Programación Orientada a Objetos

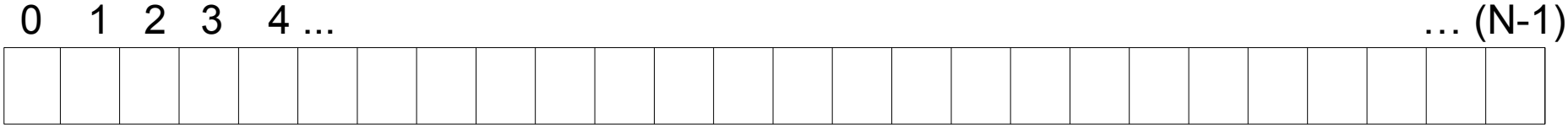
Listas Enlazadas

Teoría 05 - 14/09/2011 - Pablo Novara

Vectores dinámicos en C++

- Los elementos de un vector se almacenan de forma contigua:

```
int *v = new int[N];
```



- Es directo acceder a una posición determinada:

```
cout << *(v+i) <<endl;
```

Cantidad de posiciones (ej: 4)

v+i = 0x11E4 + i x 4 = 0x11F4

Inicio del arreglo

Tamaño del elemento (entero = 4 bytes)

Vectores dinámicos en C++

- Se debe conocer la cantidad (N) de elementos al reservar la memoria:

```
int *v = new int [N];
```

- o usar algunos trucos:
  - sobredimensionar:

