Clase Nro 4

- Nueva problemática con datos creados en tiempo de ejecución
- Estructuras de datos dinámicas
- El tipo Lista enlazada
- Operaciones de lista enlazada

Problemas de la asignación dinámica

Imaginemos algunos escenarios

Problemas de la asignación dinámica

Escenario empresarial

Se solicita un programa para la carga de productos y su correspondiente stock

Procesos sobre sus entidades:

Cargar Productos nuevos

Modificar Productos

Eliminar Productos

Listar Productos con algún criterio de búsqueda

Problemas de la asignación dinámica

Escenario en un juego

<code/>

```c
typedef struct
{
    char * Tipo;
    Image  * Asset;
    int Energia;
    int PoderDeAtaque;
}Enemigo;

typedef struct
{
    char * Tipo;
    Image  * Asset;
    int Vidas;
    int PoderDeAtaque;
    Arma * ArmaSeleccionda;
}Personajes;


Personaje * Players = (Personaje *) malloc(sizeof(Personaje) * n);

Enemigo   * Enemigos = (Enemigo *) malloc(sizeof(Enemigo) * n);
```

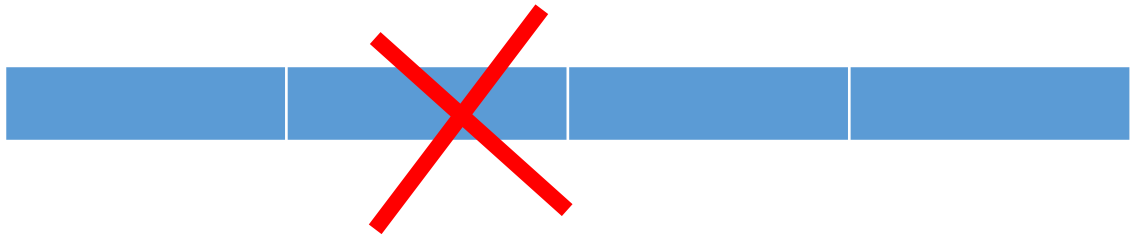
Problemas de la asignación dinámica

¿Podemos solucionarlo con un array?

Problemas para manejar lo que sucede en tiempo de ejecución

Crear Jugadores

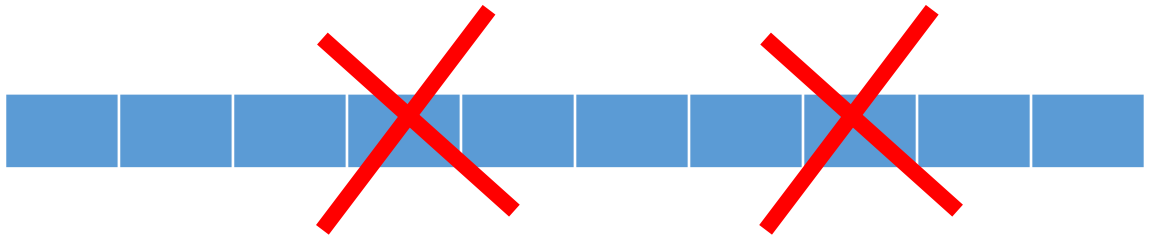
Eliminar Jugadores



Arreglos enemigos ?

Crear enemigos

Eliminar enemigos



Crear nuevos enemigos de forma aleatoria

Operaciones Necesarias en tiempo de ejecución

- Insertar elementos nuevos
- Eliminar elementos existentes
- Modificar elementos existentes
- Buscar elementos

Estructuras de datos dinámicas

Una estructura de datos dinámica es aquella en la que el tamaño ocupado en memoria puede modificarse durante la ejecución del programa.

De esta manera se pueden adquirir posiciones adicionales de memoria a medida que se necesiten durante la ejecución del programa y liberarlas cuando no se necesiten.

Estructuras de datos

Estructuras de datos dinámicas
(su tamaño crece y se encoge en tiempo de ejecución):

- Listas enlazadas
- Pilas
- Colas
- Arboles

Listas enlazadas

Es una colección de elementos organizados en secuencia que puede crecer y decrecer libremente y a cuyos elementos individuales se puede acceder, insertar y eliminar en cualquier posición

