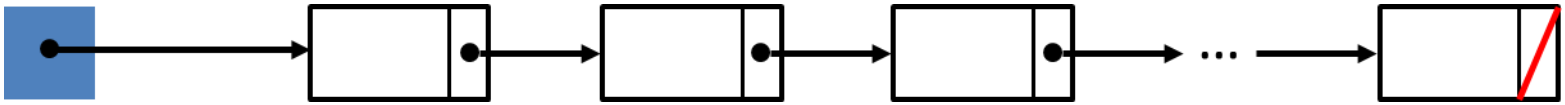
Listas Enlazadas

Cátedra de Lenguaje de Programación 1

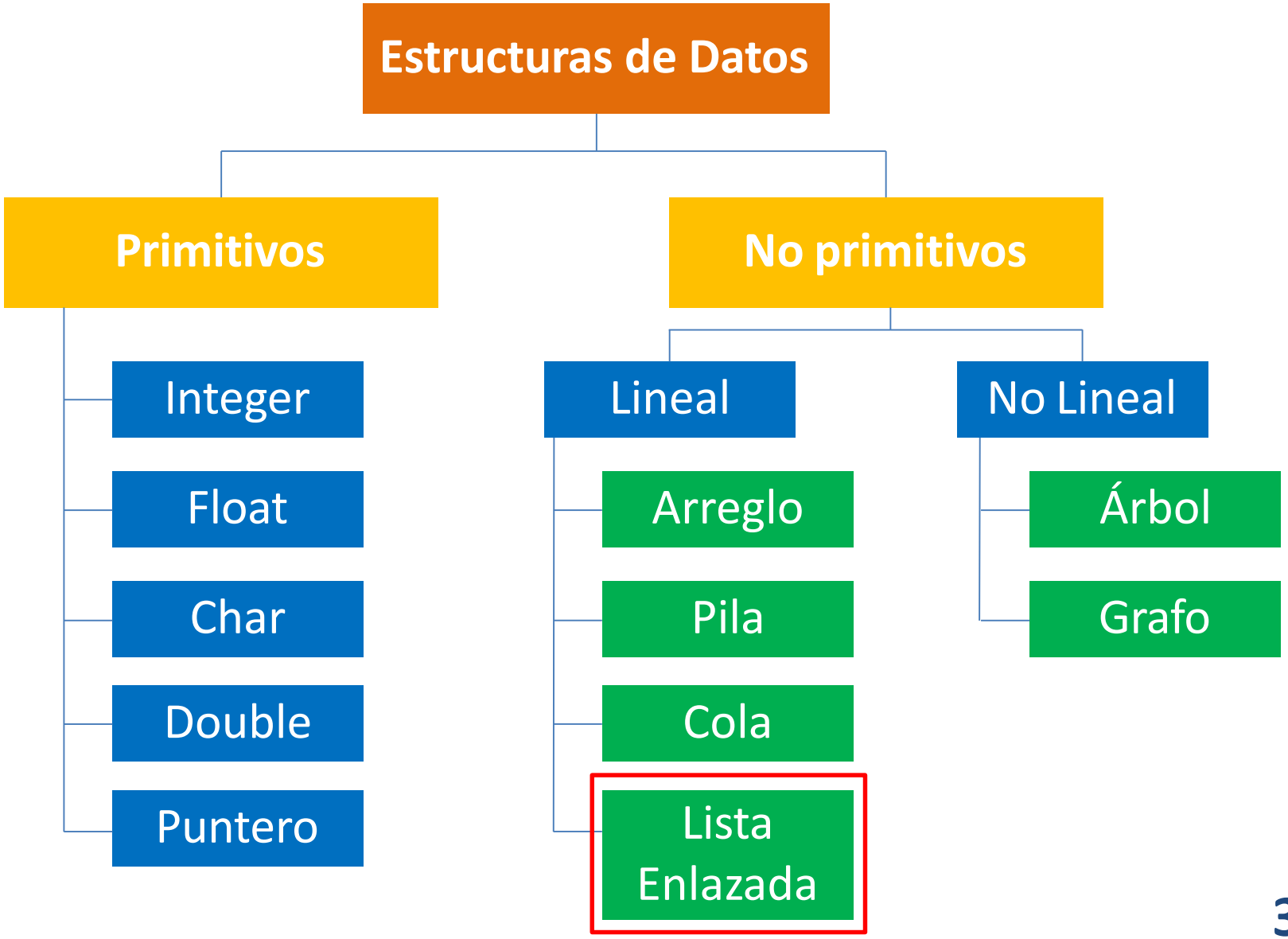
¿Qué veremos?