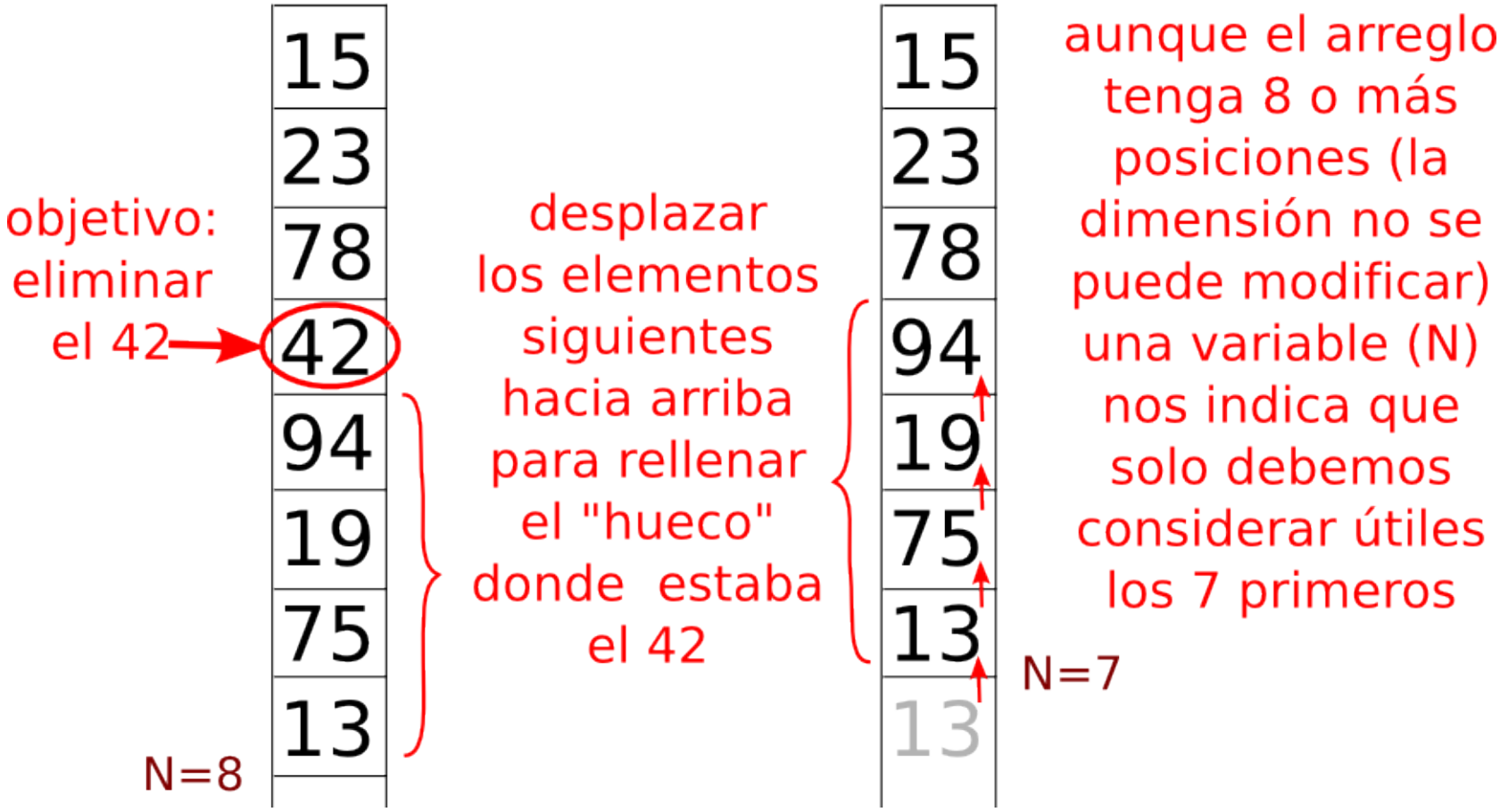
```
int *v = new int [MUCHISIMO];
```

  - realocar cuando sea necesario:

```
int *v2 = new int [N+M];
memcpy(v2,v,sizeof(int)*N);
delete []v; v=v2;
```