LISTA

Player 1

Insert

```
Personaje * Players1 = (Personaje *) malloc(sizeof(Personaje));
```

Remove

Player 2

Insert

```
Personaje * Player2 = (Personaje *) malloc(sizeof(Personaje));
```

Player 3

Insert

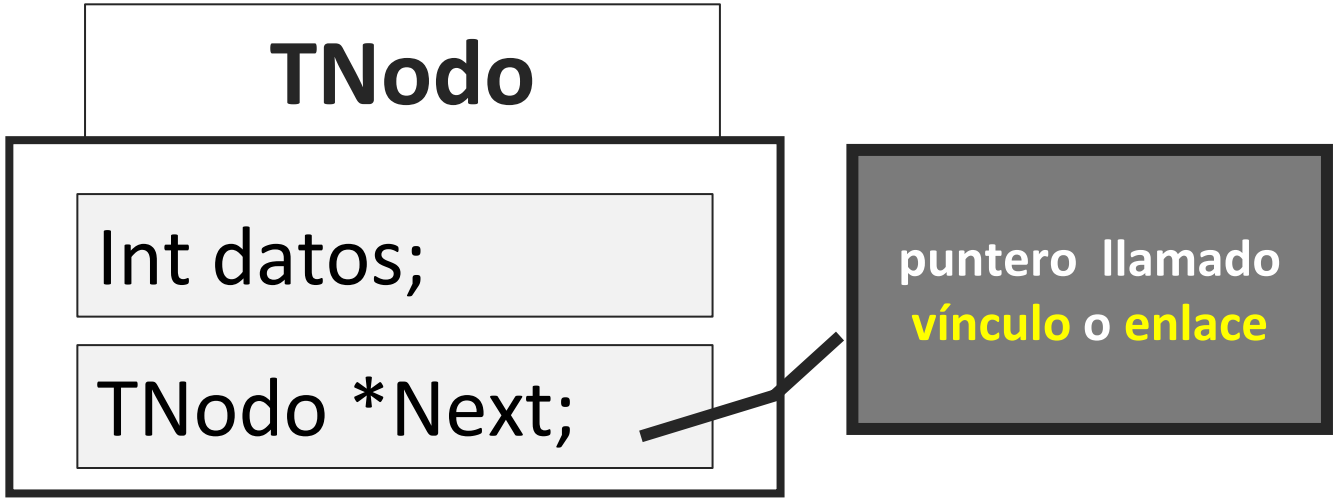
```
Personaje * Players3 = (Personaje *) malloc(sizeof(Personaje));
```

kahoot

Listas enlazadas

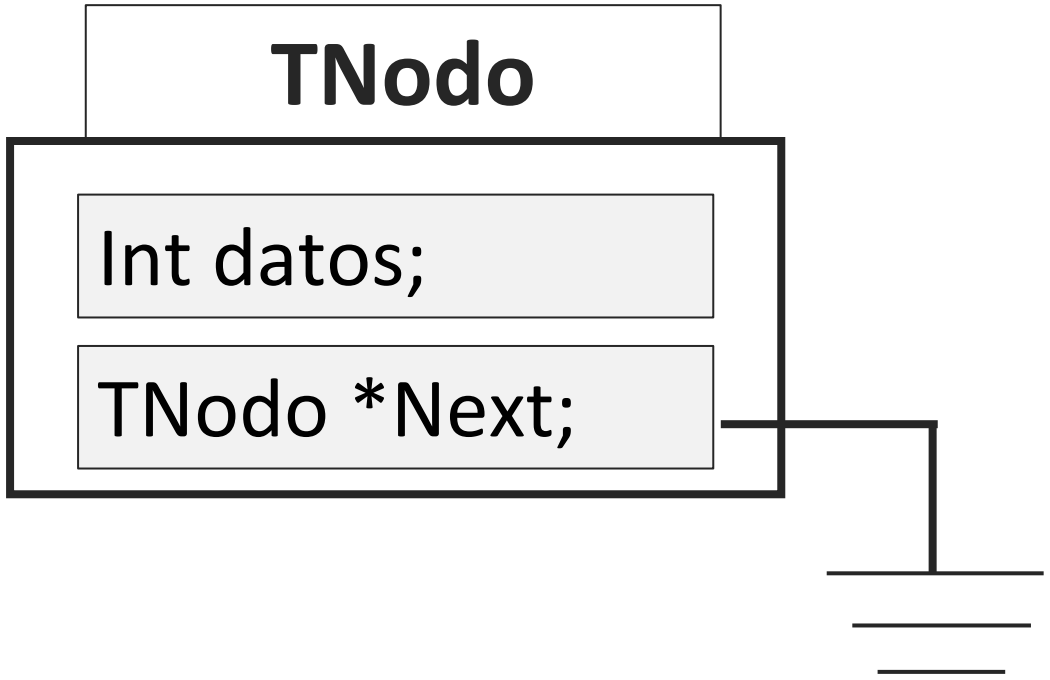
¿Cómo funcionan?

