

# CHƯƠNG 2: CẤU TRÚC TUẦN TỰ

## **DANH SÁCH LIÊN KẾT**

### (LINKED LISTS)



# Nội dung

2

- Giới thiệu
- Danh sách liên kết đơn (Single Linked List)
- Danh sách liên kết đôi (Double Linked List)
- Danh sách liên kết vòng (Circular Linked List)

# Mục tiêu

3

- Giới thiệu khái niệm cấu trúc dữ liệu động.
- Giới thiệu danh sách liên kết:
  - ▣ Các kiểu tổ chức dữ liệu theo DSLK.
  - ▣ Danh sách liên kết đơn: tổ chức, các thuật toán, ứng dụng.

# Giới thiệu - Cấu trúc dữ liệu tĩnh

4

- Cấu trúc dữ liệu tĩnh:
  - Khái niệm: Các đối tượng dữ liệu không thay đổi được kích thước, cấu trúc, ... trong suốt quá trình sống thuộc về kiểu dữ liệu tĩnh
  - Một số kiểu dữ liệu tĩnh: các cấu trúc dữ liệu được xây dựng từ các kiểu cơ sở như: **kiểu số thực, kiểu số nguyên, kiểu ký tự** ... hoặc từ các cấu trúc đơn giản như **mẫu tin, tập hợp, mảng** ...
- ➔ Các đối tượng dữ liệu được xác định thuộc những kiểu dữ liệu này thường cứng ngắt, gò bó ➔ khó diễn tả được thực tế vốn sinh động, phong phú.

# Giới thiệu - Cấu trúc dữ liệu tĩnh

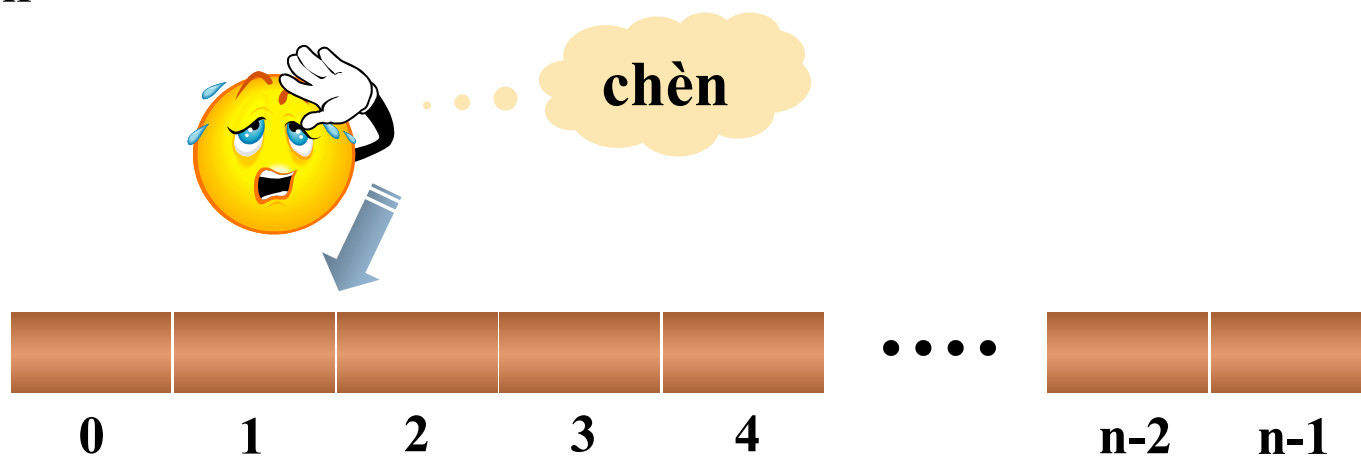
5

- Một số hạn chế của CTDL tĩnh:
  - Một số đối tượng dữ liệu trong chu kỳ sống của nó có thể thay đổi về cấu trúc, độ lớn,...
  - Ví dụ như danh sách các học viên trong một lớp học có thể tăng thêm, giảm đi ... Nếu dùng những cấu trúc dữ liệu tĩnh đã biết như mảng để biểu diễn → Những thao tác phức tạp, kém tự nhiên → chương trình khó đọc, khó bảo trì và nhất là khó có thể sử dụng bộ nhớ một cách có hiệu quả
  - Dữ liệu tĩnh sẽ chiếm vùng nhớ đã dành cho chúng suốt quá trình hoạt động của chương trình → sử dụng bộ nhớ kém hiệu quả

# Giới thiệu – Ví dụ cấu trúc dữ liệu tĩnh

6

- **Cấu trúc dữ liệu tĩnh:** Ví dụ: Mảng 1 chiều
  - ▣ Kích thước cố định (fixed size)
  - ▣ Các phần tử tuần tự theo chỉ số  $0 \Rightarrow n-1$
  - ▣ Truy cập ngẫu nhiên (random access)
  - ▣ Chèn 1 phần tử vào mảng, xóa 1 phần tử khỏi mảng tốn nhiều chi phí



# Giới thiệu - Cấu trúc dữ liệu động

7

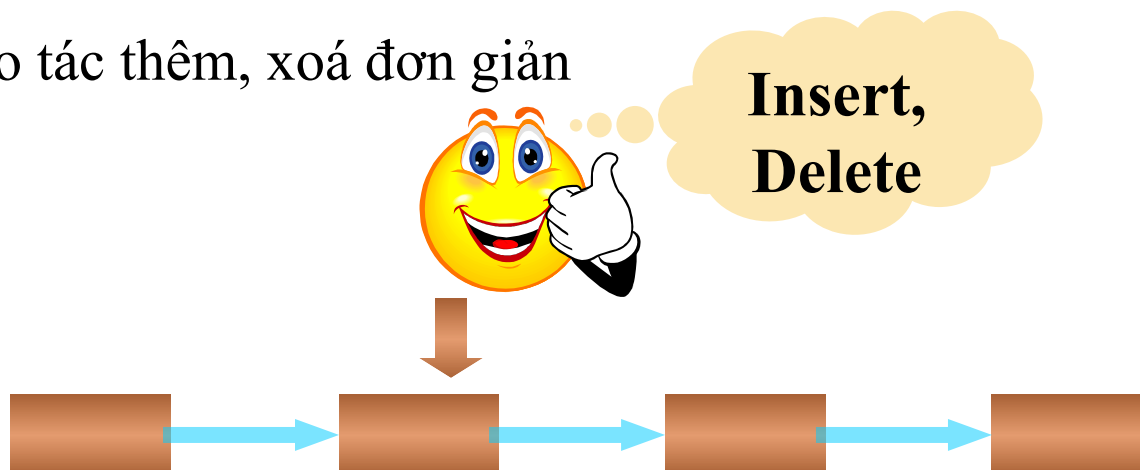
## Hướng giải quyết

- Cần xây dựng cấu trúc dữ liệu đáp ứng được các yêu cầu:
    - ▣ Linh động hơn
    - ▣ Có thể thay đổi kích thước, cấu trúc trong suốt thời gian sống
- *Cấu trúc dữ liệu động*

# Giới thiệu - Cấu trúc dữ liệu động

8

- **Cấu trúc dữ liệu động:** Ví dụ: **Danh sách liên kết, cây**
  - Cấp phát động lúc chạy chương trình
  - Các phần tử nằm rải rác ở nhiều nơi trong bộ nhớ
  - Kích thước danh sách chỉ bị giới hạn do RAM
  - Tốn bộ nhớ hơn (vì phải chứa thêm vùng liên kết)
  - Khó truy cập ngẫu nhiên
  - Thao tác thêm, xóa đơn giản





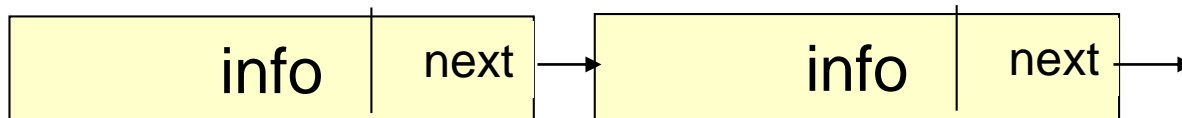
# Danh sách liên kết (List)

9

## I. Định nghĩa:

- Danh sách liên kết là một cấu trúc dữ liệu mà mỗi phần tử trong đó chứa dữ liệu và một con trỏ (tham chiếu) tới phần tử tiếp theo trong danh sách. Cấu trúc này giúp lưu trữ dữ liệu một cách linh hoạt và cho phép thêm/xóa các phần tử một cách dễ dàng.
- Một danh sách liên kết có thể có dạng như sau:
- Mỗi phần tử của nó gồm hai thành phần:
  - ▣ Phần chứa dữ liệu -Data
  - ▣ Phần chỉ vị trí của phần tử tiếp theo trong danh sách – Next

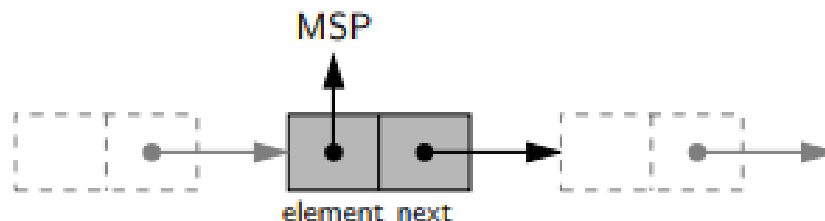
```
+---+ +---+ +---+ +---+
| 1 | -> | 2 | -> | 3 | -> | 4 |
+---+ +---+ +---+ +---+
```



# Giới thiệu - Danh sách liên kết

10

- Danh sách liên kết:
  - ▣ Mỗi phần tử của danh sách gọi là **node** (nút)
  - ▣ Mỗi **node** có 2 thành phần: **phần dữ liệu** và **phần liên kết** (**phần liên kết** chứa địa chỉ của node kế tiếp hay node trước nó)
  - ▣ Các thao tác cơ bản trên danh sách liên kết:
    - Thêm một phần tử mới
    - Xóa một phần tử
    - Tìm kiếm
    - ...



# Giới thiệu - Danh sách liên kết

11

- Tính chất danh sách liên kết:
  - ▣ DSLK có thể mở rộng và thu hẹp 1 cách linh hoạt
  - ▣ Các phần tử trong DSLK được gọi là Node, được cấp phát động.
  - ▣ Phần tử cuối cùng trong DSLK trở vào Null
  - ▣ Không lãng phí bộ nhớ nhưng cần thêm bộ nhớ để lưu phần con trỏ.
  - ▣ Đây là CTDL cấp phát động nên khi còn bộ nhớ thì sẽ còn thêm được 1 phần tử vào DSLK



# Giới thiệu - Danh sách liên kết

12

- Ưu điểm danh sách liên kết:
  - ▣ Dễ dàng mở rộng và thu hẹp kích thước
  - ▣ Có thể mở rộng với độ phức tạp là hằng số
  - ▣ Có thể cấp phát với số lượng lớn các node tùy vào bộ nhớ
- Nhược điểm danh sách liên kết:
  - ▣ Khó khăn trong việc truy cập 1 phần tử ở 1 phần tử bất kỳ  $O(n)$
  - ▣ Khó khăn trong việc cài đặt
  - ▣ Tốn thêm bộ nhớ trong phần tham chiếu bổ sung.



# Giới thiệu - Danh sách liên kết

13

## □ Độ phức tạp của các thao tác với mảng và DSLK

Thao tác	DSLK	Mảng
Truy xuất phần tử	$O(n)$	$O(1)$
Chèn/Xóa ở đầu	$O(1)$	$O(n)$ nếu mảng chưa full
Chèn ở cuối	$O(n)$	$O(1)$ nếu mảng chưa full
Xóa ở cuối	$O(n)$	$O(1)$
Chèn giữa	$O(n)$	$O(n)$ nếu mảng chưa full
Xóa giữa	$O(n)$	$O(n)$



# Giới thiệu - Danh sách liên kết

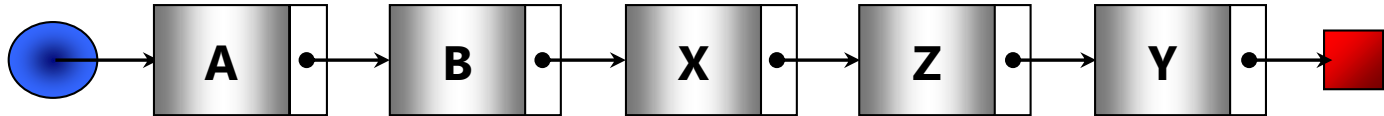
14

- Có nhiều kiểu tổ chức liên kết giữa các phần tử trong danh sách như:
  - ▣ Danh sách liên kết đơn
  - ▣ Danh sách liên kết kép
  - ▣ Danh sách liên kết vòng

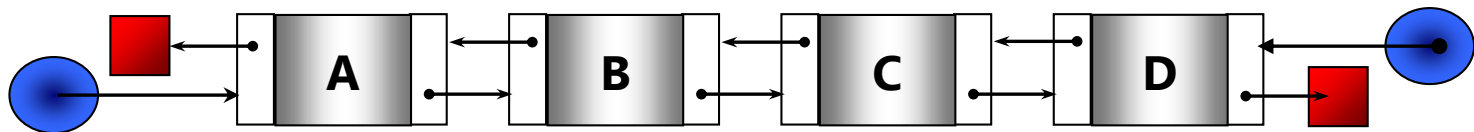
# Giới thiệu - Danh sách liên kết

15

- **Danh sách liên kết đơn:** mỗi phần tử liên kết với 1 phần tử đứng sau nó trong danh sách:



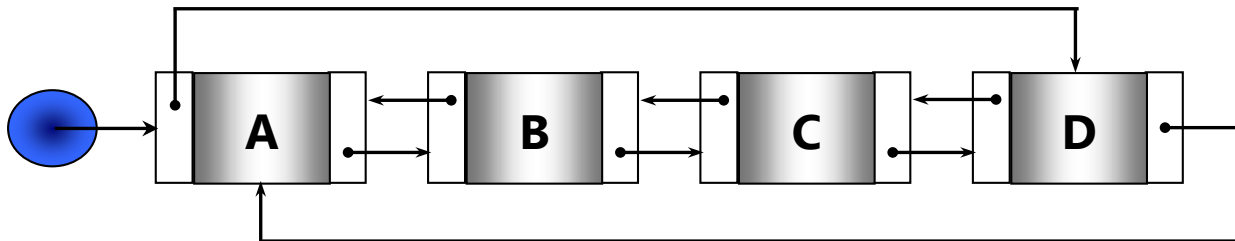
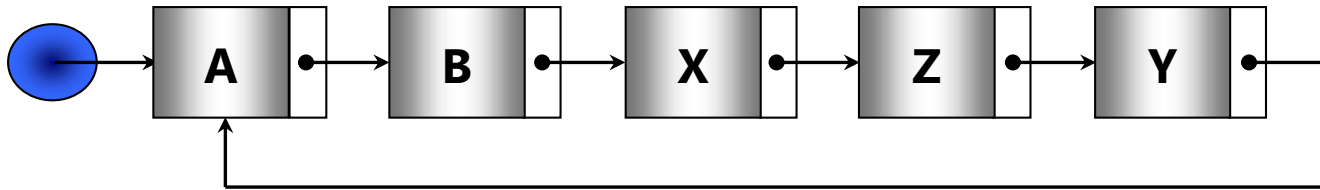
- **Danh sách liên kết kép:** mỗi phần tử liên kết với các phần tử đứng trước và sau nó trong danh sách:



# Giới thiệu - Danh sách liên kết

16

- **Danh sách liên kết vòng** : phần tử cuối danh sách liên kết với phần tử đầu danh sách:





# Nội dung

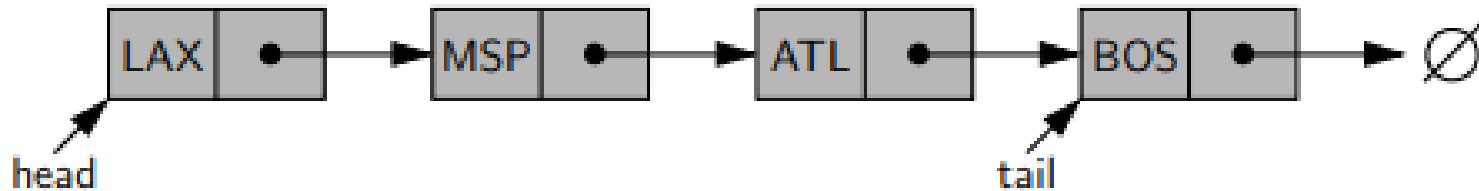
17

- Giới thiệu
- Danh sách liên kết đơn (Single Linked List)
- Danh sách liên kết kép (Double Linked List)
- Danh sách liên kết vòng (Circular Linked List)

# Danh sách liên kết đơn (DSLK đơn)

18

- Khai báo
- Các thao tác cơ bản trên DSLK đơn
- Sắp xếp trên DSLK đơn



# DSLK đơn – Khai báo

19

- Là danh sách các node mà mỗi node có 2 thành phần:
  - ▣ Thành phần **dữ liệu**: lưu trữ các thông tin về bản thân phần tử
  - ▣ Thành phần **mối liên kết**: lưu trữ địa chỉ của phần tử kế tiếp trong danh sách, hoặc lưu trữ giá trị **NULL** nếu là phần tử cuối danh sách



- ▣ Khai báo node:

```
struct Node
```

```
{
```

```
    DataType data; // DataType là kiểu đã định nghĩa trước
```

```
    Node *pNext; // con trỏ chỉ đến cấu trúc Node
```

```
};
```

```
Node* tên_nút;
```

## Định nghĩa 1 nút

```
1  #TẠO NÚT
2  class Nut:
3      def __init__(self,gia_tri):
4          self.gia_tri =gia_tri
5          self.nut_ke_tiep=None
6      #def Định nghĩa hàm khởi tạo node
7      #class
8  class DSLienKet:
9      def __init__(self):
10         self.dau = None
11         self.duoi =None
12     #def Định nghĩa danh sách ban đầu
```

# Lưu trữ DSLK đơn trong RAM

Địa chỉ

- Ví dụ : Ta có danh sách theo dạng bảng sau

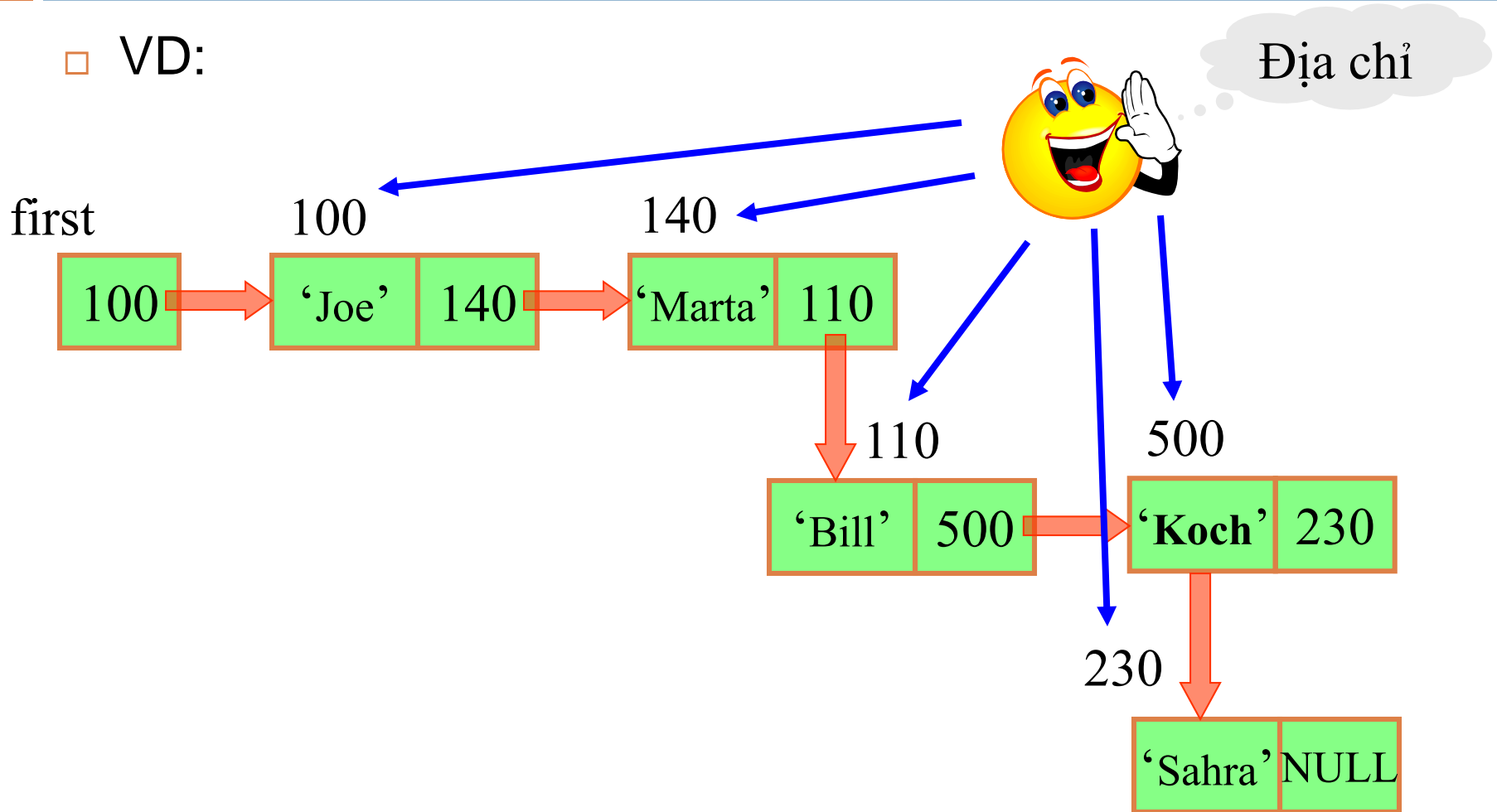
Address	Name	Age	Link
100	Joe	20	140
110	Bill	42	500
140	Marta	27	110
230	Sahra	25	NULL
...	...	...	
500	Koch	31	230

	000
Joe	100
140	
Bill	110
500	
Marta	140
110	
Sahra	230
NULL	
Kock	500
230	
	⋮

# DSLK đơn truy xuất – Minh họa

22

□ VD:



# DSLK đơn – Khai báo

23

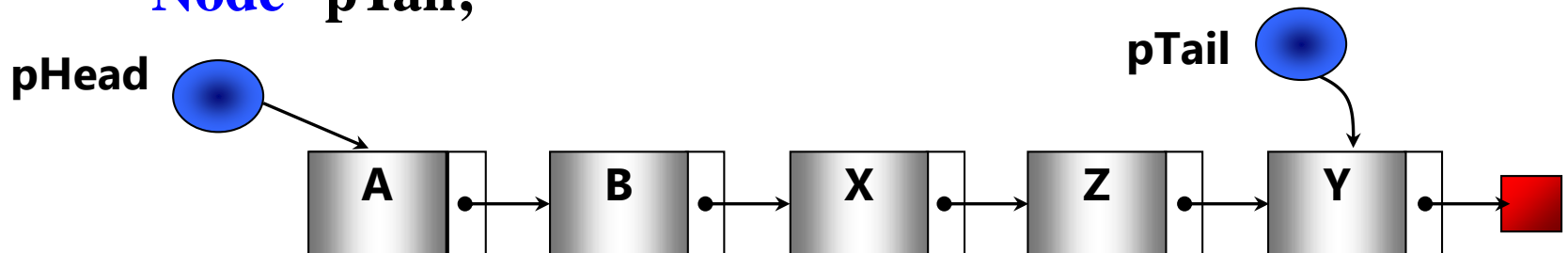
## □ Tổ chức, quản lý:

- Để quản lý một DSLK đơn chỉ cần biết địa chỉ **phần tử đầu danh sách**
- Con trỏ **pHead** sẽ được dùng để lưu trữ địa chỉ phần tử đầu danh sách. Ta có khai báo:

**Node \*pHead;**

- Để tiện lợi, có thể sử dụng thêm một con trỏ **pTail** giữ địa chỉ **phần tử cuối danh sách**. Khai báo **pTail** như sau:

**Node \*pTail;**



# Danh sách liên kết đơn (DSLK đơn)

24

- Khai báo
- Các thao tác cơ bản trên DSLK đơn
  - Tạo danh sách rỗng
  - Thêm một phần tử vào danh sách
  - Duyệt danh sách
  - Tìm kiếm
  - Xóa một phần tử ra khỏi danh sách
  - Hủy toàn bộ danh sách
  - ...
- Sắp xếp trên DSLK đơn



# DSLK đơn – Khai báo

25

Để tạo một phần tử mới cho danh sách, cần thực hiện câu lệnh:

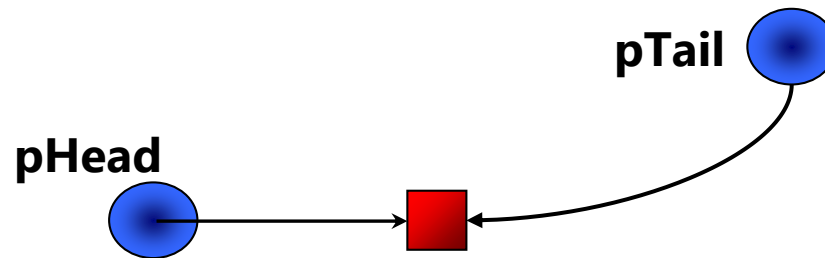
```
new_ele = GetNode(x);
```

→ new\_ele sẽ quản lý địa chỉ của phần tử mới được tạo.

