

Registros y Listas Enlazadas



Dr. Jaime Osorio Ubaldo

Registros

Un registro es una estructura de datos formado por un conjunto de elementos llamados campos, no necesariamente del mismo tipo y que permiten almacenar una serie de datos relacionados entre sí bajo un nombre común.

Arreglo:

a	c	e	i	u
---	---	---	---	---

Matriz:

1	2	6
0	1	4
0	4	9

Registros

Un registro es una estructura de datos formado por un conjunto de elementos llamados campos, no necesariamente del mismo tipo y que permiten almacenar una serie de datos relacionados entre sí bajo un nombre común.

Tipo Registro < identificador >

< Tipo de dato 1 > variable 1 ✓

< Tipo de dato 2 > variable 2 ✓

...

< Tipo de dato n > variable n ✓

Fin Registro

Registros

Ejemplo.

Tipo Registro Producto

✓ Cadena nombre ✓

✓ Entero cantidad ✓

✓ Real precio ✓

Real peso ✓

Fin registro

Arroz
12 sacos
2.5 soles
50 kg

Registros

Ejemplo.

Tipo Registro Producto

Cadena nombre

Entero cantidad

Real precio

Real peso

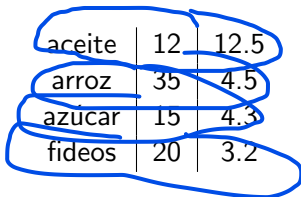
Fin registro

En C++

```
struct producto{  
    string nombre;  
    int cantidad;  
    float precio;  
    double peso;  
};
```

Asuma un arreglo de los registros de las ventas de los productos del día. Cada registro de un producto contiene campos para el nombre del producto, cantidad del producto y precio unitario del producto. Determine el ingreso al final del día.

Asuma un arreglo de los registros de las ventas de los productos del día. Cada registro de un producto contiene campos para el nombre del producto, cantidad del producto y precio unitario del producto. Determine el ingreso al final del día.



aceite	12	12.5
arroz	35	4.5
azúcar	15	4.3
fideos	20	3.2

En la siguiente tabla se muestra las ventas trimestrales, del 2020, de una empresa en varias ciudades del mundo.

ciudad	trimestre 1	trimestre 2	trimestre 3	trimestre 4
Lima	11 234	13 245	10 123	15 021
Roma	10 234	11 445	11 423	13 025
Paris	10 234	15 245	13 123	12 021
Tokio	10 236	14 645	11 123	14 021

- 1 Muestre en consola la tabla anterior.

En la siguiente tabla se muestra las ventas trimestrales, del 2020, de una empresa en varias ciudades del mundo.

ciudad	trimestre 1	trimestre 2	trimestre 3	trimestre 4
Lima	11 234	13 245	10 123	15 021
Roma	10 234	11 445	11 423	13 025
Paris	10 234	15 245	13 123	12 021
Tokio	10 236	14 645	11 123	14 021

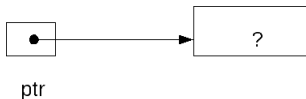
- 1 Muestre en consola la tabla anterior.
- 2 Mostrar la ciudad que tuvo la mayor y menor venta en todo el año.

En la siguiente tabla se muestra las ventas trimestrales, del 2020, de una empresa en varias ciudades del mundo.

ciudad	trimestre 1	trimestre 2	trimestre 3	trimestre 4
Lima	11 234	13 245	10 123	15 021
Roma	10 234	11 445	11 423	13 025
Paris	10 234	15 245	13 123	12 021
Tokio	10 236	14 645	11 123	14 021

- 1 Muestre en consola la tabla anterior.
- 2 Mostrar la ciudad que tuvo la mayor y menor venta en todo el año.
- 3 Ordene la tabla de forma descendente a las ventas anuales.

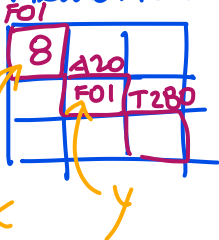
Punteros



- 1 Una variable tipo puntero es una variable cuyo contenido es la dirección de otra variable.

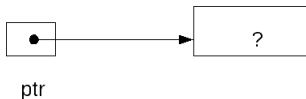
Punteros

memoria



$\text{int } x$
 $x = 8$

$\text{int } *y$
 $y = \&x$



- 1 Una variable tipo puntero es una variable cuyo contenido es la dirección de otra variable.