- Listas Enlazadas
- Operaciones básicas: definición de nodos, creación, búsqueda, inserción, borrado.
- Ejercicios



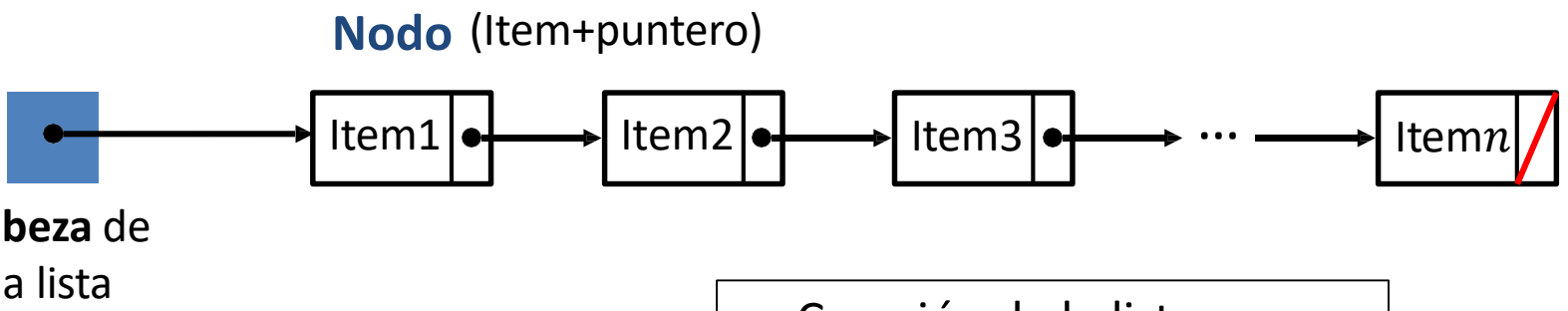
*Se recomienda leer el capítulo 12 de “Fundamentos de Programación. Algoritmos, estructuras de datos y objetos”, Joyanes Aguilar .

Estructuras de datos



Listas enlazadas

Consiste en una secuencia de **nodos**, en los que se guardan campos de datos arbitrarios y una o dos referencias, enlaces o punteros al nodo anterior y/o posterior. Es por ello que es un tipo de dato **autoreferenciado** (pues tiene una referencia a otro dato del mismo tipo).



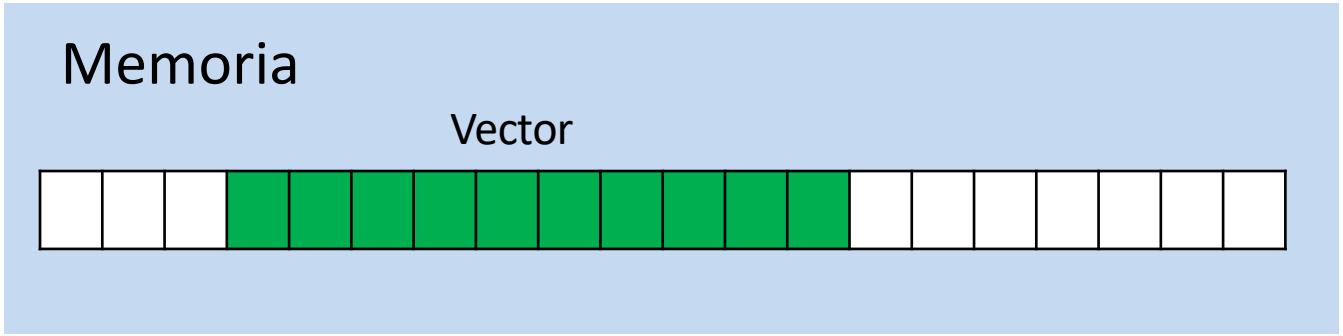
Operaciones principales:

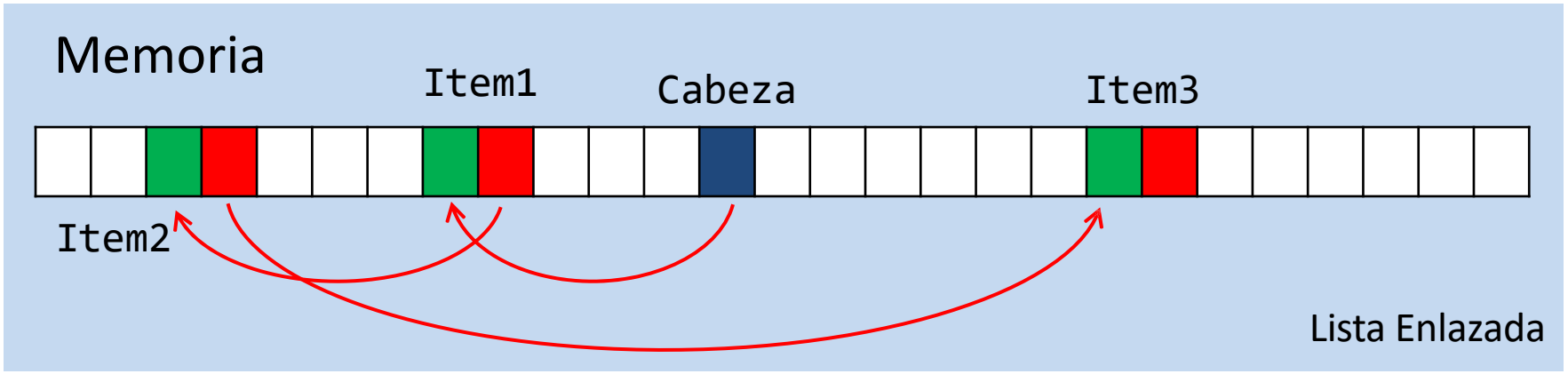
- Creación de la lista
- Inserción de un elemento
- Recorrer lista
- Búsqueda de un elemento
- Borrado de un elemento

