

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

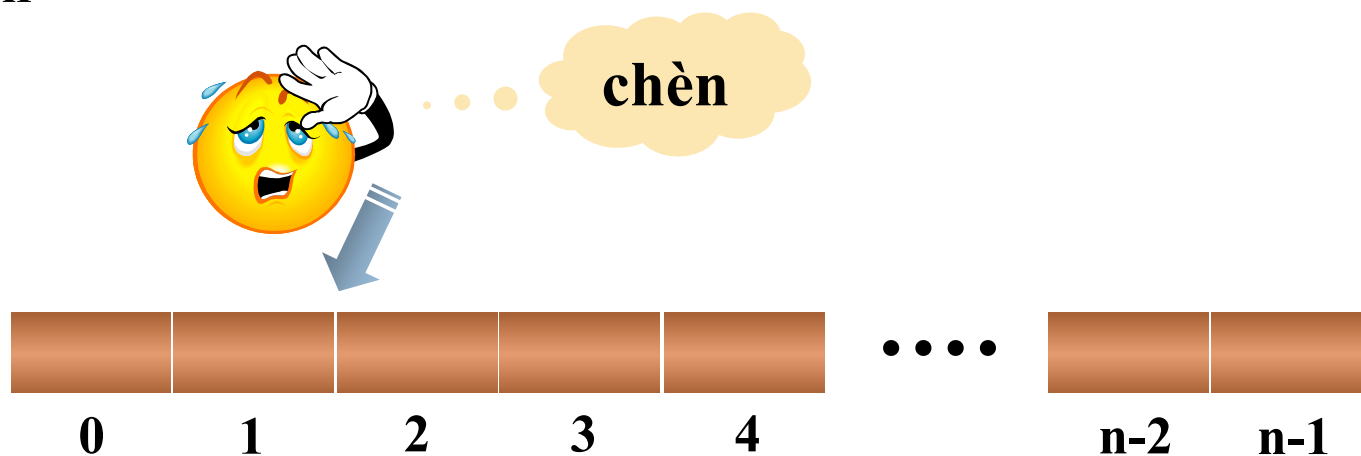
5

- 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

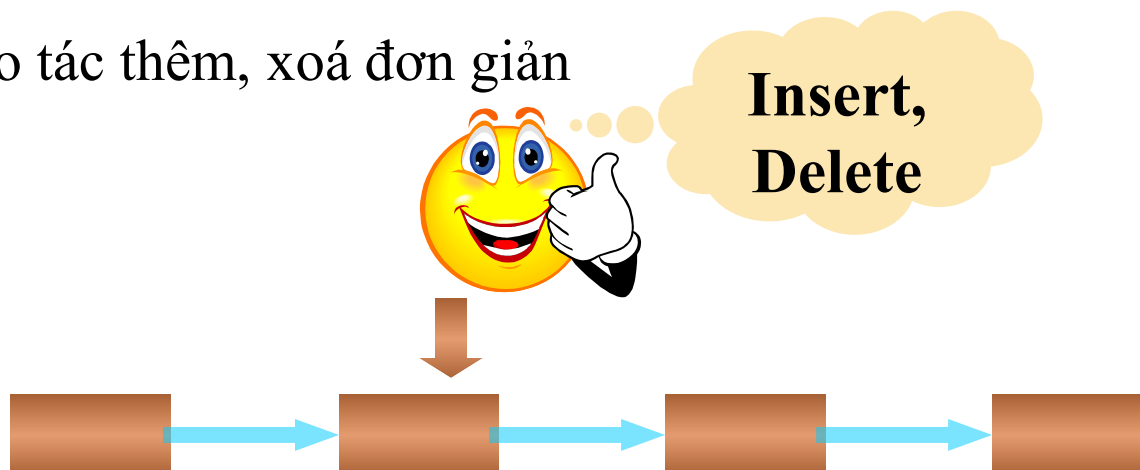
Hướng giải quyết

- 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



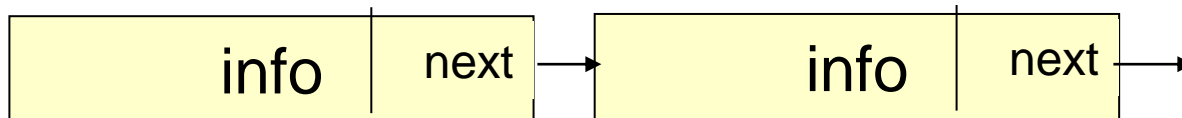
Danh sách liên kết (List)

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

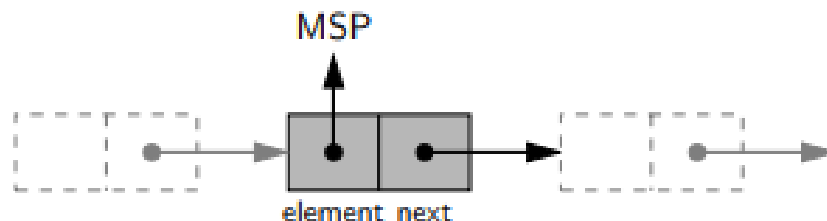
```
+---+ +---+ +---+ +---+
| 1 | -> | 2 | -> | 3 | -> | 4 |
+---+ +---+ +---+ +---+
```



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.

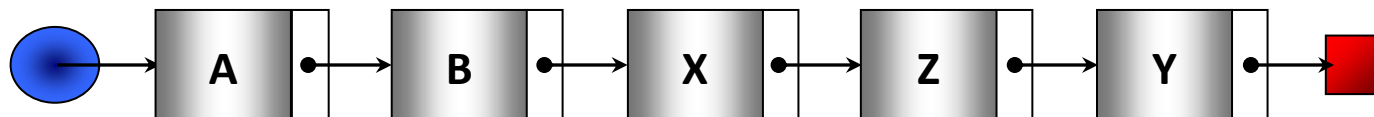


Giới thiệu - Danh sách liên kết

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

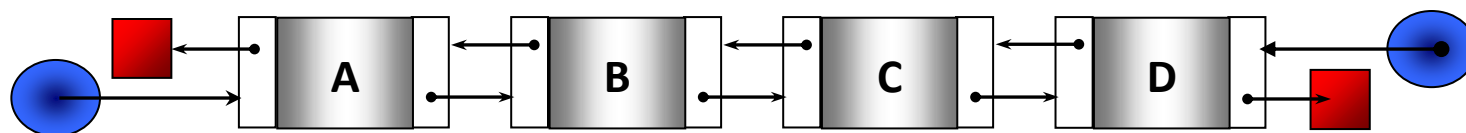
Giới thiệu - Danh sách liên kết

15

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



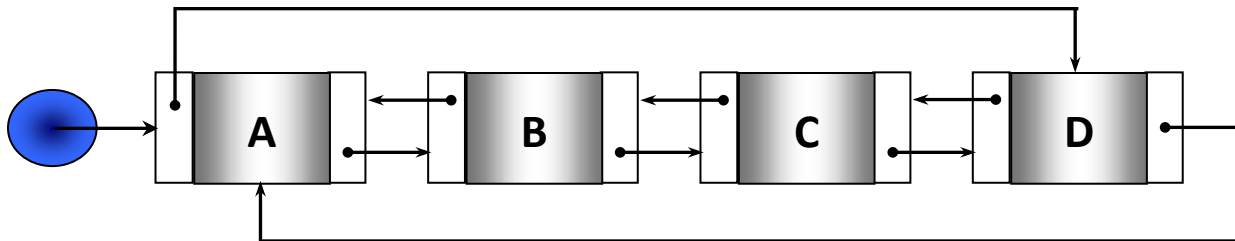
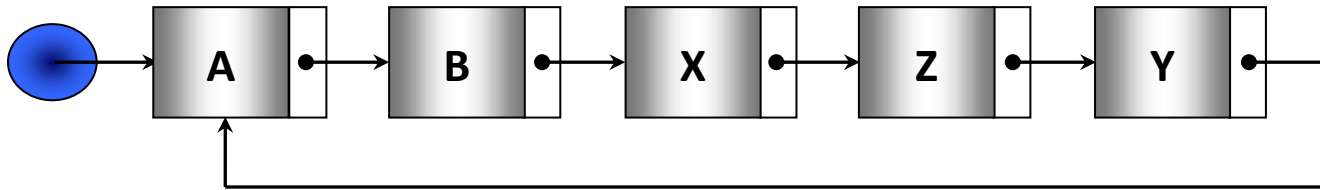
- **Danh sách liên kết kép:** mỗi phần tử liên kết với các phần tử đứng trước và sau nó trong danh sách:



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

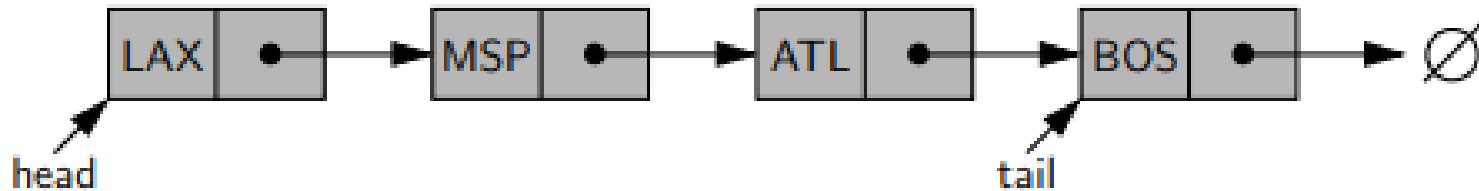
17

- Giới thiệu
- Danh sách liên kết đơn (**Single Linked List**)
- Danh sách liên kết kép (**Doule 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;
```

DSLK đơn – Khai báo

20

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

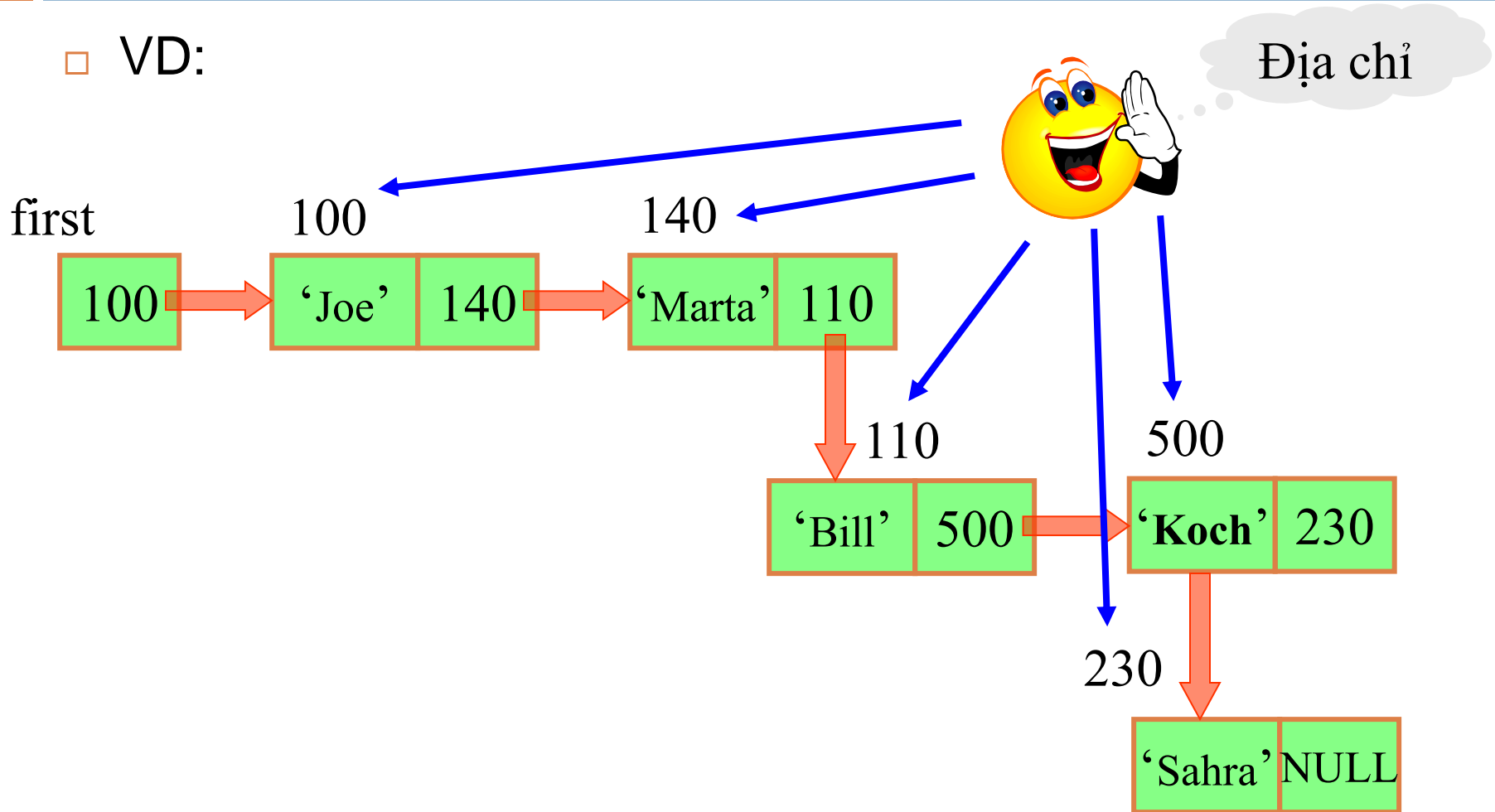
Address	Name	Age	Link
100	Joe	20	140
110	Bill	42	500
140	Marta	27	110
230	Sahra	25	NULL
...	
500	Koch	31	230

	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

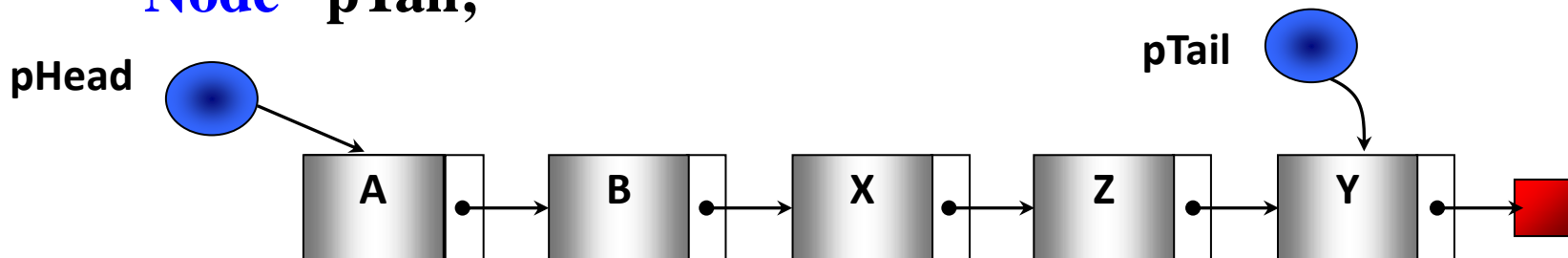
□ Tổ chức, quản lý:

- ▣ 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:

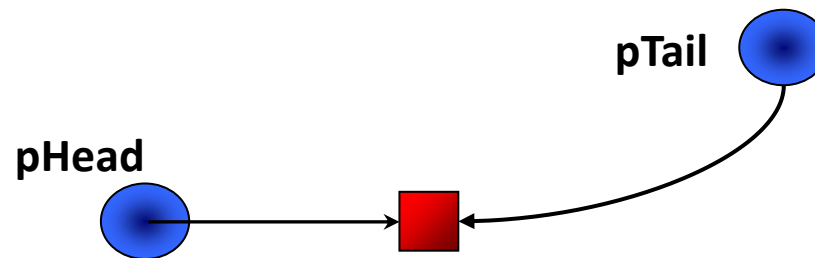
```
new_ele = GetNode(x);
```