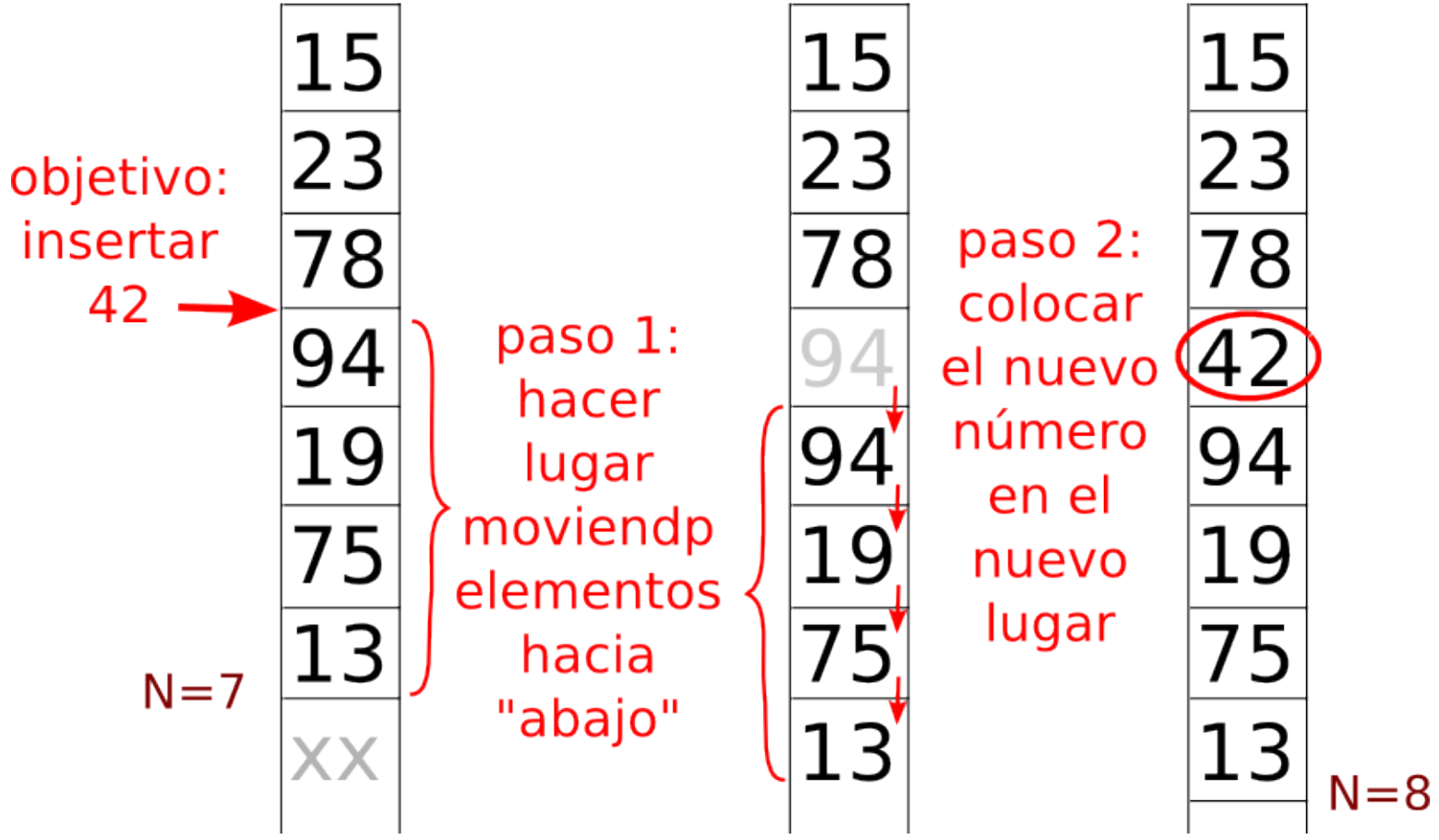
Vectores dinámicos en C++

– Al eliminar un elemento hay que realizar *muchos* desplazamientos y la memoria no se libera realmente:



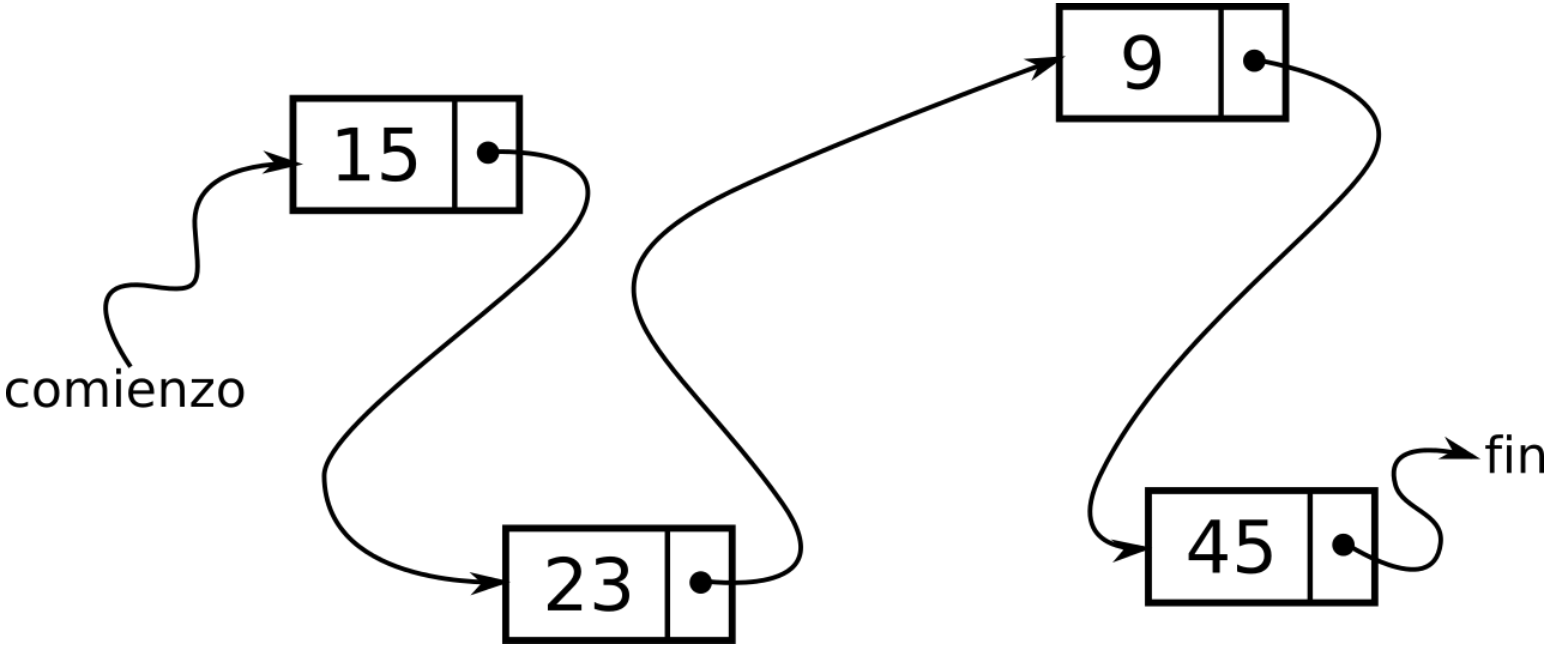
Vectores dinámicos en C++

– Al insertar un elemento hay que realizar *muchos* desplazamientos (sin contar la realocación):



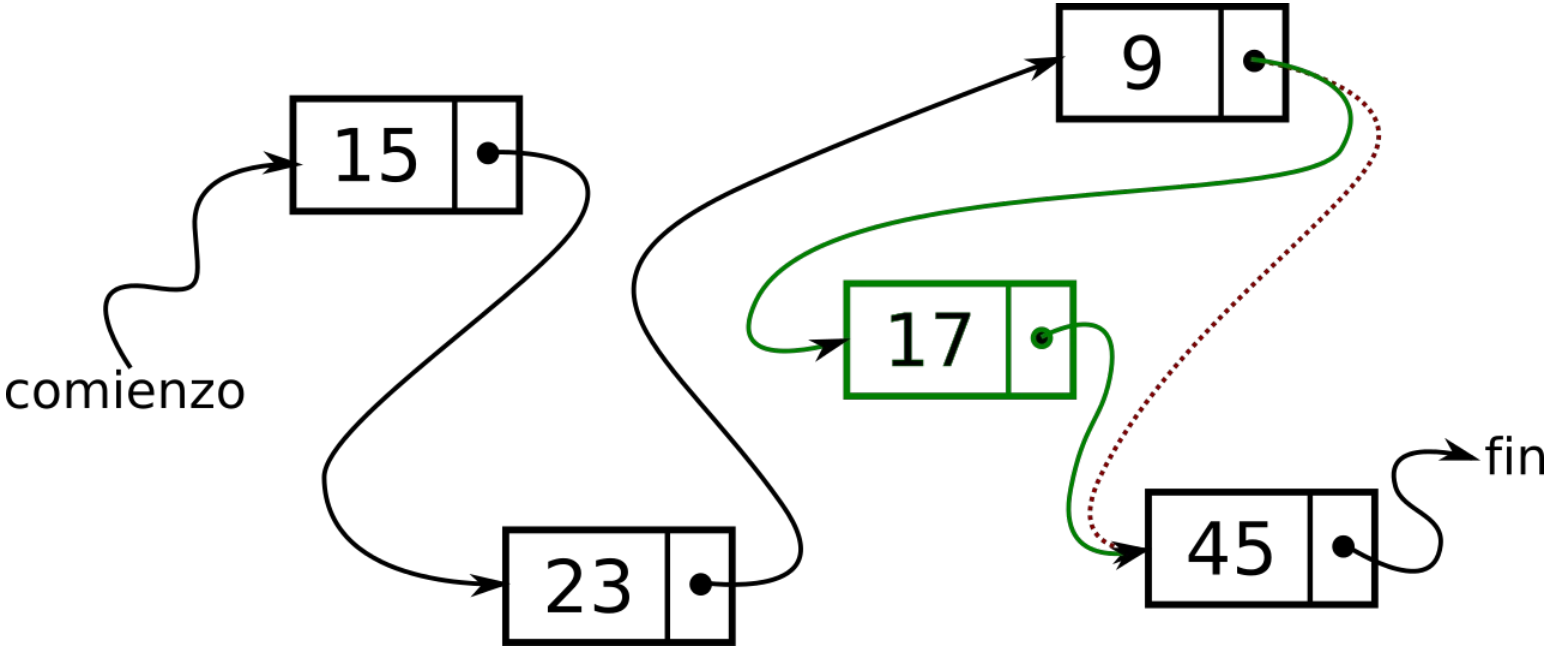
Listas Enlazadas

- La lista consiste en un **conjunto ordenado de nodos**.
- Cada nodo es una estructura que contiene un dato y **un enlace al siguiente nodo**.
- Los nodos no necesitan estar ordenados en memoria.



