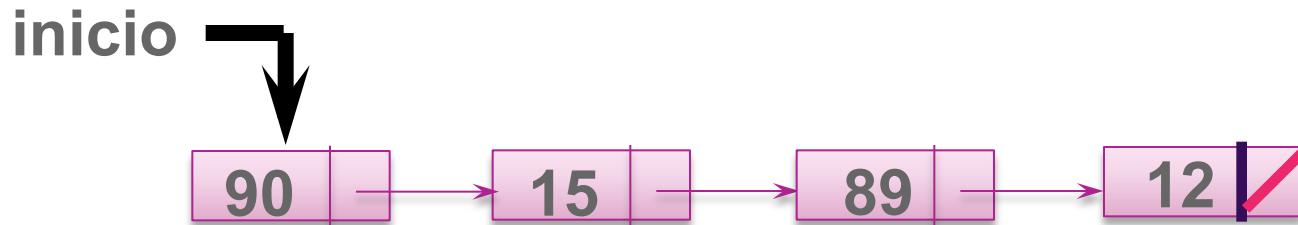


Listas Encadenadas Lineales

Ing. Armandina Leal Flores

Lista Encadenada

- ▶ Estructura de datos que se caracteriza porque cada elemento indica dónde se encuentra el siguiente.
- ▶ Su tipo de organización es lineal.
- ▶ El orden de entrada de los elementos a la estructura depende del uso que se le va a dar a la lista.

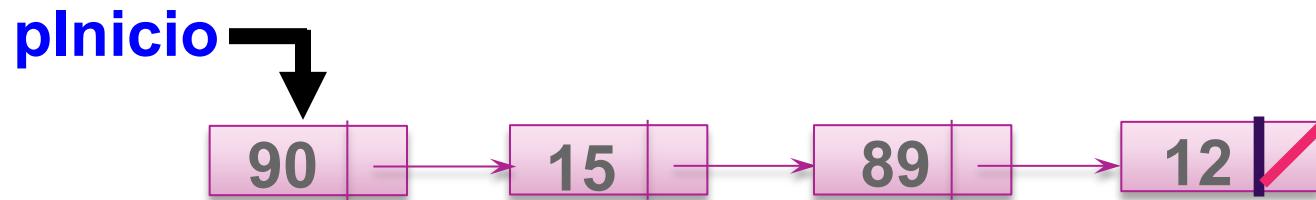


Uso de una Lista Encadenada

- ▶ En cualquier aplicación que se desconoce la cantidad de datos que se van a almacenar.
- ▶ Una Lista Encadenada se puede utilizar para guardar información ordenada.
- ▶ También se utiliza para implementar otras estructuras de datos como la Pila y la Fila.

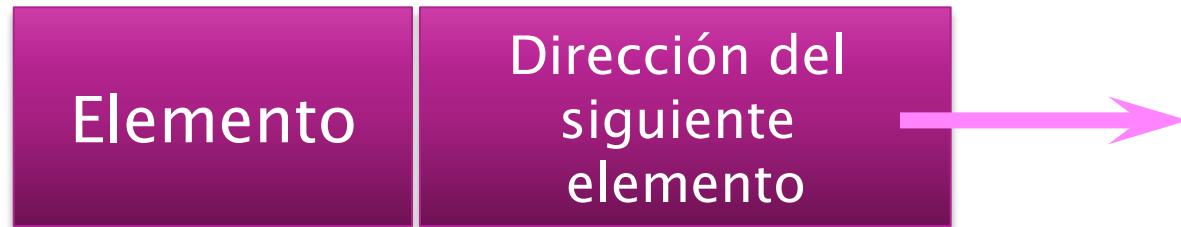
Implementación de la lista en Memoria Dinámica

- ▶ La variable apuntador **pInicio** indica cuál es el primer elemento de la lista.
- ▶ Cada elemento cuenta con un apuntador al siguiente elemento.
- ▶ El apuntador del último elemento tiene el valor **NULL** para indicar que ya se acabaron los elementos.



Nodo

- ▶ Un **NODO** en una lista encadenada es el espacio de memoria que guarda:
 - el elemento de la lista y
 - la dirección del siguiente elemento.



Implementación de la lista

- ▶ Para implementar una Lista Encadenada se necesita:
 - Implementar la clase Nodo e
 - Implementar la clase Lista
- ▶ El contenido de la clase Nodo debe estar en public para que los métodos de la lista puedan acceder directamente al contenido del nodo.

Clase Nodo

```
class Nodo
{ public:
    Nodo *pSig;
    int iInfo;
    Nodo( ) { pSig = NULL; }
    Nodo (int iDato) { iInfo = iDato; pSig = NULL; }
};
```

iInfo: variable que guarda el elemento.
Esta variable debe ser del tipo de dato que le corresponde al elemento a guardar.

pSig: variable apuntador que contendrá la dirección del siguiente Nodo.