→ 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

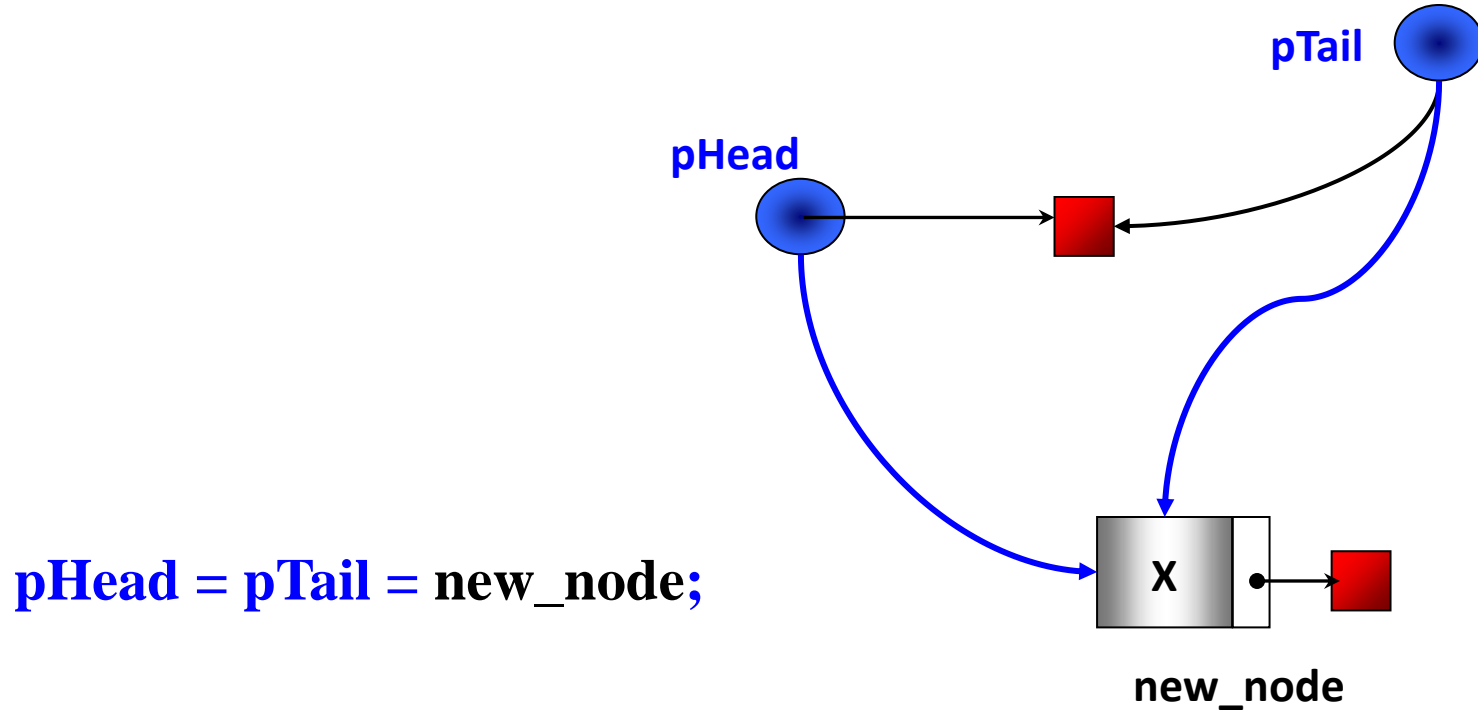
28

- **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

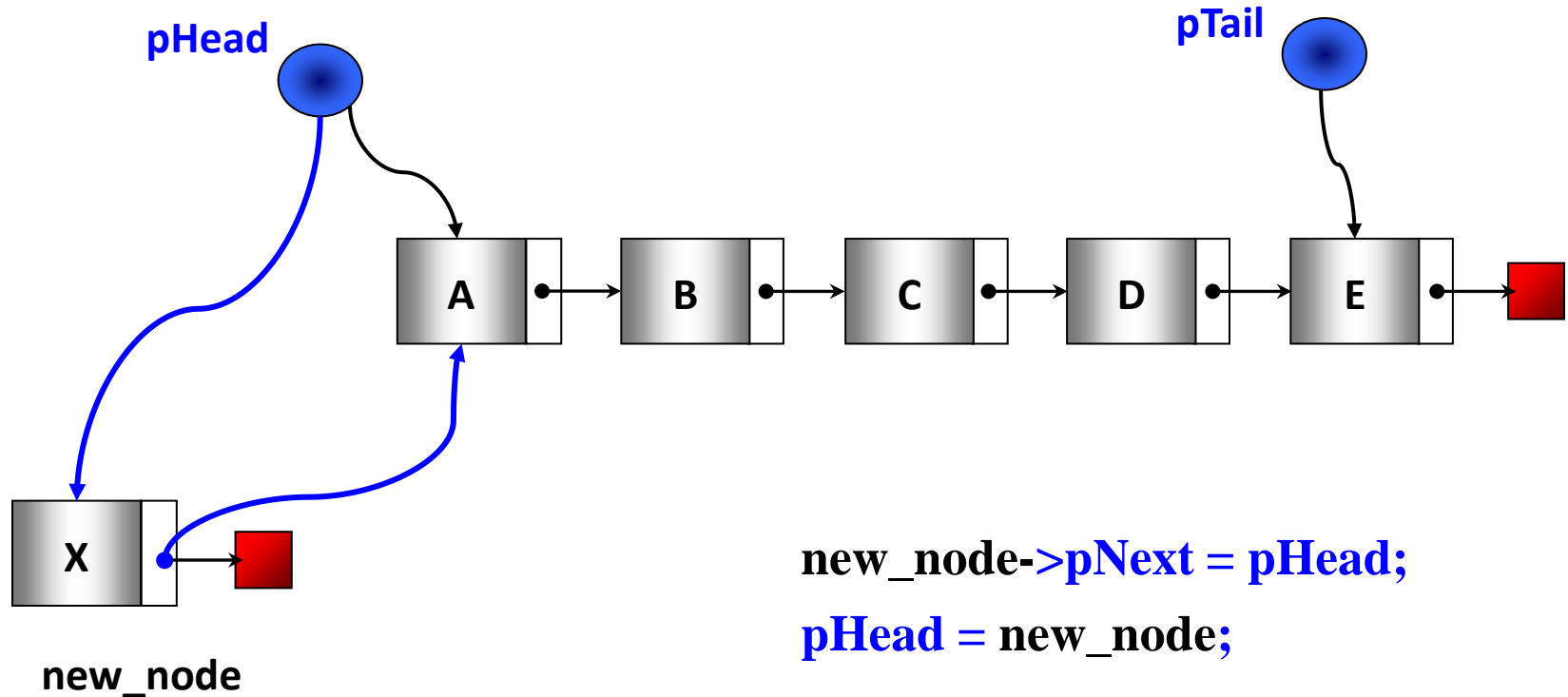
- Thêm một phần tử
 - ▣ Nếu danh sách ban đầu rỗng



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

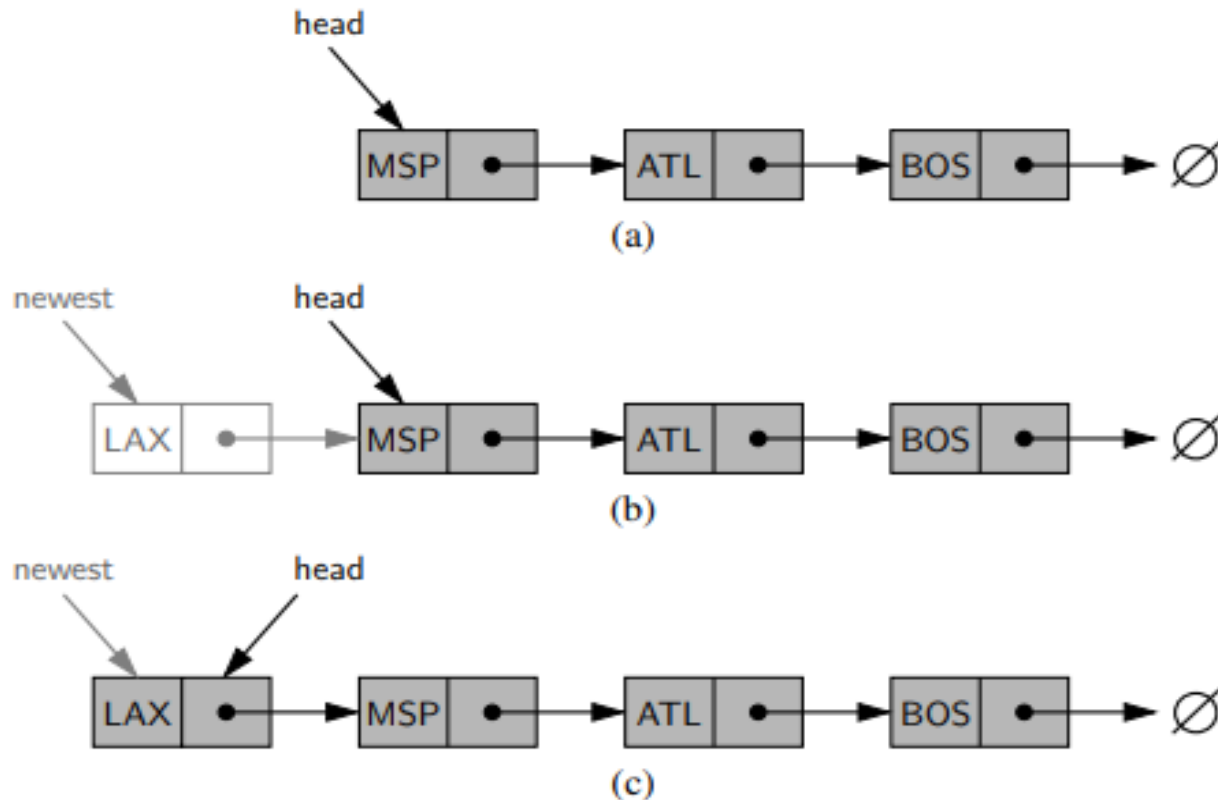
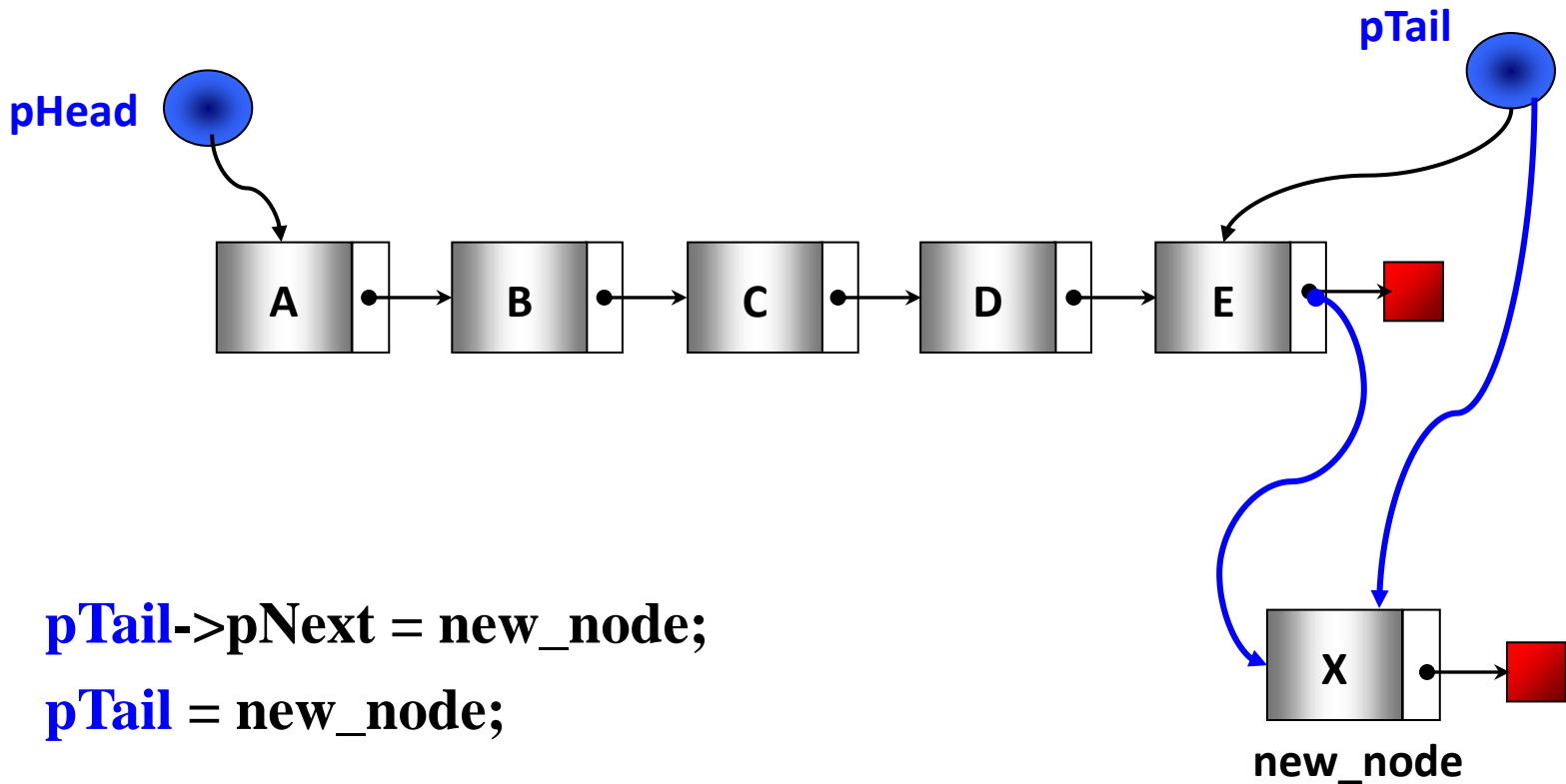


Figure 7.4: Insertion of an element at the head of a singly linked list: (a) before the insertion; (b) after creation of a new node; (c) after reassignment of the head reference.

DSLK đơn – Các thao tác cơ sở

32

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



DSLK đơn – Các thao tác cơ sở

33

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