https://es.wikipedia.org/wiki/Lista_enlazada

*Esta es sólo una representación esquemática

Listas enlazadas vs Vectores

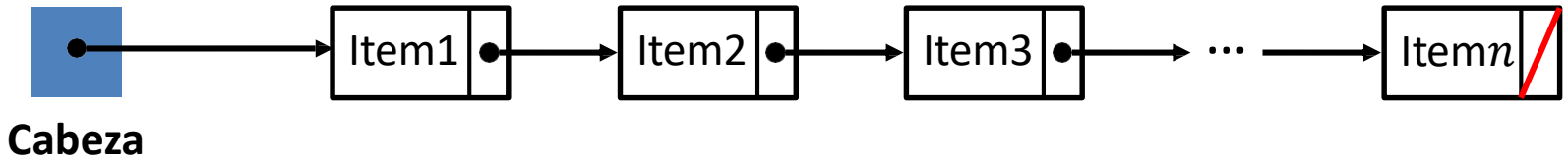




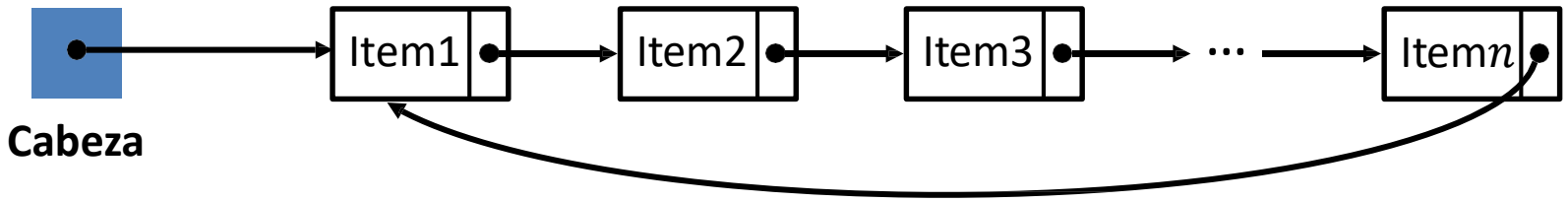
- Los vectores permiten acceso aleatorio.
- Existe un “overhead” para cada elemento de la lista enlazada.
- Se debe redimensionar el vector si se sobrepasa su tamaño máximo.

Tipos de listas enlazadas

Lista simplemente enlazada

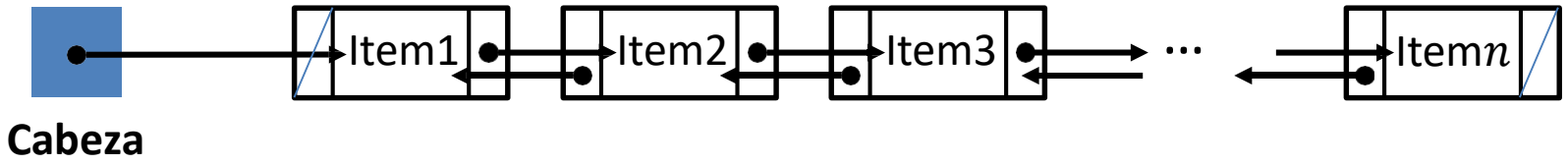


Lista circular simplemente enlazada

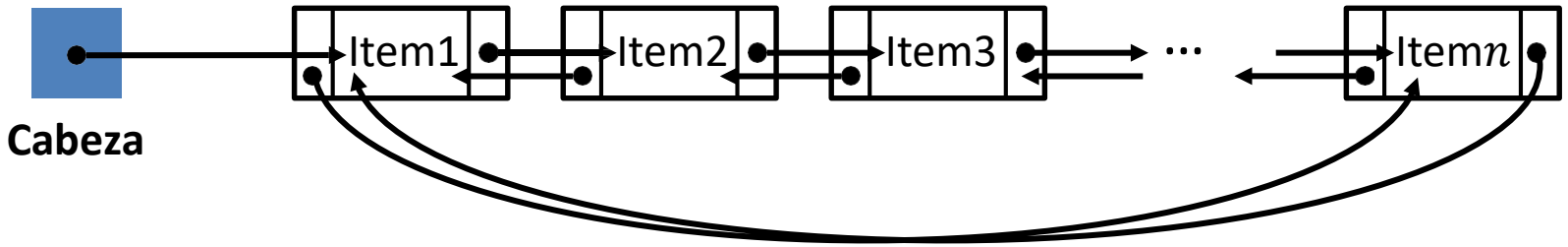


Tipos de listas enlazadas

Lista doblemente enlazada



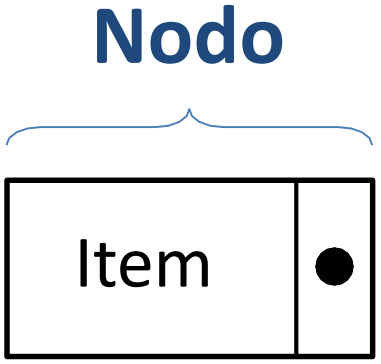
Lista circular doblemente enlazada



Definición de un nodo

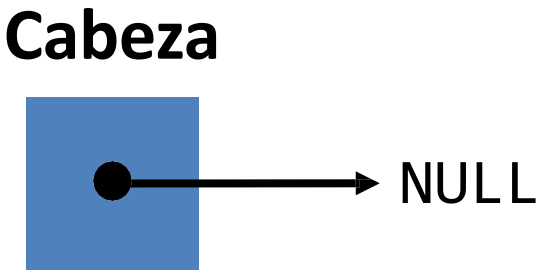
```
typedef struct{
   //definición de la estructura
   …
}Item;

typedef struct Elemento{
   Item dato;
   struct Elemento *siguiente;
}Nodo;
```