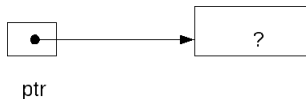
- 2 Declaración:

< tipo de dato apuntado >

* < nombre de la variable >

$\text{int } *ptr$
 $\text{float } *p$
 $\text{t_producto } *pp$

Punteros

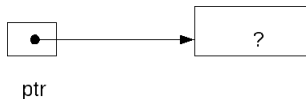


- 1 Una variable tipo puntero es una variable cuyo contenido es la dirección de otra variable.
- 2 Declaración:

*< tipo de dato apuntado > * < nombre de la variable >*

int *p | p es un variable tipo puntero que apunta a una variable tipo int.

Punteros



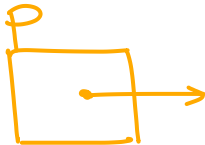
- 1 Una variable tipo puntero es una variable cuyo contenido es la dirección de otra variable.
- 2 Declaración:

*< tipo de dato apuntado > * < nombre de la variable >*

int *p	p es un variable tipo puntero que apunta a una variable tipo int.
float *q	q es un variable tipo puntero que apunta a una variable tipo float

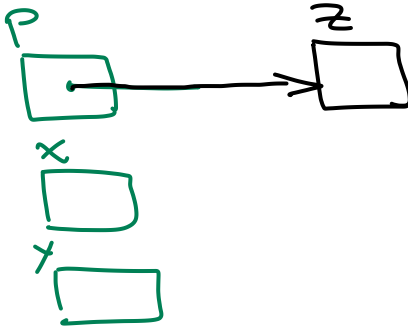
Punteros

```
float *p;
```



Punteros

```
float *p;  
float x,y,z;
```



$p = \&z$

Punteros

```
float *p;  
float x,y,z;  
x=5;  
y=-4;  
z=8.2
```

Punteros

```
float *p;  
float x,y,z;  
x=5;  
y=-4;  
z=8.2  
p=&x;
```

Punteros

```
float *p;
```

```
float x,y,z;
```

```
x=5;
```

```
y=-4;
```

```
z=8.2
```

```
p=&x;
```

Podemos leer el valor apuntado (leer x).

```
z=*p;
```

Punteros

```
float *p;
```

```
float x,y,z;
```

```
x=5;
```

```
y=-4;
```

```
z=8.2
```

```
p=&x;
```

Podemos leer el valor apuntado (leer x).

```
z=*p;
```

Podemos asignar al valor apuntado (modificar x).

```
*p=10;
```

Punteros

```
float *p;
```

```
float x,y,z;
```

```
x=5;
```

```
y=-4;
```

```
z=8.2
```

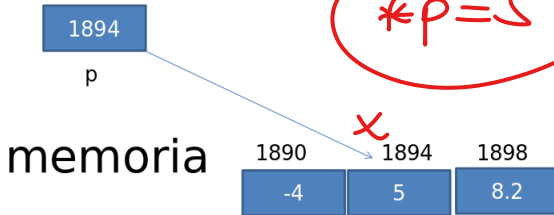
```
p=&x; ✓
```

Podemos leer el valor apuntado (leer x).

```
z=*p;
```

Podemos asignar al valor apuntado (modificar x).

```
*p=10;
```



Puntero a Estructura

```
struct producto{  
    string nombre;  
    int cantidad;  
    float precio;  
    double peso;  
};
```

Puntero a Estructura

```
struct producto{  
    string nombre;  
    int cantidad;  
    float precio;  
    double peso;  
};  
typedef struct producto *pproducto;
```

int x
pproducto p

Variable tipo puntero

Puntero a Estructura

```
struct producto{  
    string nombre;  
    int cantidad;  
    float precio;  
    double peso;  
};  
typedef struct producto *pproducto;  
pproducto p;
```


Puntero a Estructura

```
struct producto{  
    string nombre;  
    int cantidad;  
    float precio;  
    double peso;  
};
```

```
typedef struct producto *pproducto;
```

