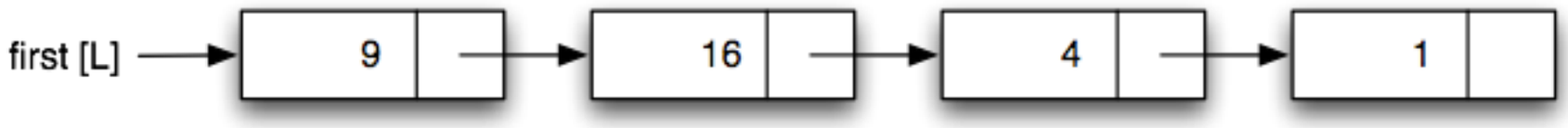
- Realiza una lista de elementos, por ejemplo, (9,16,4,1)
- **Insertar** y **remover** elementos al inicio de la lista requieren un tiempo ***O(1)***.
- Una **búsqueda** requiere iterar a lo largo de la lista: tiempo ***O(n)***.
- La lista ligada puede ser iterada del frente hacia atrás.
- El bloque básico de un **nodo** contiene:



listas ligadas

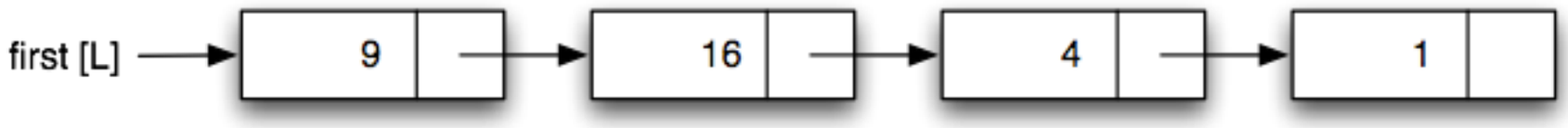
- Realiza una lista de elementos, por ejemplo, (9,16,4,1)
- **Insertar** y **remover** elementos al inicio de la lista requieren un tiempo ***O(1)***.
- Una **búsqueda** requiere iterar a lo largo de la lista: tiempo ***O(n)***.
- La lista ligada puede ser iterada del frente hacia atrás.
- El bloque básico de un **nodo** contiene:

  ➡ **item**: algún tipo de datos

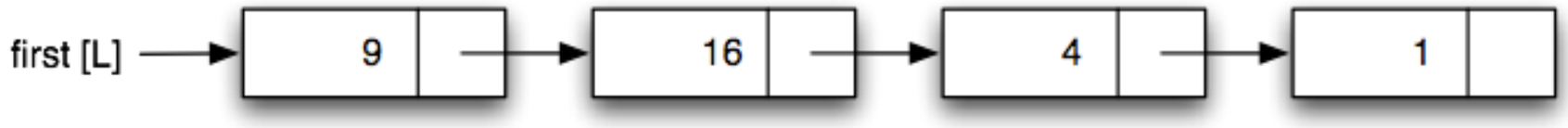


listas ligadas

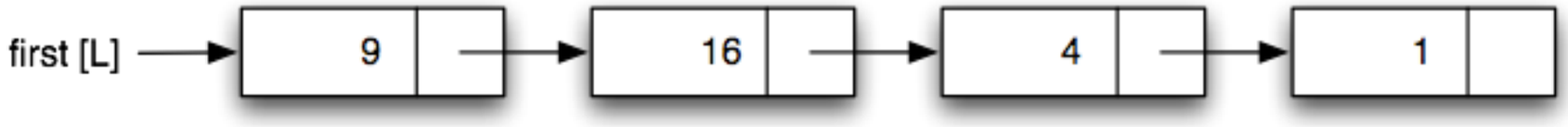
- Realiza una lista de elementos, por ejemplo, (9,16,4,1)
- **Insertar** y **remover** elementos al inicio de la lista requieren un tiempo ***O(1)***.
- Una **búsqueda** requiere iterar a lo largo de la lista: tiempo ***O(n)***.
- La lista ligada puede ser iterada del frente hacia atrás.
- El bloque básico de un **nodo** contiene:
  - ➡ **item:** algún tipo de datos
  - ➡ **next**: un ***apuntador*** al siguiente nodo, o un **apuntador nulo** si el nodo está al final de la lista.