```
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

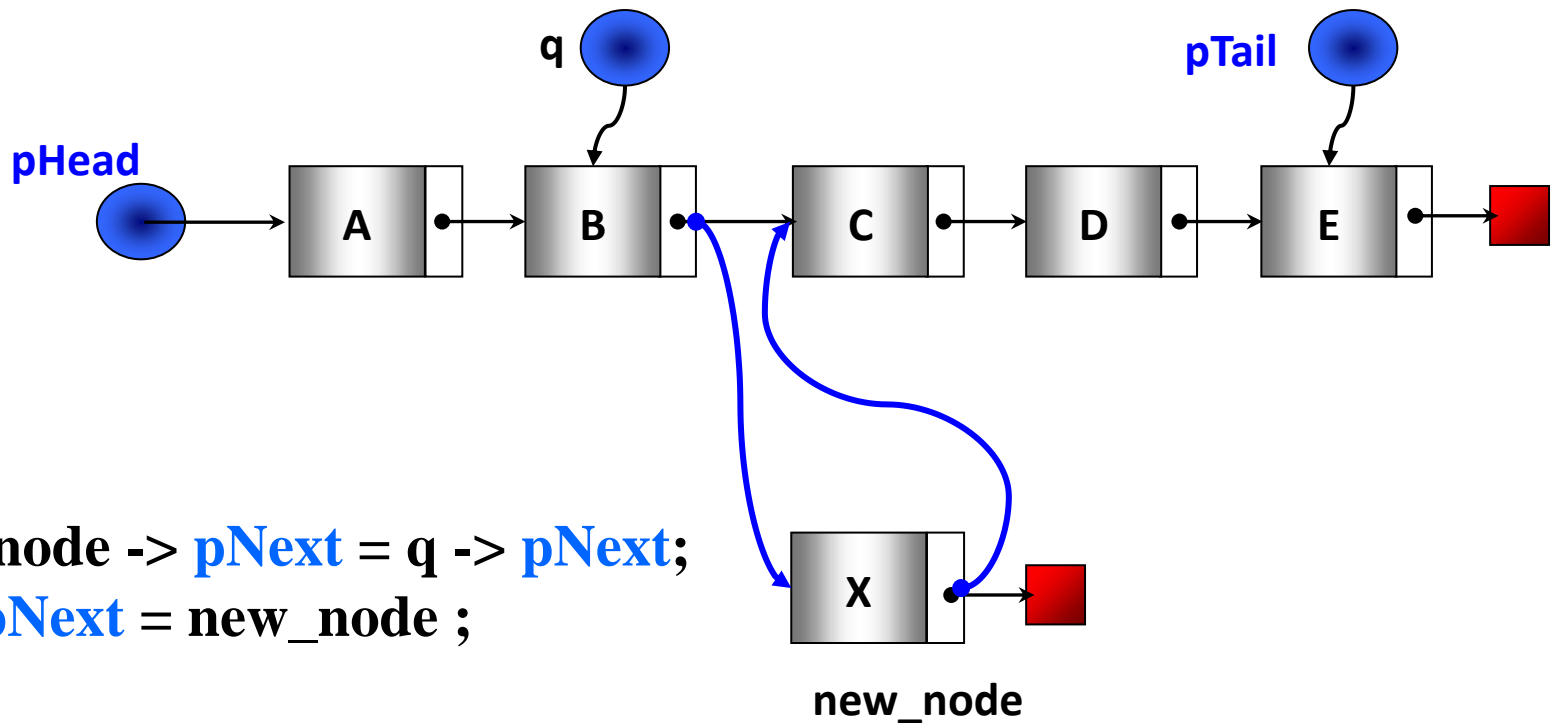
Ví dụ:

```
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

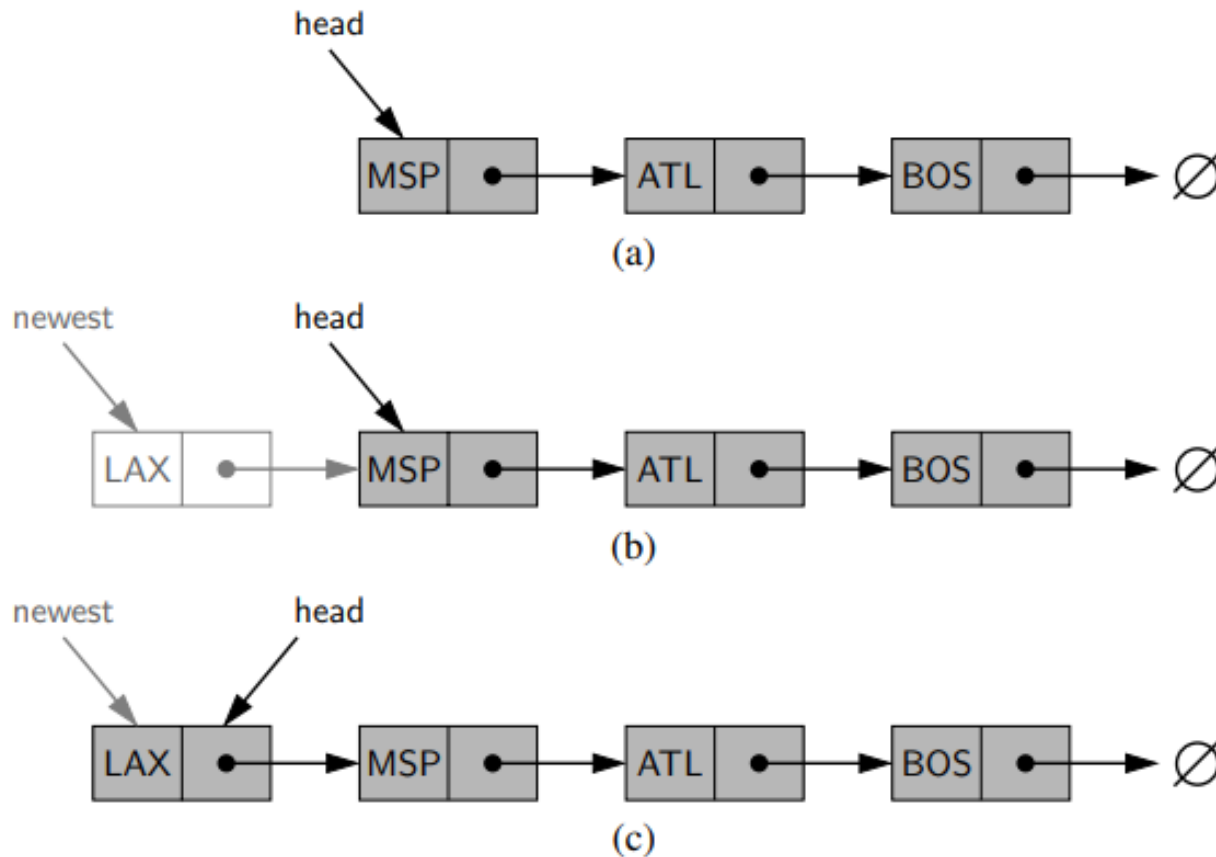


Figure 7.4: Insertion of an element at the head of a singly linked list: (a) before the insertion; (b) after creation of a new node; (c) after reassignment of the head reference.

DSLK đơn – Các thao tác cơ sở

38

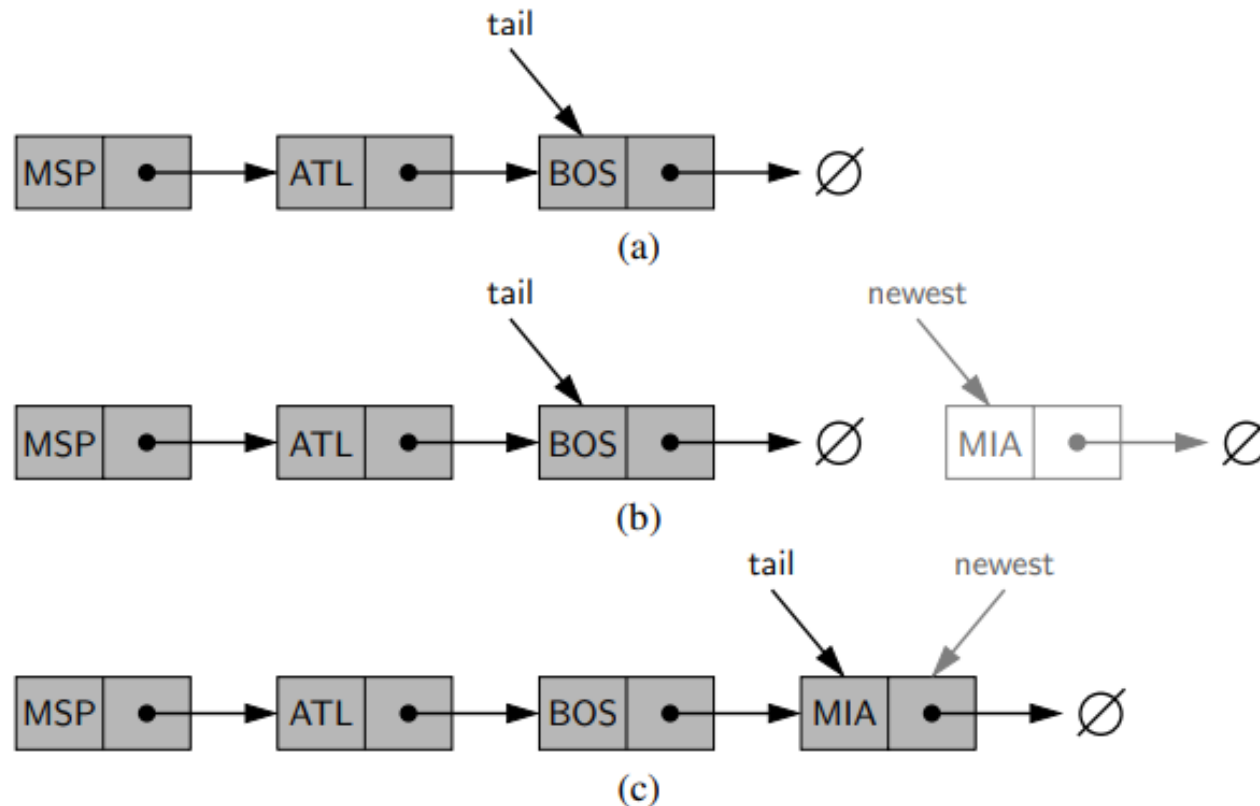


Figure 7.5: Insertion at the tail of a singly linked list: (a) before the insertion; (b) after creation of a new node; (c) after reassignment of the tail reference. Note that we must set the next link of the tail in (b) before we assign the tail variable to point to the new node in (c).

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