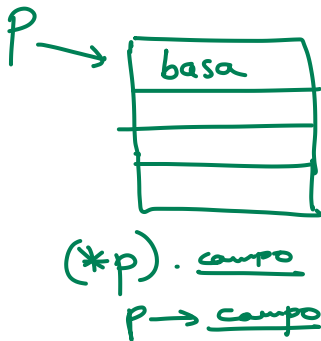
pproducto p; // declaración de la variable

p=new producto; // solicitar espacio de memoria



Puntero a Estructura

```
struct producto{  
    string nombre;  
    int cantidad;  
    float precio;  
    double peso;  
};  
typedef struct producto *pproducto;  
pproducto p;  
p=new producto;  
(*p).nombre="basa";
```

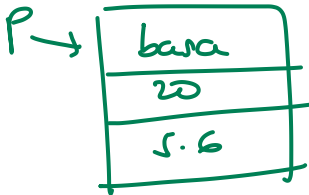


Puntero a Estructura

```
struct producto{  
    string nombre;  
    int cantidad;  
    float precio;  
    double peso;  
};  
typedef struct producto *pproducto;  
pproducto p;  
p=new struct producto;  
(*p).nombre="basa";  
(*p).cantidad=20;
```

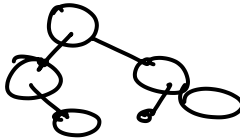
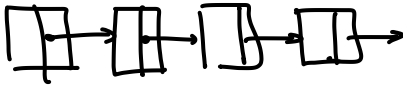
Puntero a Estructura

```
struct producto{  
    string nombre;  
    int cantidad;  
    float precio;  
    double peso;  
};  
typedef struct producto *pproducto;  
pproducto p;  
p=new producto;  
(*p).nombre="basa";  
(*p).cantidad=20;  
p->precio=5.6;
```



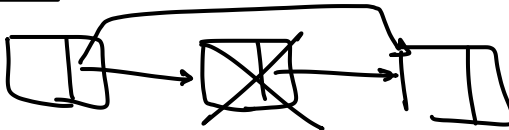
Listas Enlazadas, Pila, Cola, Árboles, Grafos

- 1 En estas estructuras es posible modificar el espacio ocupado en memoria durante la ejecución del programa.



Estructuras de Datos Dinámicas

- 1 En estas estructuras es posible modificar el espacio ocupado en memoria durante la ejecución del programa.
- 2 Permite hacer un optimo uso de la memoria, usar cuando se necesite y liberar cuando ya no se necesite.



Estructuras de Datos Dinámicas

- ① En estas estructuras es posible modificar el espacio ocupado en memoria durante la ejecución del programa.
- ② Permite hacer un optimo uso de la memoria, usar cuando se necesite y liberar cuando ya no se necesite.
- ③ Son estructuras de datos dinámicas las listas enlazadas, pilas, colas, árboles y grafos.





Una Lista se define como una sucesión de n elementos e_1, e_2, \dots, e_n ordenados de manera consecutiva, es decir el elemento e_k es el elemento anterior a e_{k+1} .

Listas



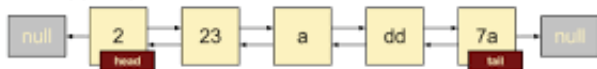
Una Lista se define como una sucesión de n elementos e_1, e_2, \dots, e_n ordenados de manera consecutiva, es decir el elemento e_k es el elemento anterior a e_{k+1} . Si la lista no tiene elemento es denominada lista vacía.



Una Lista se define como una sucesión de n elementos e_1, e_2, \dots, e_n ordenados de manera consecutiva, es decir el elemento e_k es el elemento anterior a e_{k+1} . Si la lista no tiene elemento es denominada lista vacía. Las operaciones que se pueden realizar en las listas son insertar un elemento en la posición k , eliminar el k -ésimo elemento, buscar un elemento en la lista, concatenar, dividirse en sublistas, etc.

Array vs. Linked List

Linked List

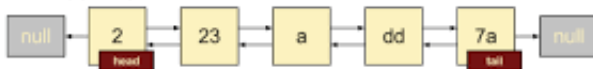


Array



Array vs. Linked List

Linked List



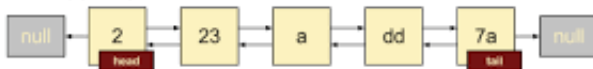
Array



- 1 En un arreglo el tamaño es predefinido pero en una lista enlazada puede crecer a medida que uno lo requiera.