listas ligadas

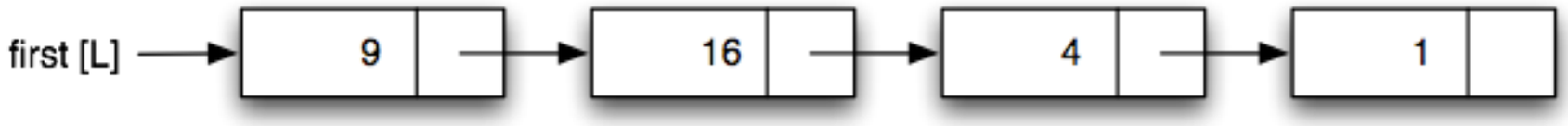


listas ligadas



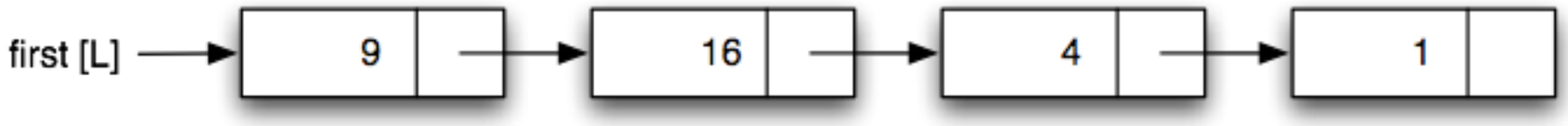
- Cuentan también de un apuntador **first o head** que apunta al primer elemento.

listas ligadas



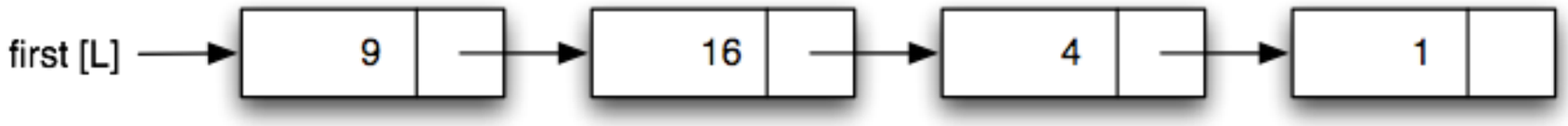
- Cuentan también de un apuntador **first o head** que apunta al primer elemento.
- Una lista se recorre apuntando primero a **first** y siguiento el apuntador **next** de cada elemento hasta llegar a ***null***. Esto es O(n).

listas ligadas



- Cuentan también de un apuntador **first o head** que apunta al primer elemento.
- Una lista se recorre apuntando primero a **first** y siguiento el apuntador **next** de cada elemento hasta llegar a ***null***. Esto es O(n).
- La forma genérica de **implementar** una **lista ligada** cuando no sabemos su tamaño es reservar una **estructura *e*** para cada elemento que queramos incluir en la lista.

listas ligadas



- Cuentan también de un apuntador **first o head** que apunta al primer elemento.
- Una lista se recorre apuntando primero a **first** y siguiento el apuntador **next** de cada elemento hasta llegar a ***null***. Esto es O(n).
- La forma genérica de **implementar** una **lista ligada** cuando no sabemos su tamaño es reservar una **estructura *e*** para cada elemento que queramos incluir en la lista.
- Esta estructura contendrá un campo **e.val** que será el valor del nodo y un campo ***e.*Next** que será el apuntador al siguiente nodo.

Definición

```
struct node {
    Item item;
    node *next;
    node (Item x, node *t) {
        item = x;
        next = t;
    }
};

typedef node *link;
```

usar **`lLink`** en lugar de **`link`** en algunos compiladores

a) **link s = ... ;**

s

b)

c)

d) (otra forma de hacer c))

Definición

```
struct node {
    Item item;
    node *next;
    node (Item x, node *t) {
        item = x;
        next = t;
    }
};

typedef node *link;
```

usar **`lLink`** en lugar de **`link`** en algunos compiladores

Uso

a) **link s = ... ;**

```
link t = new node(x, s);
```

b)

c)

d) (otra forma de hacer c))

Definición

```
struct node {
    Item item;
    node *next;
    node (Item x, node *t) {
        item = x;
        next = t;
    }
};

typedef node *link;
```

usar **`lLink`** en lugar de **`link`** en algunos compiladores

Uso

a) **link s = ... ;**

```
link t = new node(x, s);
```

b)

```
link t = new node(0,NULL);
```

c)

```
(*t).item = x;
(*t).link = t;
```

d) (otra forma de hacer c))