Clases Lista Encadenada

```
class Lista  
{ public:
```

```
    Lista ( );
```

Crea una lista vacía

```
    ~Lista( );
```

Libera todos los nodos que aún quedan en la lista

```
    void meterLista(int iValor);
```

Agrega un nodo a la lista

```
    void sacarLista(int iValor);
```

Borra un nodo de la lista

```
    bool listaVacia( );
```

Determina si la lista está vacía

```
    int observaLista();
```

Regresa el siguiente elemento a sacar de la lista

```
private:
```

```
    Nodo *pInicio;
```

Apuntador al primer elemento de la lista

```
};
```

Se está suponiendo que la lista almacena valores tipo *int*

Ejemplo:

Agregar el primer nodo a la lista

ALTERNATIVA 1

```
Nodo *pP;           // P es una variable apuntador  
pP = new Nodo;     // crea el nodo con sig = NULL  
pP -> iInfo = iDato; // guarda el Dato en el nodo  
pInicio = pP;       // inicio apunta al nuevo nodo
```

ALTERNATIVA 2

```
Nodo *pP;           // P es una variable apuntador  
pP = new Nodo (iDato); // crea el nodo con info igual a Dato  
                      // y sig igual a NULL  
pInicio = pP;         //inicio apunta al nuevo nodo
```

Ejemplo:

Agregar un nodo con dato 12 después del nodo pP

Nodo *pQ;

pInicio

90

pP

15

16

pQ = new Nodo(iDato);

pQ

12

pQ -> pSig = pP -> pSig;

pInicio

90

pP

15

16

pQ

12

pP -> pSig = pQ;

pInicio

90

pP

15

16

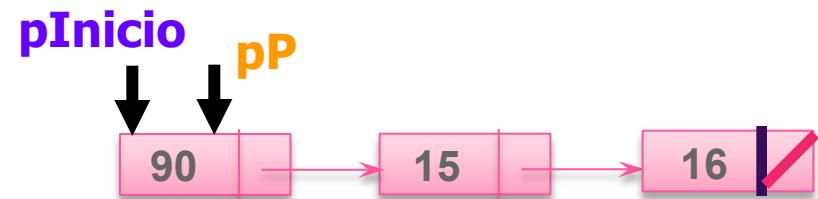
pQ

12

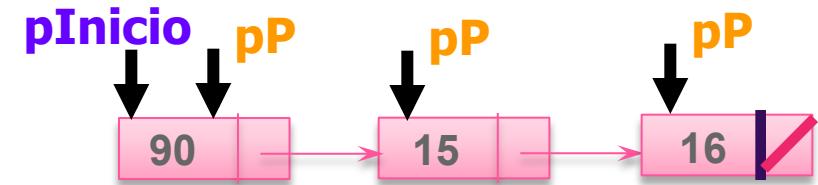
Ejemplo:

Despliega el contenido de la lista

```
Nodo *pP;  
pP = pInicio;
```



```
while (pP != NULL)  
{  
    cout << pP -> info << " ";  
    pP = pP -> pSig;  
}
```



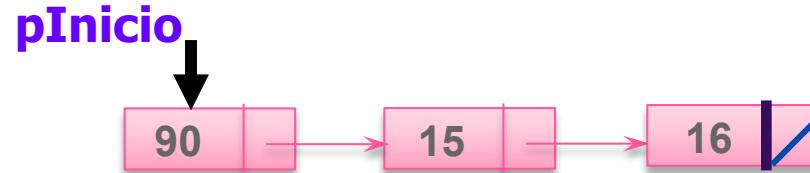
pP avanza en cada ciclo

Desplegado en Pantalla

90 15 16

Ejemplo: Agrega un nodo con dato 12 al inicio de la lista

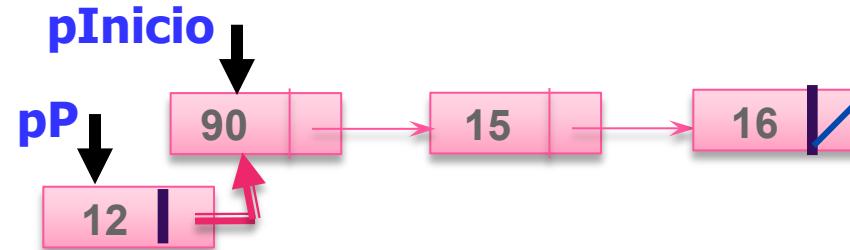
Nodo *pP;



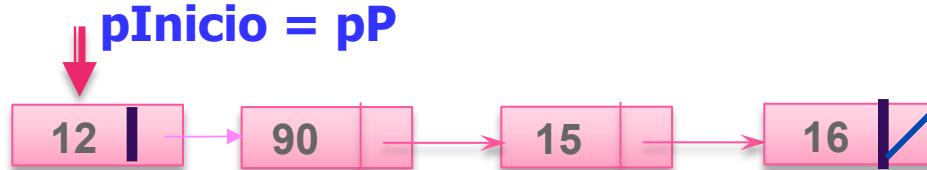
pP = new Nodo(iDato);



pP -> pSig = pInicio;