```
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()**

```
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;  
}
```

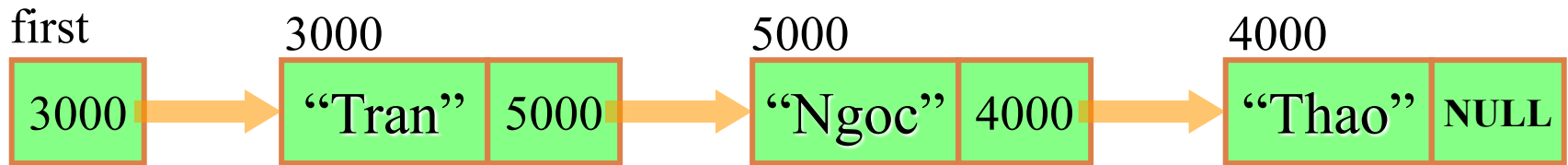
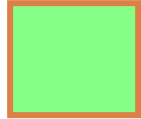


**Chuyển
thành vòng
lặp for??**

DSLK – Minh họa in danh sách

45

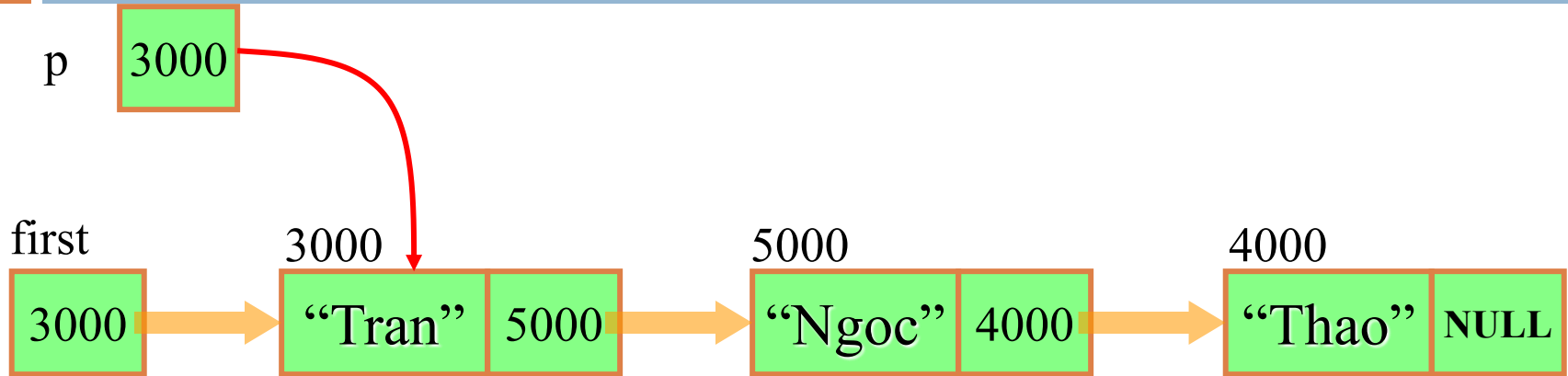
p



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

DSLK – Minh họa in danh sách

46



```
p = first;
```

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

```
{
```

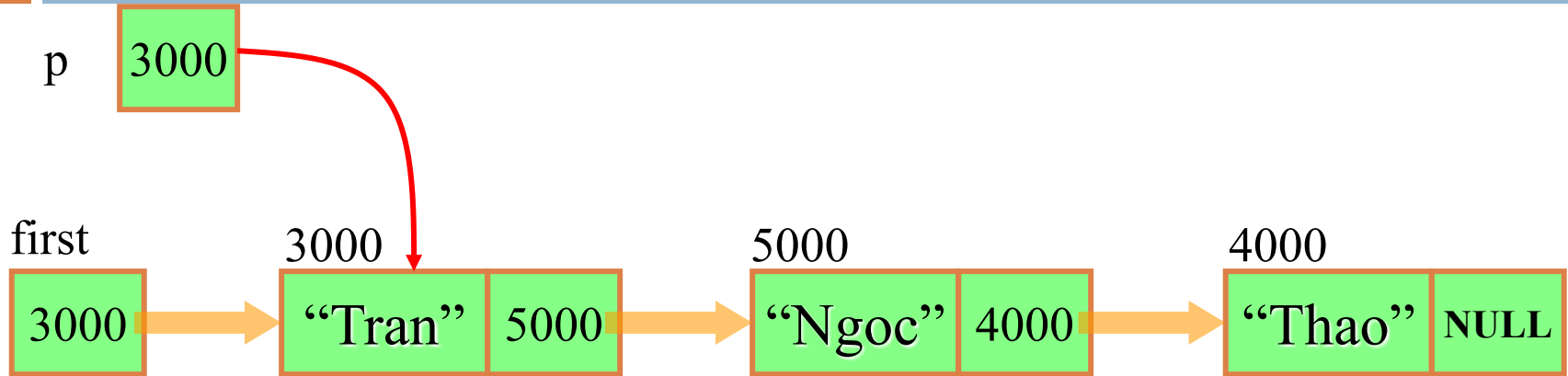
```
    printf("%d\t",p->data);
```

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

```
}
```

DSLK – Minh họa in danh sách

47



```
p = first;
```

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

```
{
```

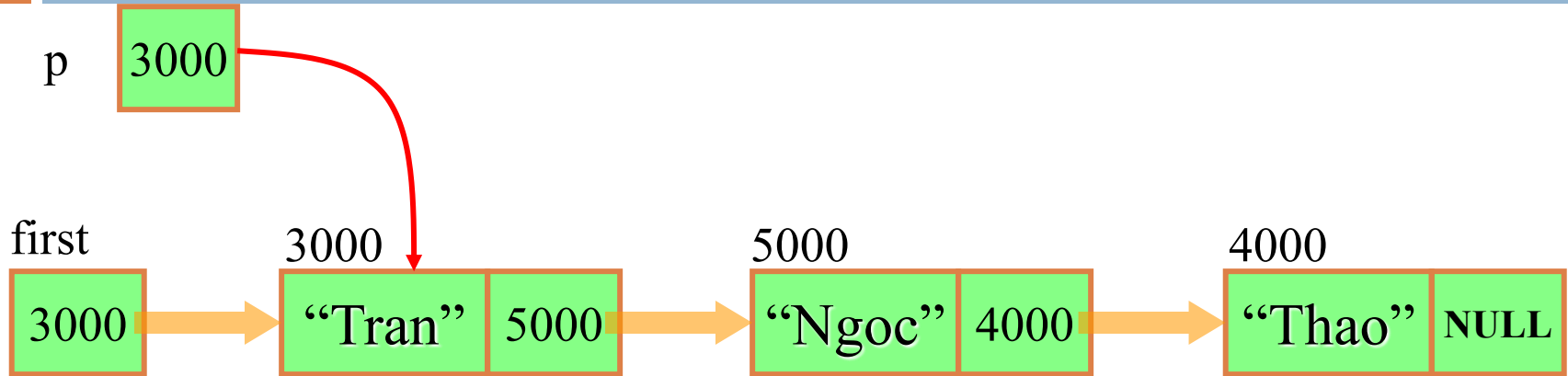
```
    printf("%d\t",p->data);
```

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

```
}
```

DSLK – Minh họa in danh sách

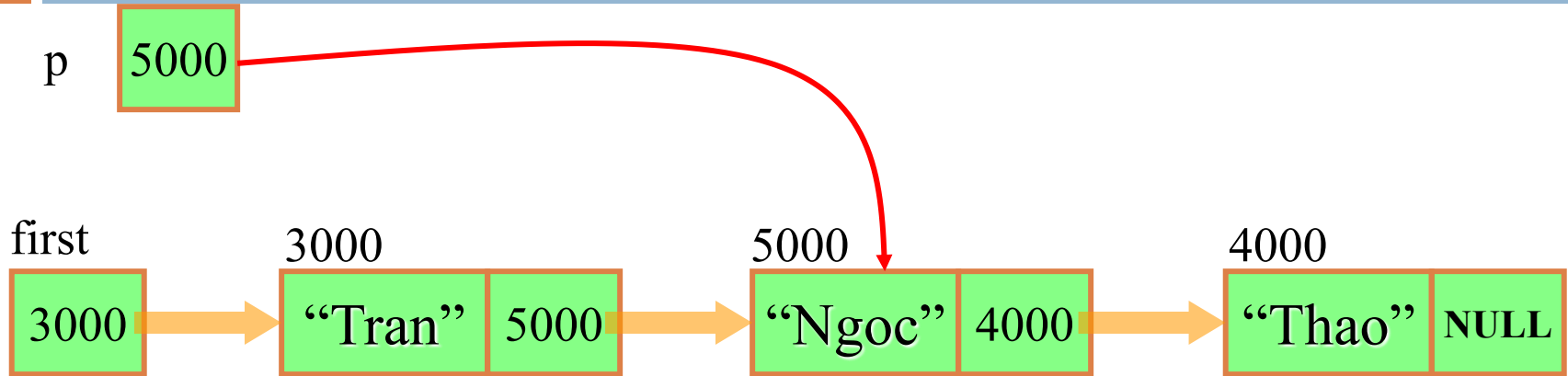
48



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


DSLK – Minh họa in danh sách

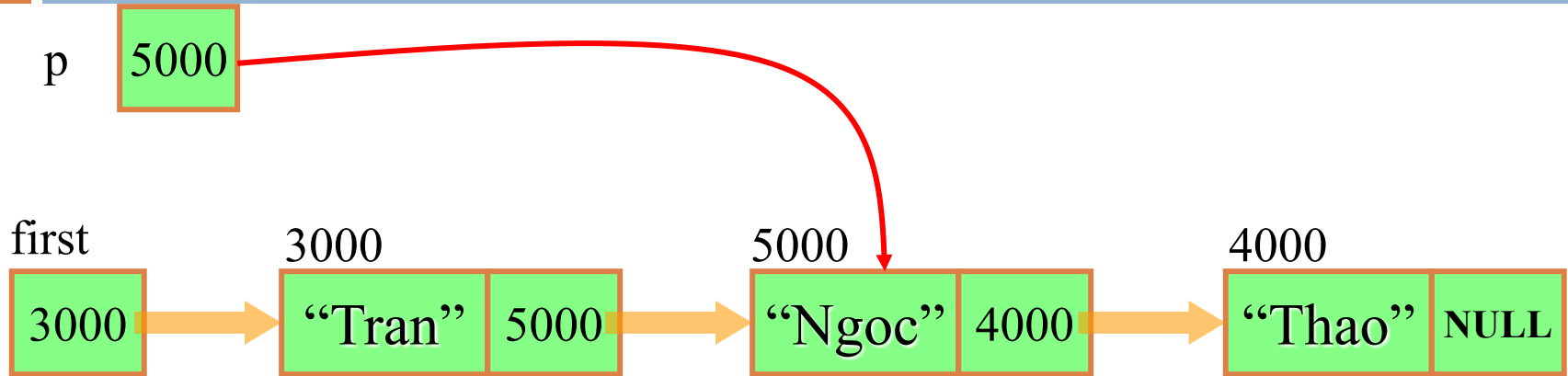
49



```
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)
```

```
{
```