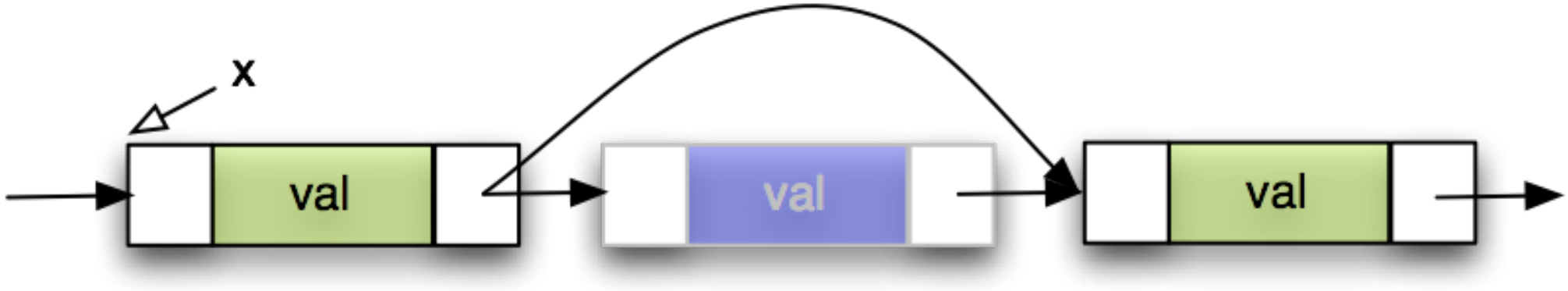
```
t->item = x;
t->link = t;
```

listas ligadas: borrar un nodo t que sigue a x

a)

b)

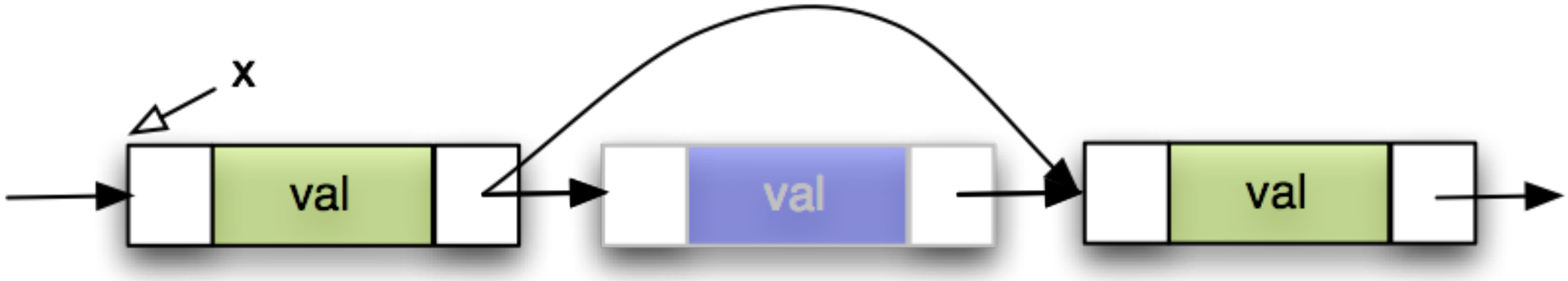
listas ligadas: borrar un nodo t que sigue a x



a)

b)

listas ligadas: borrar un nodo t que sigue a x

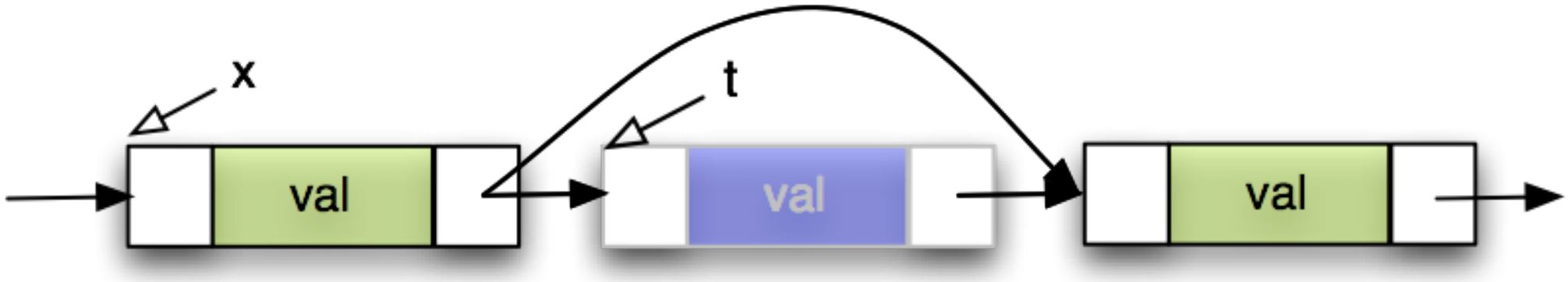


a)

```
t = x->next;
x->next = t->next;
```

b)

