ESTRUCTURAS
DE
DATOS DINÁMICAS

- Las **ESTRUCTURAS DE DATOS ESTÁTICAS** son aquellas en las que el tamaño ocupado en la memoria se define antes de que el programa se ejecute y no puede ser modificado durante la ejecución del programa.

- Las **ESTRUCTURAS DE DATOS DINÁMICAS** son aquellas en las que el tamaño podrá modificarse durante la ejecución del programa; teóricamente no hay límites a su tamaño, salvo el que impone la memoria disponible en la computadora.

PUNTERO

Un puntero es una variable que contiene la dirección de memoria de otra variable.

- Un puntero es una variable que apunta o referencia a una ubicación de memoria en la cual hay datos.

- Es un tipo de dato que “apunta” a otro valor almacenado en memoria.

ESTRUCTURA DE DATOS LINEALES

Las **estructuras de datos lineales** son aquellas en las que los elementos ocupan lugares sucesivos en la estructura y cada uno de ellos tiene un único sucesor y un único predecesor, es decir, sus elementos están ubicados uno al lado del otro relacionados en forma lineal.

Hay tres tipos de **estructuras de datos lineales**:

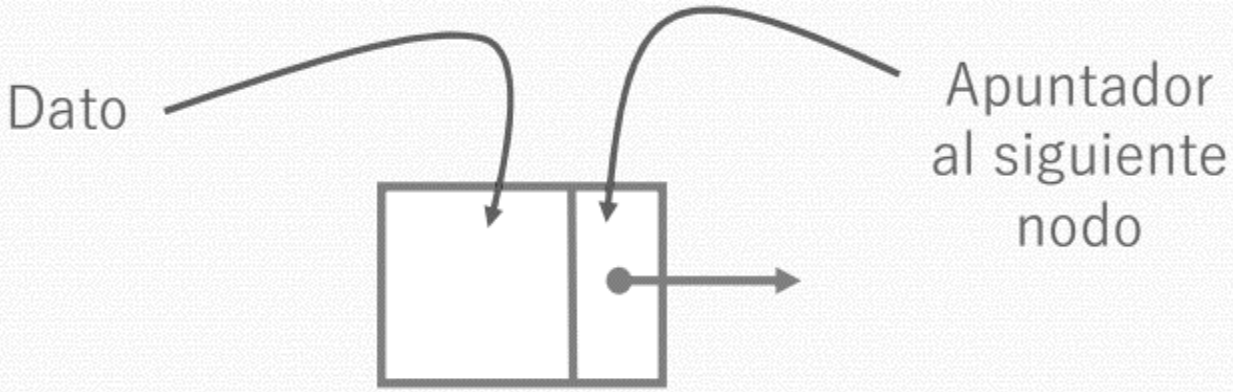
- Listas enlazadas
- Pilas
- Colas

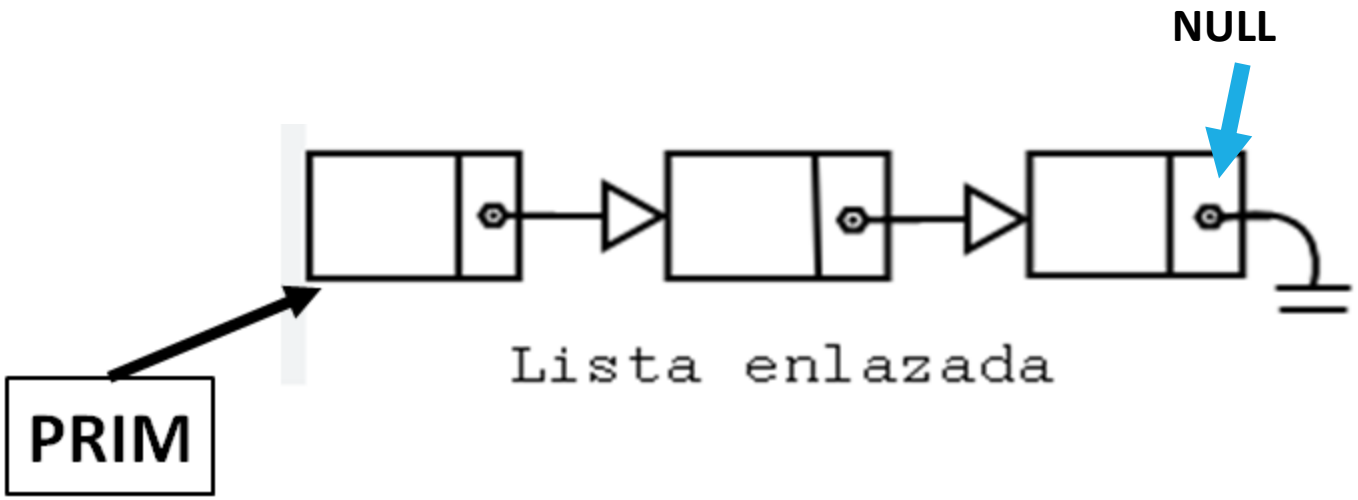
LISTAS ENLAZADAS

Una lista simplemente enlazada (también llamada lista lineal o lista enlazada simple) es una estructura de datos lineal y dinámica que utiliza un conjunto de nodos para almacenar datos en secuencia, enlazados mediante un apuntador o referencia.

Cada nodo tiene 2 secciones:

1. *Dato (puede ser simple o compuesto)*
2. *Apuntador o referencia que enlaza al siguiente nodo en secuencia lógica*





- *La lista simple tiene un apuntador inicial*
- *El último nodo de la lista apunta a nulo*