Listas Enlazadas

– Insertar/Eliminar un nodo en cualquier posición consiste en modificar solo dos enlaces.



Vector Vs Lista

Vector:

+ Acceso **aleatorio**

– Insertar/Eliminar poco eficiente

– Sobredimensión

– Realocación

+ Toda la memoria se reserva y libera en un solo paso

Lista:

– Acceso **secuencial**

+ Insertar/Eliminar muy eficiente

+ No hay sobredimensión

– Para una cantidad fija de datos, ocupan más memoria y requieren más operaciones

– Hay que mantener los enlaces

Estructuras de Datos Basadas en Listas Enlazadas

- **Lista propiamente dicha**: permite insertar y consultar o eliminar en cualquier posición.
- **Pila**: permite insertar y consultar/eliminar solo por un extremo.
- **Cola**: permite insertar por un extremo y consultar/eliminar por el otro.
- **Doble cola**: permite insertar, consultar y eliminar por ambos extremos.

Listas Enlazadas en C++

- Se implementa mediante dos clases:

  - Clase Nodo:
    - contiene un dato y uno o dos punteros a nodos (siguiente/anterior)

  - Clase Lista:
    - punteros privados a los nodos extremos
    - métodos públicos para insertar y eliminar elementos en la lista

Listas Enlazadas en C++

– Clase Nodo (ej. para lista de enteros):

```
struct Nodo {
    int dato;
    Nodo *siguiente;
    Nodo (int d, Nodo *p=NULL) {
        dato=d; siguiente=p;
    }
};
```

¿Que pasa con el principio de ocultación?

Listas Enlazadas en C++

– Clase Pila:

```
class Pila {
    Nodo *tope;
public:
    Pila(); // constructor
    ~Pila(); // destructor
    void Push(int n); // insertar
    int Top(); // ver
    void Pop(); // quitar
    bool Vacia();
};
```

Listas Enlazadas en C++

- Clase Pila:

```
Pila::Pila() : tope(NULL) { }

void Pila::Push(int n) {
    tope = new Nodo(n,tope);
}

void Pila::Pop() {
    Nodo *aux=tope;
    tope=tope->siguiente;
    delete aux;
}
```

Listas Enlazadas en C++

- Clase Pila:

```
int Pila::Top() {
    return tope->dato;
}

bool Pila::Vacia() {
    return tope==NULL;
}

Pila::~Pila() {
    while (!Vacia())
        Pop();
}
```

Listas Enlazadas en C++

– Clase Cola:

```
class Cola {

    Nodo *primero; // primero en la lista enlazada
    Nodo *ultimo; // ultimo en la lista enlazada

public:

    Cola(); // constructor
    ~Cola(); // destructor
    void Push(int n); // insertar
    int Front(); // ver
    void Pop(); // quitar

};
```

Listas Enlazadas en C++

– Clase Cola:

```
Cola::Cola() :primero(NULL),ultimo(NULL) {}
```

- ```
  void Cola::Push(int n) {
      Nodo *aux = new Nodo(n,NULL);
      if (ultimo) ultimo = ultimo->siguiente = aux;
      else ultimo = primero = aux;
  }

  void Cola::Pop() {
      Nodo *aux = primero;
      primero = primero->siguiente;
      delete aux;
      if (!primero) ultimo=NULL;
  }
  ```
  ```

Listas Enlazadas en C++

– Clase Cola:

```
int Cola::Front() {
    return primero->dato;
}

bool Cola::Vacia() {
    return ultimo==NULL;
}

Cola::~Cola() {
    while (ultimo)
        Pop();
}
```