listas ligadas: borrar un nodo t que sigue a x

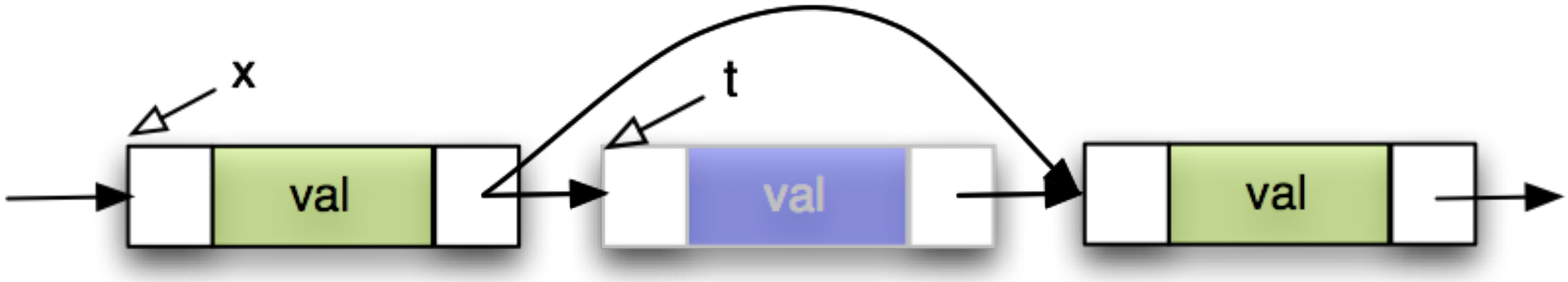


a)

```
t = x->next;
x->next = t->next;
```

b)

listas ligadas: borrar un nodo t que sigue a x

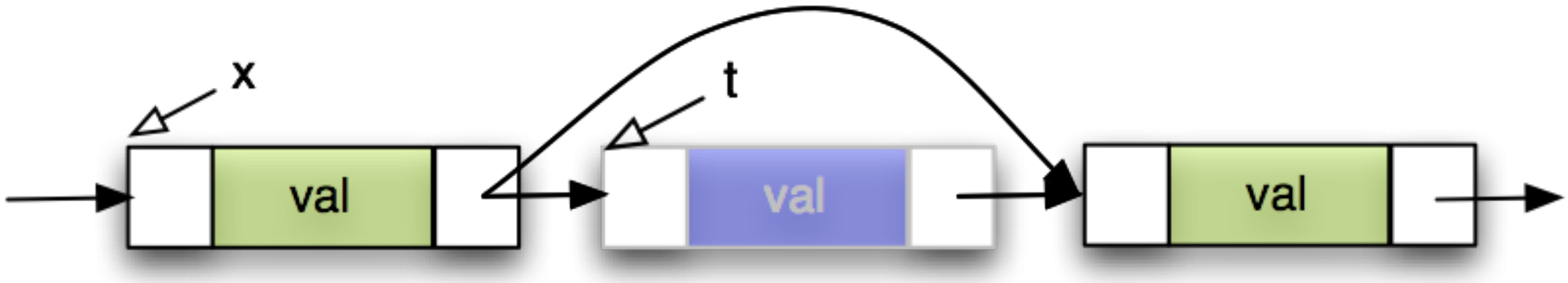


a)

```
t = x->next;
x->next = t->next;

delete t;
```

b)