Estructuras de datos Autorreferencial




```
Typedef struct TNodo
{
        int datos;
        TNodo *Next
};
```

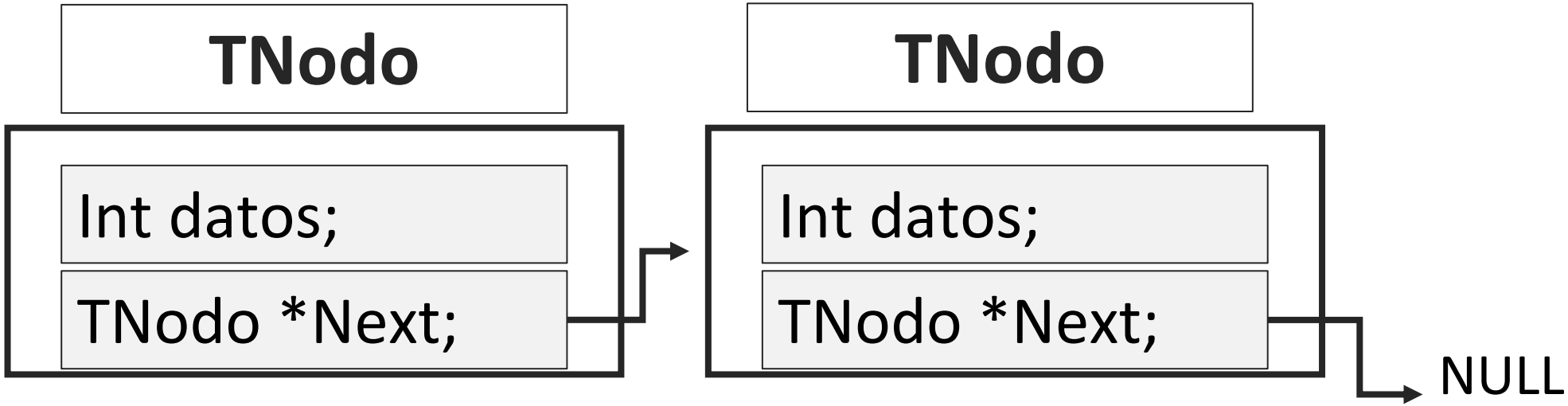
Estructuras de datos Autoreferencial




```
Typedef struct TNodo
{
        int datos;
        TNodo *Next
};
```

Estructuras de datos

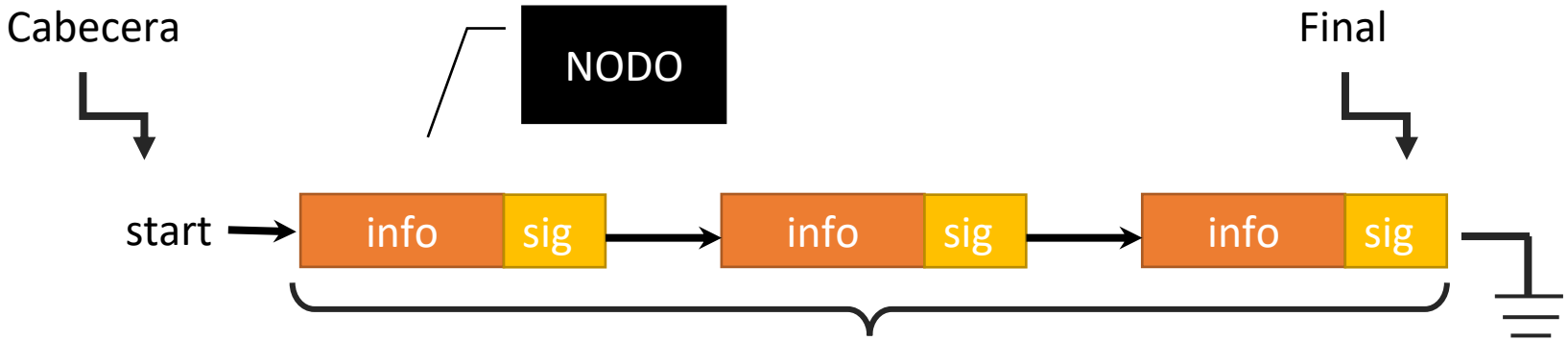
Un conjunto de estos **Nodos** unidos a través del puntero **Next**, se denomina una "**lista simplemente enlazada**":



Las **listas enlazadas** pueden ser **simples**, **dobles** o **circulares**.

Lista enlazada

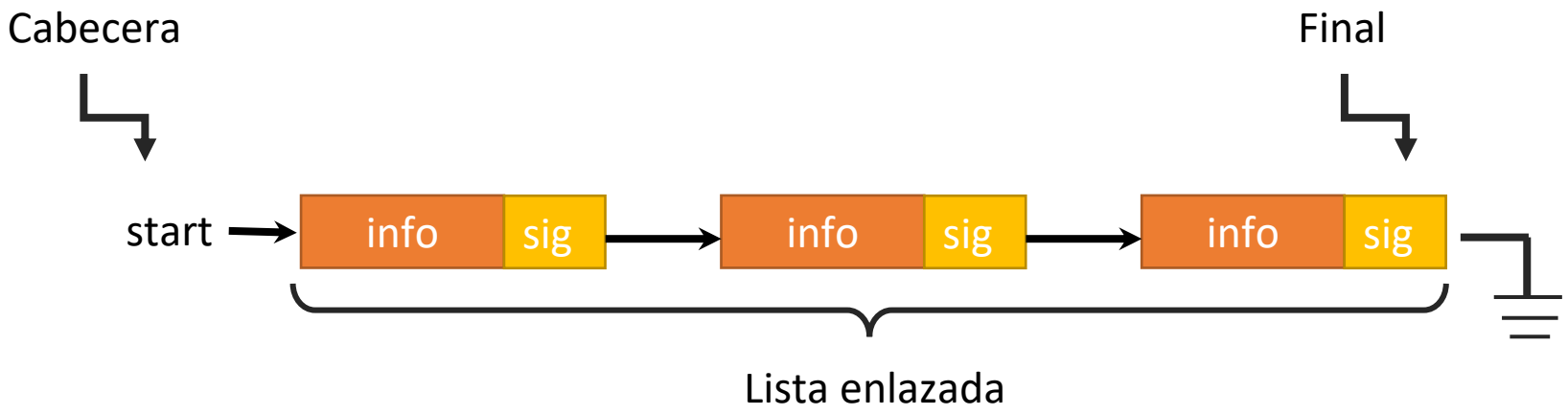
- Cada elemento agregado contendrá un **puntero al próximo elemento**.
- La lista cuenta con un puntero (**Start**) que señale al nodo inicial de la lista y el puntero del último nodo debe apuntar a **NULL** para indicar que termina.