# DSLK đơn – Các thao tác cơ sở

26

- Tạo danh sách rỗng



```
8 class DSLienKet:
9     def __init__(self):
10         self.dau = None
11         self.duoi = None
```

# DSLK đơn

27

- **Các thao tác cơ bản**
  - Tạo danh sách rỗng
  - Thêm một phần tử vào danh sách
  - Duyệt danh sách
  - Tìm kiếm một giá trị trên danh sách
  - Xóa một phần tử ra khỏi danh sách
  - Hủy toàn bộ danh sách
  - ...

# DSLK đơn – Các thao tác cơ sở

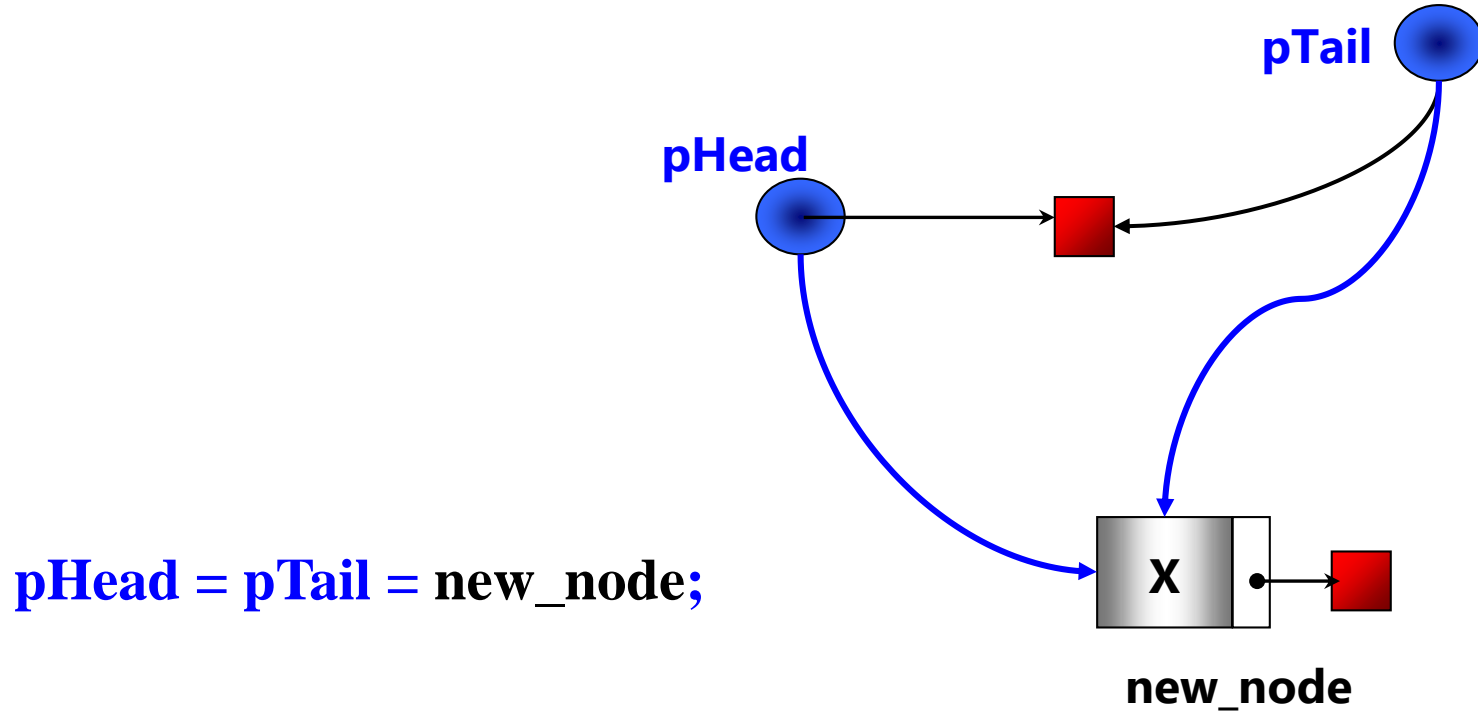
28

- **Thêm một phần tử vào danh sách:** Có 3 vị trí thêm
  - Gắn vào đầu danh sách
  - Gắn vào cuối danh sách
  - Chèn vào sau nút q trong danh sách
- Chú ý trường hợp danh sách ban đầu rỗng

# DSLK đơn – Các thao tác cơ sở

29

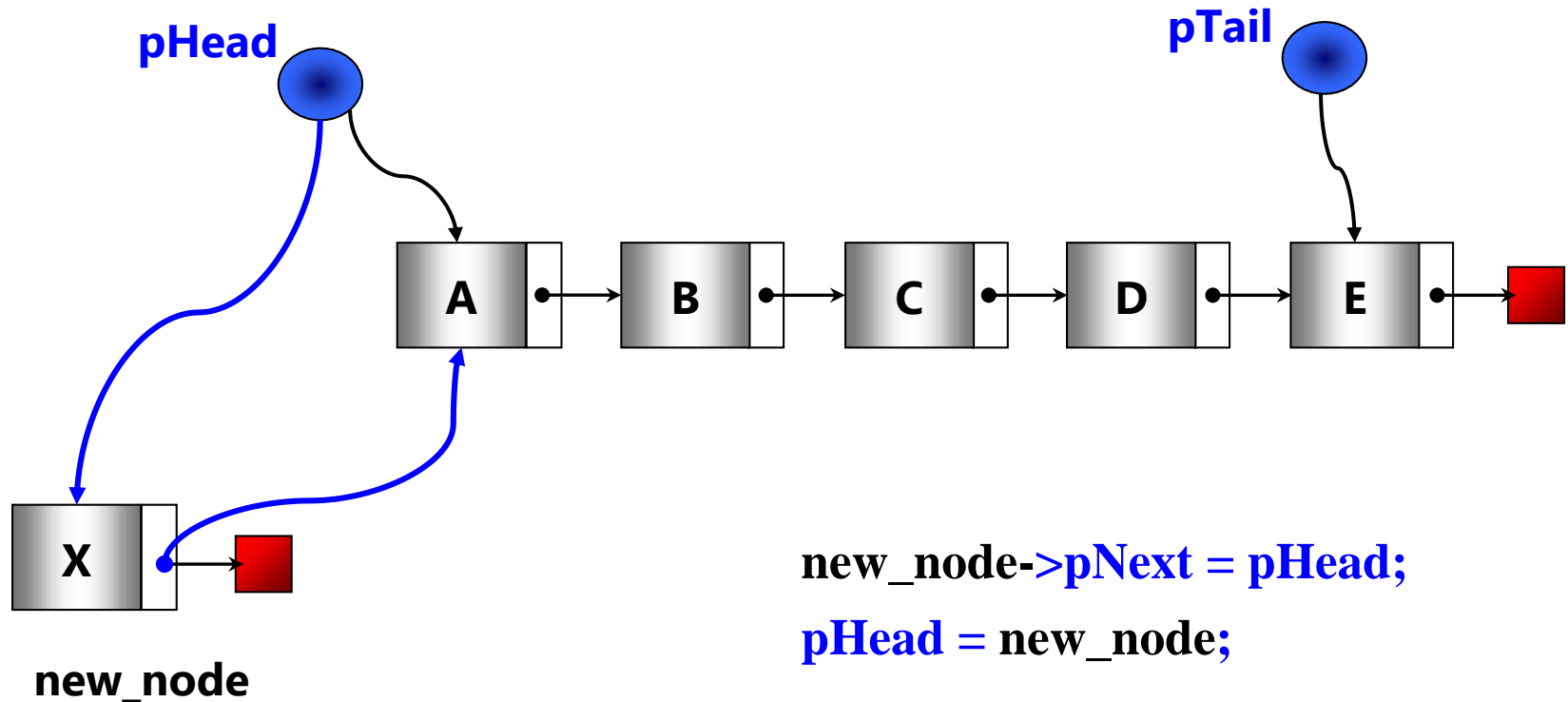
- Thêm một phần tử
  - ▣ Nếu danh sách ban đầu rỗng



# DSLK đơn – Các thao tác cơ sở

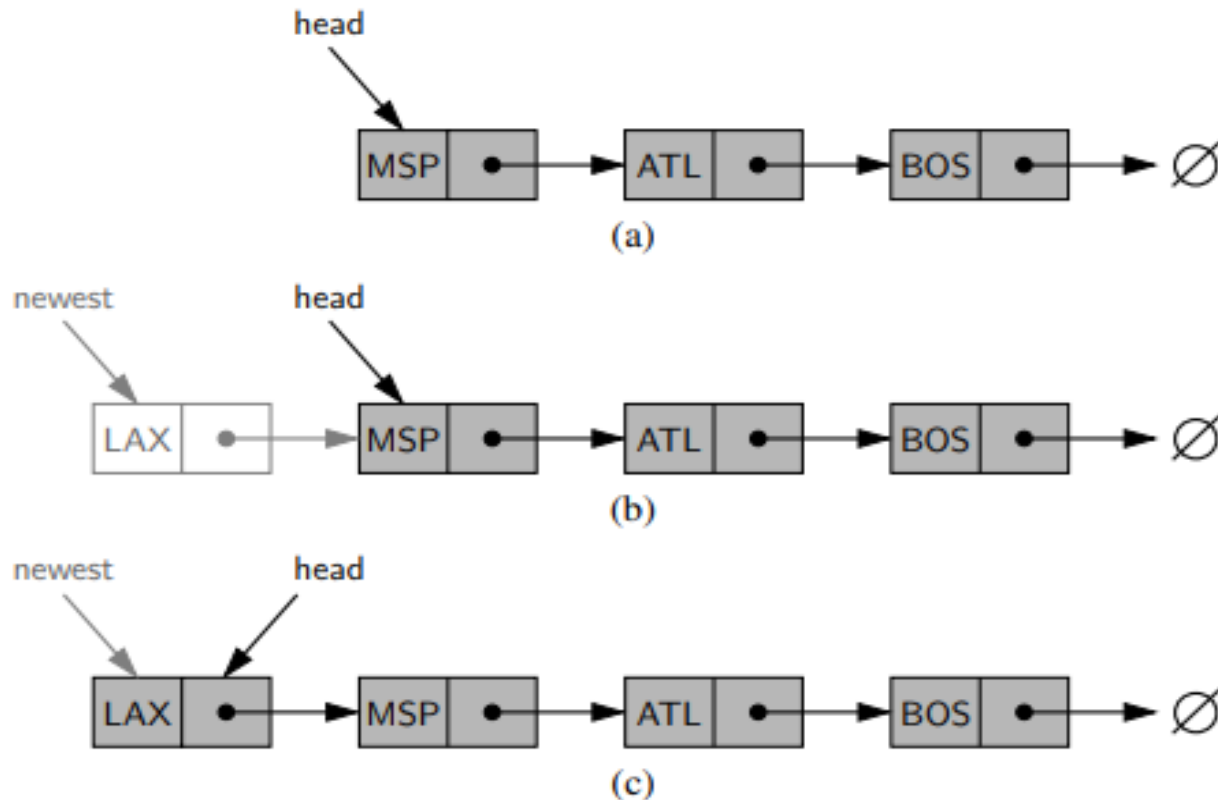
30

- Thêm một phần tử
  - ▣ Gắn node vào đầu danh sách



# DSLK đơn – Các thao tác cơ sở

31

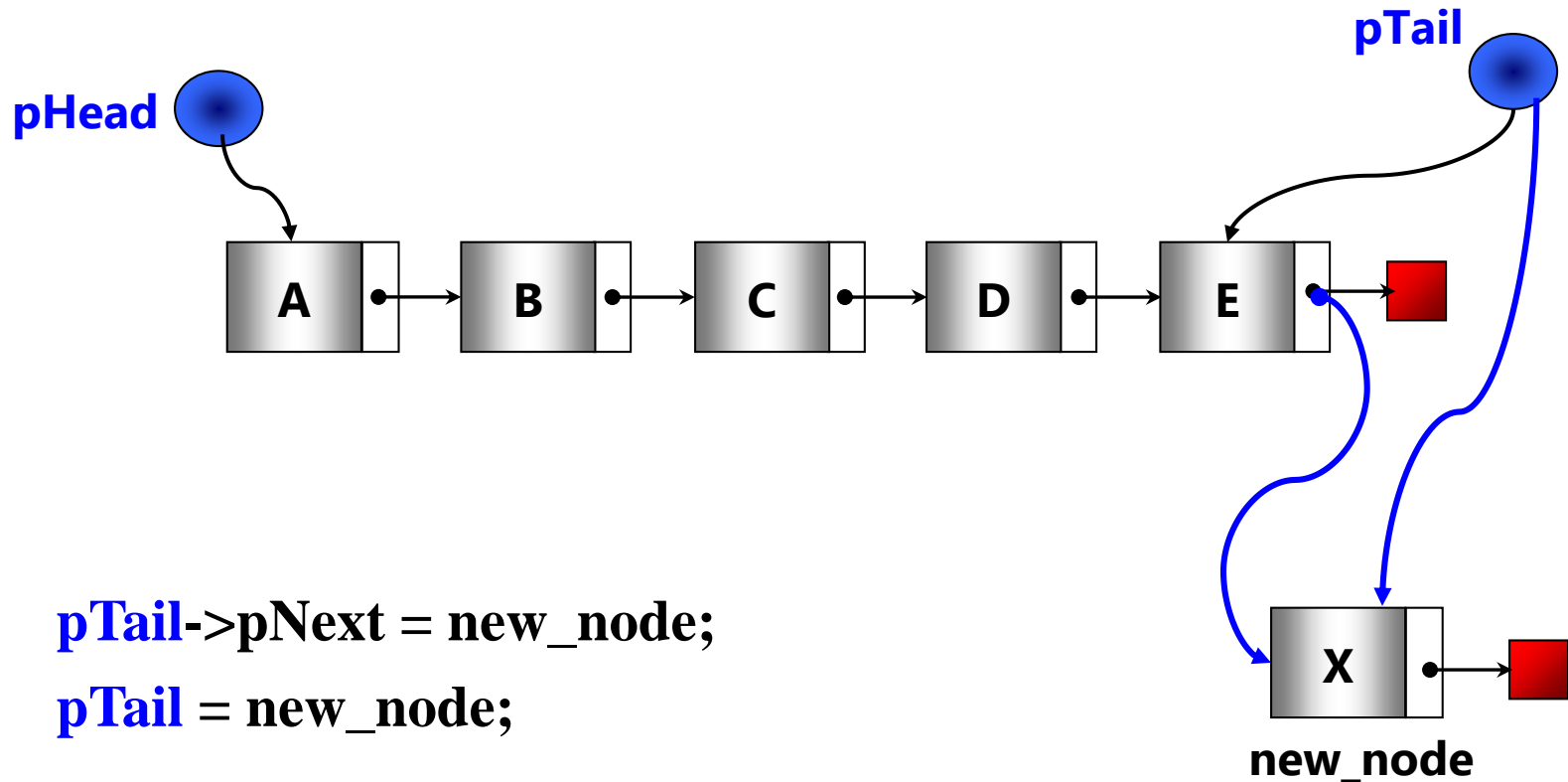


**Figure 7.4:** Insertion of an element at the head of a singly linked list: (a) before the insertion; (b) after creation of a new node; (c) after reassignment of the head reference.

# DSLK đơn – Các thao tác cơ sở

32

- Thêm một phần tử
  - ▣ Gắn node vào cuối danh sách:



**pTail**->pNext = new\_node;

**pTail** = new\_node;



# DSLK đơn – Các thao tác cơ sở

33

Cài đặt: Gắn nút vào đầu DS và cuối danh sách

Chú ý

```
30     def them(self,gia_tri):
31         nut =Nut(gia_tri)
32         if self.dau==None:#thêm đầu
33             self.dau =nut
34             self.duoi =nut
35         else:#thêm cuối
36             self.duoi.nut_ke_tiep=nut
37             self.duoi =nut
38         #
39     #def
```

# DSLK đơn – Các thao tác cơ sở

34

**Thuật toán: Thêm một thành phần dữ liệu vào đầu DS**

*// input: danh sách l*

*// output: danh sách l với phần tử chứa X ở đầu DS*

- Nhập dữ liệu cho X (???)
- Tạo nút mới chứa dữ liệu X (???)
- Nếu tạo được:
  - ▣ Gắn nút mới vào đầu và cuối danh sách (???)

# DSLK đơn – Các thao tác cơ sở

35

Ví dụ:

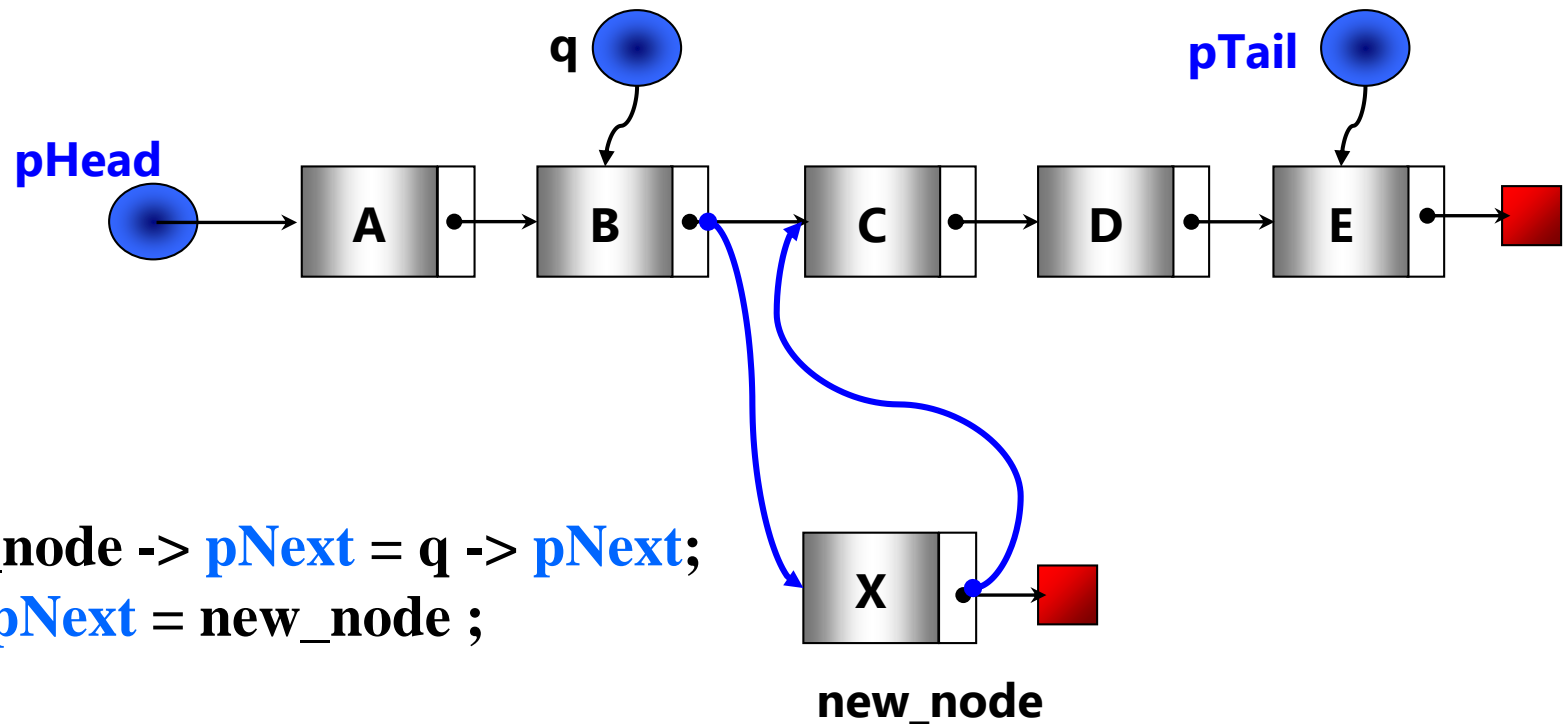
Chú ý

```
1  from DSLK import *
2  def main():
3      ds =DSLienKet()
4      ds.in_ds()
5      # a. Thêm
6      print('1. Thêm-----')
7      so =12 # có thể thay bằng so =int(input("Nhập Số cần thêm"))
8      print(f'Them {so}')
9      ds.them(so)
10     ds.in_ds()
11
12     so =10
13     print(f'Them {so}')
14     ds.them(so)
15     ds.in_ds()
```