listas ligadas: borrar un nodo t que sigue a x



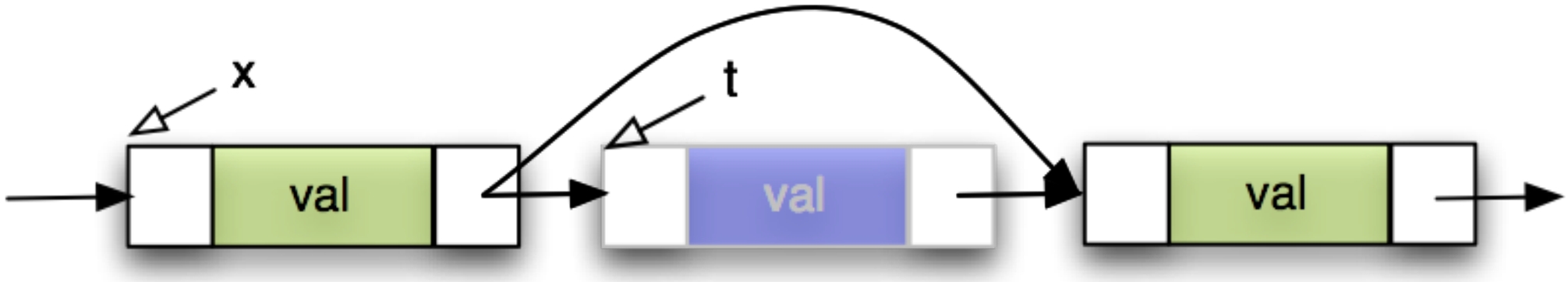
a)

```
t = x->next;
x->next = t->next;

delete t;
```

b)

```
x->next = x->next->next;
```

listas ligadas: borrar un nodo t que sigue a x



a)

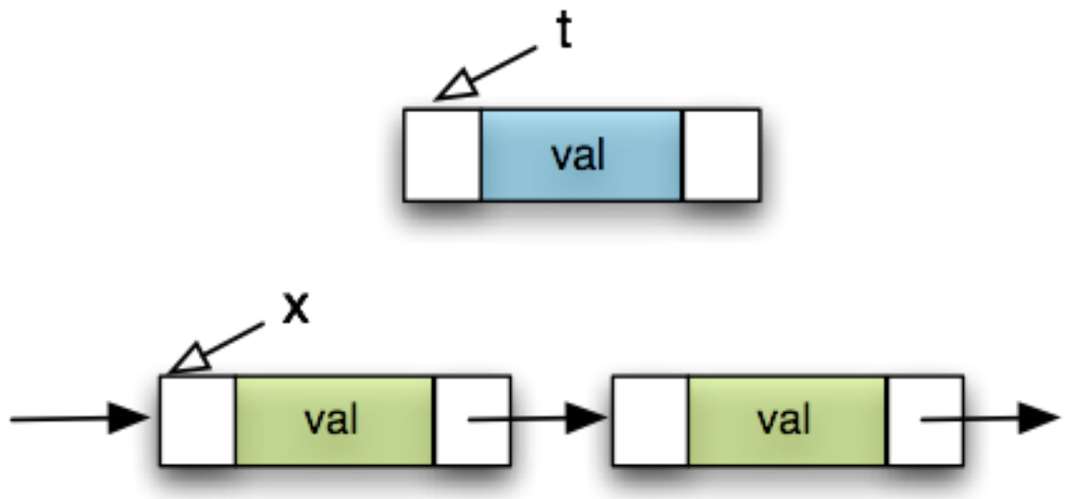
```
t = x->next;
x->next = t->next;

delete t;
```

b)

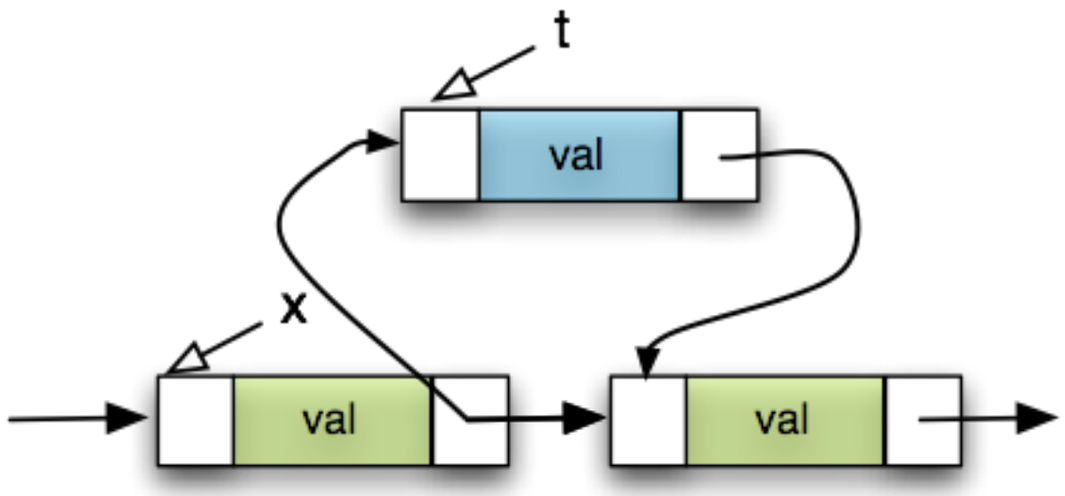
```
x->next = x->next->next;
```

- ¿cuál de las dos maneras es la correcta?