Array vs. Linked List

Linked List



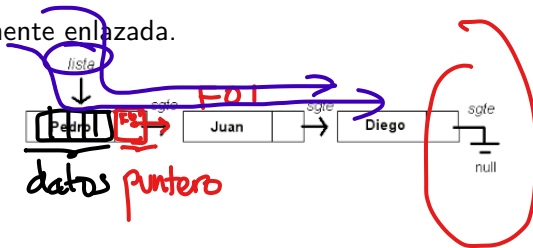
Array



- 1 En un arreglo el tamaño es predefinido pero en una lista enlazada puede crecer a medida que uno lo requiera.
- 2 En un arreglo todos son del mismo dato, en una lista puede tener distintos tipos de datos.

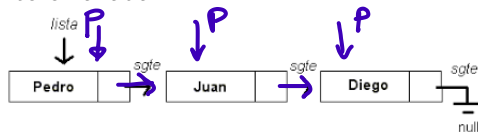
Lista con memoria dinámica

1 Lista simplemente enlazada.

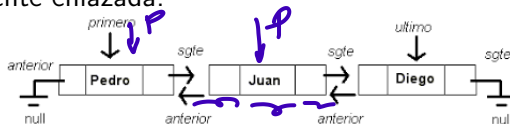


Lista con memoria dinámica

1 Lista simplemente enlazada.

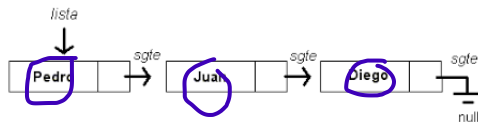


2 Lista doblemente enlazada.

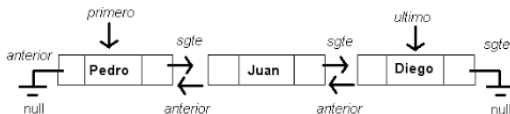


Lista con memoria dinámica

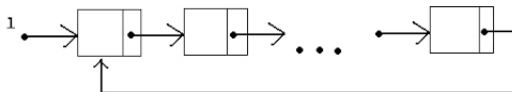
1 Lista simplemente enlazada.



2 Lista doblemente enlazada.



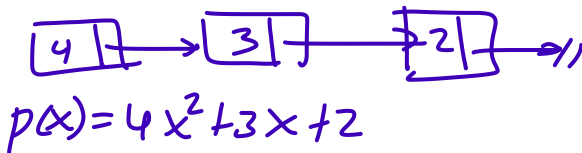
3 Lista circular.



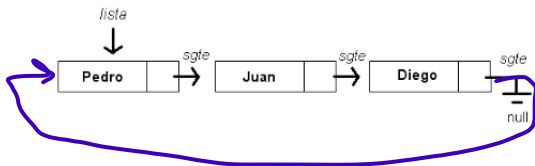
- 1 Puede ser utilizada para un cronograma de actividades diarias.

- 1 Puede ser utilizada para un cronograma de actividades diarias.
- 2 Puede ser utilizada para almacenar una sucesión de imágenes de un experimento.

- 1 Puede ser utilizada para un cronograma de actividades diarias.
- 2 Puede ser utilizada para almacenar una sucesión de imagenes de un experimento.
- 3 Representación de polinomios.

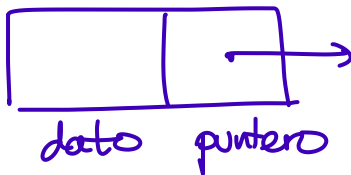
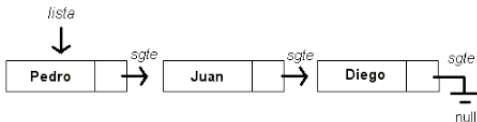


- 1 Puede ser utilizada para un cronograma de actividades diarias.
- 2 Puede ser utilizada para almacenar una sucesión de imagenes de un experimento.
- 3 Representación de polinomios.
- 4 Lista circular se podría utilizar también con el reproductor de música, para que en el momento en que termine la última canción vuelva a empezar con la primera, etc.