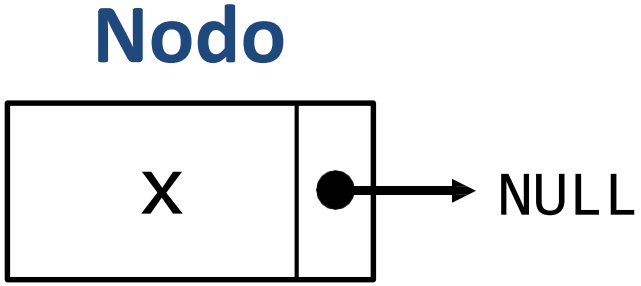
Declaración de una lista vacía

```
Nodo  *crearLista(){
    Nodo *cabeza = NULL;
    return cabeza;
}
```



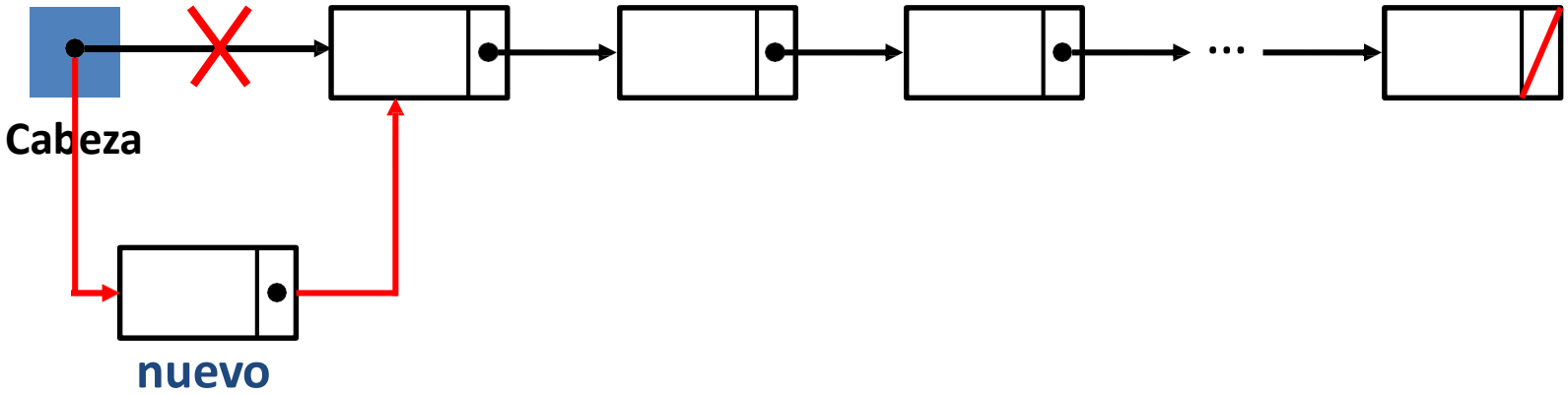
Creación de un nodo

```
Nodo *crearNodo(Item x){
   Nodo *a;
   a=malloc(sizeof(Nodo));
   a->dato=x;
   a-
   >siguiente=NULL;
   return a;
}
```