# DSLK đơn – Các thao tác cơ sở

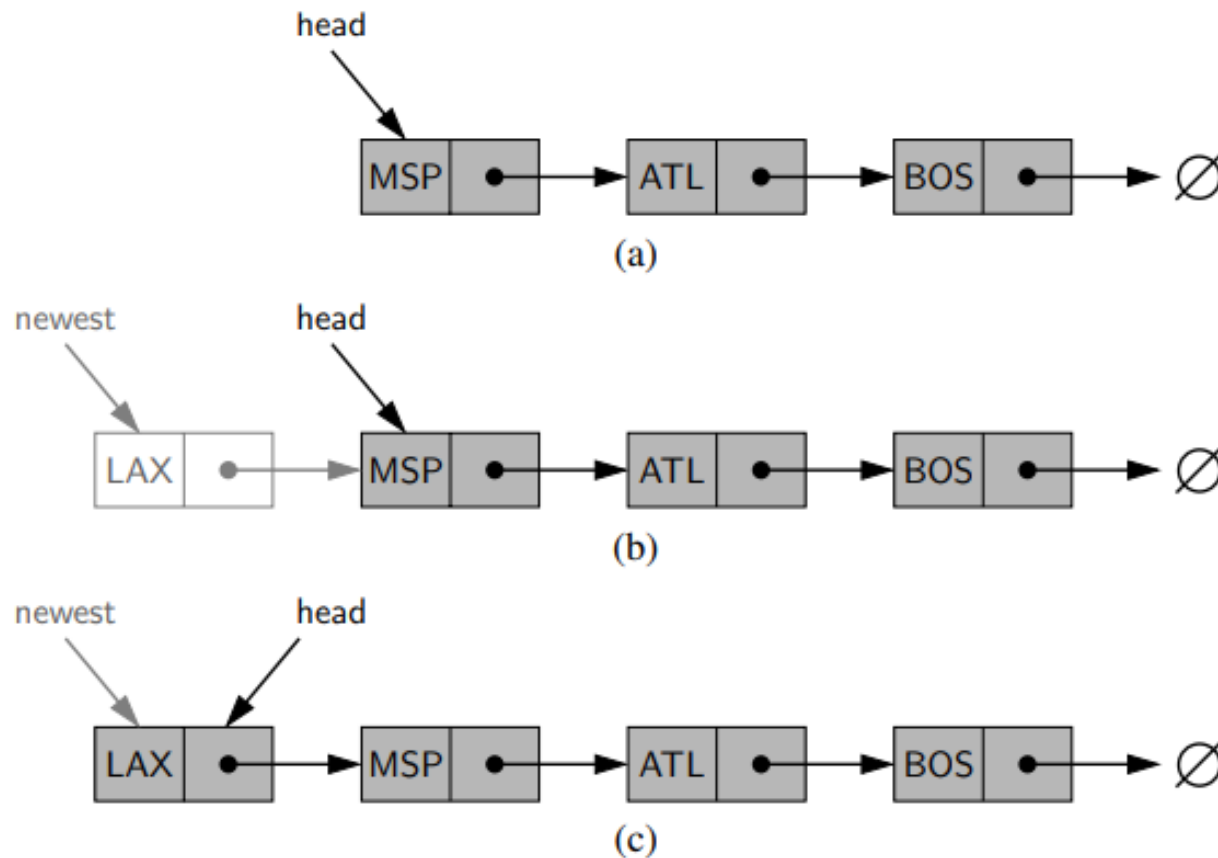
36

- Thêm một phần tử
  - ▣ Chèn một phần tử vào sau nút q



# DSLK đơn – Các thao tác cơ sở

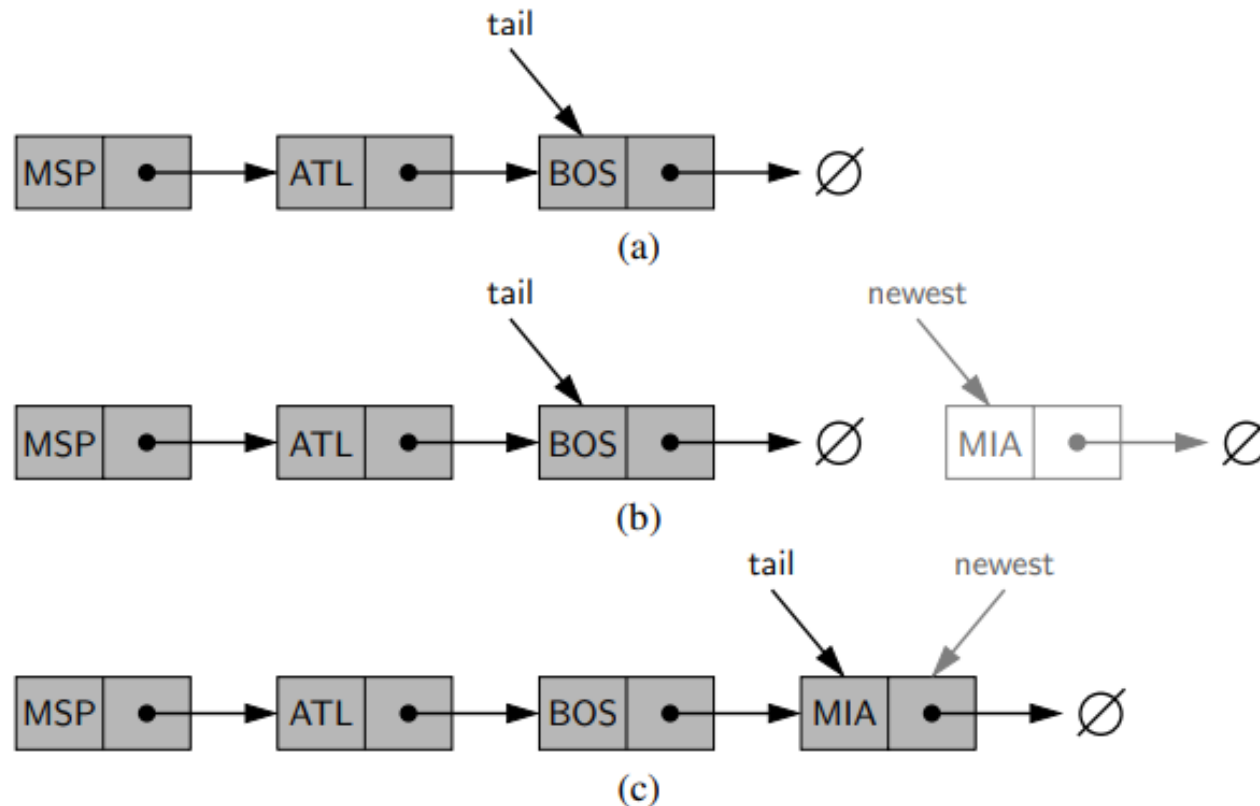
37



**Figure 7.4:** Insertion of an element at the head of a singly linked list: (a) before the insertion; (b) after creation of a new node; (c) after reassignment of the head reference.

# DSLK đơn – Các thao tác cơ sở

38



**Figure 7.5:** Insertion at the tail of a singly linked list: (a) before the insertion; (b) after creation of a new node; (c) after reassignment of the tail reference. Note that we must set the next link of the tail in (b) before we assign the tail variable to point to the new node in (c).

# DSLK đơn – Các thao tác cơ sở

39

**Thuật toán:** Thêm một thành phần dữ liệu vào sau q

*// input: danh sách thành phần dữ liệu X*

*// output: danh sách với phần tử chứa X ở cuối DS*

- Nhập dữ liệu cho nút q (???)
- Tìm nút q (???)
- Nếu tồn tại q trong ds thì:
  - ▣ Nhập dữ liệu cho X (???)
  - ▣ Tạo nút mới chứa dữ liệu X (???)
  - ▣ Nếu tạo được:
    - Gắn nút mới vào sau nút q (???)
- Ngược lại thì báo lỗi

# DSLK đơn – Các thao tác cơ sở

40

**Cài đặt:** Chèn một phần tử vào sau nút q

**Chú ý**

```
40 def chen(self,chi_muc,gia_tri):
41     #pass
42     nut =Nut(gia_tri)
43     truoc =None
44     hien_tai =self.dau
45     i=0
46     while i<chi_muc and hien_tai !=None:
47         i+=1
48         truoc = hien_tai
49         hien_tai = hien_tai.nut_ke_tiep
50     #while
51     if truoc == None:
52         #chèn vào đầu danh sách
53         nut.nut_ke_tiep =self.dau
54         self.dau = nut
55         if self.duoi == None:
56             self.duoi = nut
57         ##trước là 1 nút cụ thể
58     else:
```

```
58     else:
59         if hien_tai == None:
60             #Thêm vào cuối danh sách
61             self.duoi.nut_ke_tiep = nut
62             self.duoi = nut
63         else:
64             #Thêm vào giữa danh sách
65             truoc.nut_ke_tiep = nut
66             nut.nut_ke_tiep =hien_tai
67         #if
68         #if
69     #def
```



# DSLK đơn – Các thao tác cơ sở

41

**Cài đặt:** Chèn một phần tử vào sau nút q, lệnh trong hàm `main()`

Chú ý

```
17     print('2. Chèn-----')
18     so = 8
19     vt = 0
20     print(f'Chen {so} vào vị trí {vt}')
21     ds.chen(vt,so)
22     ds.in_ds()
23
24     so = 15
25     vt = 1
26     print(f'Chen {so} vào vị trí {vt}')
27     ds.chen(vt,so)
28     ds.in_ds()
29
30     so = 17
31     vt = 3
32     print(f'Chen {so} vào vị trí {vt}')
33     ds.chen(vt,so)
34     ds.in_ds()
```

# DSLK đơn

42

## □ Các thao tác cơ bản

- Tạo danh sách rỗng
- Thêm một phần tử vào danh sách
- Duyệt danh sách
- Tìm kiếm một giá trị trên danh sách
- Xóa một phần tử ra khỏi danh sách
- Hủy toàn bộ danh sách
- ...

# DSLK đơn – Các thao tác cơ sở

43

## □ **Duyệt danh sách**

- ▣ Là thao tác thường được thực hiện khi có nhu cầu muốn lấy lần lượt từng phần tử trong danh sách để xử lý, chẳng hạn xử lý:
  - **Xuất** các phần tử trong danh sách
  - **Đếm** các phần tử trong danh sách
  - **Tính** tổng các phần tử trong danh sách
  - **Tìm** tất cả các phần tử danh sách thoả điều kiện nào đó
  - **Hủy** toàn bộ danh sách (và giải phóng bộ nhớ)
  - ...


# DSLK đơn – Các thao tác cơ sở

44

## □ Duyệt danh sách

- Bước 1:  $p = \text{pHead}$ ; *//Cho p trở đến phần tử đầu danh sách*
- Bước 2: Trong khi (chưa hết danh sách) thực hiện:
  - B2.1 : Xử lý phần tử  $p$
  - B2.2 :  $p = p \rightarrow \text{pNext}$ ; *// Cho p trở tới phần tử kế*

```
Node *p = l.pHead;  
while ( p!=NULL )  
{  
    // xử lý cụ thể p tùy ứng dụng  
    p = p->pNext;  
}
```

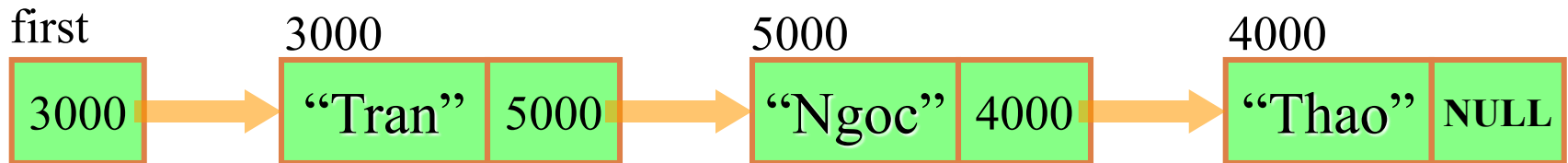


**Chuyển  
thành vòng  
lặp for??**

# DSLK – Minh họa in danh sách

45

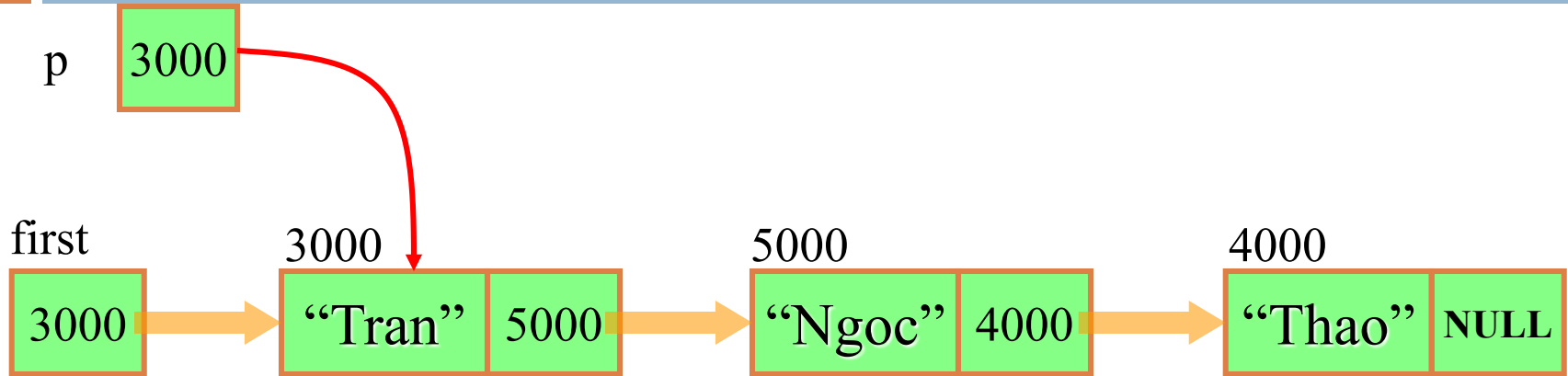
p



```
p = first;  
while (p!=NULL)  
{  
    cout<<p->data;  
    p = p->link;  
}
```

# DSLK – Minh họa in danh sách

46



```
p = first;
```

```
while (p!=NULL)
```

```
{
```

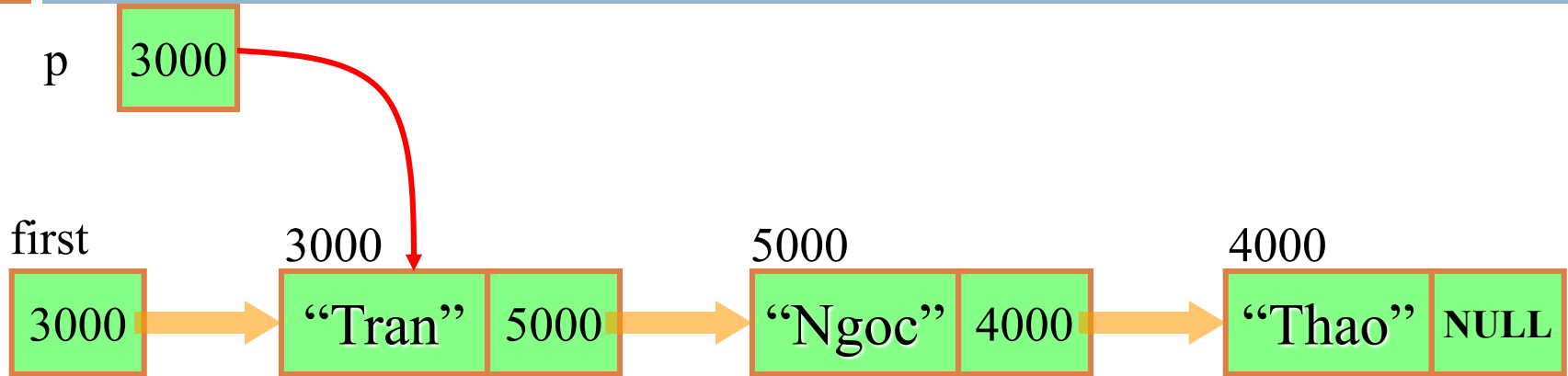
```
    printf("%d\t",p->data);
```

```
    p = p->link;
```

```
}
```

# DSLK – Minh họa in danh sách

47



```
p = first;
```

```
while (p!=NULL)
```

```
{
```

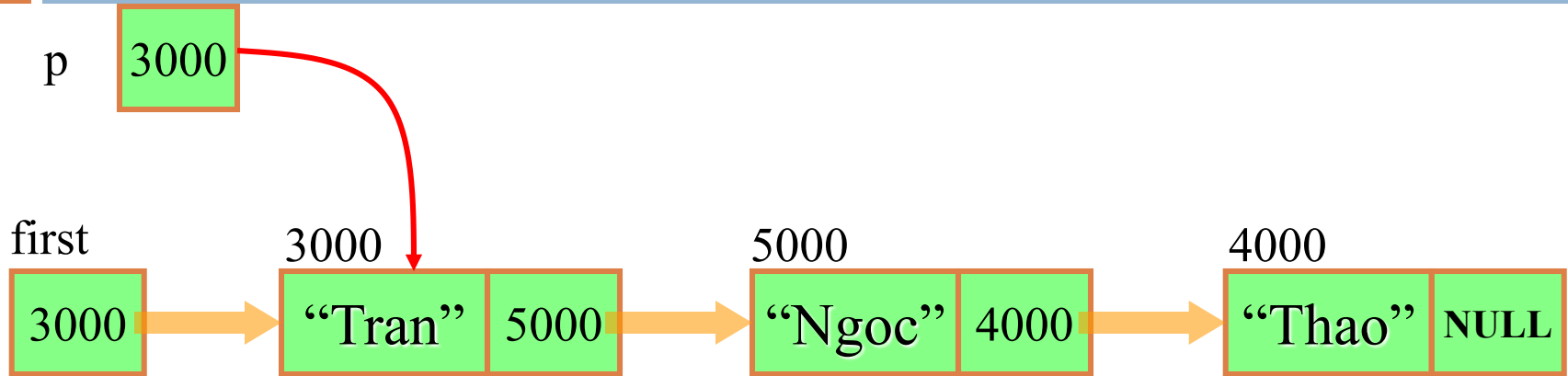
```
    printf("%d\t",p->data);
```

```
    p = p->link;
```

```
}
```

# DSLK – Minh họa in danh sách

48

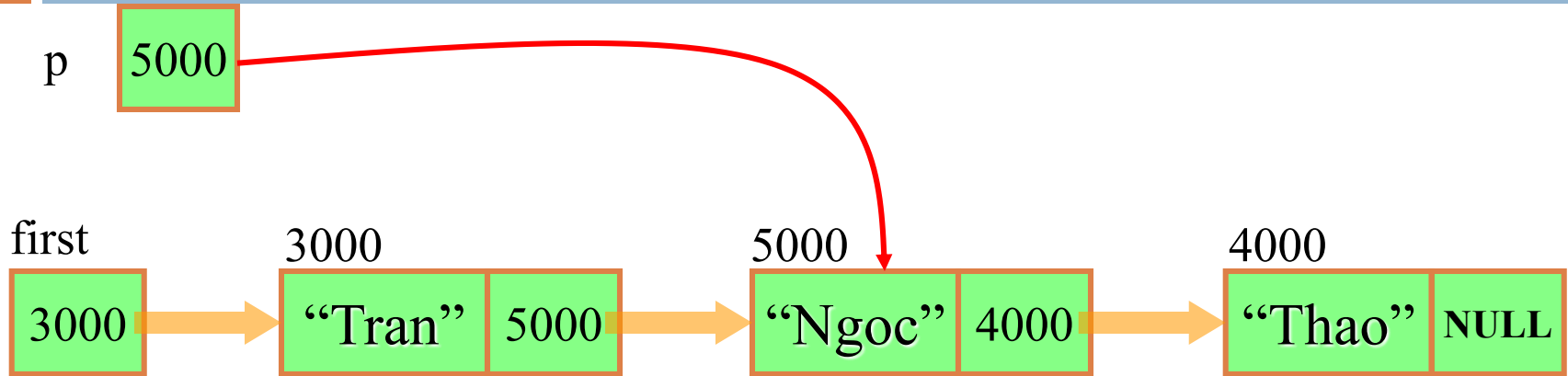


```
p = first;
while (p!=NULL)
{
    printf("%d\t",p->data);
    p = p->link;
}
```



# DSLK – Minh họa in danh sách

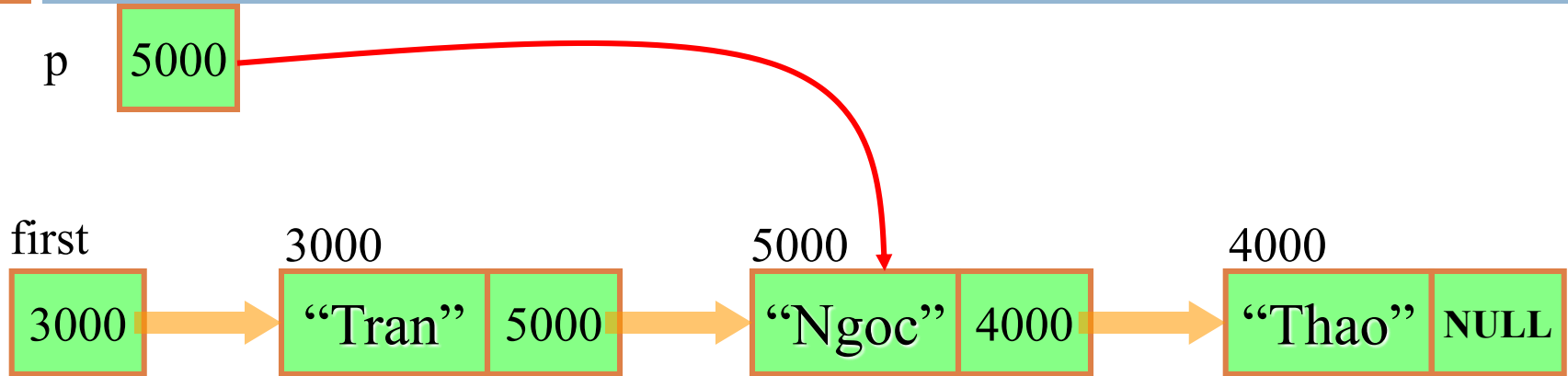
49



```
p = first;  
while (p!=NULL)  
{  
    printf("%d\t",p->data);  
    p = p->link;  
}
```

# DSLK – Minh họa in danh sách

50



```
p = first;
```

```
while (p!=NULL)
```

```
{
```

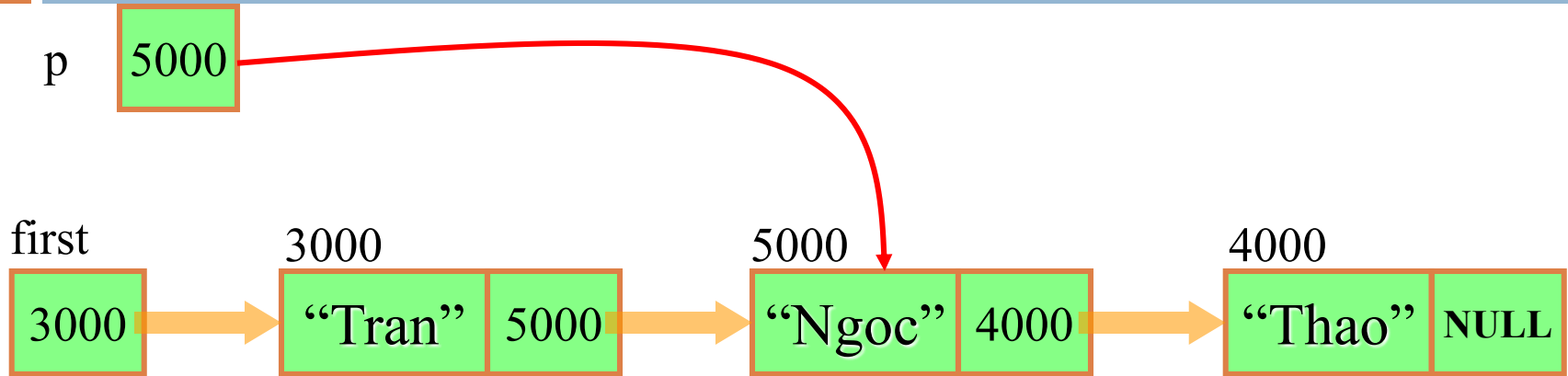
```
    printf("%d\t",p->data);
```

```
    p = p->link;
```

```
}
```

# DSLK – Minh họa in danh sách

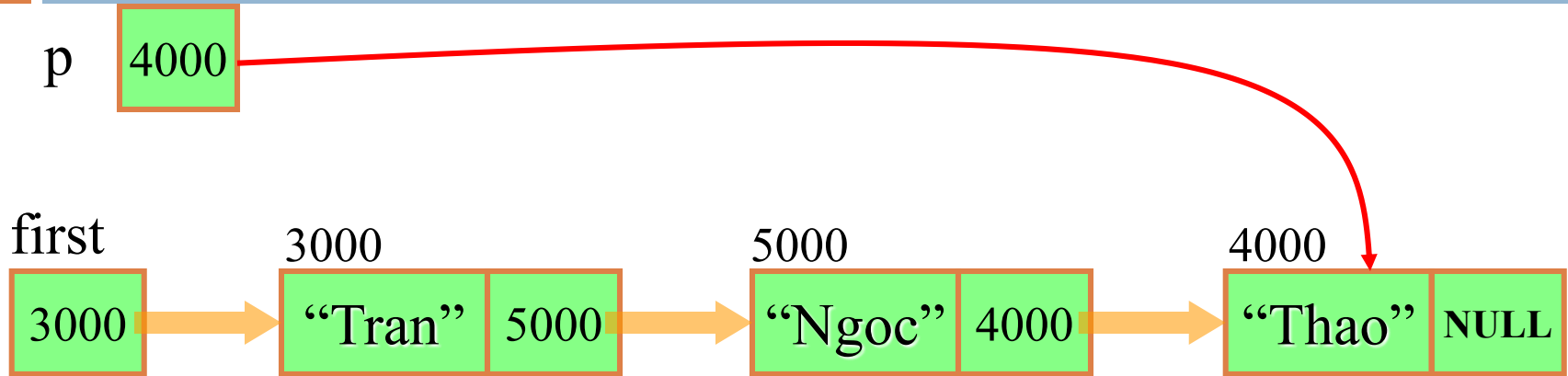
51



```
p = first;  
while (p!=NULL)  
{  
    printf("%d\t",p->data);  
    p = p->link;  
}
```