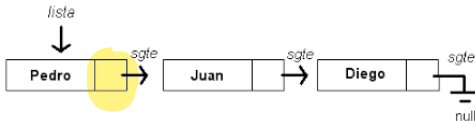


Lista Simplemente Enlazada

lista



Lista Simplemente Enlazada

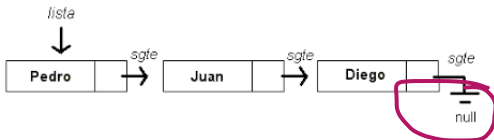


Debemos definir la estructura nodo

```
struct nodo{  
    int dato;  
    nodo *sig;  
};  
typedef nodo *pnodo;
```

↑ como un tipo de dato

Lista Simplemente Enlazada



Debemos definir la estructura nodo

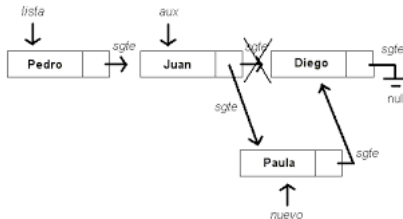
```
struct nodo{
int dato;
nodo *sig;
};
typedef struct nodo *pnodo;
```

Luego, declaramos una variable de tipo pnodo inicializamos

```
pnodo pL;
pL=NULL
```

lista →
LISTA VACÍA
lista →

Insertar nodo



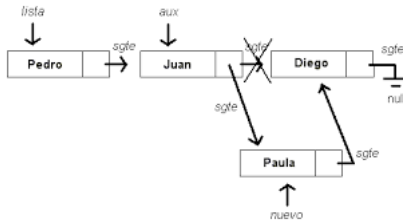
lista

pnode p
p = new node

P →

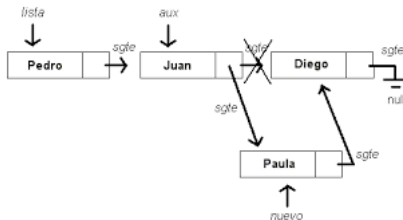
lista = P

Insertar nodo



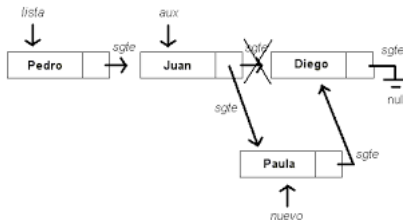
```
void insertar(int x){
```

Insertar nodo

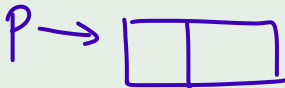


```
void insertar(int x){  
    pnode p;
```

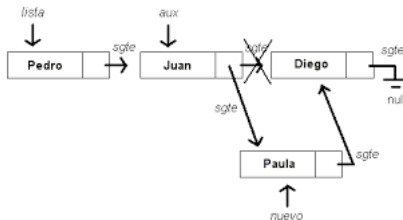
Insertar nodo



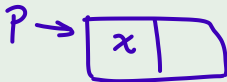
```
void insertar(int x){  
  pnode p;  
  p = new nodo;
```



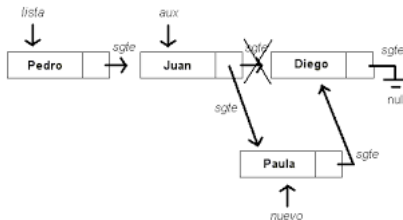
Insertar nodo



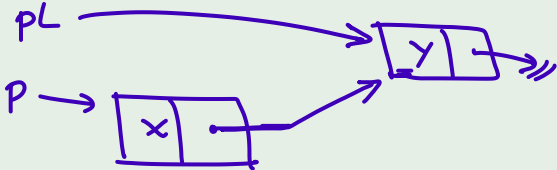
```
void insertar(int x){  
  pnode p;  
  p = new nodo;  
  (*p).dato = x;
```



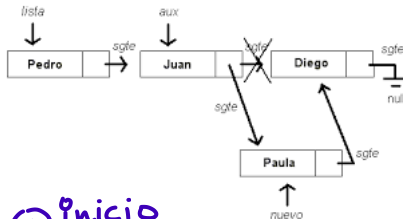
Insertar nodo



```
void insertar(int x){  
    pnode p;  
    p = new nodo;  
    (*p).dato = x;  
    (*p).sig = pL;
```