Lista simplemente enlazada

Lista enlazada

- En una **lista enlazada** se crean Nodos nuevo en memoria a medida que se necesita.
- Para **añadir un nuevo elemento** reservamos **memoria** para él y lo añadimos a la **lista**.
- Para **eliminar el elemento** simplemente lo sacamos de la **lista** y liberamos la memoria usada.



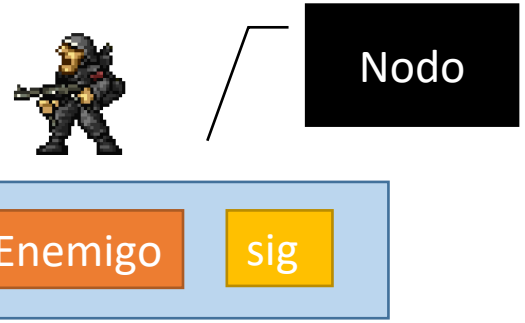
Lista enlazada

Cada **elemento o nodo** de una **lista enlazada simple** consta de**:**

- Una **estructura** de uno o más campos.
  Donde guardaremos la información que nos interesa retener
- Un **puntero** del mismo Tipo Nodo.
  Que utilizamos saber la **posición** en memoria del siguiente **elemento de la lista**.)

<code/>

```
typedef struct
{
    char * Tipo;
    Image  * Asset;
    int Energia;
    int PoderDeAtaque;
}Enemigo;

typedef struct TNodoEnemigo
{
    Enemigo Enemigo;
    TNodoEnemigo * Sig;
} TNodoEnemigo;
```

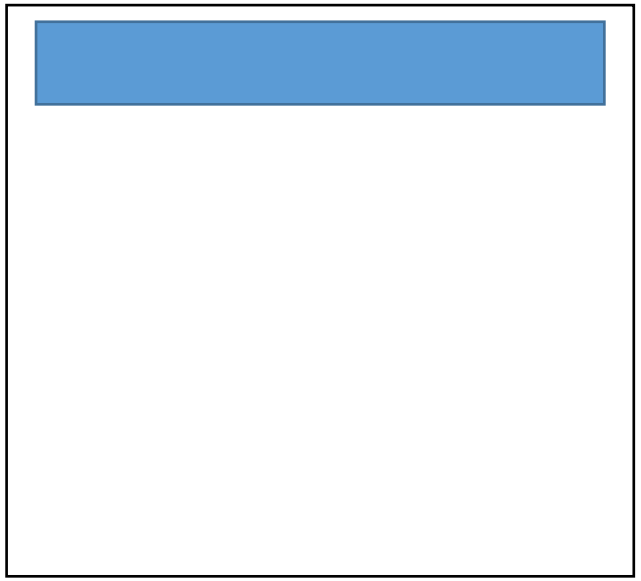


Lista enlazada

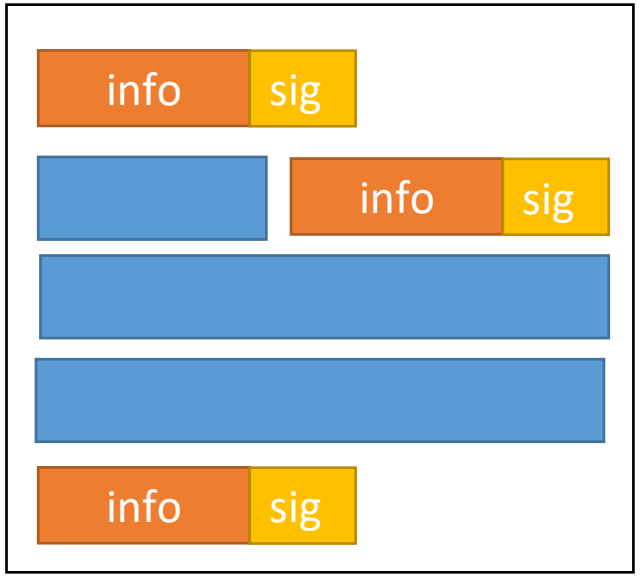
Los elementos se distribuyen de forma dispersa por la memoria:

- Los bloques de información no ocupan posiciones consecutivas en la memoria.
- El orden de la lista la establecen los enlaces entre bloques de información.