# DSLK – Minh họa in danh sách

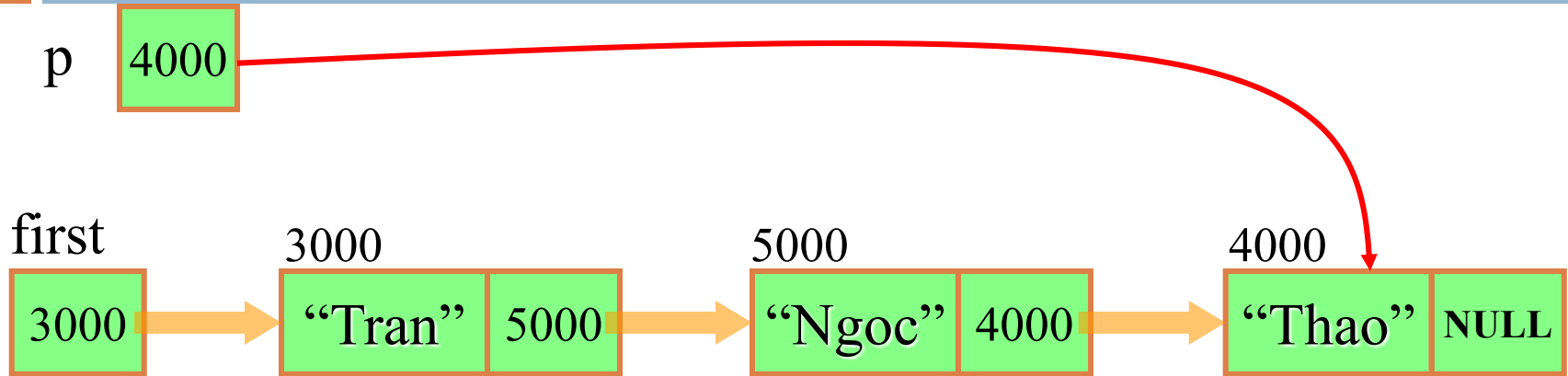
52



```
p = first;  
while (p!=NULL)  
{  
    printf("%d\t",p->data);  
    p = p->link;  
}
```

# DSLK – Minh họa in danh sách

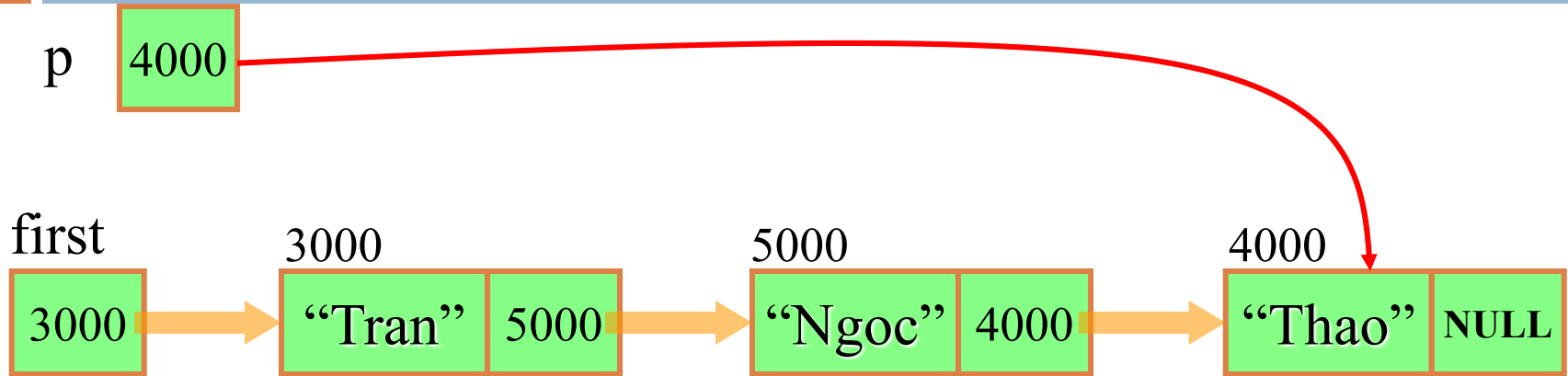
53



```
p = first;  
while (p!=NULL)  
{  
    printf("%d\t",p->data);  
    p = p->link;  
}
```

# DSLK – Minh họa in danh sách

54



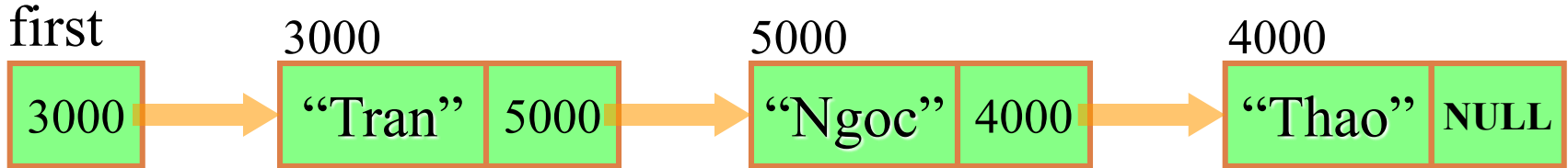
```
p = first;  
while (p!=NULL)  
{  
    printf("%d\t",p->data);  
    p = p->link;  
}
```

# DSLK – Minh họa in danh sách

55

p

NULL



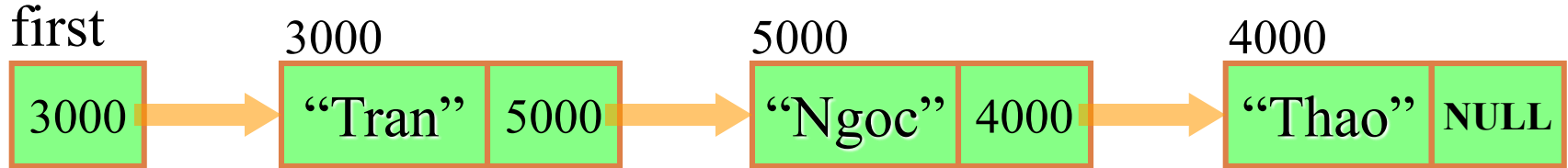
```
p = first;
while (p!=NULL)
{
    printf("%d\t",p->data);
    p = p->link;
}
```

# DSLK – Minh họa in danh sách

56

p

NULL



```
p = first;  
while (p!=NULL)  
{  
    printf("%d\t",p->data);  
    p = p->link;  
}
```

Kết thúc





# DSLK đơn – Các thao tác cơ sở

57

Ví dụ: In các phần tử trong danh sách

**C h ú ý**

```
14 def in_ds(self):
15     stt = 0
16     hien_tai = self.dau
17     kq = 'DS['
18     while hien_tai != None:
19         stt += 1
20         if stt == 1: # DS có 1 phần tử
21             kq += ' ' + str(hien_tai.gia_tri)
22         else:
23             kq += ' -> ' + str(hien_tai.gia_tri)
24         #if
25         hien_tai = hien_tai.nut_ke_tiep
26     #while
27     kq += ']'
28     print(kq)
29 #def
```

**void Output (List l)**

```
{
    Node* p = l.pHead;
    while (p != NULL)
    {
        cout << p->data << " ";
        p = p->pNext;
    }
    cout << endl;
}
```

# DSLK đơn – Các thao tác cơ sở

58

Ví dụ: Lệnh gọi hàm In các phần tử trong danh sách trong hàm  
main()

**C h ú ý**

```
ds.in_ds()
```

# DSLK đơn – Các thao tác cơ sở

59

- Đếm số nút trong danh sách:

```
int CountNodes (List l)
{
    int count = 0;
    Node *p = l.pHead;
    while (p!=NULL)
    {
        count++;
        p = p->pNext;
    }
    return count;
}
```



Gọi hàm???

# DSLK đơn

60

- **Các thao tác cơ bản**
  - Tạo danh sách rỗng
  - Thêm một phần tử vào danh sách
  - Duyệt danh sách
  - Tìm kiếm một giá trị trên danh sách
  - Xóa một phần tử ra khỏi danh sách
  - Hủy toàn bộ danh sách
  - ...

# DSLK đơn – Các thao tác cơ sở

61

- Tìm kiếm một phần tử có khóa x

```
Node* Search (List l, int x)
{
    Node* p = l.pHead;
    while ( p!=NULL ) {
        if ( p->data==x )
            return p;
        p=p->pNext;
    }
    return NULL;
}
```



Gọi hàm???

# DSLK đơn – Các thao tác cơ sở

62

- Tìm kiếm một phần tử có khóa x

Chú ý

```
70     def tim(self,gia_tri):
71         #pass
72         hien_tai = self.dau
73         vi_tri = 0
74         while hien_tai != None and hien_tai.gia_tri != gia_tri:
75             hien_tai = hien_tai.nut_ke_tiep
76             vi_tri += 1
77         #while
78         if hien_tai == None:
79             return None #Không tìm thấy
80         else:
81             return vi_tri
82         #if
83     #def
```

# DSLK đơn – Các thao tác cơ sở

63

- **Lệnh gọi hàm Tìm kiếm một phần tử có khóa x trong main()**

**Chú ý**

```
35     # c. Tìm
36     print('3. Tìm -----')
37     ds.in_ds()
38     so = 99
39     print(f'Tìm {so}')
40     vt = ds.tim(so)
41     print(f'So {so} tại vị trí {vt}')
42
43     ds.in_ds()
44     so = 15
45     print(f'Tìm {so}')
46     vt = ds.tim(so)
47     print(f'So {so} tại vị trí {vt}')
```

# DSLK đơn

64

- **Các thao tác cơ bản**
  - Tạo danh sách rỗng
  - Thêm một phần tử vào danh sách
  - Duyệt danh sách
  - Tìm kiếm một giá trị trên danh sách
  - Xóa một phần tử ra khỏi danh sách
  - Hủy toàn bộ danh sách
  - ...



# DSLK đơn – Các thao tác cơ sở

65

- Xóa một node của danh sách
  - ▣ Xóa node đầu danh sách
  - ▣ Xóa node sau node q trong danh sách
  - ▣ Xóa node có khoá k

# DSLK đơn – Các thao tác cơ sở

66

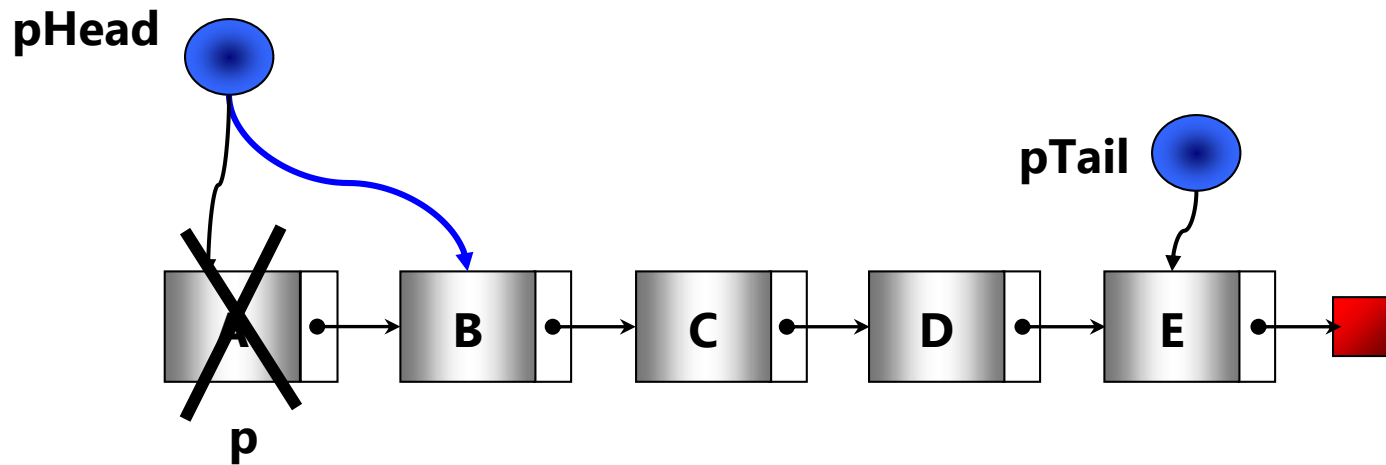
## Thuật toán: Xóa node đầu danh sách

- ❑ Bước 1: Nếu danh sách rỗng thì không xóa được và thoát ct, ngược lại qua Bước 2
- ❑ Bước 2: Gọi **p** là node đầu của danh sách ( $p = \text{pHead}$ )
- ❑ Bước 3: Cho **pHead** trở vào node sau node **p** ( $\text{pHead} = p \rightarrow \text{pNext}$ )
- ❑ Bước 4: Nếu không còn node nào thì **pTail** = NULL
- ❑ Bước 5: Giải phóng vùng nhớ mà **p** trỏ tới

# DSLK đơn – Các thao tác cơ sở

67

- Minh họa: **Xóa node đầu** danh sách



**pHead = p->pNext;**

**delete p;**

# DSLK đơn – Các thao tác cơ sở

68

Cài đặt: **Xóa node đầu** danh sách

// xóa được: hàm trả về 1

// xóa không được: hàm trả về 0

```
int removeHead (List &l){
```

```
    if (l.pHead == NULL)
```

```
        return 0;
```

```
    Node* p=l.pHead;
```

```
    l.pHead = p->pNext;
```

```
    if (l.pHead == NULL) l.pTail=NULL; //Nếu danh sách rỗng
```

```
    delete p;
```

```
    return 1;
```

```
}
```

# DSLK đơn – Các thao tác cơ sở

69

- Xóa một node của danh sách
  - ▣ Xóa node đầu danh sách
  - ▣ Xóa node sau node q trong danh sách
  - ▣ Xóa node có khoá k

# DSLK đơn – Các thao tác cơ sở

70

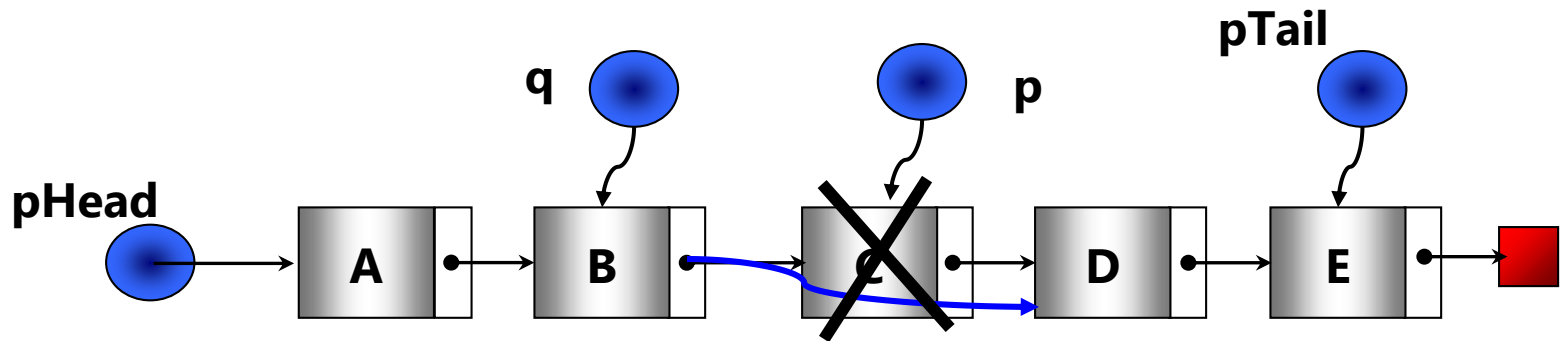
## Thuật toán: Xóa node sau node $q$ trong danh sách:

- ▣ Điều kiện để có thể xóa được node sau  $q$  là:
  - $q$  phải khác NULL ( $q \neq \text{NULL}$ )
  - Node sau  $q$  phải khác NULL ( $q \rightarrow \text{pNext} \neq \text{NULL}$ )
- ▣ Thuật toán:
  - Bước 1: Gọi  $p$  là node sau  $q$
  - Bước 2: Cho  $q$  trở vào node đứng sau  $p$
  - Bước 3: Nếu  $p$  là phần tử cuối thì  $p\text{Tail}$  là  $q$
  - Bước 4: Giải phóng vùng nhớ mà  $p$  trở tới

# DSLK đơn – Các thao tác cơ sở

71

Minh họa: **Xóa node sau node  $q$**  trong danh sách



$q \rightarrow \text{pNext} = p \rightarrow \text{pNext};$

**delete** p;

# DSLK đơn – Các thao tác cơ sở

72

Cài đặt: **Xóa node sau node *q* trong danh sách**

```
// xóa được: hàm trả về 1
```

```
// xóa không được: hàm trả về 0
```

```
int removeAfter (List &l, Node* q ){  
    if (q !=NULL && q->pNext !=NULL) {  
        Node* p = q->pNext;  
        q->pNext = p->pNext;  
        if (p==l.pTail) l.pTail = q;  
        delete p;  
        return 1;  
    }  
    else return 0;  
}
```



# DSLK đơn – Các thao tác cơ sở

73

- Xóa một node của danh sách
  - ▣ Xóa node đầu của danh sách
  - ▣ Xóa node sau node q trong danh sách
  - ▣ Xóa node có khoá k

# DSLK đơn – Các thao tác cơ sở

74

## Thuật toán: Xóa 1 node có khoá k

- Bước 1:
  - ▣ Tìm node có khóa k (gọi là p) và node đứng trước nó (gọi là q)
- Bước 2:
  - ▣ Nếu  $(p \neq \text{NULL})$  thì // tìm thấy k
    - Hủy p ra khỏi danh sách: tương tự hủy phần tử sau q
  - ▣ Ngược lại
    - Báo không có k

# DSLK đơn – Các thao tác cơ sở

75

pHead

pTail

- Cài đặt:  
Xóa 1  
node có  
khóa k

```
int removeNode (List &l, int k)
```

```
{
```

```
    Node *p = l.pHead;
```

```
    Node *q = NULL;
```

```
    while (p != NULL)
```

```
    {
```

```
        if (p->data == k) break;
```

```
        q = p;
```

```
        p = p->pNext;
```

```
    }
```

```
    if (p == NULL) { cout<<"Không tìm thấy k"; return 0; }
```

```
    else if (q == NULL)
```

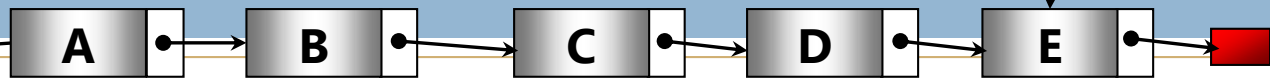
```
        // thực hiện xóa phần tử đầu ds là p
```

```
    else
```

```
        // thực hiện xóa phần tử p sau q
```

```
}
```

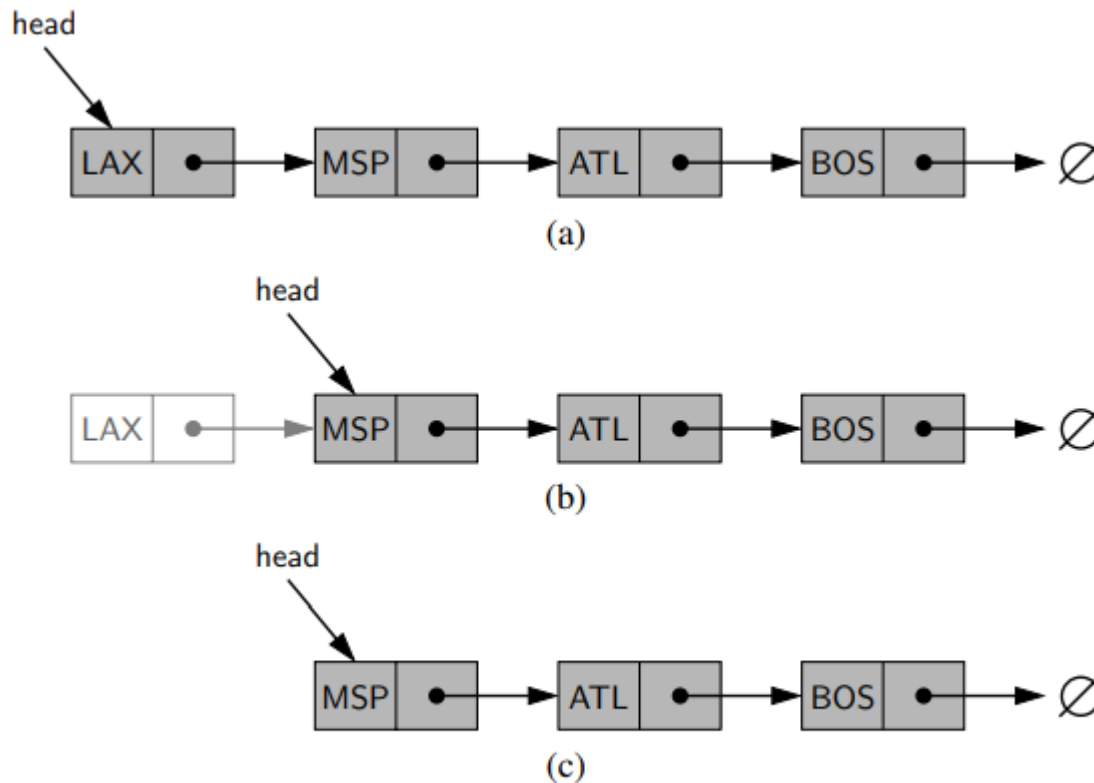
Tìm phần tử **p** có khóa k và  
phần tử **q** đứng trước nó



# DSLK đơn

76

## □ Hàm xóa 1 phần tử trong DSLK

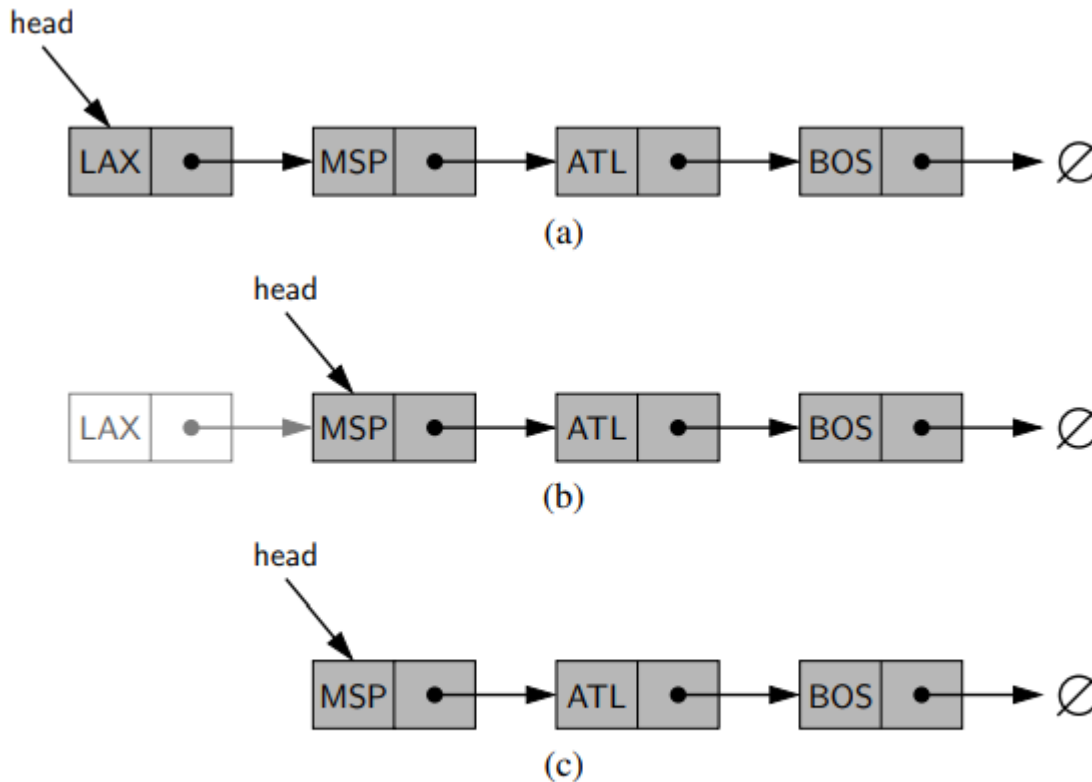


**Figure 7.6:** Removal of an element at the head of a singly linked list: (a) before the removal; (b) after “linking out” the old head; (c) final configuration.

# DSLK đơn

77

## □ Hàm xóa 1 phần tử trong DSLK



**Figure 7.6:** Removal of an element at the head of a singly linked list: (a) before the removal; (b) after “linking out” the old head; (c) final configuration.