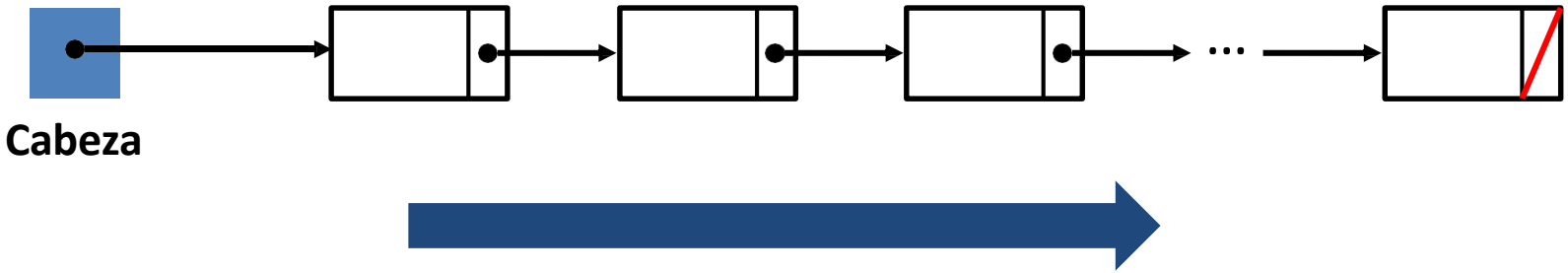
Insertar un nuevo elemento

Al inicio de la lista



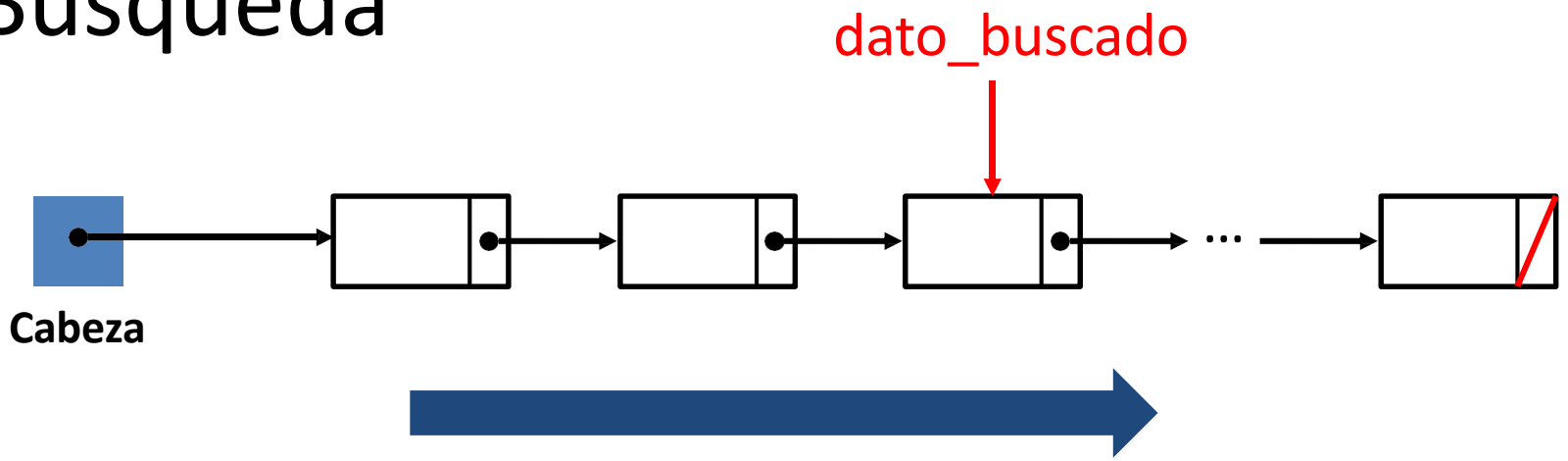

```
void inserInicio(Nodo **cabeza, Item entrada){
    Nodo *nuevo;
    nuevo = crearNodo(entrada);
    nuevo->siguiente = *cabeza;
    *cabeza = nuevo;
}
```

Mostrar elementos de la lista




```
void imprimirLista(Nodo *cabeza){
    Nodo *actual;
    printf("La lista es:\n");
    for(actual=cabeza; actual!=NULL; actual=actual->siguiente)
        printf("%d\t",actual->dato.nro);
    printf("\n\n");
}
```

Búsqueda

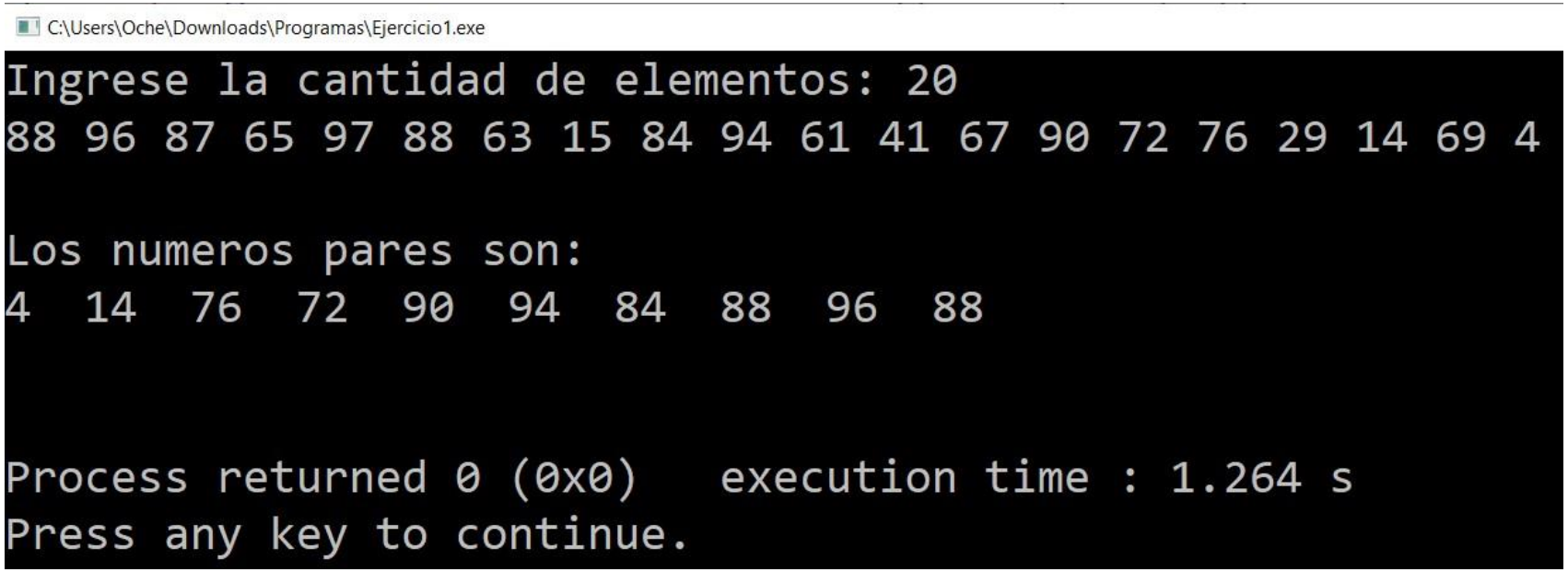



```
Nodo *buscar(Nodo *cabeza, Item dato_buscado){
    Nodo *actual;
    for(actual=cabeza;actual!=NULL;actual=actual->siguiente)
        if(dato_buscado.nro == actual->dato.nro) return actual;
    return NULL;
}
```