Arreglo en M.RAM



Lista Enlazada en M.RAM



Lista enlazada

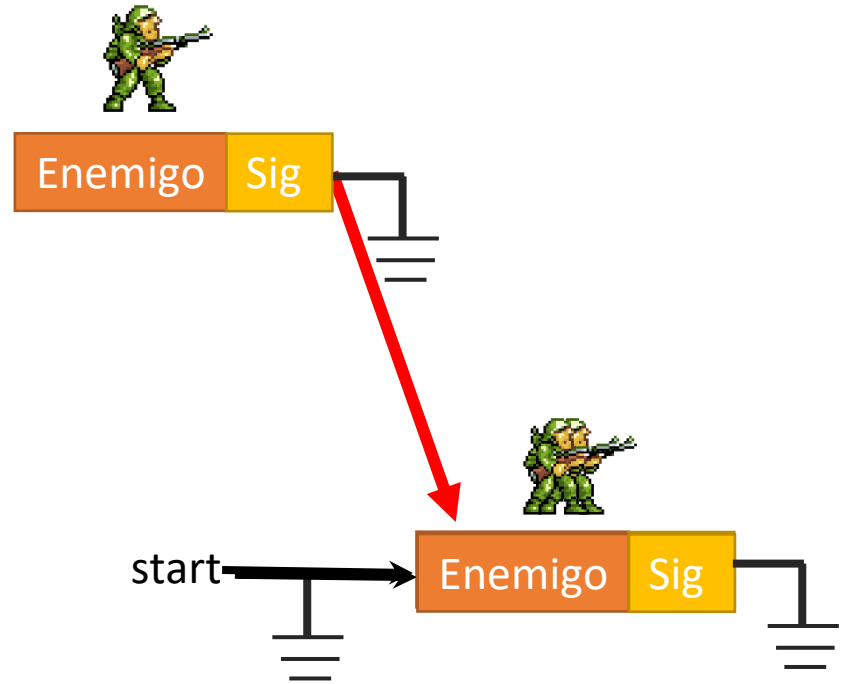
Puede modificarse de forma dinámica:

- No hay un número máximo de elementos de la lista (salvo limitaciones de la capacidad de almacenamiento de la máquina).
- Cada elemento requiere una **reserva dinámica de memoria** al ser creado (liberar memoria al ser eliminado).

Apliquemos a nuestro ejemplo

<code/>

```
typedef struct
{
    char * Tipo;
    Image  * Asset;
    int Energia;
    int PoderDeAtaque;
}Enemigo;


typedef struct TNodoEnemigo
{
    Enemigo Enemigo;
    TNodoEnemigo * Sig;
} TNodoEnemigo;

TNodoEnemigo * Start = NULL;
TNodoEnemigo * NodoE1 = (TNodoEnemigo  *) malloc(sizeof(TNodoEnemigo));
NodoE1->Enemigo.Energia = 100;
NodoE1->Sig = NULL;
Start = NodoE1;
TNodoEnemigo * NodoE2 = (TNodoEnemigo  *) malloc(sizeof(TNodoEnemigo));
NodoE2->Sig = NULL;
NodoE2->Sig = Start;
Start = NodoE2;
```



kahoot

Lista enlazada – Implementación Práctica

Lista enlazada – Operaciones

- Declaración de los tipos nodo y puntero a nodo.
- Inicialización o creación.
- Insertar elementos en una lista.
- Eliminar elementos de una lista.
- Buscar elementos de una lista (comprobar la existencia de elementos en una lista).
- Recorrer una lista enlazada (visitar cada nodo de la lista).

Lista enlazada

Operaciones

- Declaración de los tipos nodo y puntero a nodo.
- Inicialización o creación.
- Insertar elementos en una lista.
- Eliminar elementos de una lista.
- Buscar elementos de una lista (comprobar la existencia de elementos en una lista).
- Recorrer una lista enlazada (visitar cada nodo de la lista)

Lista enlazada

Estructura Nodo

Nuevo nodo

```
typedef struct Tnodo
{
        int valor; //dato
        Tnodo * Sig; //puntero al siguiente
}Tnodo;
```

Lista enlazada

Nuevo Nodo

Nuevo nodo

```
typedef struct Tnodo
{
        int valor;
        Tnodo* Sig;

}Tnodo;
```

```
Tnodo * CrearNodo(int valor)
{
        Tnodo * NNodo = (Tnodo *) malloc (sizeof(Tnodo));
        Nnodo->Valor = valor;
        NNodo->siguiente = null;
        return NNodo;
}
```