# Chú ý

```
84 def xoa(self,gia_tri):
85     #pass
86     hien_tai = self.dau
87     truoc = None
88     while hien_tai != None and hien_tai.gia_tri !=gia_tri:
89         truoc = hien_tai
90         hien_tai =hien_tai.nut_ke_tiep
91     #while
92     if hien_tai != None:
93         #Tìm thấy
94         if hien_tai == self.dau and hien_tai ==self.dau:
95             #xoá phần tử duy nhất
96             self.dau = self.duoi = None
97         elif hien_tai == self.dau:
98             #Xoá phần tử đầu tiên và không duy nhất
99             self.dau = self.dau.nut_ke_tiep
100        elif hien_tai == self.duoi:
101            #xoá phần tử cuối và không duy nhất
102            truoc.nut_ke_tiep = None
103            self.duoi = truoc
104        else:
105            #xoá ở giữa
106            truoc.nut_ke_tiep = hien_tai.nut_ke_tiep
107    #if
108    del hien_tai
```

- **Gọi Hàm xoá 1 phần tử trong DSLK**

```
50     # 4. Xoá
51     print('4. Xoá -----')
52     so =19
53     print(f'Xoa {so}')
54     ds.xoa(so)
55     ds.in_ds()
56
57     so =15
58     print(f'Xoa {so}')
59     ds.xoa(so)
60     ds.in_ds()
```

# DSLK đơn Chú ý

80

## □ Hàm cập nhập danh sách

```
110     def cap_nhat(self,vi_tri,gia_tri):
111         #pass
112         hien_tai =self.dau
113         i = 0
114         while i < vi_tri and hien_tai != None:
115             i += 1
116             hien_tai =hien_tai.nut_ke_tiep
117         #while
118         if hien_tai !=None:
119             hien_tai.gia_tri = gia_tri
120         #if
121     #def
```



## □ Gọi Hàm cập nhập danh sách

```
62     print('5. Cập nhật -----')
63     vt = 6
64     gia_trí = 23
65     print(f'Cập nhật vị trí {vt} với giá trị {gia_trí}')
66     ds.cap_nhat(vt,gia_trí)
67     ds.in_ds()
68
69     vt = 2
70     gia_trí = 9
71     print(f'Cập nhật vị trí {vt} với giá trị {gia_trí}')
72     ds.cap_nhat(vt,gia_trí)
73     ds.in_ds()
```

# DSLK đơn – Các thao tác cơ sở

82

## □ Hủy toàn bộ danh sách

- ▣ Để hủy toàn bộ danh sách, thao tác xử lý bao gồm hành động giải phóng một phần tử, do vậy phải cập nhật các liên kết liên quan:
- ▣ Thuật toán:
  - Bước 1: Trong khi (chưa hết danh sách) thực hiện:
    - B1.1:
      - $p = \text{pHead};$
      - $\text{pHead} = \text{pHead} \rightarrow \text{pNext};$  *// Cho p trở tới phần tử kế*
    - B1.2:
      - Hủy p;
  - Bước 2:
    - $\text{pTail} = \text{NULL};$  *//Bảo đảm tính nhất quán khi xâu rỗng*

# DSLK đơn – Các thao tác cơ sở

83

- Cài đặt: **Hủy** toàn bộ danh sách

```
void RemoveList (List &l)
{
    Node *p;
    while (l.pHead!=NULL)
    {
        p = l.pHead;
        l.pHead = p->pNext;
        delete p;
    }
    l.pTail = NULL;
}
```



Gọi hàm???

□ Hàm xoá hết danh sách

```
122     def xoa_het(self):
123         #pass
124         hien_tai =self.dau
125         self.dau = self.duoi = None
126         while hien_tai != None:
127             tam = hien_tai
128             hien_tai = hien_tai.nut_ke_tiep
129             del tam
130         #while
131     #def
```

- **Gọi Hàm xoá hết danh sách**

```
74     #6. Xoá hết
75     print('6.Xoá hết -----')
76     print('Xoá hết')
77     ds.xoa_het()
78     ds.in_ds()
79     #def
```

```
79     #def
80     if __name__ == '__main__':
81         main()
82     #if
```

# Sắp xếp trên DSLK đơn

87

- Các cách tiếp cận:
  - ▣ **Phương án 1:** Hoán vị nội dung các phần tử trong danh sách (thao tác trên vùng **data**)
    - Sử dụng thêm vùng nhớ trung gian → thích hợp cho DSLK với thành phần data có kích thước nhỏ
  - ▣ **Phương án 2:** Thay đổi các mối liên kết (thao tác trên vùng **pNext**)
    - Phức tạp hơn

# Sắp xếp trên DSLK đơn: Hoán vị data

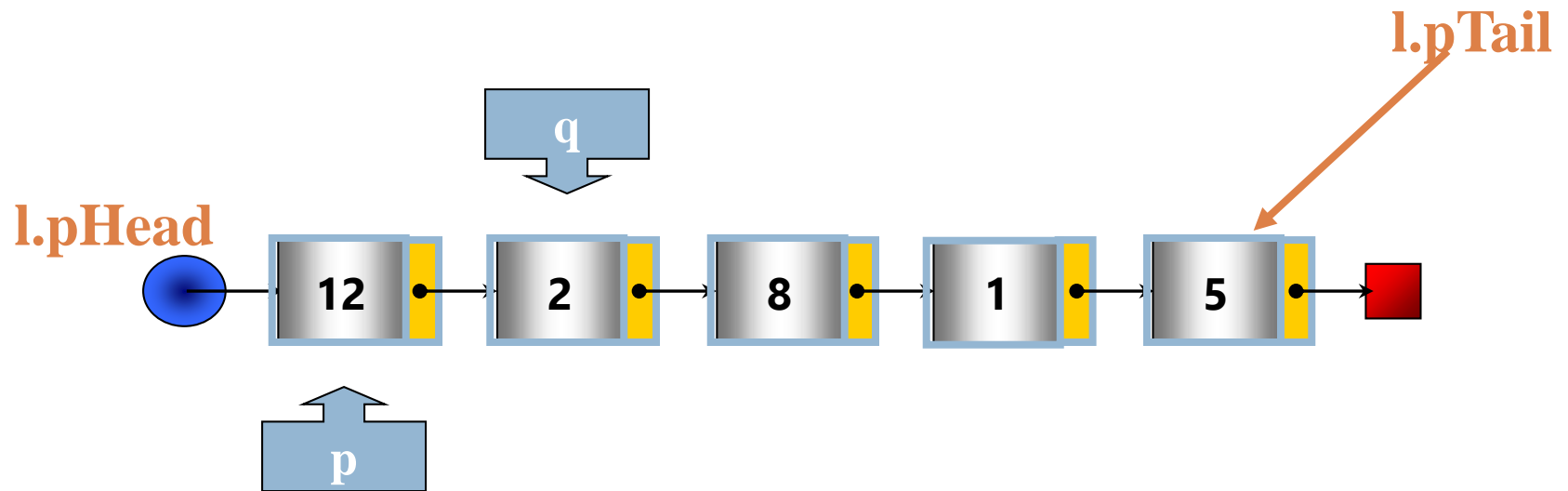
88

- Cài đặt bằng pp đổi chỗ trực tiếp (*Interchange Sort*)

```
void InterChangeSort (List &l)
{
    for (Node* p=l.pHead; p!=l.pTail; p=p->pNext)
        for (Node* q=p->pNext; q!=NULL; q=q->pNext)
            if (p->data > q->data)
                Swap (p->data, q->data);
}
```

# Sắp xếp trên DSLK đơn: Hoán vị data

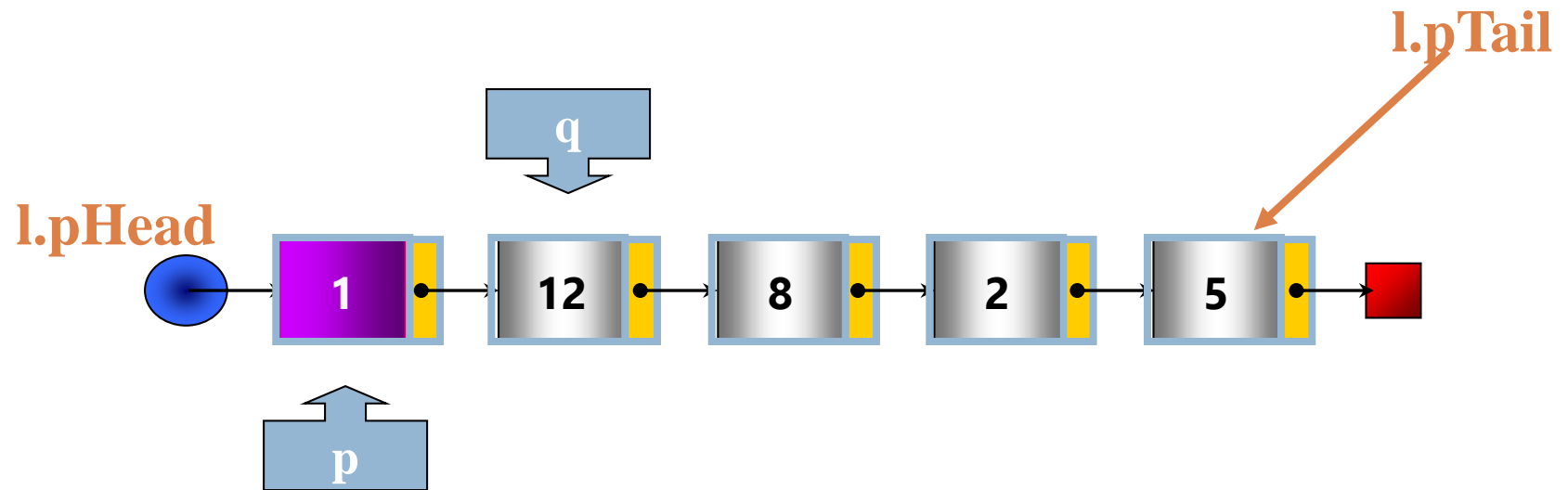
89





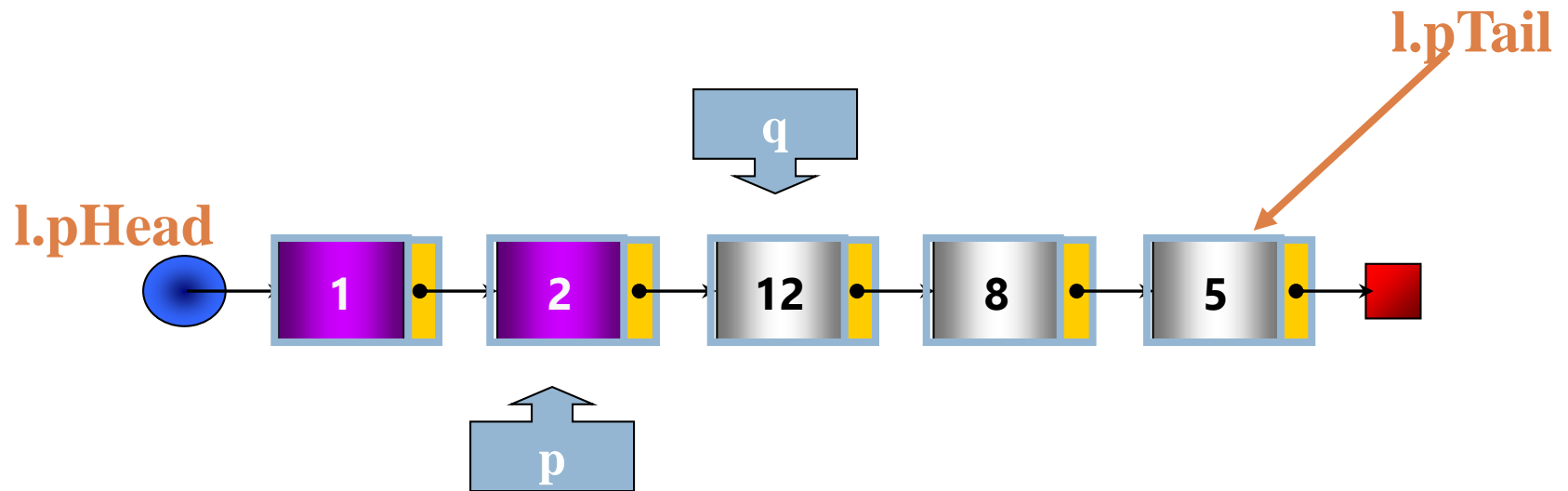
# Sắp xếp trên DSLK đơn: Hoán vị data

90



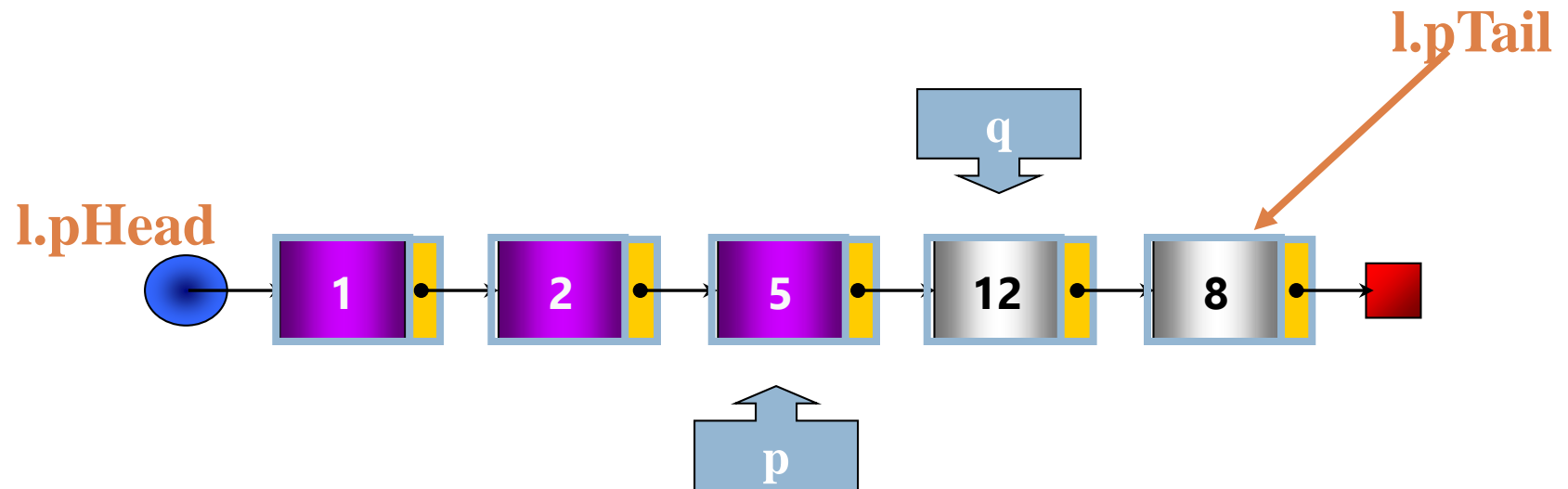
# Sắp xếp trên DSLK đơn: Hoán vị data

91



# Sắp xếp trên DSLK đơn: Hoán vị data

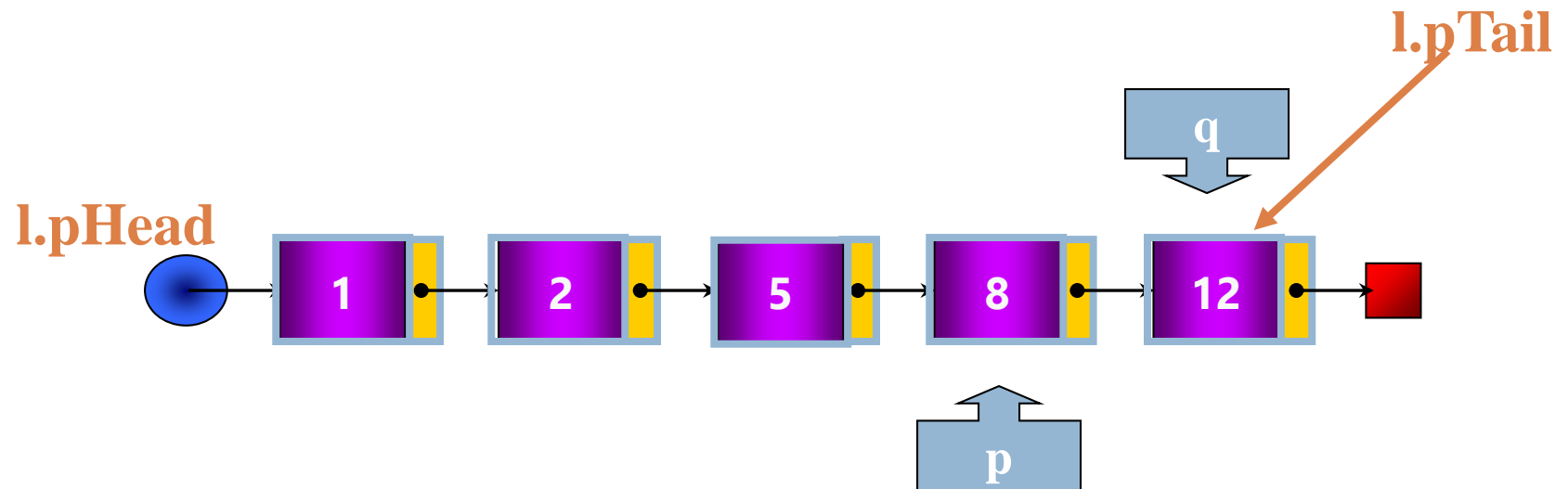
92



# Sắp xếp trên DSLK đơn: Hoán vị data

93

Dừng



# Sắp xếp bằng pp chọn trực tiếp

## (*Selection sort*)

94

```
void ListSelectionSort (List &l)
{
    for ( Node* p = l.pHead ; p != l.pTail ; p = p->pNext )
    {
        Node* min = p;
        for ( Node* q = p-> pNext; q != NULL ; q = q-> pNext )
            if ( min->data > q->data ) min = q ;
        Swap(min->data, p->data);
    }
}
```

# Sắp xếp trên DSLK đơn: Thay đổi liên kết

95

- Một trong những cách thay đổi liên kết đơn giản nhất là tạo một danh sách mới là danh sách có thứ tự từ danh sách cũ (GT.101)
  - ▣ Bước 1: Khởi tạo danh sách mới Result là rỗng;
  - ▣ Bước 2: Tìm phần tử nhỏ nhất min trong danh sách cũ l;
  - ▣ Bước 3: Tách min khỏi danh sách l;
  - ▣ Bước 4: Chèn min vào cuối danh sách Result;
  - ▣ Bước 5: Lặp lại bước 2 khi chưa hết danh sách cũ l;

```

void SortList( List &l ){
    List lResult;
    Init( lResult );
    Node *p,*q, *min, *minprev;
    while( l.pHead != NULL ){
        min = l.pHead;
        q = min->pNext;
        p = l.pHead;
        minprev = NULL;
        while ( q != NULL )
        {
            if (min->data > q->data ) {
                min = q;
                minprev = p;
            }
            p = q;
            q = q->pNext;
        }
    }
}

```

```

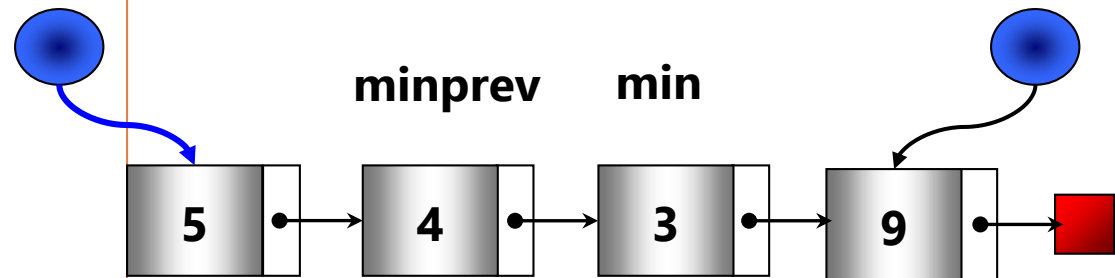
if ( minprev != NULL )
    minprev->pNext = min->pNext;
else
    l.pHead = min->pNext;

min->pNext = NULL;

addTail( lResult, min );
}

l = lResult;
}

```



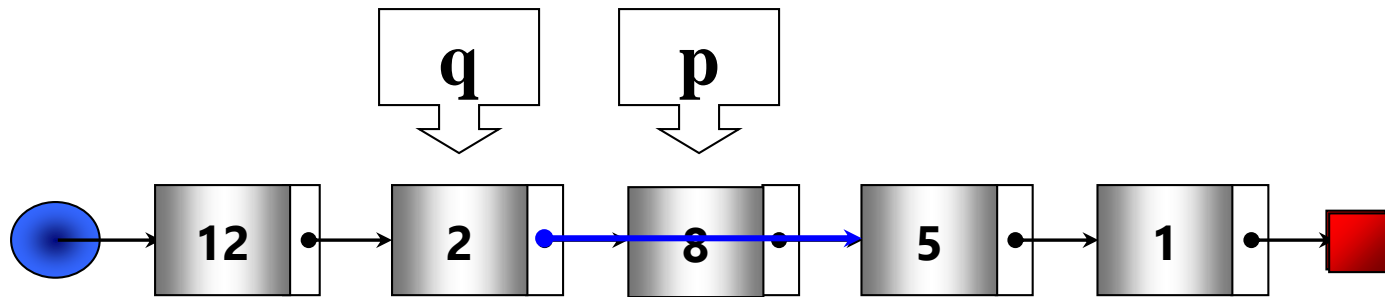
# Sắp xếp Thay đổi các mối liên kết

97

- Thay vì hoán đổi giá trị, ta sẽ tìm cách thay đổi trình tự móc nối của các phần tử sao cho tạo lập nên được thứ tự mong muốn  $\Rightarrow$  chỉ thao tác trên các móc nối (link).
  - Kích thước của trường link:
    - ▣ Không phụ thuộc vào bản chất dữ liệu lưu trong xâu
    - ▣ Bằng kích thước 1 con trỏ (2 hoặc 4 byte trong môi trường 16 bit, 4 hoặc 8 byte trong môi trường 32 bit...)
  - Thao tác trên các móc nối thường phức tạp hơn thao tác trực tiếp trên dữ liệu.
- $\Rightarrow$  Cần cân nhắc khi chọn cách tiếp cận: Nếu dữ liệu không quá lớn thì nên chọn phương án 1 hoặc một thuật toán hiệu quả nào đó.



# Phương pháp lấy **Node** ra khỏi danh sách giữ nguyên địa chỉ của **Node**



1 . **q->link = p->link ;** // *p->link* chứa địa chỉ sau *p*

2 . **q->link = NULL ;** // *p* không liên kết phần tử *Node*

# Quick Sort : Thuật toán

99

*//input: chuỗi (first, last)*

*//output: chuỗi đã được sắp tăng dần*

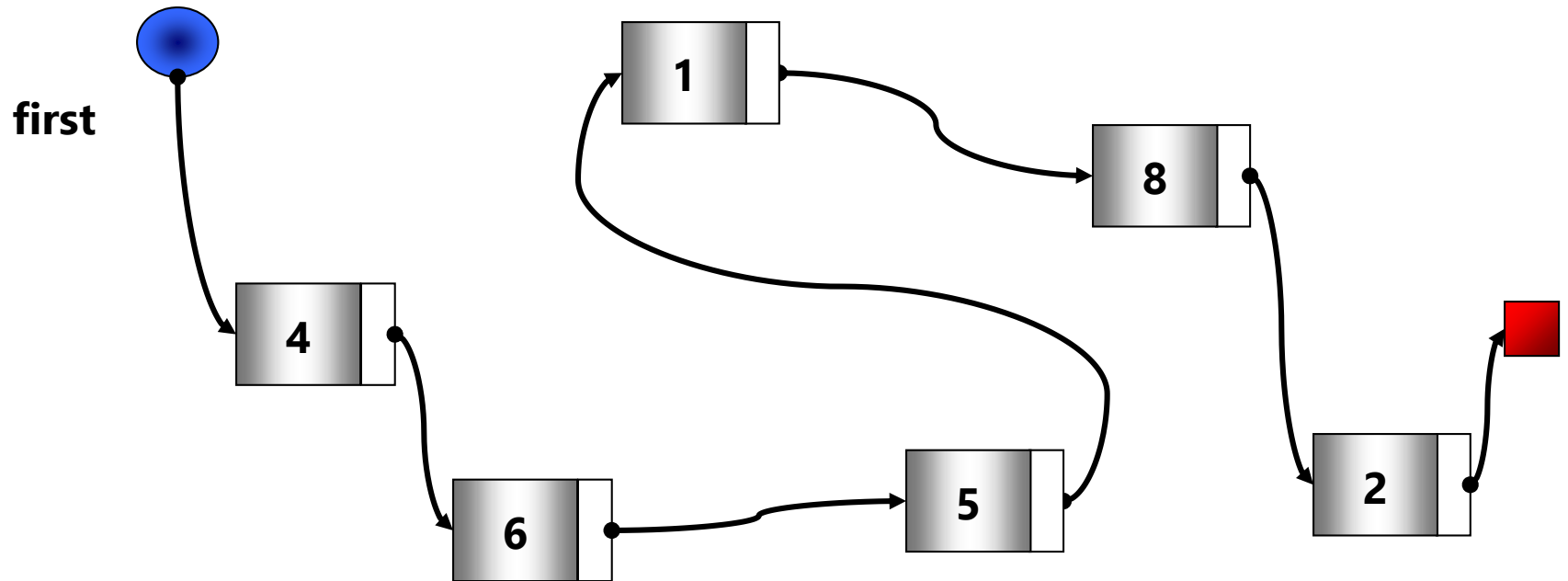
- Bước 1: Nếu chuỗi có ít hơn 2 phần tử

*Dừng; //chuỗi đã có thứ tự*

- Bước 2: Chọn X là phần tử đầu chuỗi L làm ngưỡng. Trích X ra khỏi L.
- Bước 3: Tách chuỗi L ra làm 2 chuỗi  $L_1$  (gồm các phần tử nhỏ hơn hay bằng X) và  $L_2$  (gồm các phần tử lớn hơn X).
- Bước 4: Sắp xếp **Quick Sort** ( $L_1$ ).
- Bước 5: Sắp xếp **Quick Sort** ( $L_2$ ).
- Bước 6: Nối  $L_1$ , X, và  $L_2$  lại theo trình tự ta có chuỗi L đã được sắp xếp.