Nota: recordar que destino y dato son de tipo Item, por lo que se debe tener cuidado a la hora de comparar (este código es esquemático)

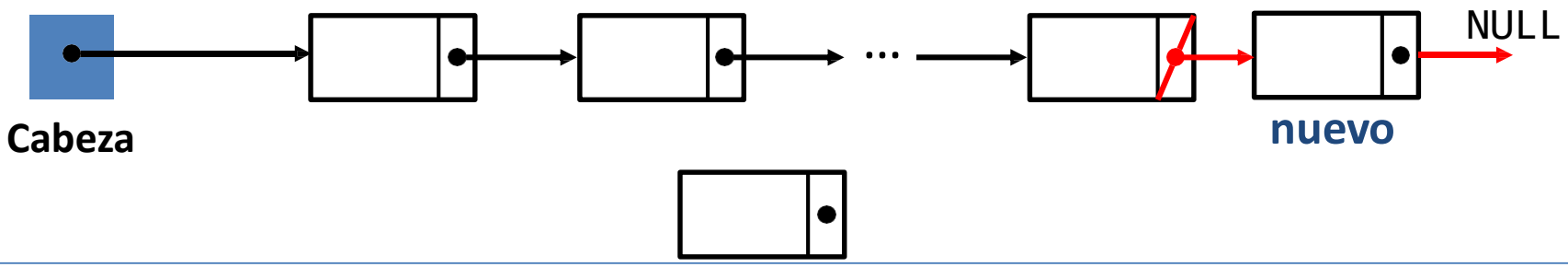
Ejemplo de aplicación

Se va a formar una lista enlazada de N números aleatorios enteros (del 0 al 100). El programa que realiza esta tarea inserta los nuevos nodos por la cabeza de la lista. Una vez creada la lista, se recorre los nodos para mostrar los números pares.



Insertar un nuevo elemento

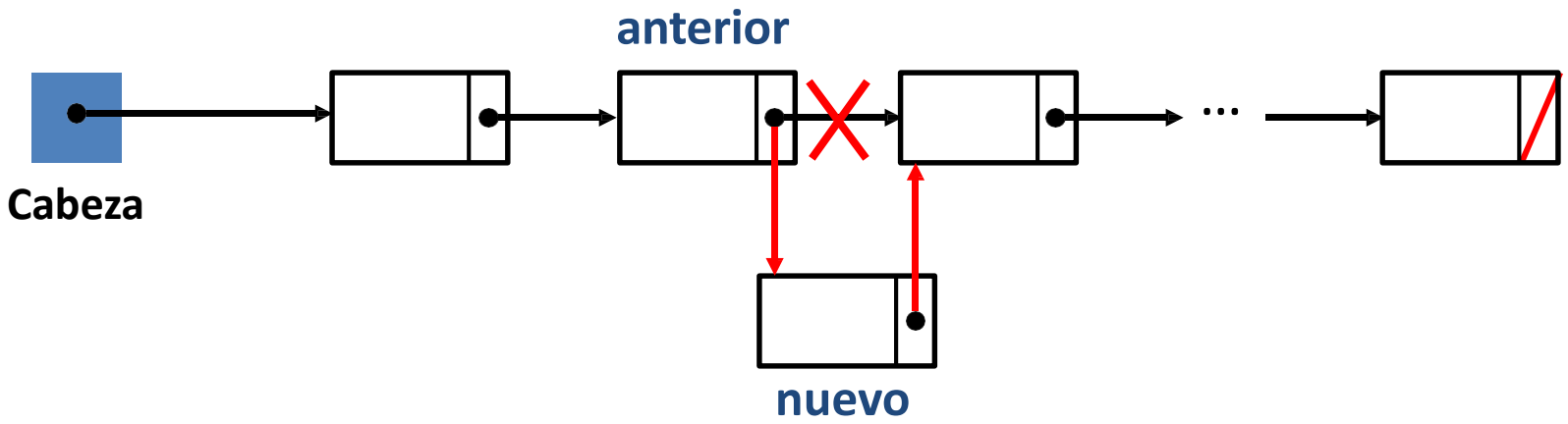
Al final de la lista




```
void inserFinal(Nodo **cabeza, Item entrada){
    Nodo *ultimo;
    ultimo  =  *cabeza;
    if(ultimo==NULL){ //caso particular
        *cabeza=crearNodo(entrada);
    }
    else{ //vamos hasta el ultimo nodo
        while(ultimo->siguiente!=NULL) ultimo = ultimo->siguiente;
        ultimo->siguiente = crearNodo(entrada);
    }
}
```