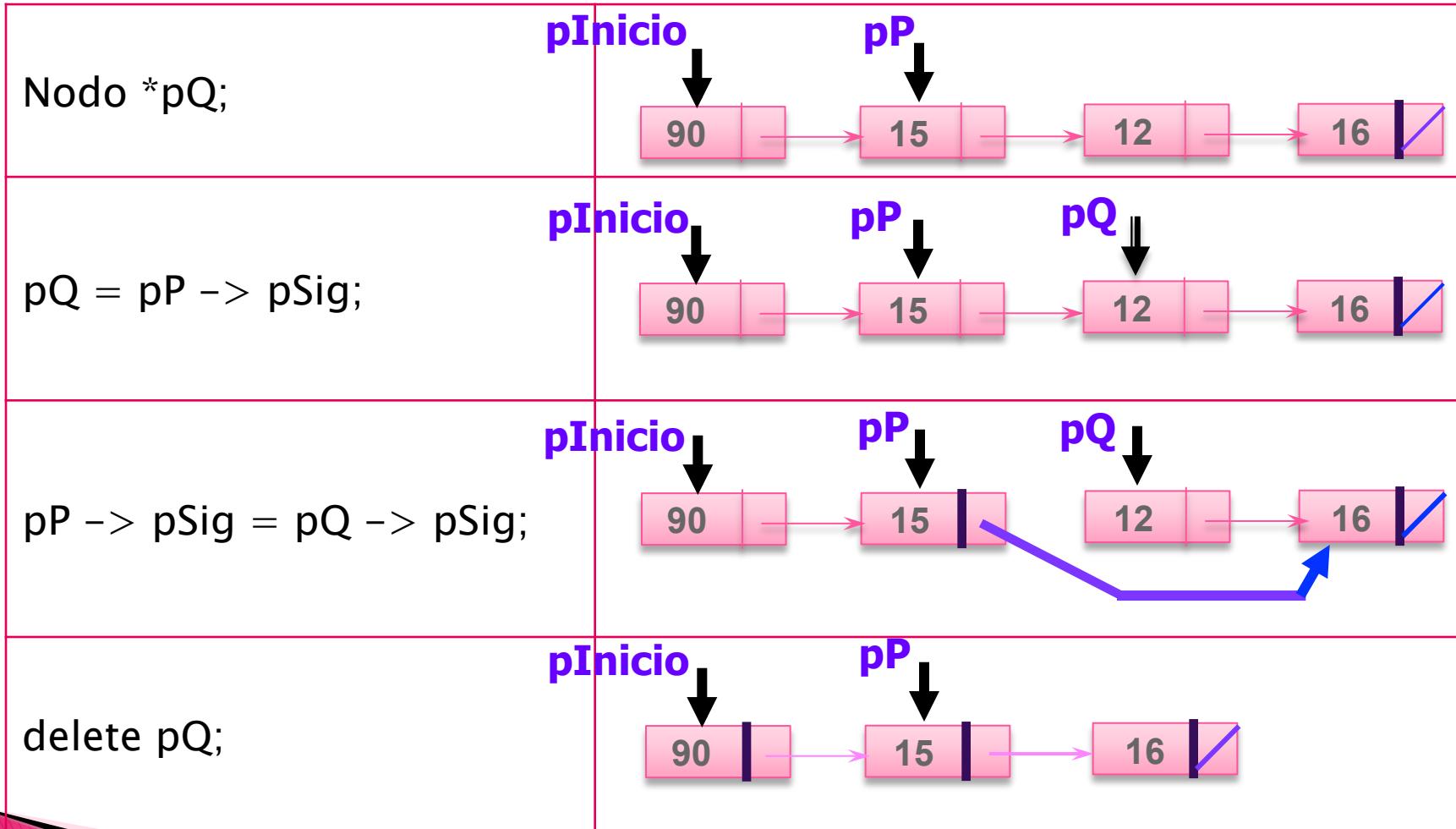


pInicio = pP;



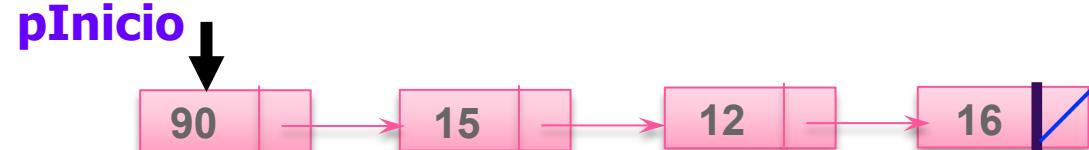
Si **pInicio=NULL**, agrega el primer nodo de la lista?

Ejemplo: Borra el nodo que está después del nodo pP



Ejemplo: Borra el primer nodo de la lista

Nodo *pP;



pP = pInicio;



pInicio = pInicio -> pSig;



delete pP;