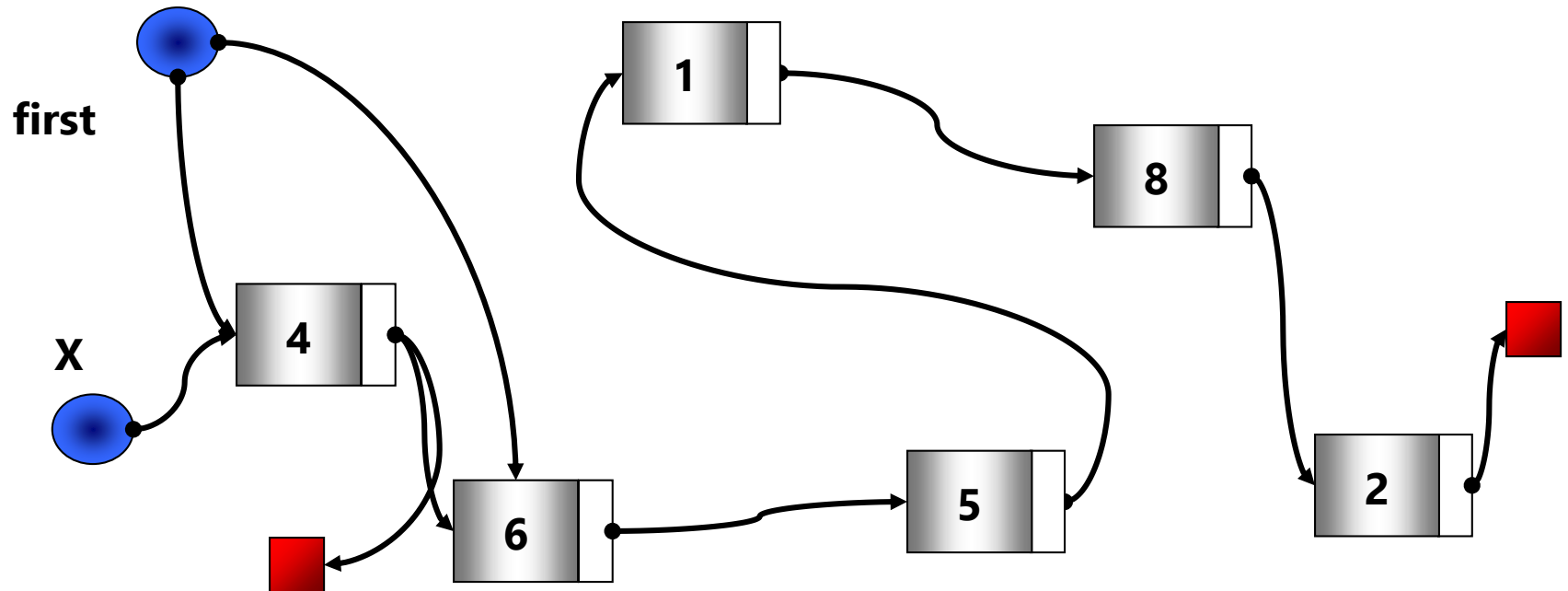
# Sắp xếp quick sort

100



# Quick sort : phân hoạch

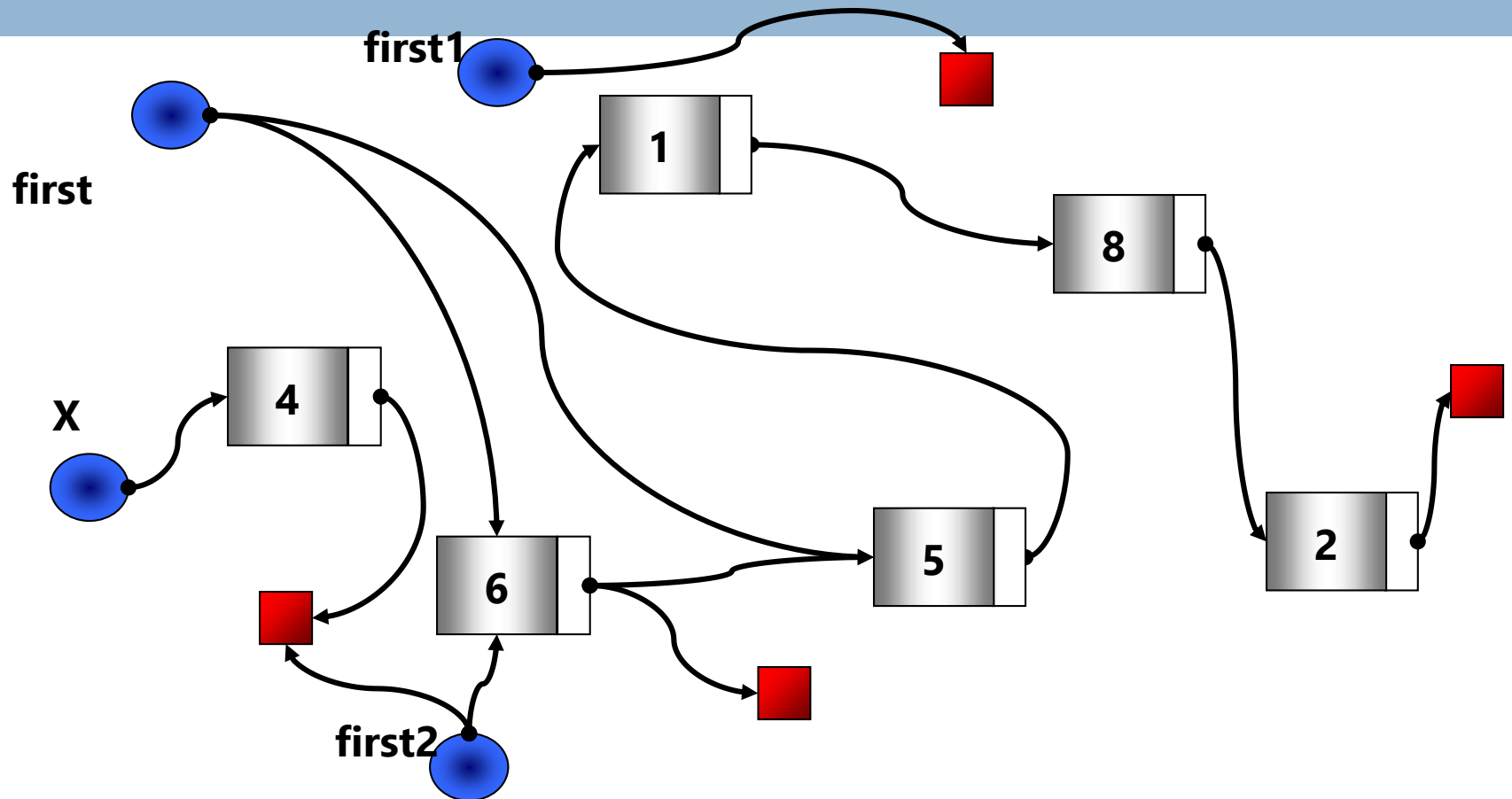
101



**Chọn phần tử đầu xâu làm ngưỡng**

# Quick sort : phân hoạch

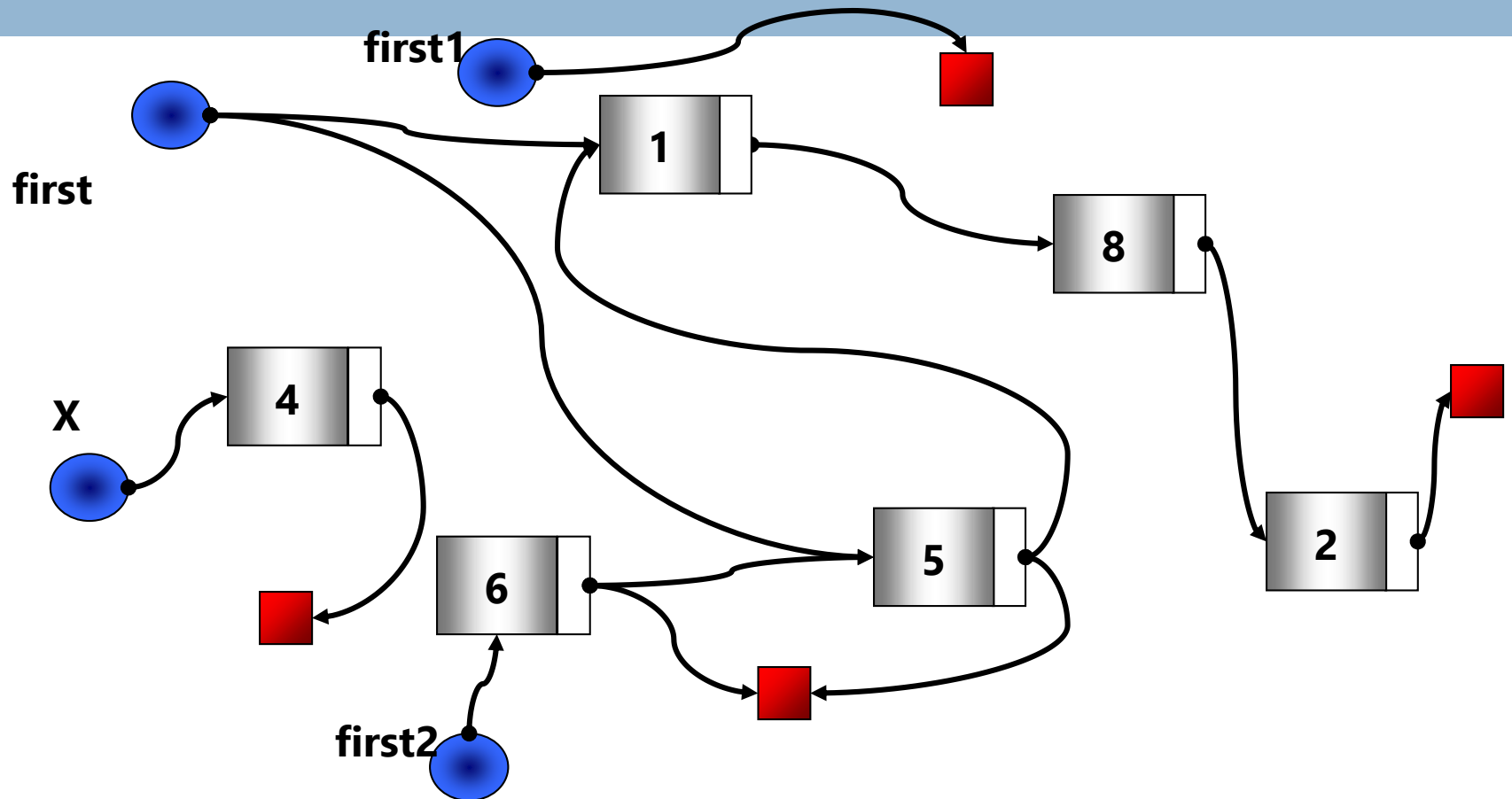
102



**Tách xâu hiện hành thành 2 xâu**

# Quick sort : phân hoạch

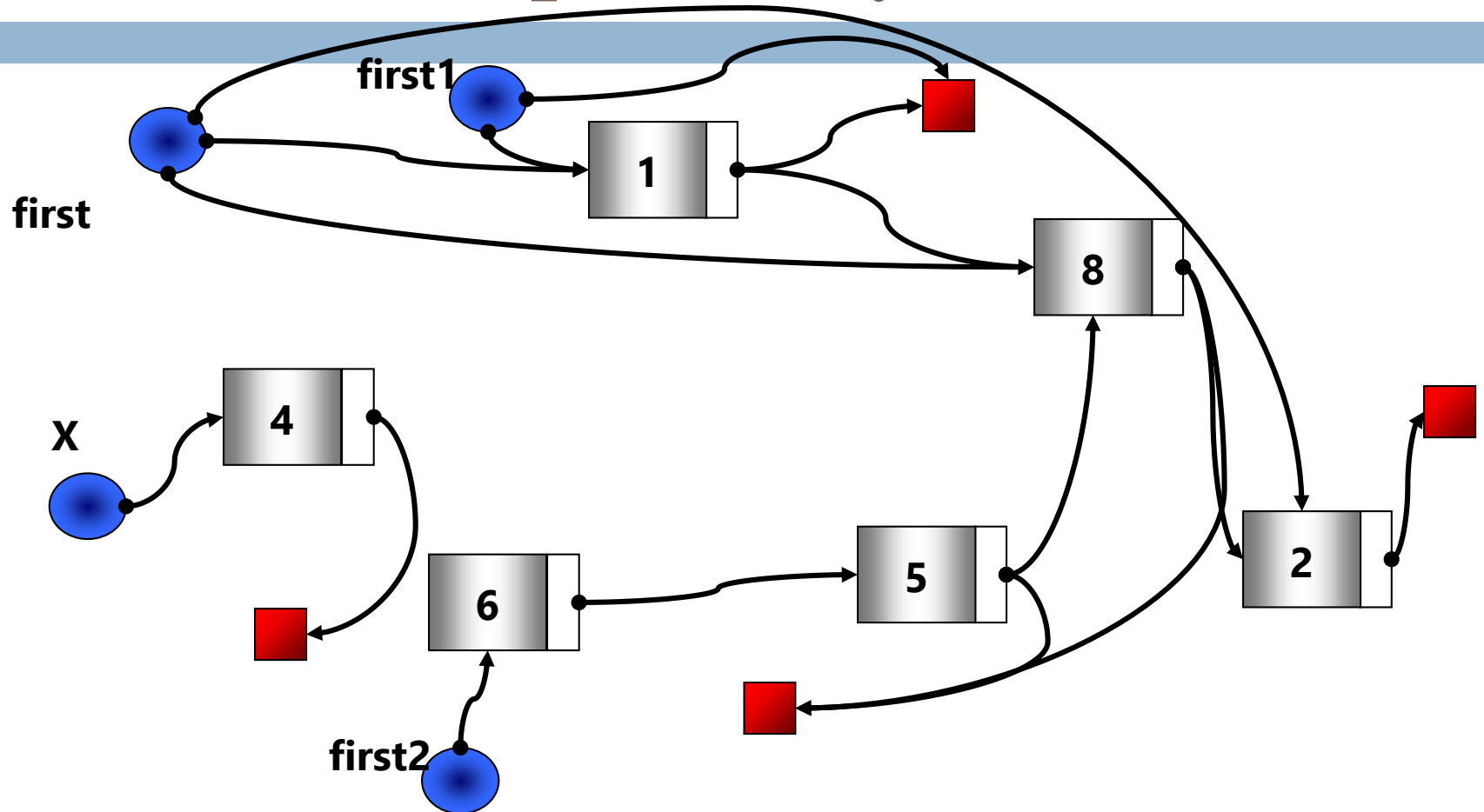
103



**Tách xâu hiện hành thành 2 xâu**

# Quick sort : phân hoạch

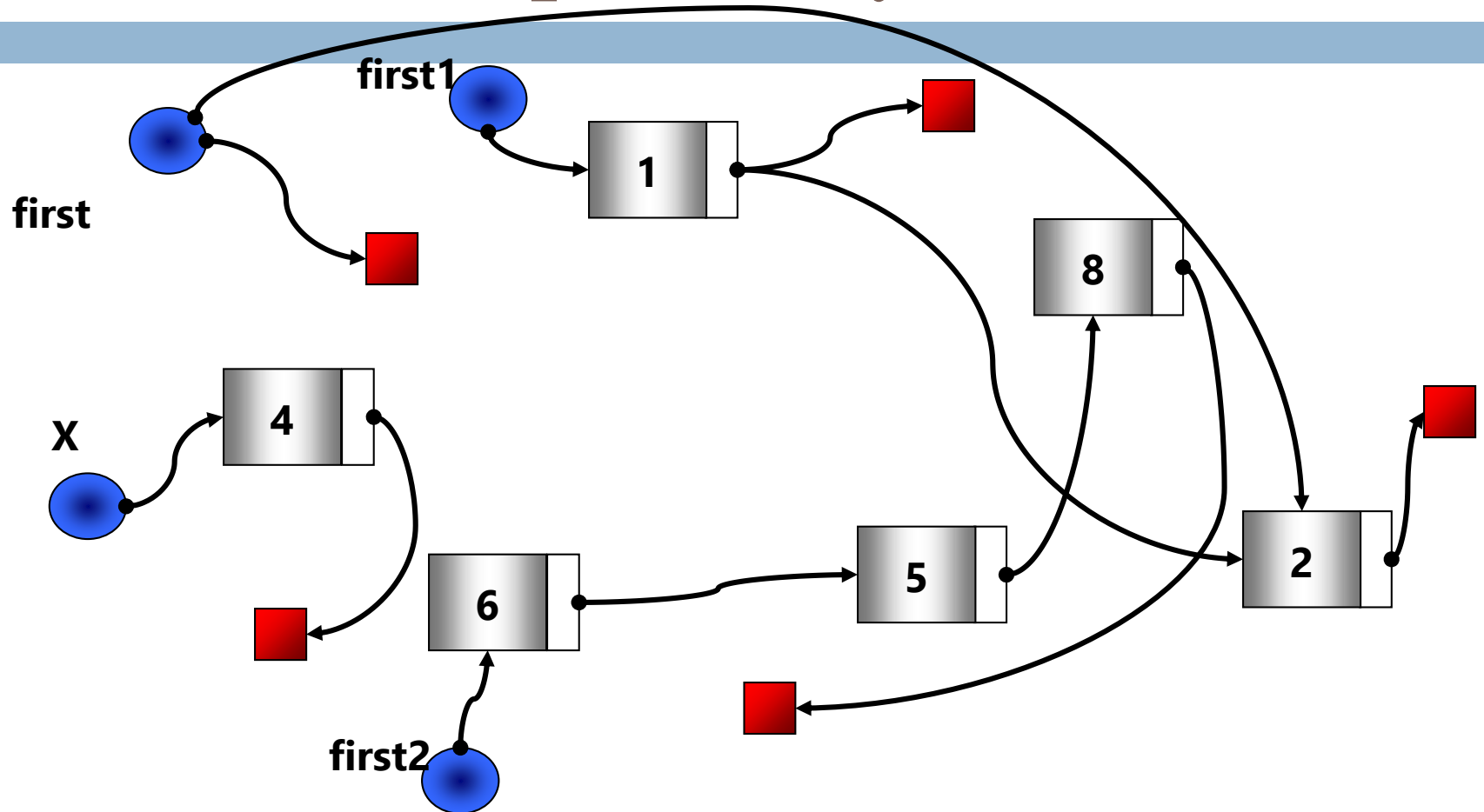
104



**Tách xâu hiện hành thành 2 xâu**

# Quick sort : phân hoạch

105

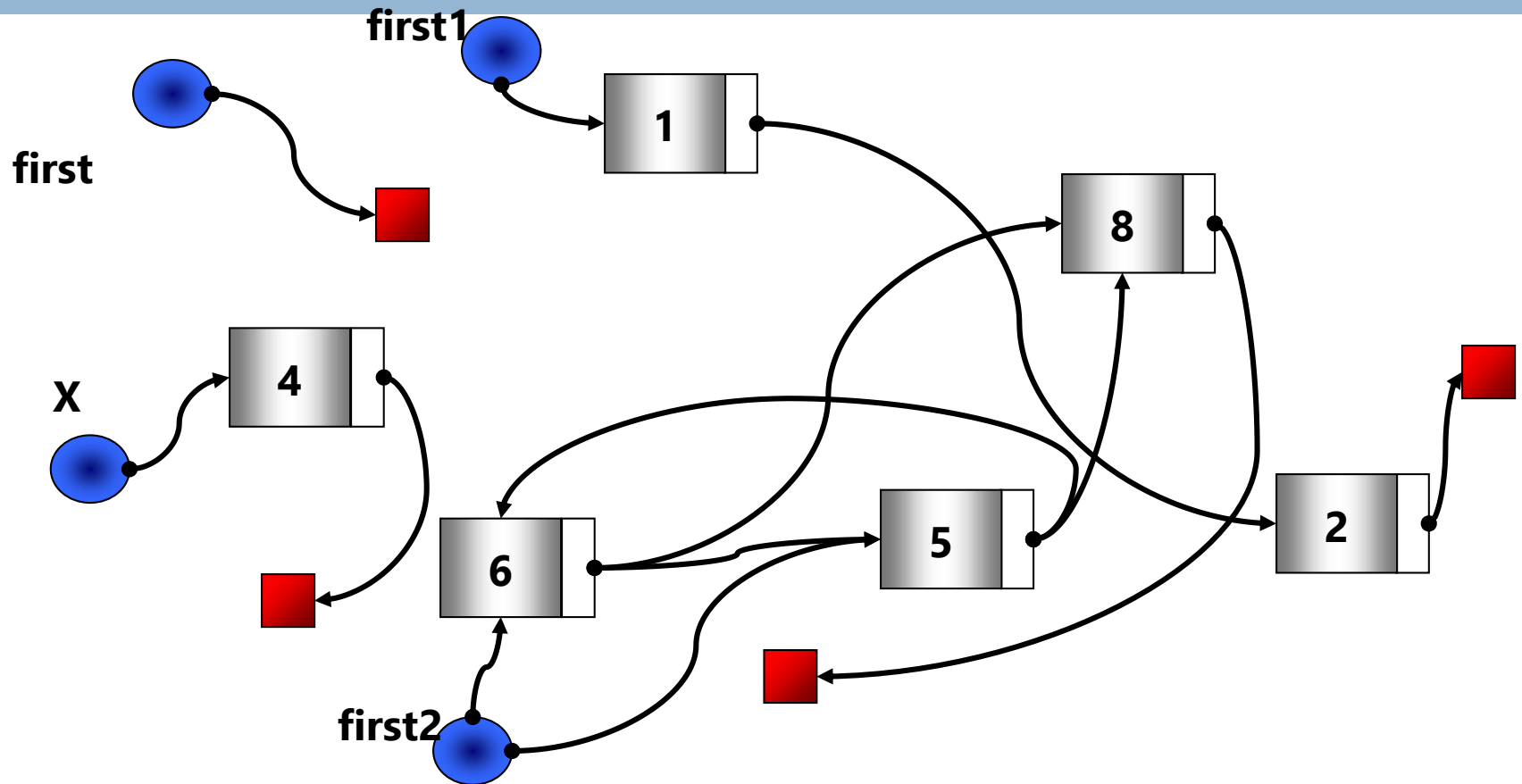


**Tách xâu hiện hành thành 2 xâu**



# Quick sort

106

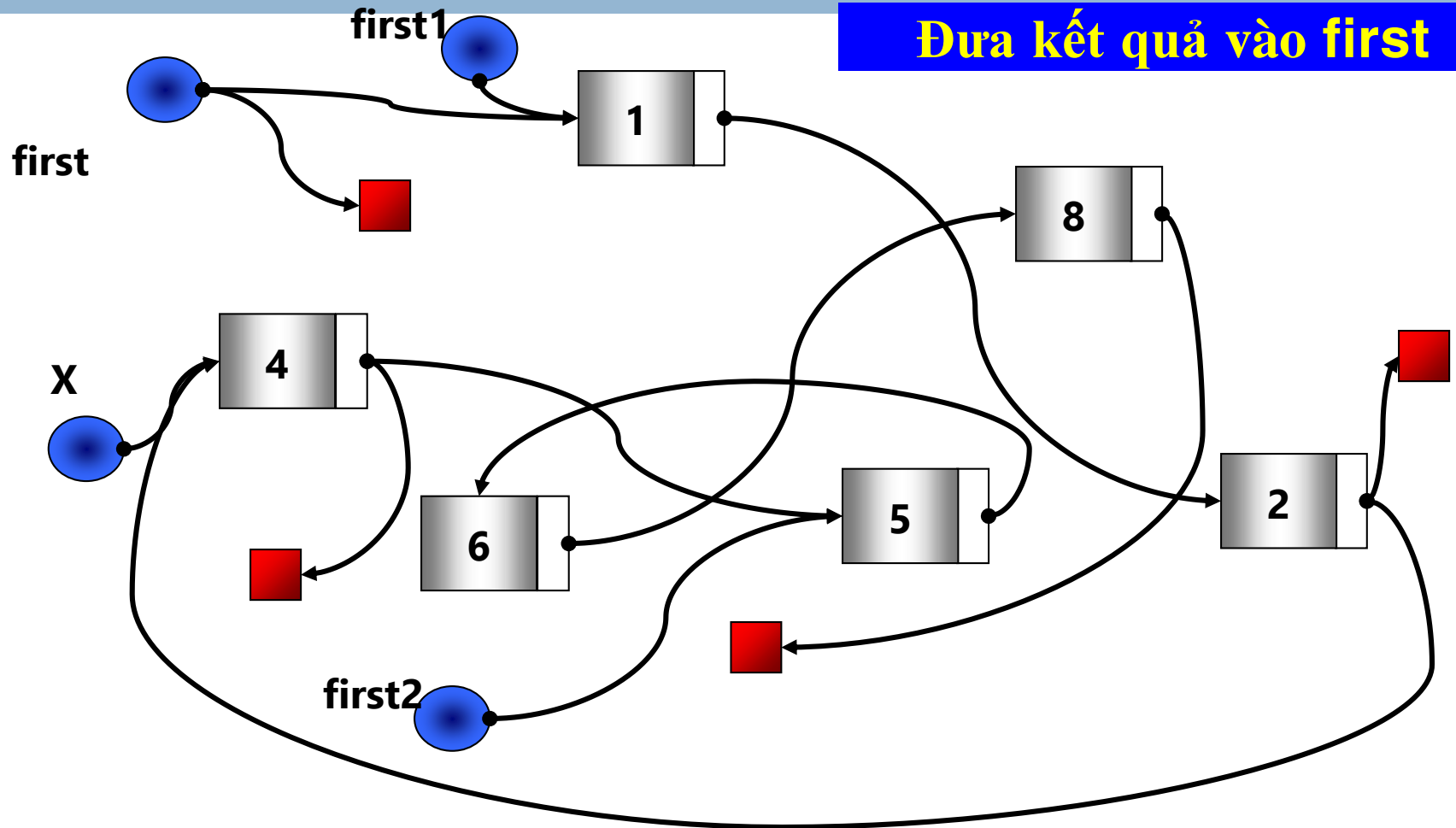


**Sắp xếp các xâu l1, l2**

# Quick sort

107

Đưa kết quả vào first



# Nối 2 danh sách

108

```
void SListAppend(SLIST &l, LIST &l2)
{
    if (l2.first == NULL) return;
    if (l.first == NULL)
        l = l2;
    else {
        l.first->link = l2.first;
        l.last = l2.last;
    }
    Init(l2);
}
```

```
void SListQSort(SLIST &l) {  
    NODE *X, *p;  
    SLIST l1, l2;  
    if (list.first == list.last) return;  
    Init(l1);      Init(l2);  
    X = l.first; l.first=x->link;  
    while (l.first != NULL) {  
        p = l.first;  
        if (p->data <= X->data) AddFirst(l1, p);  
        else AddFirst(l2, p);  
    }  
    SListQSort(l1);      SListQSort(l2);  
    SListAppend(l, l1);  
    AddFirst(l, X);  
    SListAppend(l, l2);  
}
```

# Quick sort : nhận xét

110

## Nhận xét:

- ▣ Quick sort trên cây đơn giản hơn phiên bản của nó trên mảng một chiều
- ▣ Khi dùng quick sort sắp xếp một cây đơn, chỉ có một chọn lựa phần tử cầm canh duy nhất hợp lý là phần tử đầu cây. Chọn bất kỳ phần tử nào khác cũng làm tăng chi phí một cách không cần thiết do cấu trúc tự nhiên của cây.

# Bài tập

111

- Thêm phần tử có giá trị  $x$  sau phần tử có giá trị  $y$
- Thêm vào danh sách không có khóa trùng
- Thêm vào danh sách có thứ tự

# Bài tập

112

Thông tin của một quyển sách trong thư viện gồm các thông tin: Tên sách (chuỗi), Tác giả (chuỗi, tối đa 5 tác giả), Nhà xuất bản (chuỗi), Năm xuất bản (số nguyên), giá int

1. Hãy tạo danh sách liên kết đơn chứa thông tin các quyển sách có trong thư viện (được nhập từ bàn phím). Người dùng không nhập nữa thì thôi
2. Thêm 1 sách mới vào đầu, cuối danh sách. Chú ý nếu tên sách có rồi thì bắt nhập tên khác
3. Xuất danh sách các cuốn sách
4. Cho biết số lượng các quyển sách của một tác giả bất kỳ (nhập từ bàn phím).
5. Trong năm YYYY (nhập từ bàn phím), nhà xuất bản ABC (nhập từ bàn phím) đã phát hành những quyển sách nào.
6. Tìm 1 sách có tên là x, nếu có xuất thông tin sách vừa tìm thấy và cho phép người dùng thêm vào 1 sách mới, nếu tên sách đó có rồi thì thông báo đã có nhập lại tên khác
7. Xóa 1 sách khỏi danh sách theo 3 cách: xóa đầu, xóa cuối và xóa 1 sách có tên là k

# Nội dung (Giáo trình 278)

113

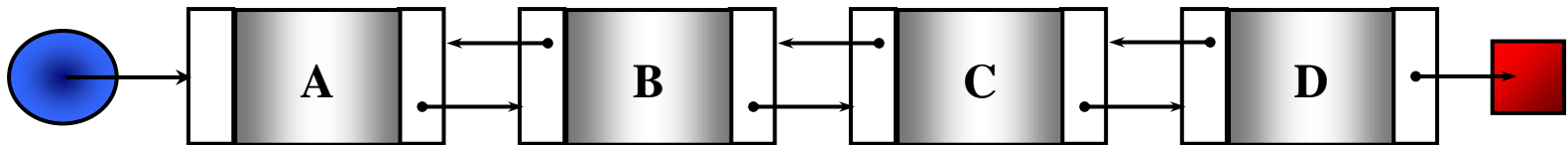
- Giới thiệu
- Danh sách liên kết đơn (Single Linked List)
- Danh sách liên kết đôi (Double Linked List)
- Danh sách liên kết vòng (Circular Linked List)



# Danh sách liên kết đôi (DSLK đôi)

114