Insertar nodo



① Inicio

PL → X

② Insertar(4)

P → 4

③ Insertar(2)

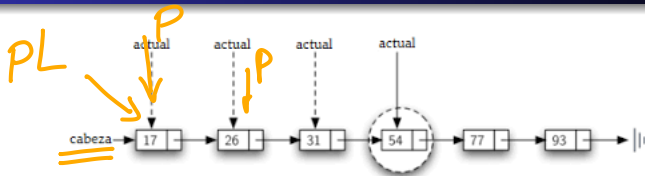
P → 2

PL → 4

PL → 2 → 4

```
void insertar(int x){  
    pnode p;  
    p = new nodo;  
    (*p).dato = x;  
    (*p).sig = pL;  
    pL = p;  
}
```

Buscar nodo



buscar(54)

p nodo p

$p = PL$

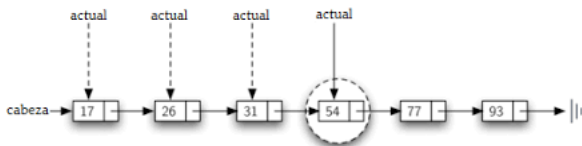
se $(54 = p \rightarrow \text{dato})$

OK ✓

sino

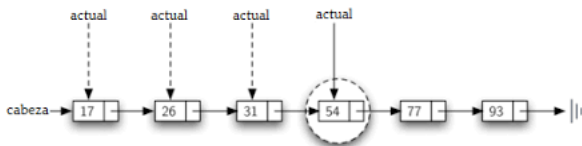
$p = p \rightarrow \text{sig}$

Buscar nodo



```
pnode buscar(int x)
```

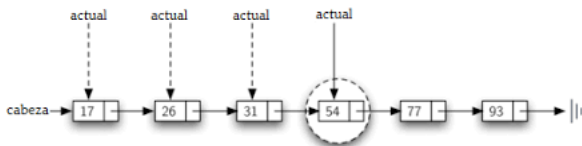
Buscar nodo



```
pnode buscar(int x){  
    pnode p;  
    if ( pL == NULL )
```

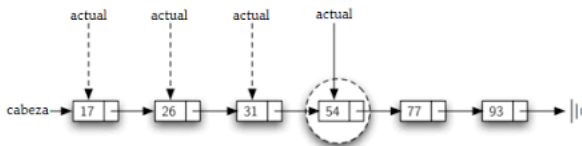
← verificando que la lista no se encuentre vacía.

Buscar nodo



```
pnode buscar(int x){  
  pnode p;  
  if ( pL == NULL )  
    return NULL;
```

Buscar nodo



```
pnode buscar(int x){
```

```
  pnode p;
```

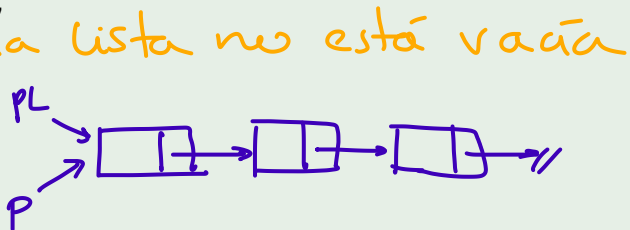
```
  if ( pL == NULL )
```

```
    return NULL;
```

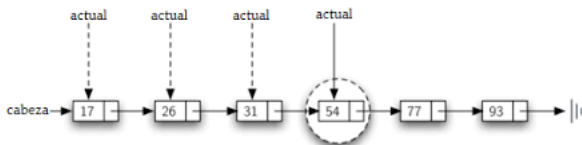
```
  else
```

```
  {
```

```
    p = pL;
```

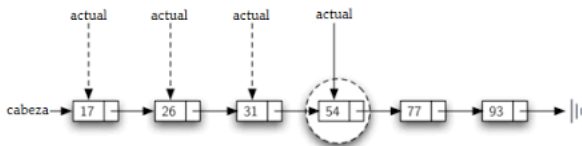


Buscar nodo

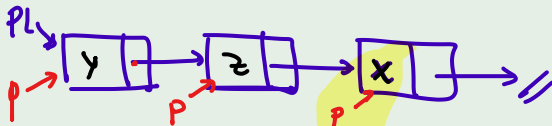


```
pnode buscar(int x){  
    pnode p;  
    if ( pL == NULL )  
        return NULL;  
    else  
    {  
        p = pL;  
        while ( p != NULL && (*p).dato != x )  
            p = (*p).sig;  
    }  
}
```