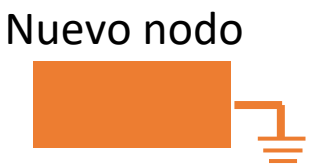
Lista enlazada

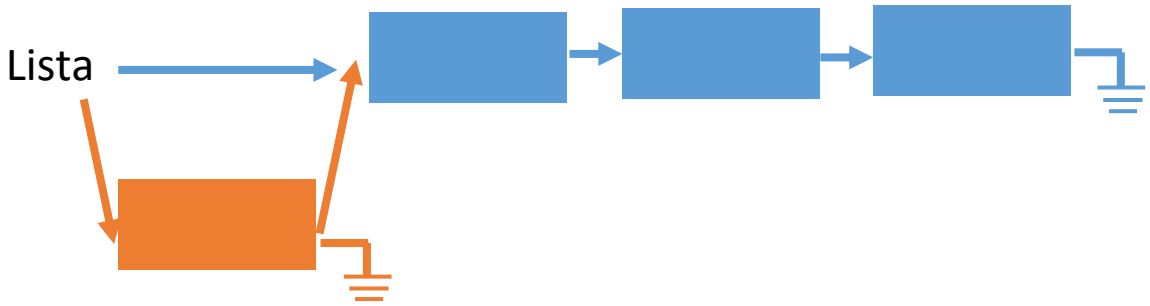
Inserción: Lista vacía


```
TNodo * Start;

TNodo * CrearListaVacia()
{
        return null;
}


Start = CrearListaVacia();
```

Lista enlazada

Inserción: al inicio de la lista



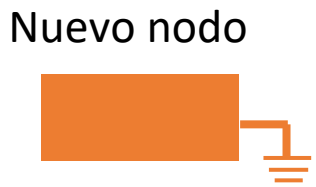


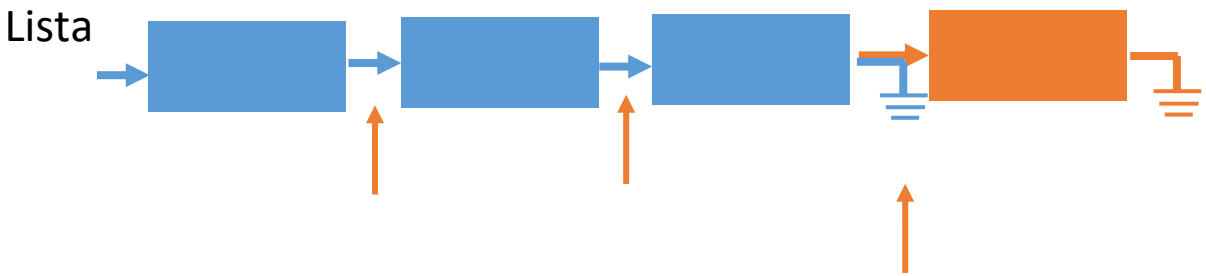

```
typedef struct Tnodo
{
        int valor;
        Tnodo Sig;

}Tnodo;
```

```
void InsertarNodo(Tnodo ** Start , Tnodo
*NuevoNodo)
{
    NuevoNodo -> siguiente = *Start;
   *Start  = NuevoNodo ;
}
```

Lista enlazada

Inserción: al final de la lista

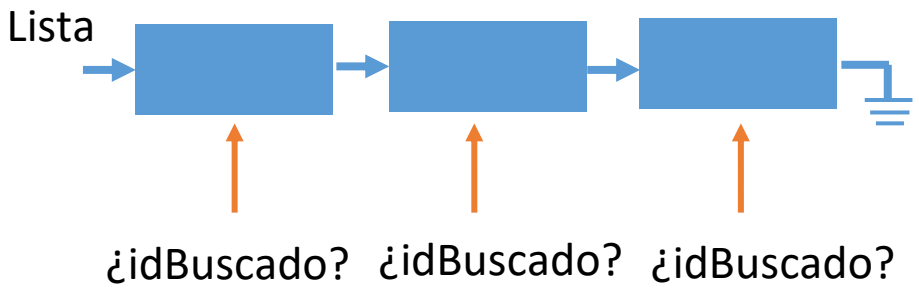





```
void InsertarAlFinal(Tnodo * Start, tnodo *
NuevoNodo)
{
  Tnodo * Aux = Start;
   while(Aux->siguiente)
   {
            Aux = Aux -> siguiente;
   }
   Aux -> siguiente = NuevoNodo;

}
```