- Là danh sách mà trong đó mỗi nút có liên kết với 1 nút đứng trước nó và 1 nút đứng sau nó



# DSLK đôi – Khai báo cấu trúc

115

□ Dùng hai con trỏ:

□ **pPrev** liên kết với node đứng trước

□ **pNext** liên kết với node đứng sau

```
struct DNode
```

```
{
```

```
    DataType
```

```
    data;
```

```
    DNode*
```

```
    pPrev;
```

```
    // trỏ đến phần tử đứng trước
```

```
    DNode*
```

```
    pNext;
```

```
    // trỏ đến phần tử đứng sau
```

```
};
```

```
struct DList
```

```
{
```

```
    DNode*
```

```
    pHead;
```

```
    // trỏ đến phần tử đầu ds
```

```
    DNode*
```

```
    pTail;
```

```
    // trỏ đến phần tử cuối ds
```

```
};
```

# DSLK đôi – Tạo nút mới

116

- Hàm tạo nút mới:

```
DNode* getNode (DataType x)
```

```
{
```

```
    DNode *p;
```

```
    p = new DNode; // Cấp phát vùng nhớ cho phần tử
```

```
    if (p==NULL) {
```

```
        cout<<"Không đủ bộ nhớ"; return NULL;
```

```
    }
```

```
    p->data = x; // Gán thông tin cho phần tử p
```

```
    p->pPrev = p->pNext = NULL;
```

```
    return p;
```

```
}
```



Gọi hàm??

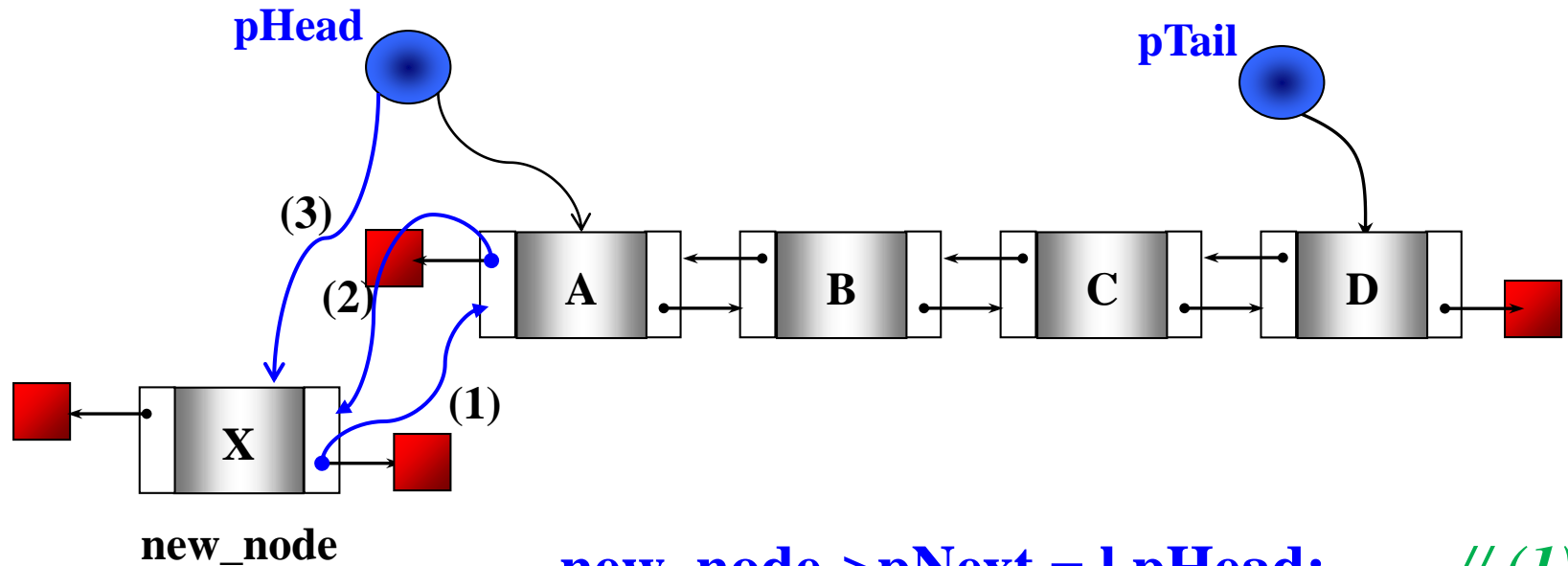
# DSLK đôi – Thêm 1 nút vào ds

117

- Có 4 cách thêm:
  1. Chèn vào đầu danh sách
  2. Chèn vào cuối danh sách
  3. Chèn vào danh sách sau một phần tử q
  4. Chèn vào danh sách trước một phần tử q
- Chú ý trường hợp khi danh sách ban đầu rỗng

# Minh họa: Thêm vào đầu ds

118



$\text{new\_node} \rightarrow \text{pNext} = \text{l.pHead};$  // (1)

$\text{l.pHead} \rightarrow \text{pPrev} = \text{new\_node};$  // (2)

$\text{l.pHead} = \text{new\_node};$  // (3)

# Cài đặt: Thêm vào đầu ds

119

```
void addHead ( DList &l, DNode* new_node )
```

```
{
```

```
    if ( l.pHead==NULL )
```

```
        l.pHead = l.pTail = new_node;
```

```
    else
```

```
        { new_node->pNext = l.pHead;
```

// (1)

```
        l.pHead->pPrev = new_node;
```

// (2)

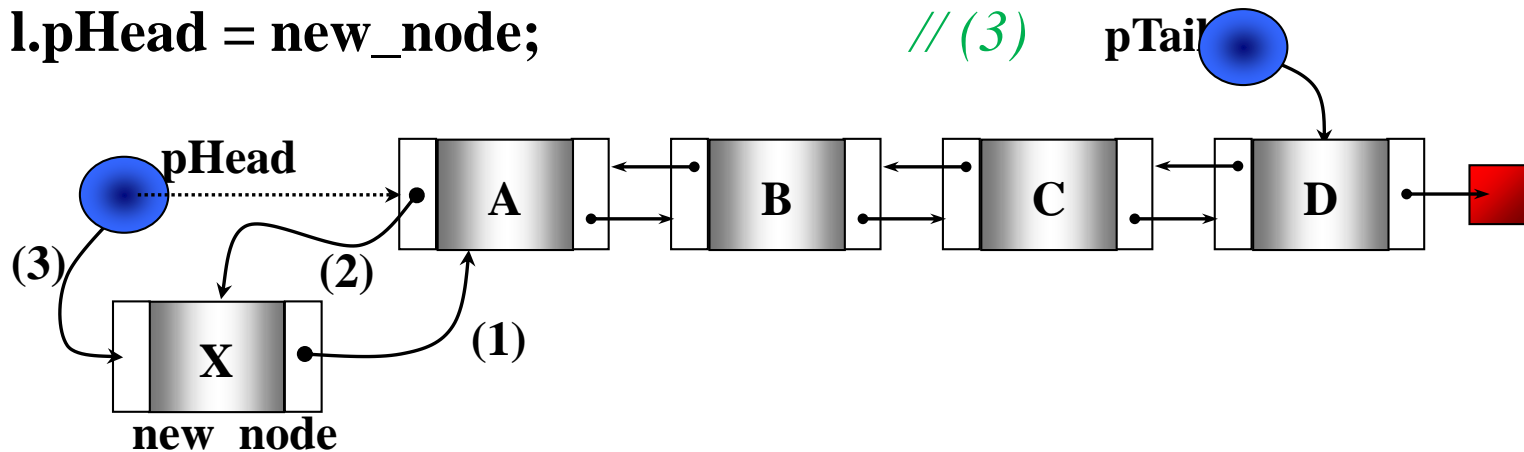
```
        l.pHead = new_node;
```

// (3)

```
    }
```

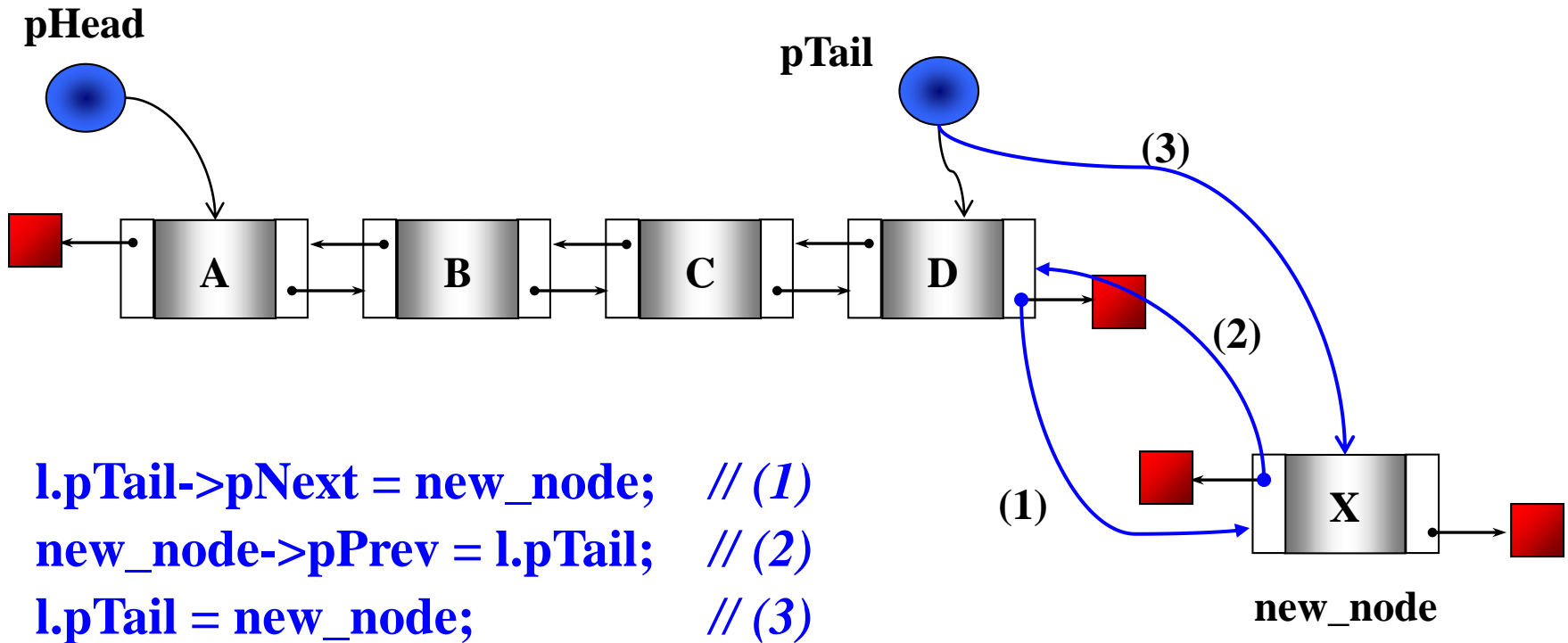
```
}
```

Gọi hàm??



# Minh họa: Thêm vào cuối ds

120



# Cài đặt – Thêm vào cuối ds

121

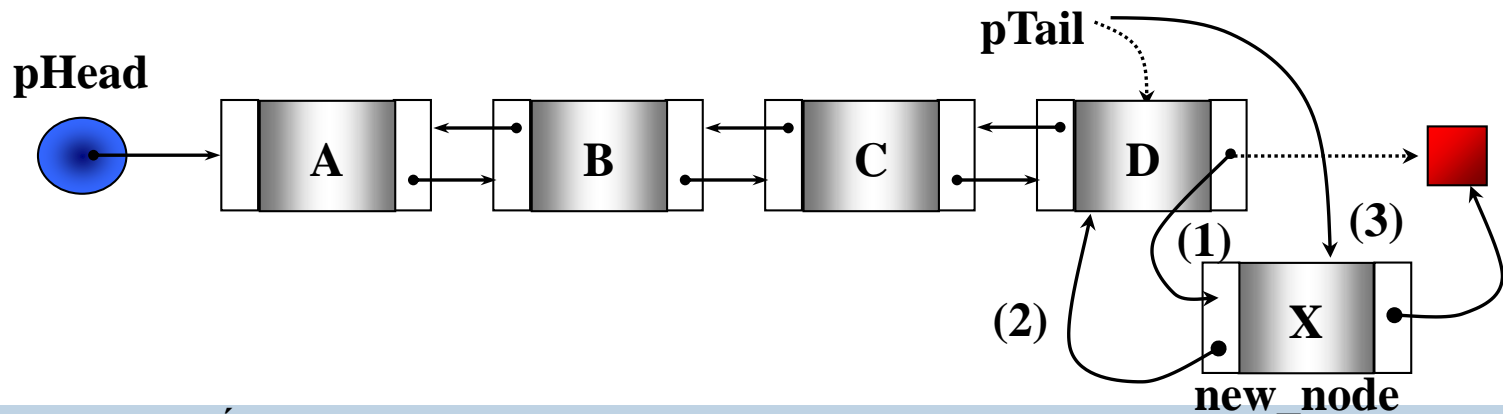
```
void addTail ( DList &l, DNode *new_node )  
{  
    if ( l.pHead==NULL )  
        l.pHead = l.pTail = new_node;  
    else  
    {  
        l.pTail->pNext = new_node;  
        new_node->pPrev = l.pTail;  
        l.pTail = new_node;  
    }  
}
```

Gọi hàm??

// (1)

// (2)

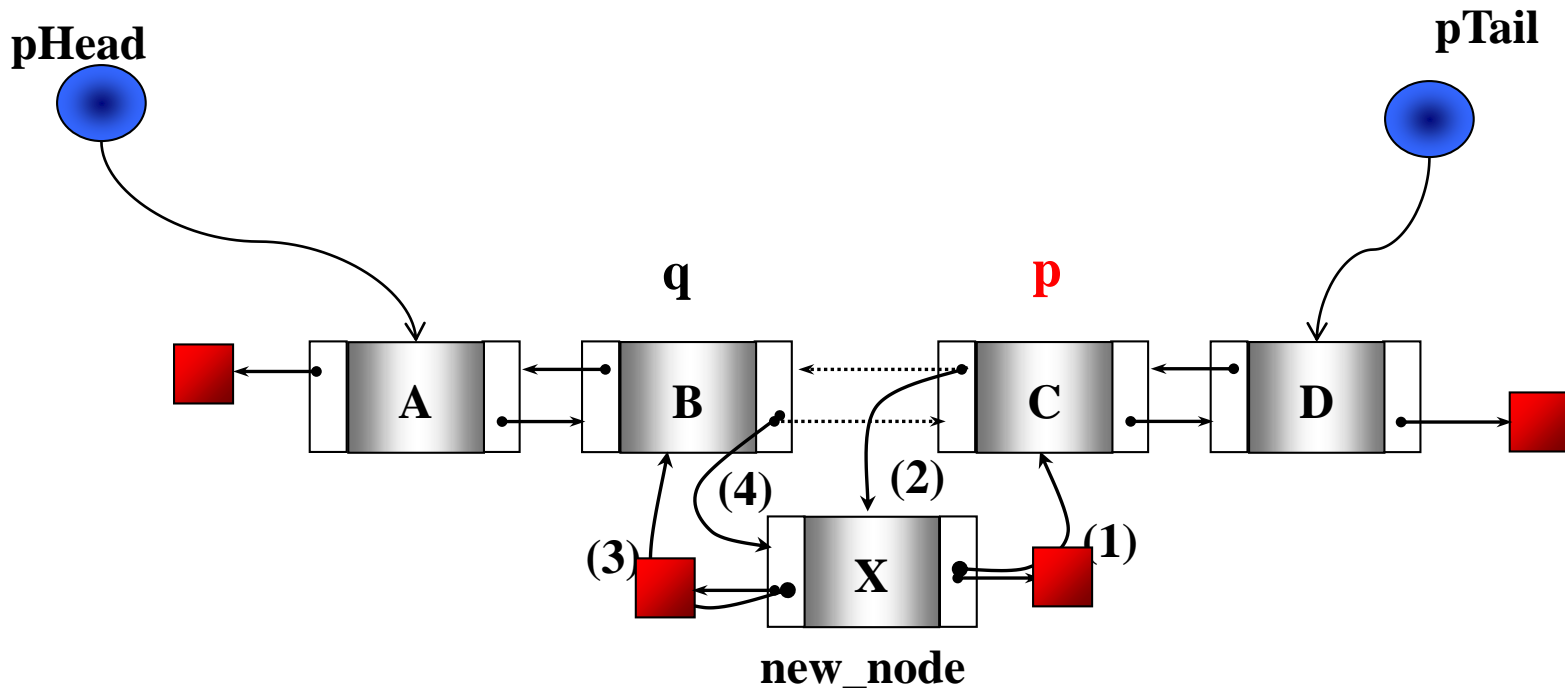
// (3)





# Minh họa: Chèn vào sau q

122



```
new_node->pNext = p;    //(1)
if ( p != NULL ) p->pPrev = new_node; //(2)
new_node->pPrev = q;    //(3)
q->pNext = new_node;   //(4)
```

# Cài đặt: Chèn vào sau q

123

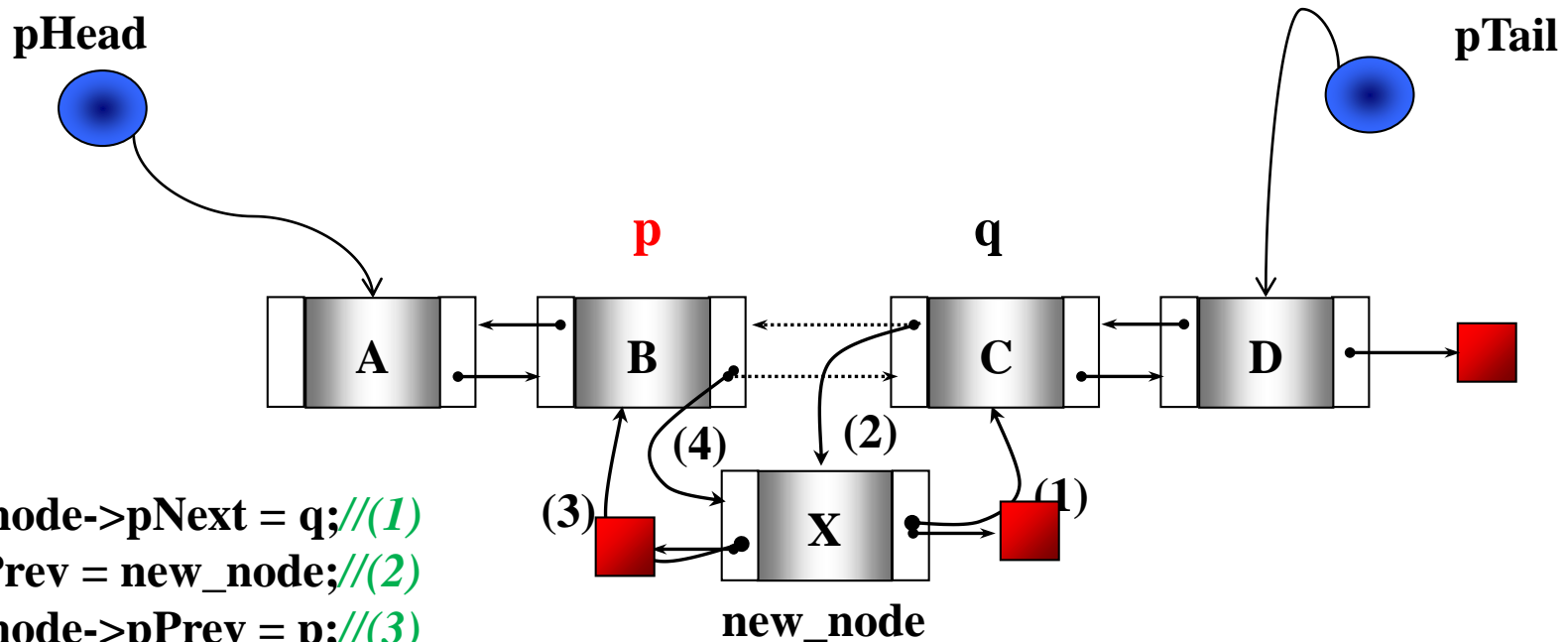
```
void addAfter (DList &l, DNode *q, DNode *new_node)
{
    DNode *p = q->pNext;
    if (q!=NULL) {
        new_node->pNext = p;           //(1)
        if ( p != NULL ) p->pPrev = new_node;   //(2)
        new_node->pPrev = q;           //(3)
        q->pNext = new_node;          //(4)
        if ( q == l.pTail ) l.pTail = new_node;
    }
}
```



Gọi hàm??