Buscar nodo

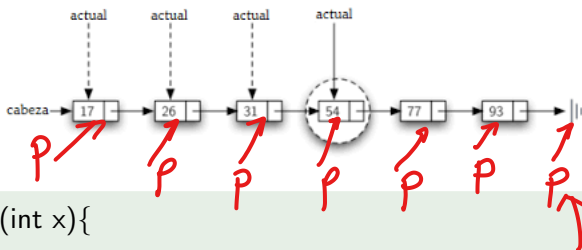


```
pnode buscar(int x){  
    pnode p;  
    if ( pL == NULL )  
        return NULL;  
    else  
    {  
        p = pL;  
        while ( p != NULL && (*p).dato != x )  
            p = (*p).sig;  
    }
```



Buscar nodo

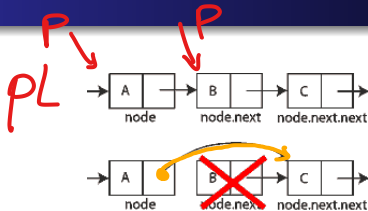
$x = 40$



```
pnode buscar(int x){  
    pnode p;  
    if ( pL == NULL )  
        return NULL;  
    else  
    {  
        p = pL;  
        while ( p != NULL && (*p).dato != x )  
            p = (*p).sig;  
        return p;  
    }  
}
```

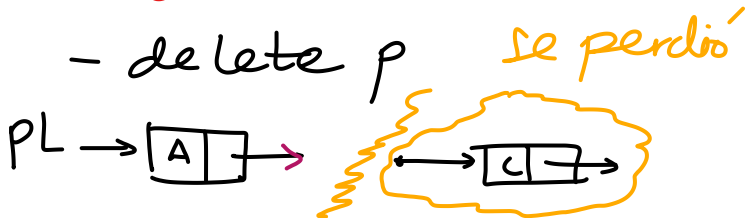
retorna NULL
NO LO
encontro

Eliminar nodo

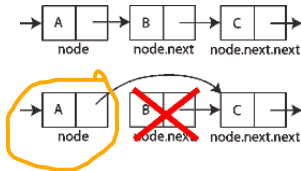


① buscar (B)

② eliminar(B)



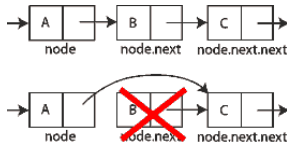
Eliminar nodo



Primero implementaremos la función buscar nodo anterior

```
pnodo buscaranterior(int x) {  
pnodo a,p;
```

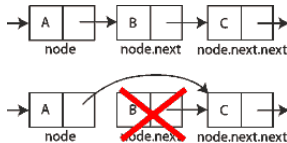
Eliminar nodo



Primero implementaremos la función buscar nodo anterior

```
pnode buscaranterior(int x) {  
    pnode a,p;  
    a = NULL;
```

Eliminar nodo



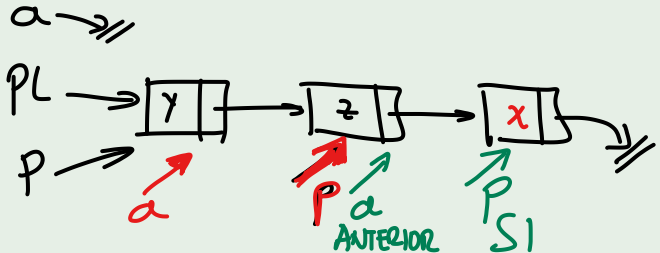
Primero implementaremos la función buscar nodo anterior

```
pnode buscaranterior(int x) {
```

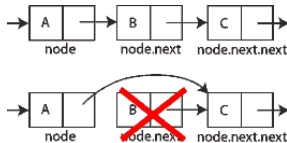
```
  pnode a,p;
```

```
  a = NULL;
```

```
  p = pL;
```