Almacenamiento de los datos en una lista simple

- **Ordenados:**
  - **Se recorren lógicamente los nodos de la listasimple**
  - **Se ubica la posición definitiva de un nuevo nodo Se mantiene el orden lógico de los datos.**

- **Desordenados**
  - **El nuevo dato se agrega al final de la lista simple**

OPERACIONES CON LISTAS

- **Crear una lista.**
- **Eliminar en elemento de la lista**
- **Insertar un elemento en la lista**
- **Mostrar los elementos de una lista.**
- **Buscar un elemento en la lista**

```
Acción Lista es
Ambiente
Tipo
      punt:puntero a elto;
      registro: elto
          valor: entero;
          proximo: puntero a elto;
        fin registro;

prim,p: punt;
cant,i,dato: intero;
```

```
Algoritmo
i=0;
Escribir (“Ingresar cantidad de elementos de la lista”);
Leer (cant);

nuevo (p);
si (p = null)  entonces escribir (“Error”);
   sino
              Escribir (“Ingrese dato”);
              Leer (dato);
               *p.dato:=dato;
               *p.proximo:=null;
                prim:=p;
                i:=i+1;
                nuevo (p);
```

SE CREA EL PRIMER NODO DE LA LISTA

```
Mientras (p<>Null) ∧ (i<cant) hacer
                    Escribir (“Ingrese dato”);
                    Leer (dato);
                    *p.valor:=dato;
                    *p.proximo:=prim;
                     prim:=p;
                     i:=i+1;
                    Nuevo (p);
  fin mientras

  si (i= cant) entonces escribir (“Lista completa)”;
     sino escribir (“La lista tiene menos elementos”);

  escribir (“Mostrar los elementos de la lista”);
   p:=prim;
   mientras (p<> Null) hacer
       escribir (*p.valor);
        p:=*p.proximo
    fin mientras
sin si

fin acción
```