# Minh họa: Chèn vào trước q

124



# Cài đặt: Chèn vào trước q

125

```
void addBefore ( DList &l, DNode *q, DNode* new_node )
{
    DNode* p = q->pPrev;
    if ( q!=NULL )
    {
        new_node->pNext = q;           //(1)
        q->pPrev = new_node;          //(2)
        new_node->pPrev = p;           //(3)
        if ( p != NULL ) p->pNext = new_node; //(4)
        if ( q == l.pHead ) l.pHead = new_node;
    }
}
```



Gọi hàm??

# Cài đặt: Duyệt danh sách

126

```
void Output(DList l)
```

```
{    DNode *tmp = l.pHead;
    printf("\nXuat theo chieu nguc:\n");
    while(tmp->pNext!= NULL)
    {        //printf("%d\t",tmp->data);
            tmp = tmp->pNext;
    }
    //printf("%d\t",tmp->data);
    printf("\nXuat theo chieu thuan:\n");
    while(tmp!=NULL)
    {        printf("%d\t",tmp->data);
            tmp = tmp->pPrev;
```



**Gọi hàm??**

# DSLK đôi – Hủy phần tử

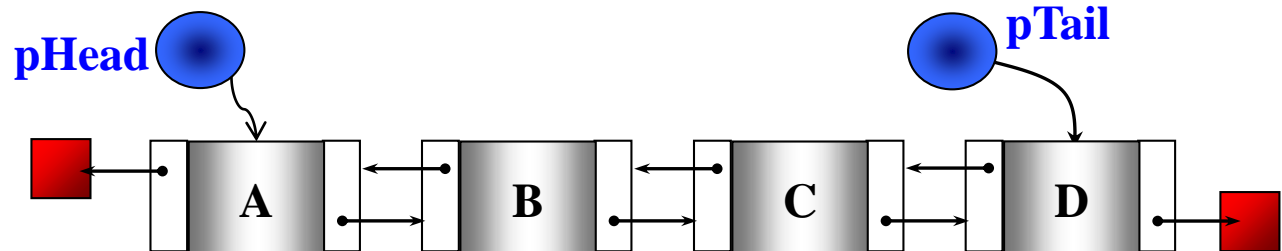
127

- Có 5 thao tác thông dụng hủy một phần tử ra khỏi danh sách liên kết đôi:
  1. Hủy phần tử đầu ds
  2. Hủy phần tử cuối ds
  3. Hủy một phần tử đứng sau phần tử q
  4. Hủy một phần tử đứng trước phần tử q
  5. Hủy 1 phần tử có khóa k

# DSLK đôi – Hủy đầu ds

128

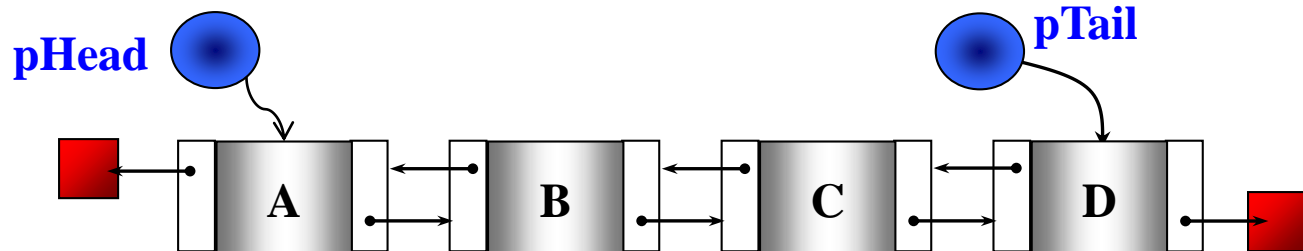
```
int removeHead ( DList &l )  
{  
    if ( l.pHead == NULL ) return 0;  
    DNode *p = l.pHead;  
    l.pHead = l.pHead->pNext;  
    delete p;  
    if ( l.pHead != NULL )    l.pHead->pPrev = NULL;  
    else                    l.pTail = NULL;  
    return 1;  
}
```



# DSLK đôi – Hủy cuối ds

129

```
int removeTail ( DList &l )  
{  
    if ( l.pTail == NULL ) return 0;  
    DNode *p = l.pTail;  
    l.pTail = l.pTail->pPrev;  
    delete p;  
    if ( l.pTail != NULL ) l.pTail->pNext = NULL;  
    else l.pHead = NULL;  
    return 1;  
}
```

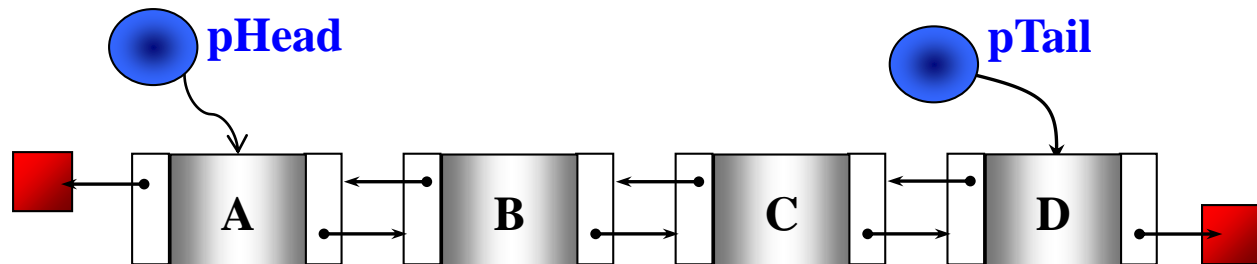




# DSLK đôi – Hủy phần tử sau q

130

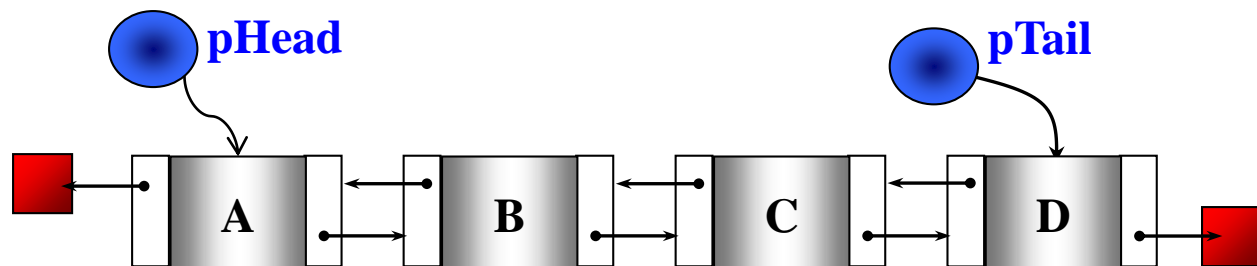
```
int removeAfter ( DList &l, DNode *q )
{
    if ( q == NULL ) return 0;
    DNode *p = q ->pNext ;
    if ( p != NULL )
    {
        q->pNext = p->pNext;
        if ( p == l.pTail ) l.pTail = q;
        else p->pNext->pPrev = q;
        delete p;
        return 1;
    }
    else return 0;
}
```



# DSLK đôi – Hủy phần tử trước q

131

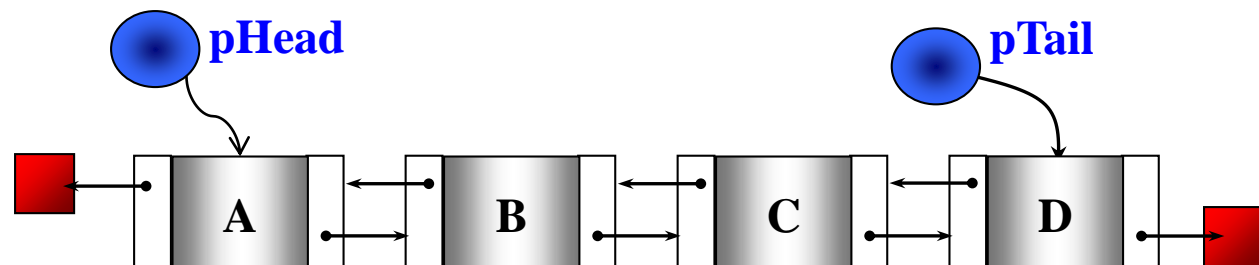
```
int removeBefore ( DList &l, DNode *q )
{
    if ( q == NULL ) return 0;
    DNode *p = q ->pPrev;
    if ( p != NULL )
    {
        q->pPrev = p->pPrev;
        if ( p == l.pHead )    l.pHead = q;
        else    p->pPrev->pNext = q;
        delete p;
        return 1;
    }
    else return 0;
}
```



# DSLK đôi – Hủy phần tử có khóa k

132

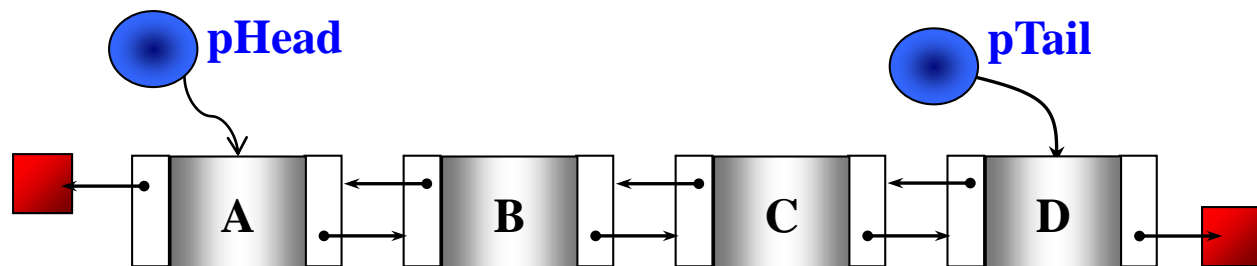
```
int removeNode ( DList &l, int k )  
{  
    DNode *p = l.pHead;  
    while ( p != NULL )  
    {  
        if ( p->data == k ) {  
  
            break;}  
        p = p->pNext;  
    }  
}
```



# DSLK đôi – Hủy phần tử có khóa k

133

```
if ( p == NULL ) return 0; // Không tìm thấy k
DNode *q = p->pPrev;
if ( q != NULL )           // Xóa nút p sau q
    return removeAfter ( l, q );
else                       // Xóa p là nút đầu ds
    return removeHead ( l );
}
```



# DSLK đôi – Nhận xét

134

- DSLK đôi về mặt cơ bản có tính chất giống như DSLK đơn
- Tuy nhiên DSLK đôi có mỗi liên kết hai chiều nên từ một phần tử bất kỳ có thể truy xuất một phần tử bất kỳ khác
- Trong khi trên DSLK đơn ta chỉ có thể truy xuất đến các phần tử đứng sau một phần tử cho trước
- Điều này dẫn đến việc ta có thể dễ dàng hủy phần tử cuối DSLK đôi, còn trên DSLK đơn thao tác này tốn chi phí  $O(n)$
- Bù lại, xâu đôi tốn chi phí gấp đôi so với xâu đơn cho việc lưu trữ các mỗi liên kết. Điều này khiến việc cập nhật cũng nặng nề hơn trong một số trường hợp. Như vậy ta cần cân nhắc lựa chọn CTDL hợp lý khi cài đặt cho một ứng dụng cụ thể

# Bài tập

135

- Tạo menu và thực hiện các chức năng sau trên DSLK đơn chứa số nguyên:
  1. Thêm một số pt vào cuối ds
  2. Thêm 1 pt vào trước pt nào đó
  3. In ds
  4. In ds theo thứ tự ngược
  5. Tìm GTNN, GTLN trong ds
  6. Tính tổng số âm, tổng số dương trong ds
  7. Tính tích các số trong ds
  8. Tính tổng bình phương của các số trong ds
  9. Nhập x, xuất các số là bội số của x
  10. Nhập x, xuất các số là ước số của x
  11. Nhập x, tìm giá trị đầu tiên trong ds mà  $>x$

# Bài tập (tt)

136

12. Xuất số nguyên tố cuối cùng trong ds
13. Đếm các số nguyên tố
14. Kiểm tra xem ds có phải đã được sắp tăng không
15. Kiểm tra xem ds có các pt đối xứng nhau hay không
16. Xóa pt cuối
17. Xóa pt đầu
18. Hủy toàn bộ ds

# Nội dung

137

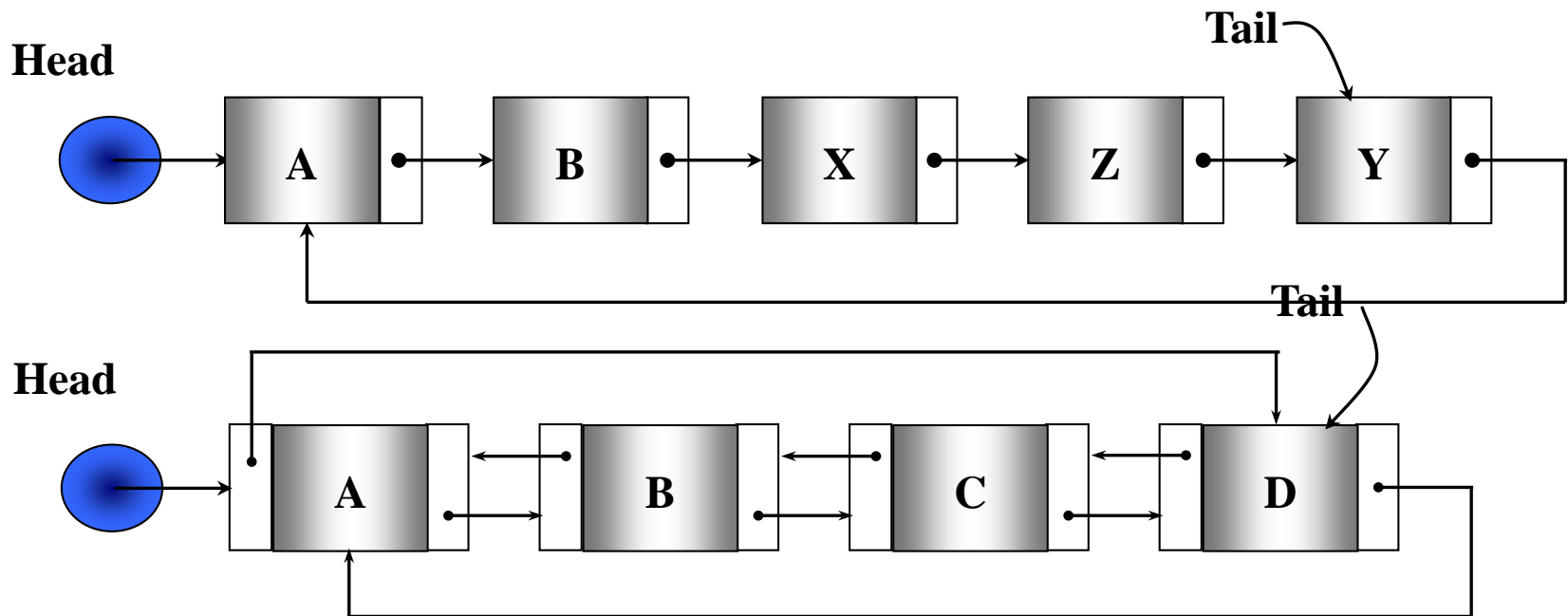
- Giới thiệu
- Danh sách liên kết đơn (Single Linked List)
- Danh sách liên kết đôi (Double Linked List)
- Danh sách liên kết vòng (Circular Linked List)



# Danh sách liên kết vòng (DSLK vòng)

138

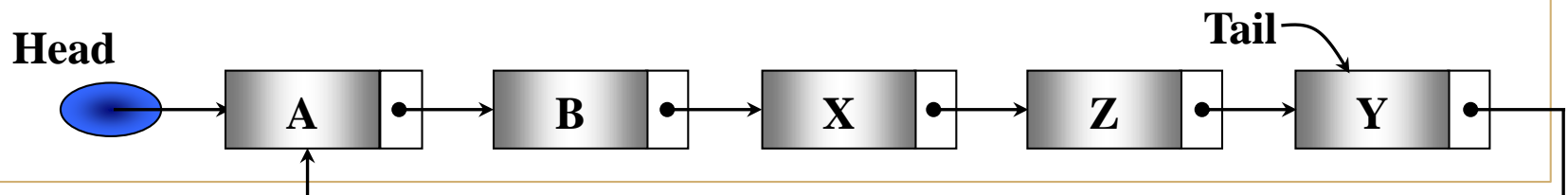
- Là một danh sách liên kết đơn (hoặc đôi) mà nút cuối danh sách, thay vì trỏ đến **NULL**, sẽ trỏ tới nút đầu danh sách
- Đối với danh sách vòng, có thể xuất phát từ một phần tử bất kỳ để duyệt toàn bộ danh sách



# DSLK vòng – Thêm vào đầu ds

139

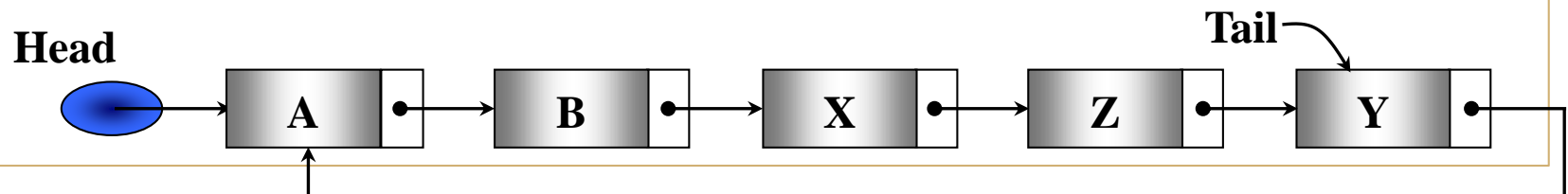
```
void addHead (List &l, Node *new_node)
{
    if (l.pHead == NULL) {
        l.pHead = l.pTail = new_node;
        l.pTail->pNext = l.pHead;
    }
    else{
        new_node->pNext = l.pHead;
        l.pTail->pNext = new_node;
        l.pHead = new_node;
    }
}
```



# DSLK vòng – Thêm vào cuối ds

140

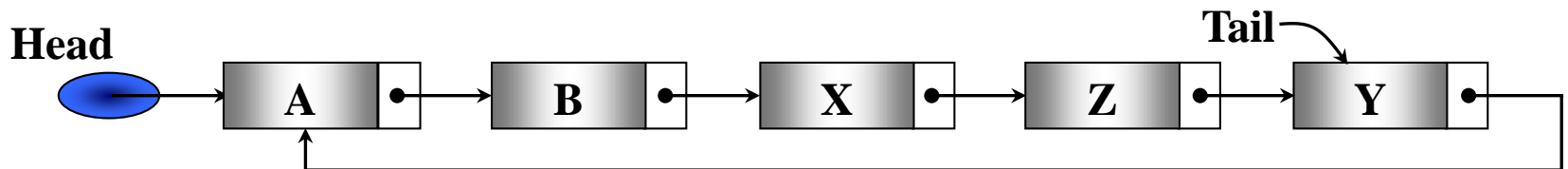
```
void addTail (List &l, Node *new_node)
{
    if (l.pHead == NULL) {
        l.pHead = l.pTail = new_node;
        l.pTail->pNext = l.pHead;
    }
    else{
        new_node->pNext = l.pHead;
        l.pTail->pNext = new_node;
        l.pTail = new_node;
    }
}
```



# DSLK vòng – Hủy nút đầu ds

141

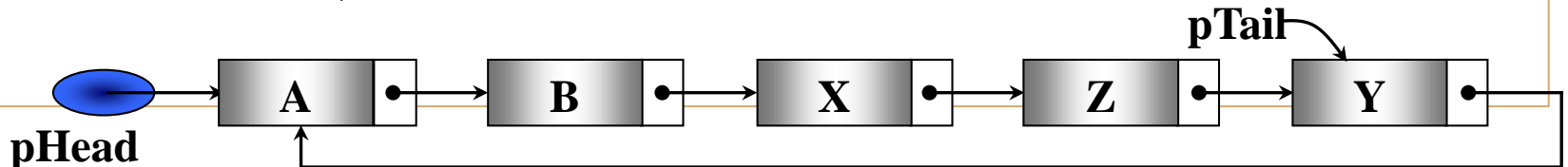
```
int removeHead (List &l){  
    Node *p = l.pHead;  
    if ( p == NULL ) return 0;  
    if ( l.pHead == l.pTail )  
        l.pHead = l.pTail = NULL;  
    else  
        { l.pHead = p->pNext; l.pTail->pNext = l.pHead; }  
    delete p;  
    return 1;  
}
```



# DSLK vòng – Hủy phần tử sau q

142

```
int removeAfter (List &l, Node *q)
{
    if ( q == NULL ) return 0;
    Node *p = q ->pNext ;
    if ( p == q )    l.pHead = l.pTail = NULL;
    else{
        q->Next = p->pNext;
        if (p == l.pTail)    l.pTail = q;
    }
    delete p;
    return 1;
}
```



# DSLK vòng – Duyệt danh sách

143

- Danh sách vòng không có phần tử đầu danh sách rõ rệt, nhưng ta có thể đánh dấu một phần tử bất kỳ trên danh sách xem như phần tử đầu xâu để kiểm tra việc duyệt đã qua hết các phần tử của danh sách hay chưa

```
Node *p = l.pHead;  
do{  
    // do something with p  
    p = p->pNext;  
} while (p != l.pHead);    // chưa đi giáp vòng
```

# Ví dụ: Tìm kiếm

144

```
Node* Search ( List &l, int x )
{
    Node *p = l.pHead;
    do{
        if ( p->data == x ) return p;
        p = p->pNext;
    } while ( p != l.pHead );    // chưa đi giáp vòng
    return NULL;
}
```