Lista enlazada

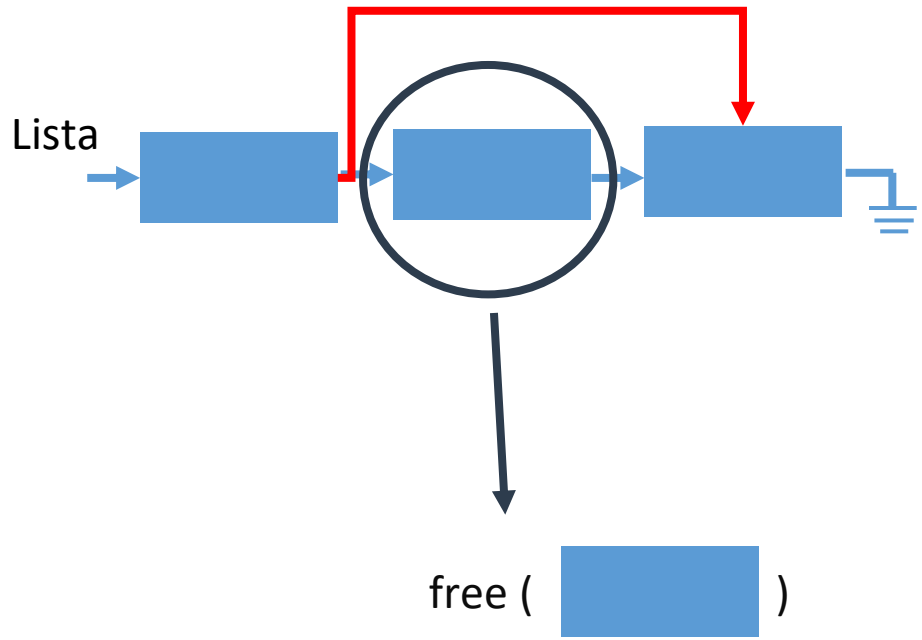
Buscando un elemento en la lista




```
Tnodo * buscarNodo(Tnodo * Start, int IdBuscado)
{
   Tnodo * Aux = Start;
   while(Aux && aux -> id != idBuscado)
   {
            Aux = Aux -> siguiente;
   }
   return Aux
}
```

Lista enlazada

Eliminación: un nodo

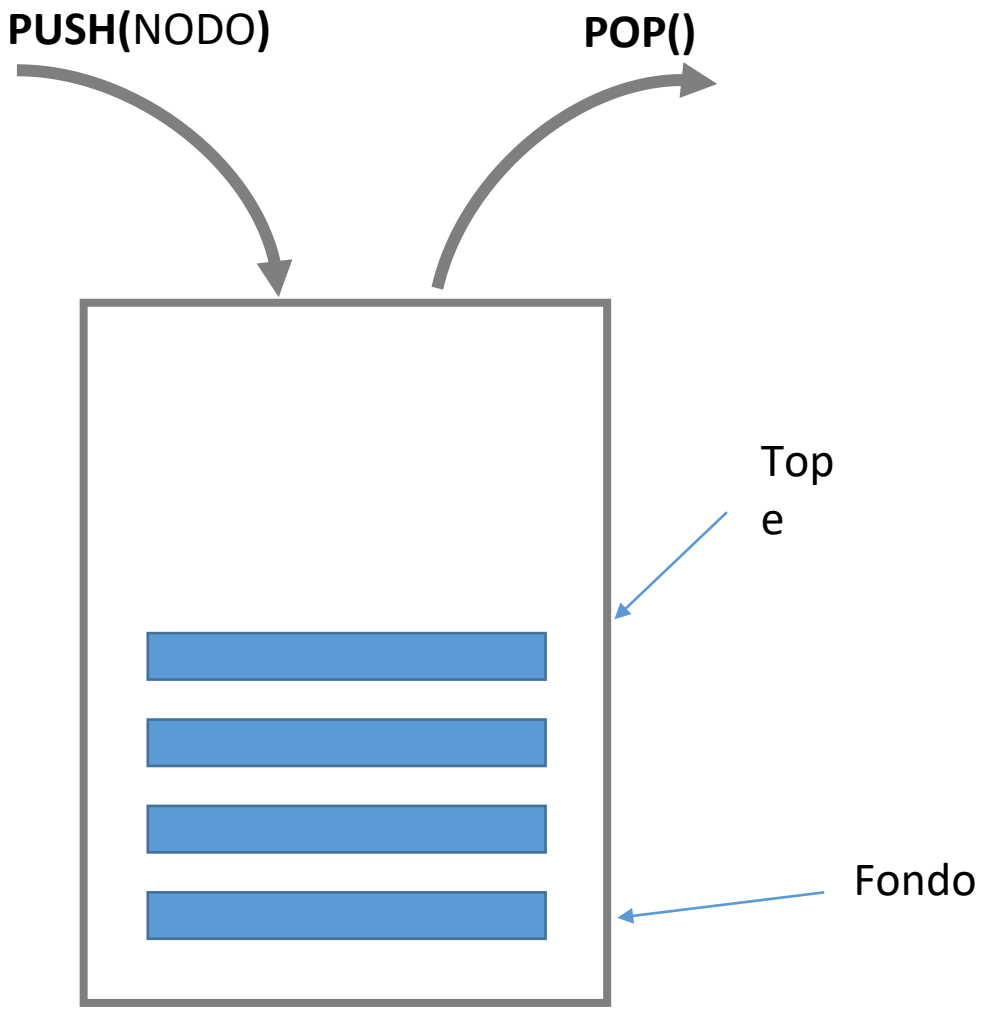



```c
void EliminarNodo(Tnodo **Start, int dato)
{
    Tnodo ** aux = Start;

    // Iteramos sobre la lista hasta encontrar el dato o alcanzar el final de la lista.
    while (*aux != NULL && (*aux)->dato != dato) {
        aux = &(*aux)->siguiente;
    }

    // Si encontramos el nodo con el dato especificado, lo eliminamos.
    if (*aux)
    {
        Tnodo *temp = *aux;  // Guardamos el nodo a eliminar en una variable temporal.
        *aux = (*aux)->siguiente;  // Desvinculamos el nodo de la lista.
        free(temp); // Liberamos la memoria ocupada por el nodo.
    }
}
```