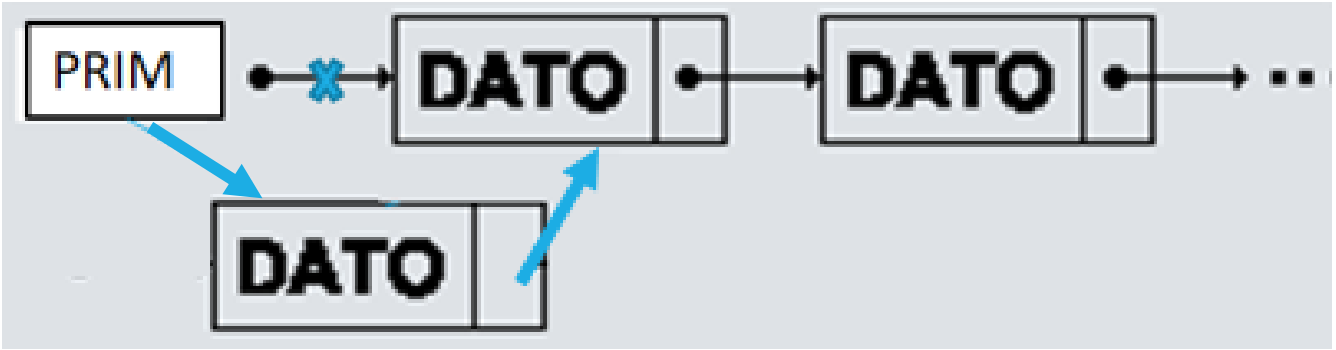
A partir del algoritmo anterior, indicar la cantidad de números pares e impares que tiene la lista

```
par:=0;
impar:=0;
p:=prim;
   mientras (p<> Null) hacer
     si (*p.valor MOD 2=0) entonces par:=par+1:
        sino impar:=impar+1;
      fin si
       p:=*p.proximo
    fin mientras

   escribir (“Cantidad de números pares:”,par);
   escribir (“Cantidad de números impares:”,impar);
```

Operación Insertar al principio de la lista

A partir del algoritmo anterior realizar la operación indicada:

```
nuevo (p);
si (p = null)  entonces escribir (“Error”);
 sino
      t:=prim;
      escribir (“Ingrese dato”);
       Leer (dato);
         *p.valor:=dato;
         *p.proximo:=prim;
          prim:= p;
```



```
// Definición de la estructura

Nodostruct Nodo {    int dato;    struct Nodo *siguiente;};

// Inicialización de la cabeza
struct Nodo *cabeza = NULL;

// Insertar un nodo al principio
// (Este es solo un ejemplo, la inserción real puede requerir más pasos)

struct Nodo *nuevo_nodo = (struct Nodo *) malloc(sizeof(struct Nodo));
nuevo_nodo->dato = 10;
nuevo_nodo->siguiente = NULL;
cabeza = nuevo_nodo;
 // cabeza ahora apunta al nuevo nodo (el primer nodo de la lista)
```

DEFINICIÓN DE LA ESTRUCTURA NODO EN C

```
struct Nodo
 {
    int dato;
    struct Nodo* siguiente;
};
```

- int dato: almacena un valor entero.
- struct Nodo* siguiente: apunta al siguiente nodo de la lista.

La declaración `struct Nodo** cabeza` define `cabeza` como un puntero a un puntero de tipo `Nodo`. En otras palabras, `cabeza` almacena la

2.Función crearNodo

```
struct Nodo* crearNodo(int valor)
{
   struct Nodo* nuevo = (struct Nodo*) malloc(sizeof(struct Nodo));

   nuevo->dato = valor;
   nuevo->siguiente = NULL;
   return nuevo;
}
```

- Usa malloc para reservar memoria dinámica.
- Asigna el valor recibido al campo dato.
- El siguiente es NULL porque aún no se conecta a ningún nodo.
- Devuelve el puntero al nuevo nodo.

3. Función agregarAlFinal

void agregarAlFinal(struct Nodo** cabeza, int valor)

- Agrega un nuevo nodo **al final de la lista**.
- Usa struct Nodo** cabeza para poder modificar la cabeza desde fuera de la función (puntero al puntero).
- Si la lista está vacía (*cabeza == NULL), el nuevo nodo será el primero.
- Si no, recorre la lista hasta el último nodo y lo enlaza al nuevo nodo.