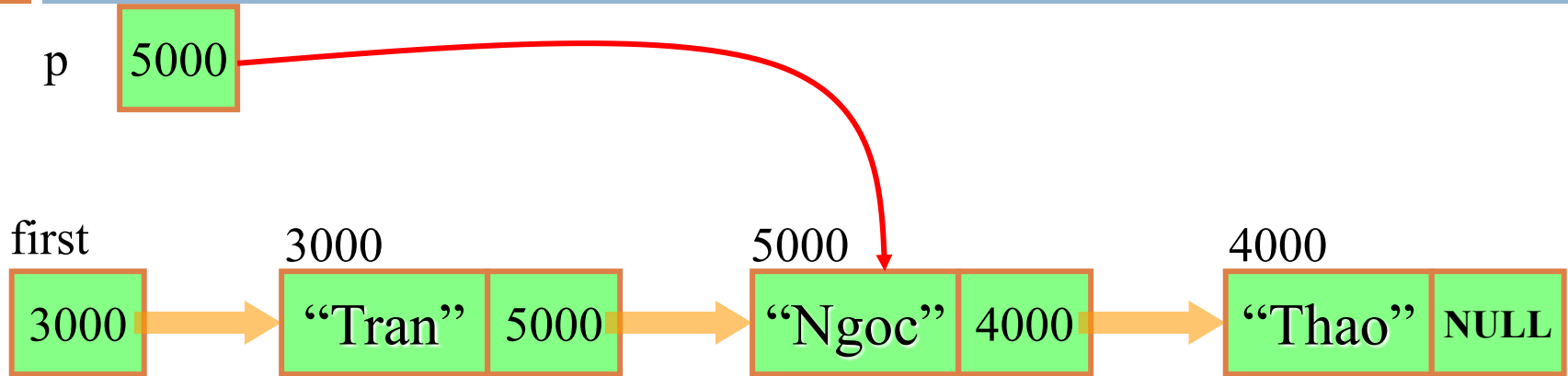
```
    printf("%d\t",p->data);
```

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

```
}
```

DSLK – Minh họa in danh sách

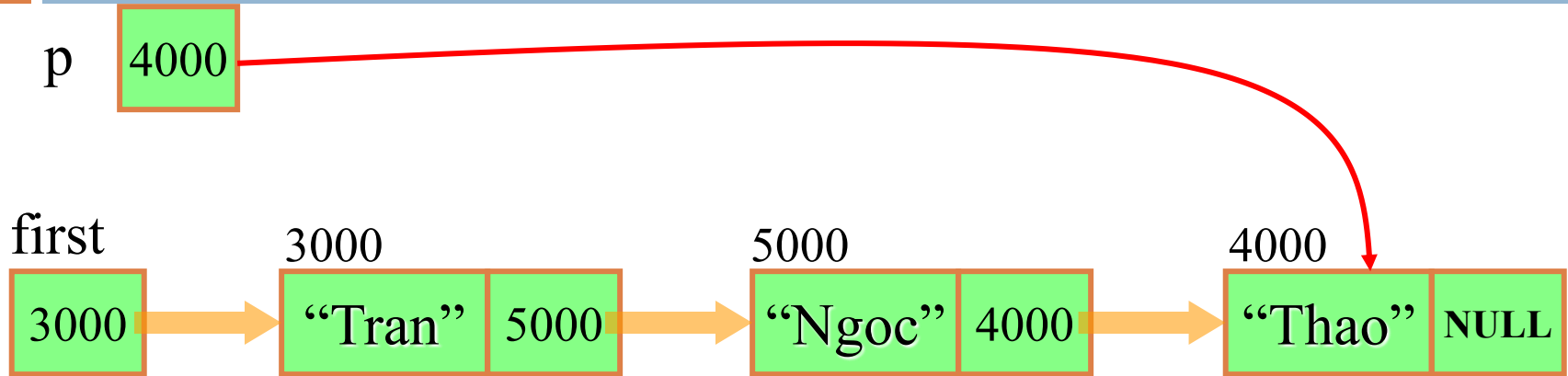
51



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

DSLK – Minh họa in danh sách

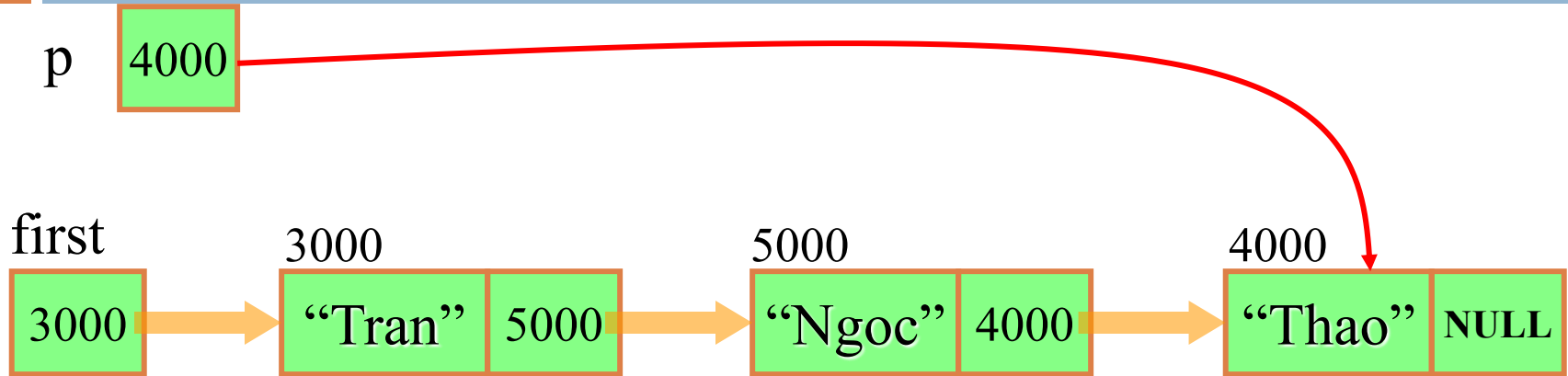
52



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

DSLK – Minh họa in danh sách

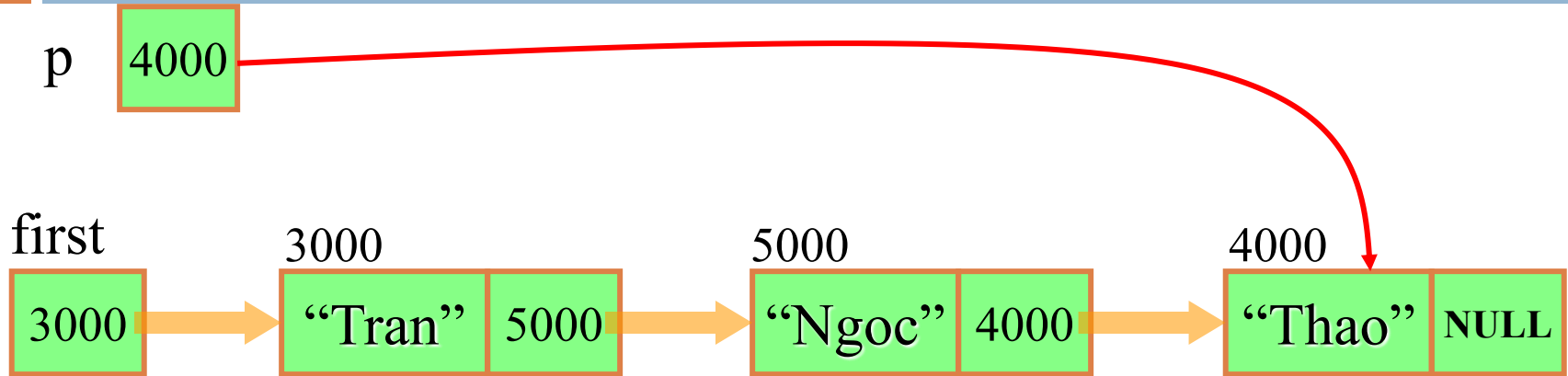
53



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

DSLK – Minh họa in danh sách

54



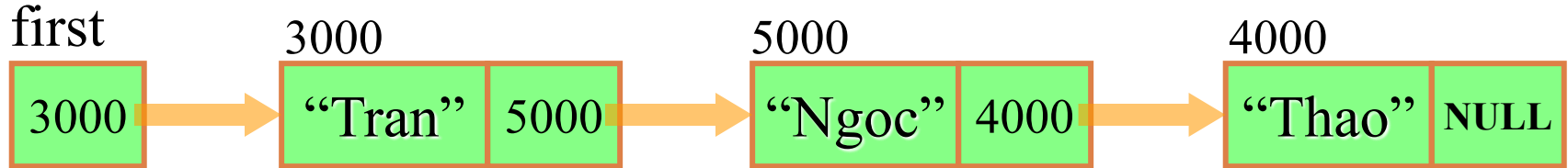
```
p = first;  
while (p!=NULL)  
{  
    printf("%d\t",p->data);  
    p = p->link;  
}
```

DSLK – Minh họa in danh sách