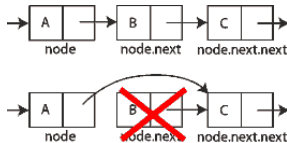
Eliminar nodo



Primero implementaremos la función buscar nodo anterior

```
pnode buscaranterior(int x) {  
    pnode a,p;  
    a = NULL;  
    p = pL;  
    while ((*p).dato != x)
```

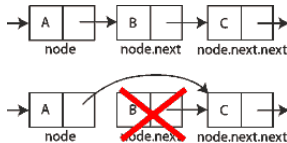
Eliminar nodo



Primero implementaremos la función buscar nodo anterior

```
pnode buscaranterior(int x) {  
    pnode a,p;  
    a = NULL;  
    p = pL;  
    while ((*p).dato != x){  
        a = p;
```

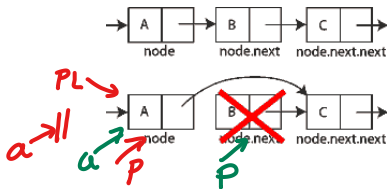
Eliminar nodo



Primero implementaremos la función buscar nodo anterior

```
pnode buscaranterior(int x) {  
    pnode a,p;  
    a = NULL;  
    p = pL;  
    while ((*p).dato != x){  
        a = p;  
        p = (*p).sig; }  
}
```

Eliminar nodo



Primero implementaremos la función buscar nodo anterior

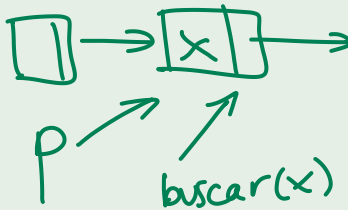
```
pnodo buscaranterior(int x) {  
    pnodo a,p;  
    a = NULL;  
    p = pL;  
    while ((*p).dato != x){  
        a = p; ✓  
        p = (*p).sig; }  
    return a;  
}
```

Eliminar nodo

```
void eliminar( int x )  
{  
    pnode a,p;
```


Eliminar nodo

```
void eliminar( int x )  
{  
  pnode a,p;  
  p = buscar( x );
```

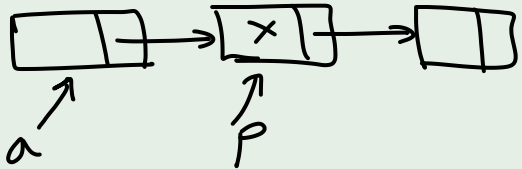


Eliminar nodo

```
void eliminar( int x )  
{  
    pnode a,p;  
    p = buscar( x );  
    if ( p == NULL )  
        cout<<"El elemento no esta en la lista;"  
    else
```

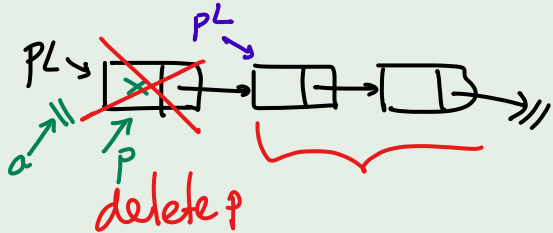
Eliminar nodo

```
void eliminar( int x )  
{  
    pnode a,p;  
    p = buscar( x );  
    if ( p == NULL )  
        cout<<"El elemento no esta en la lista;"  
    else{  
        a = buscaranterior(x);
```



Eliminar nodo

```
void eliminar( int x )
{
    pnodo a,p;
    p = buscar( x );
    if ( p == NULL )
        cout<<"El elemento no esta en la lista;
    else{
        a = buscaranterior(x);
        if ( a == NULL )
            pL = (*p).sig;
```



Eliminar nodo

```
void eliminar( int x )
{
    pnode a,p;
    p = buscar( x );
    if ( p == NULL )
        cout<<El elemento no esta en la lista;
    else{
        a = buscaranterior(x);
        if ( a == NULL )
            pL = (*p).sig;
        else
            (*a).sig = (*p).sig;
```

Eliminar nodo

```
void eliminar( int x )
{
    pnode a,p;
    p = buscar( x );
    if ( p == NULL )
        cout<<"El elemento no esta en la lista;
    else{
        a = buscaranterior(x);
        if ( a == NULL )
            pL = (*p).sig;
        else
            (*a).sig = (*p).sig;
        delete p; }
}
```