La declaración struct Nodo** cabeza define cabeza como un puntero a un puntero de tipo Nodo.

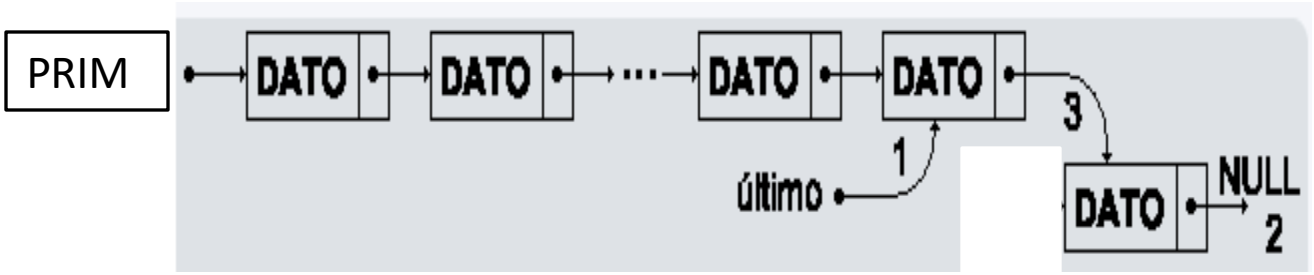
En otras palabras, cabeza almacena la dirección de memoria de otro puntero, que a su vez apunta a una estructura Nodo.

Esto es comúnmente utilizado en C/C++ para trabajar con listas enlazadas, donde cabeza puede apuntar al primer nodo de la lista.

Operación Insertar al final de la lista

A partir del algoritmo anterior realizar la operación indicada:

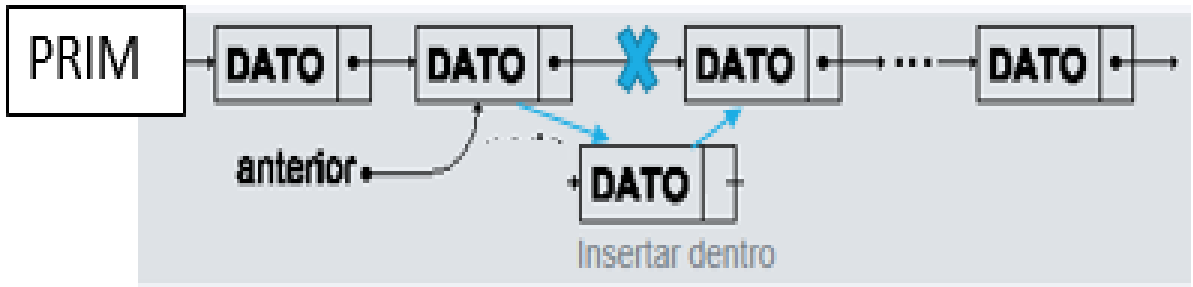
```
nuevo (p);
si (p = null)  entonces escribir (“Error”);
 sino
     t:=prim;
     mientras (t<> Null) hacer
           aux:= t;
          t:=*t.proximo;
      fin mientras

        escribir (“Ingrese dato”);
        Leer (dato);
        *p.dato:=dato;
        *p.proximo:=null;
         *aux.proximo:= p;
```