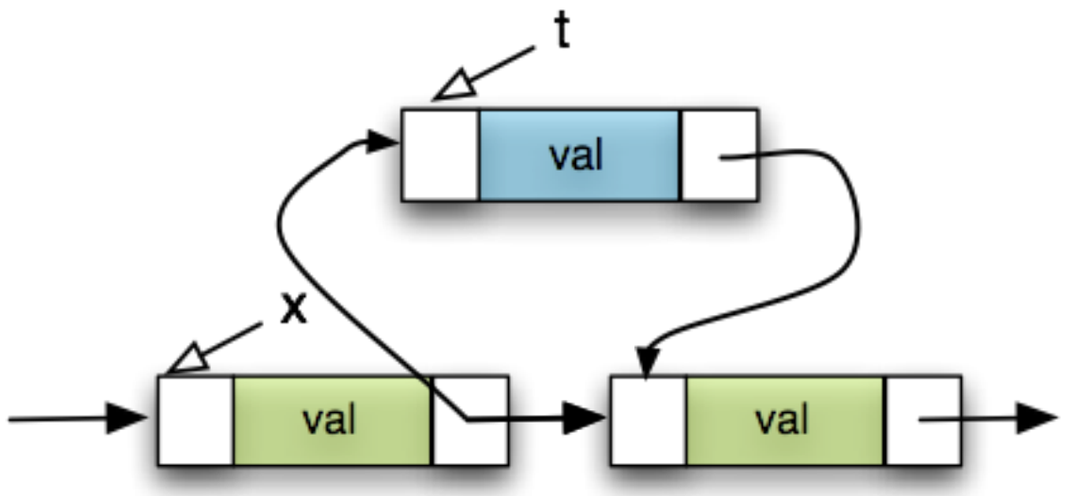
listas ligadas: insertar un nodo t despues de x



listas ligadas: insertar un nodo t despues de x

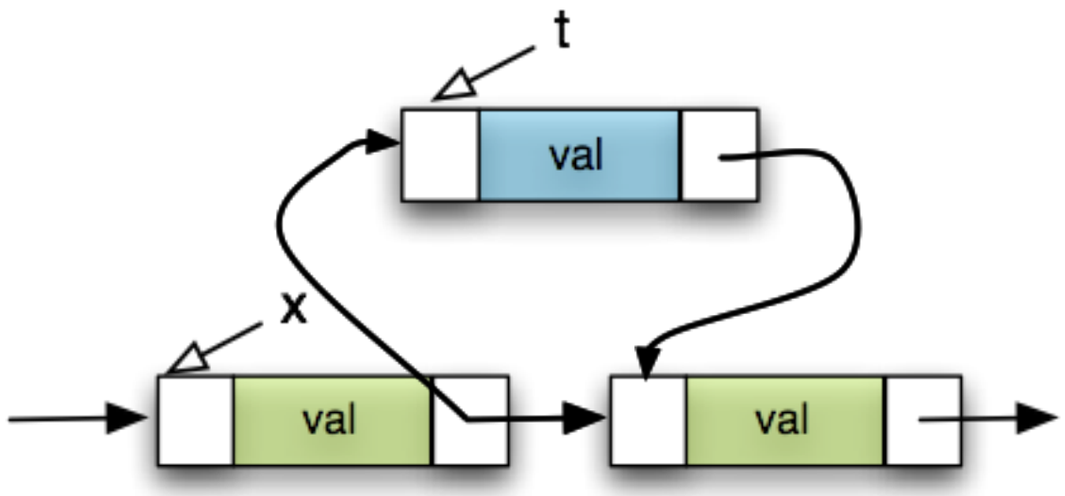


listas ligadas: insertar un nodo t despues de x




```
t->next = x->next;
x->next = t;
```

listas ligadas: insertar un nodo t despues de x




```
t->next = x->next;
x->next = t;
```