Insertar un nuevo elemento

Entre dos nodos de la lista

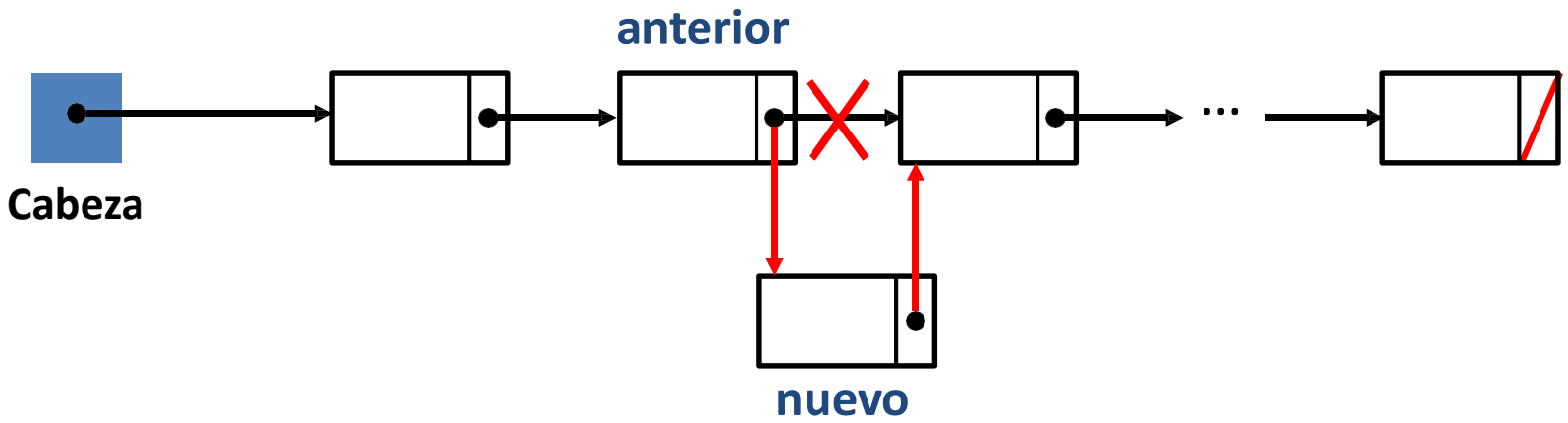


Como entrada tendremos:

- Inicio de la lista enlazada
- Dirección del nodo **anterior** (si es NULL, agregamos al inicio)
- Item del elemento a insertar

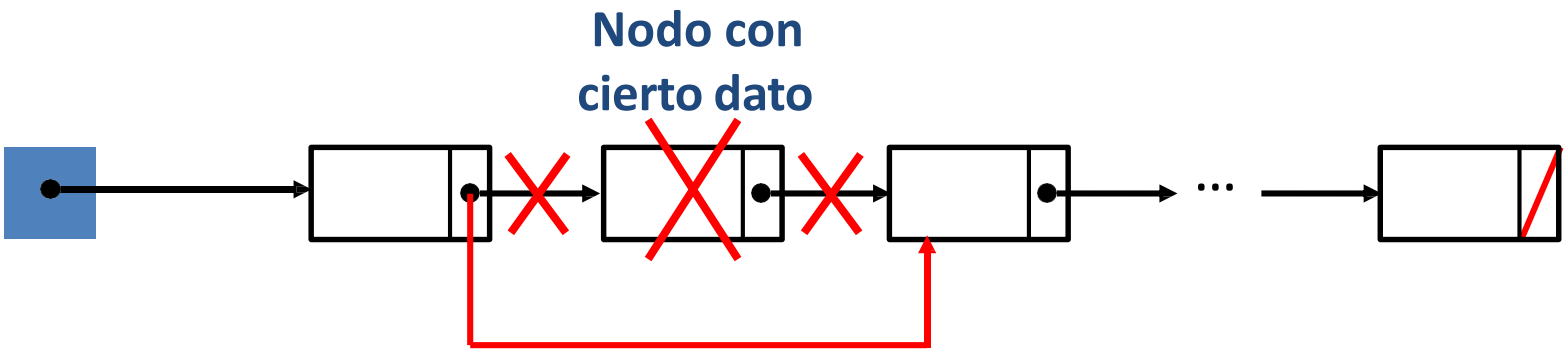
Insertar un nuevo elemento

Entre dos nodos de la lista




```
void insertar(Nodo **cabeza, Nodo *anterior, Item entrada){
    if(anterior==NULL || *cabeza==NULL)
        inserInicio(cabeza,entrada);
    else{
        Nodo *nuevo = crearNodo(entrada);
        nuevo->siguiente = anterior->siguiente;
        anterior->siguiente = nuevo;
    }
}
```

Eliminación de un nodo de la lista



El algoritmo tiene las siguientes etapas y casos:

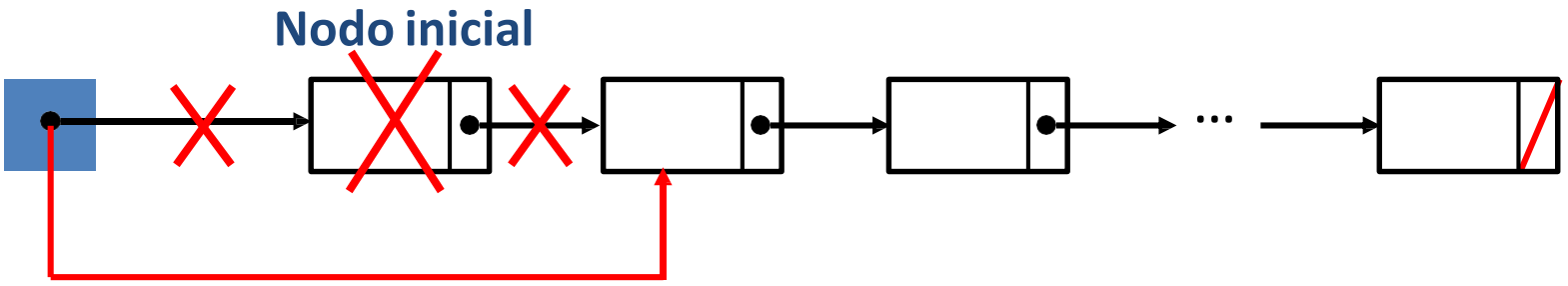
- Búsqueda del nodo que contiene el dato (se ha de tener la dirección del nodo a eliminar y la dirección del anterior).
- El puntero siguiente del nodo anterior deberá apuntar al siguiente del nodo a eliminar.
- En caso de que el nodo a eliminar sea el primero de la lista, se modifica `cabeza` para que tenga la dirección del nodo siguiente.
- Por último, se libera la memoria ocupada por el nodo.

Eliminación de un nodo de la lista

```
void eliminarNodo(Nodo **cabeza, Item dato_buscado){
    Nodo* actual = *cabeza;
    Nodo *anterior = NULL;
    while((actual!=NULL) && (actual->dato.nro != dato_buscado.nro)){
        //Actualizamos los valores de anterior y actual
        anterior = actual;
        actual = actual -> siguiente;
    }
    if(actual==NULL){ /*Significa que no se encontró el nodo con ese
dato (o que la lista está vacía)*/
        if(*cabeza == NULL) printf("La lista esta vacia!\n");
        else printf("No se encontro el elemento en la lista\n");
    }
    else{
        if(*cabeza == actual) /*significa que queremos borrar el primer
elemento*/
            *cabeza = actual->siguiente;
        else
            anterior -> siguiente = actual -> siguiente;
        free(actual); /*Se libera el nodo. Tener en cuenta el tipo de
dato con el que se está trabajando!*/
    }
}
```

*misma observación que en la diapositiva 13

Eliminación de toda la lista



Básicamente, eliminaremos siempre el primer elemento, hasta que la cabeza apunte a un NULL

```
void eliminarLista(Nodo **cab){
    Nodo *primero; /*eliminaremos siempre el nodo al inicio de
la lista*/
    while(*cab != NULL){
        primero = *cab;
        *cab = primero->siguiente;
        free(primero);
    }
}
```

Ejercicio 1

Se desea crear una lista enlazada de números enteros ordenada. La lista se organiza de tal forma que el nodo cabecera tenga el menor elemento, y así en orden creciente los demás nodos. Al iniciar el programa, se debe desplegar (y por supuesto, implementar) estas opciones para el usuario:

- Agregar elemento con un cierto valor
- Eliminar un elemento con un cierto valor. Si no se encuentra, indicarlo con un mensaje.
- Mostrar la lista
- Salir del programa

Ejercicio 2

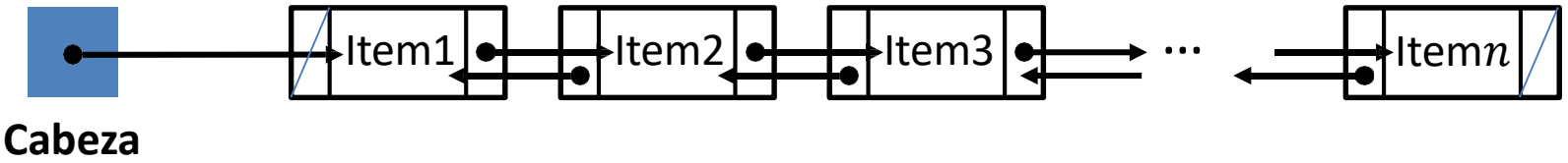
Se desea crear y organizar una lista enlazada de empleados, la cual estará ordenada alfabéticamente de acuerdo a los nombres. Los datos de los empleados son: nombre, departamento, salario. Al iniciar el programa, se debe desplegar (y por supuesto, implementar) estas opciones para el usuario:

- Agregar un empleado
- Eliminar un empleado con un cierto nombre. Si no se encuentra, indicarlo con un mensaje.
- Mostrar la lista de empleados y sus datos.
- Obtener el promedio de salarios de los empleados, así como los salarios mínimo y máximo.
- Salir del programa

Ejercicio 3

Definir todas las operaciones vistas para listas simplemente enlazadas para listas doblemente enlazadas.

Lista doblemente enlazada



Ejercicio 4

Se cargan dos listas enlazadas con números enteros positivos (cada una se carga hasta insertar un número negativo, el cual no se agrega a la lista).

Se pide escribir una función **compararListas()** que devuelva 1 (uno) si las listas son iguales, o 0 (cero) si las listas son distintas. Se considera que dos listas son iguales si tienen los mismos números en igual cantidad, independientemente de sus posiciones en las listas. Además, el programa debe mostrar las listas introducidas.

Por ejemplo, las siguientes dos listas son iguales:

1	2	3	4	2	4
4	4	2	2	1	3