Operación Insertar al medio de la lista

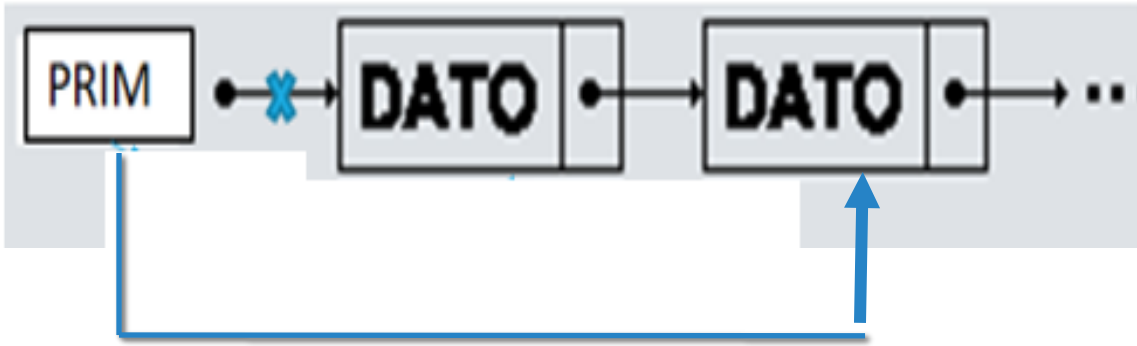
A partir del algoritmo anterior realizar la operación indicada:

```
nuevo (p);
si (p = null)  entonces escribir (“Error”);
 sino
      escribir (“Ingrese dato a insertar”);
      Leer (dato);
      t:=prim;

     mientras (t<> Null ) ∧ (*t.valor < dato) )hacer
            aux:= t;
           t:=*t.proximo;
      fin mientras

      si (t=null) entonces escribir (“No se encontró un valor menor al ingresado”);
          sino
               *p.valor:=dato;
              *p.proximo:=t;
             *aux.proximo:= p;
       fin si
```



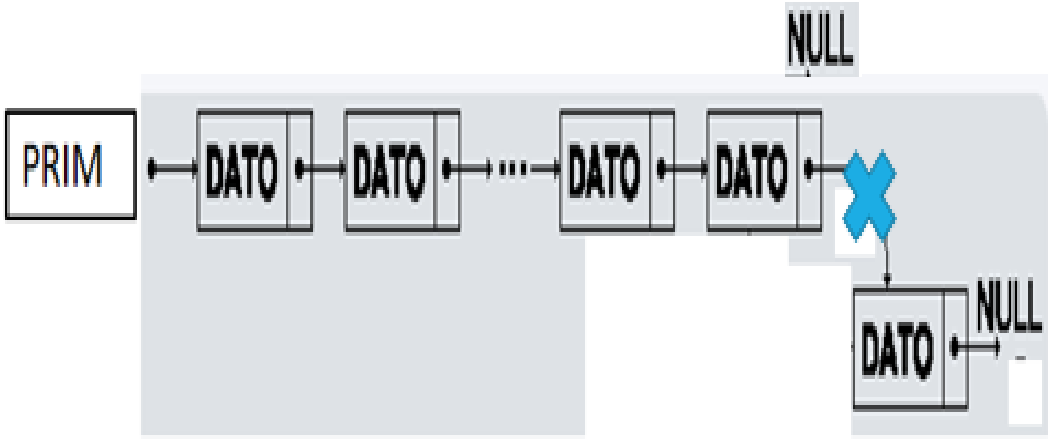
Eliminar elemento al principio de la lista

```
t:=prim;
prim:=*t.proximo;
disponer (t);
```



Eliminar el ultimo elemento de la lista

```
escribir (“Encontrar el ultimo elemento de la lista”);
t:=prim;
mientras (*t.proximo<> Null) hacer
        aux:= t;
       t:=*t.proximo;
 fin mientras

    *aux.proximo:= null;
      disponer(t);
```



Eliminar elemento en el medio de la lista

```
escribir (“Ingrese dato a eliminar”);
leer (dato);
t:=prim;
escribir (“Buscar elemento  en la lista”);
mientras (t<> Null ) ∧ (*t.valor <> dato) )hacer
      aux:= t;
     t:=*t.proximo;
fin mientras

si (t=null) entonces escribir (“No se encontró el elemento en la lista”);
    sino
          *aux.proximo:= *t.proximo;
            disponer(t);
 fin si
```