Otras estructuras de datos que utilizan nodos

Tipo Pila

Estructura tipo LIFO (Last In First Out)

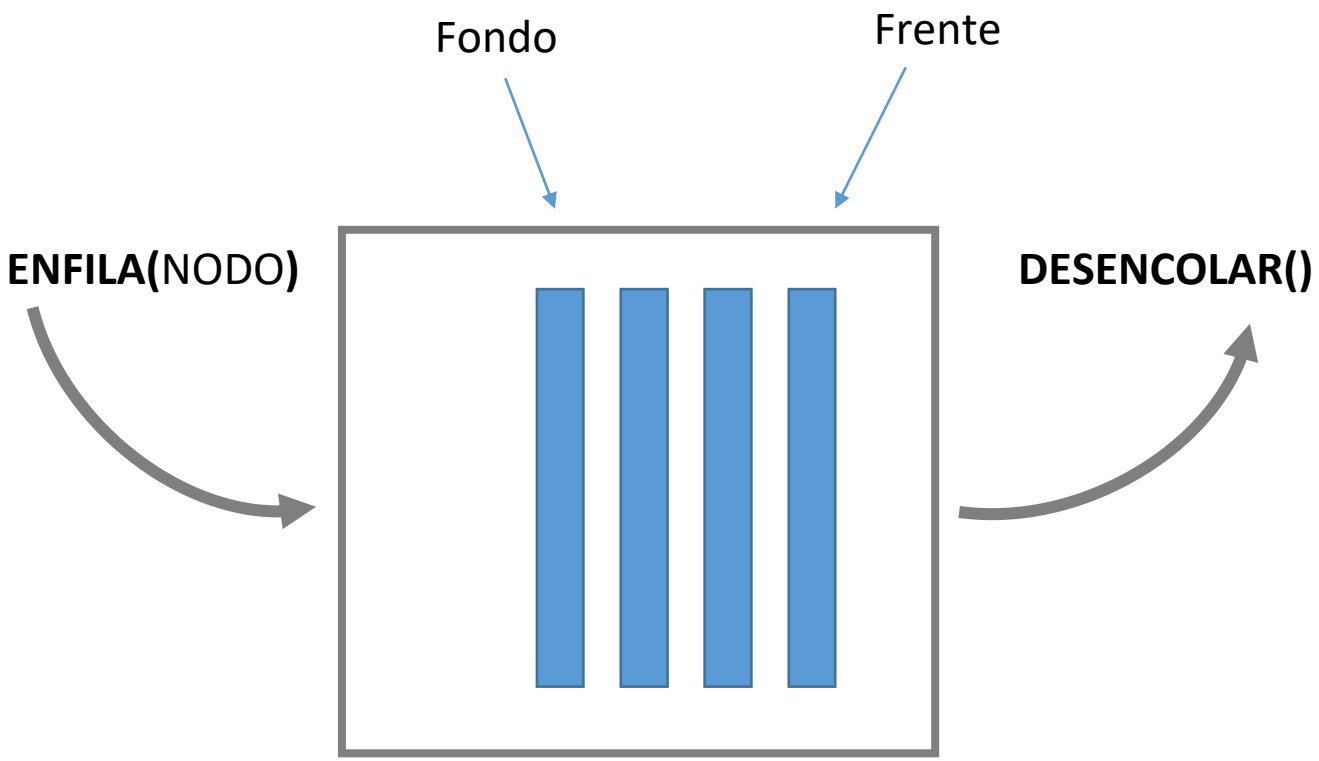


Operaciones Comunes

- **Crear (constructor)**
- **Tamaño (*size*)**
- **Apilar (*push*)**
- **Desapilar *(pop*)**
- **Leer último (*top* o *peek*)**
- **esVacía (*empty*)**

Tipo Cola O fila

Estructura tipo FIFO (First In First Out)



Operaciones Comunes

Crear: se crea la cola vacía.

Encolar: se agrega un elemento a la cola.

Desencolar: se elimina el elemento frontal de la cola y se lo devuelve para su uso

•**Frente**: se devuelve el elemento frontal de la cola, es decir, el primer elemento que entró.