55

p

NULL



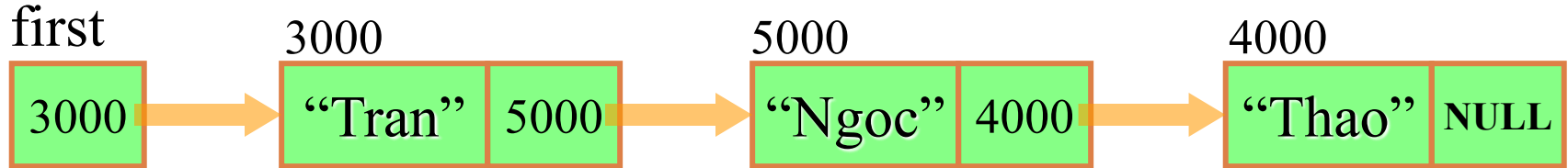
```
p = first;
while (p!=NULL)
{
    printf("%d\t",p->data);
    p = p->link;
}
```

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

```
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<<endl;
        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()

```
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

```
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()**

```
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

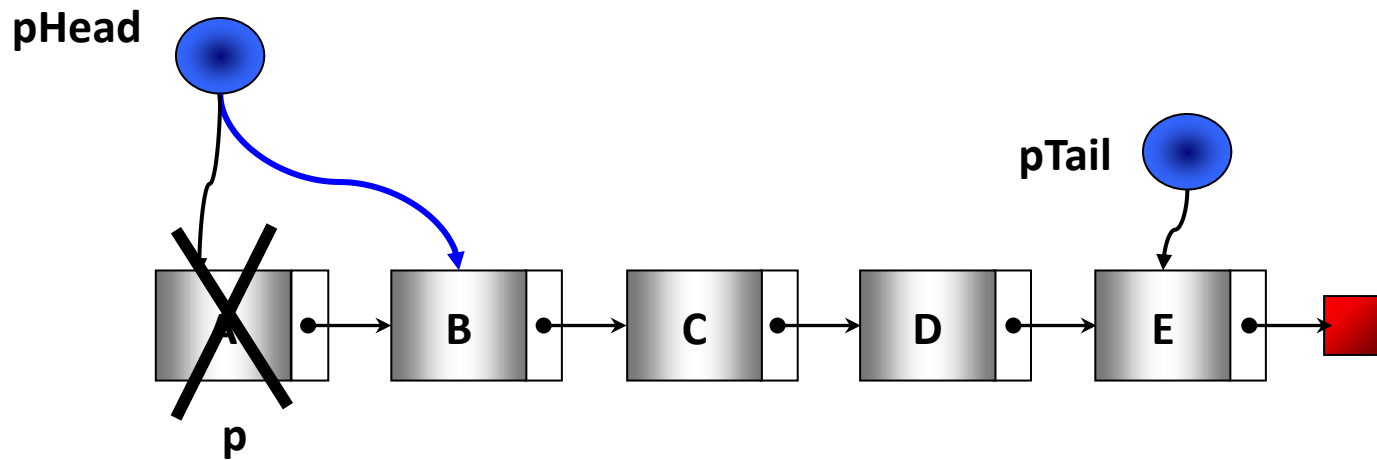
Thuật toán: Xóa node đầu danh sách

- ❑ 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

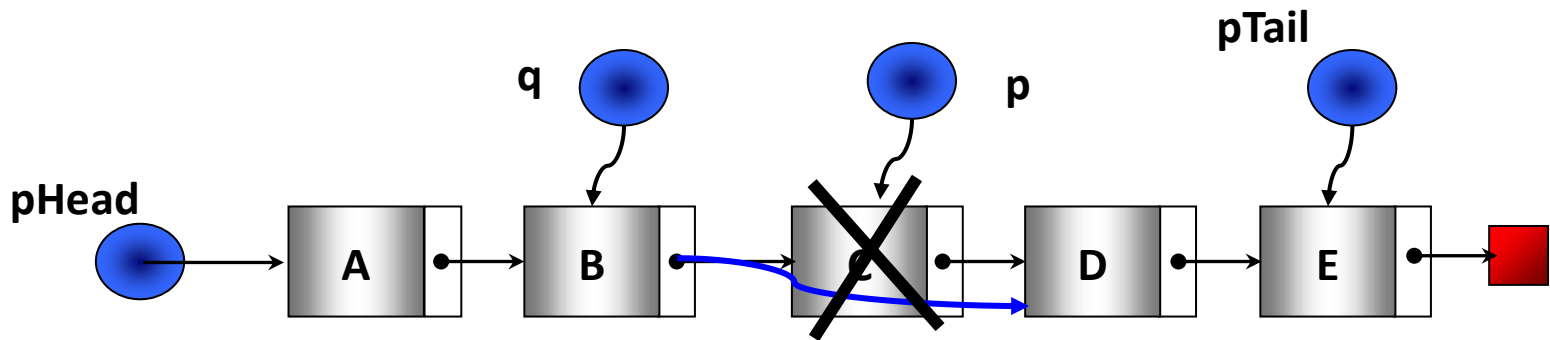
Thuật toán: Xóa node sau node **q** trong danh sách:

- ▣ 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 **pTail** 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

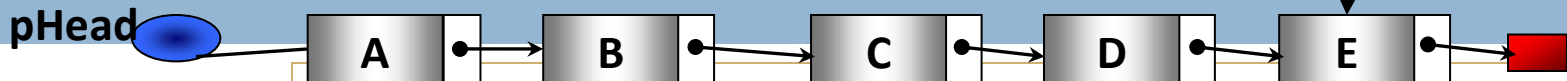
74

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



- 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

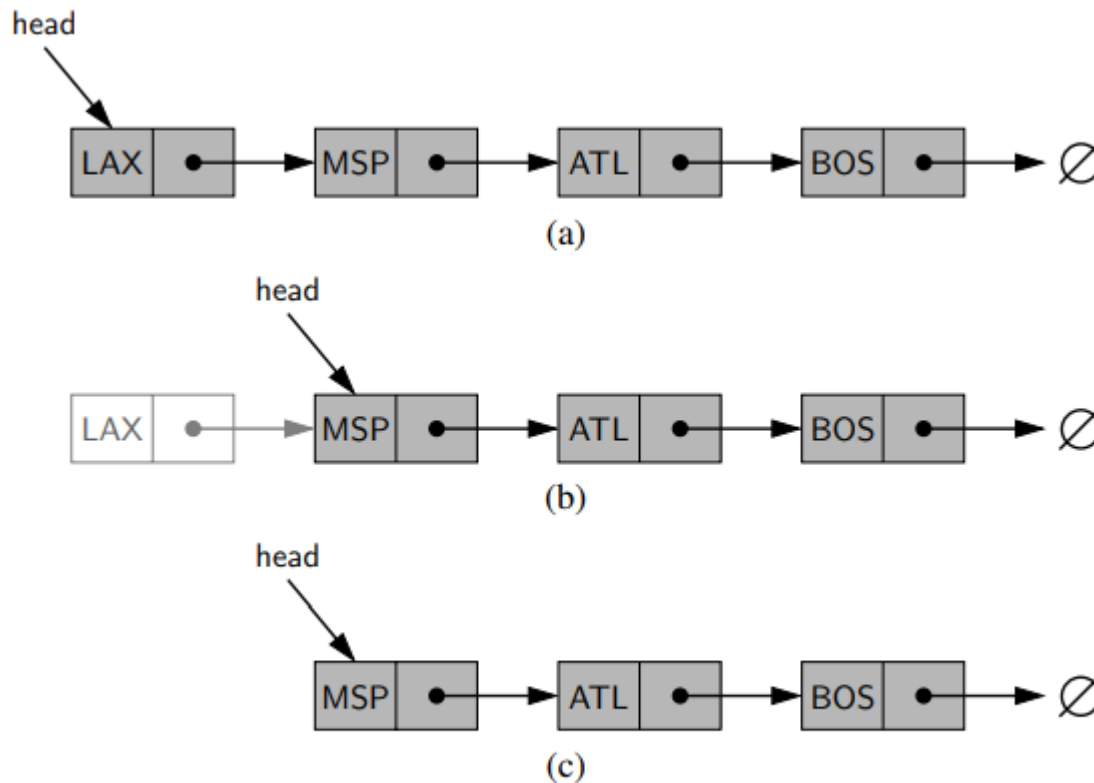


Figure 7.6: Removal of an element at the head of a singly linked list: (a) before the removal; (b) after “linking out” the old head; (c) final configuration.

```
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.duoi:
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
109         #def
```

DSLK đơn

78

□ 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

79

□ 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
```

DSLK đơn

80

□ Gọi Hàm cập nhập danh sách

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


DSLK đơn – Các thao tác cơ sở

81

□ 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

82

- 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???

DSLK đơn

83

□ 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
```

DSLK đơn

84

□ 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

85

- 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

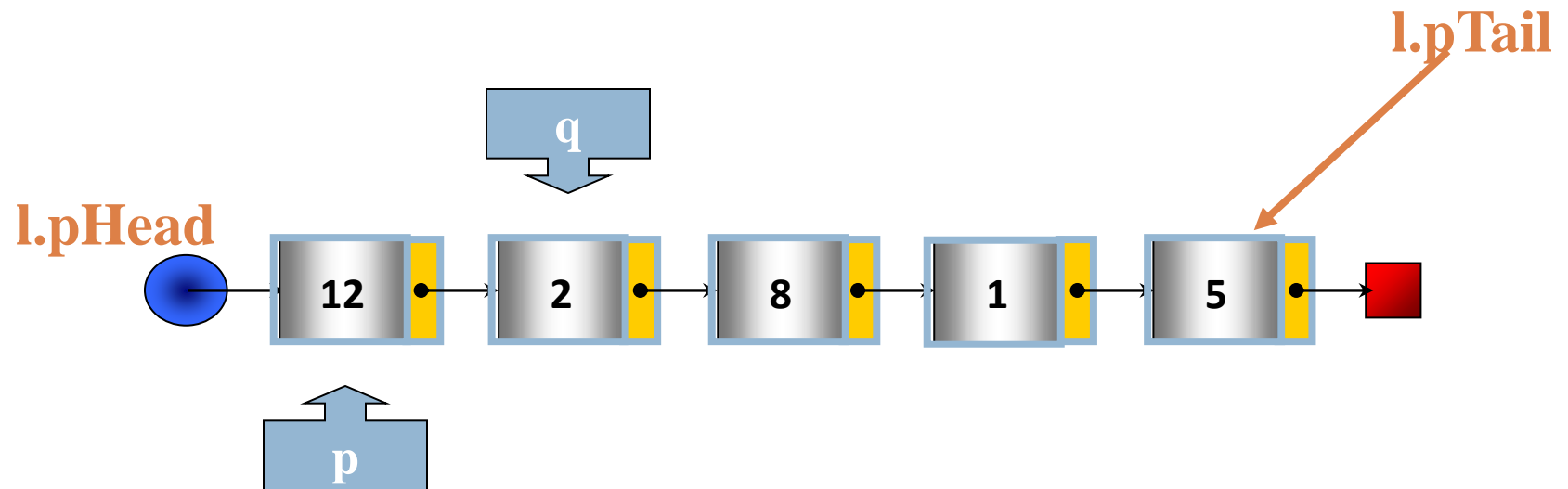
86

- 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);
}
```

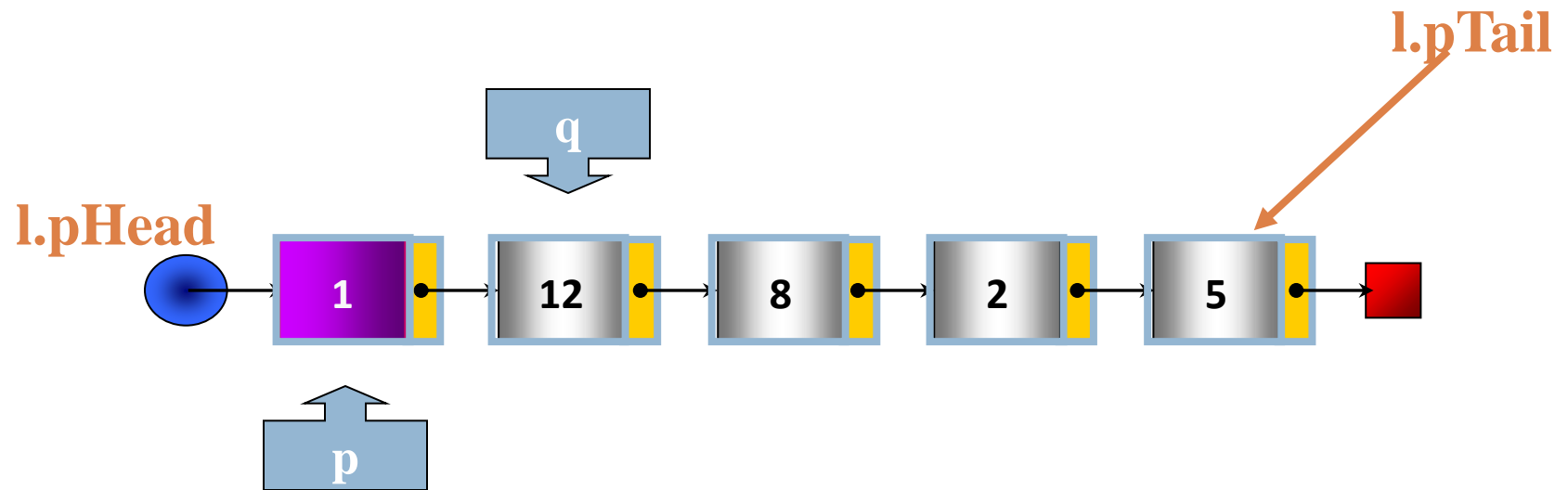
Sắp xếp trên DSLK đơn: Hoán vị data

87



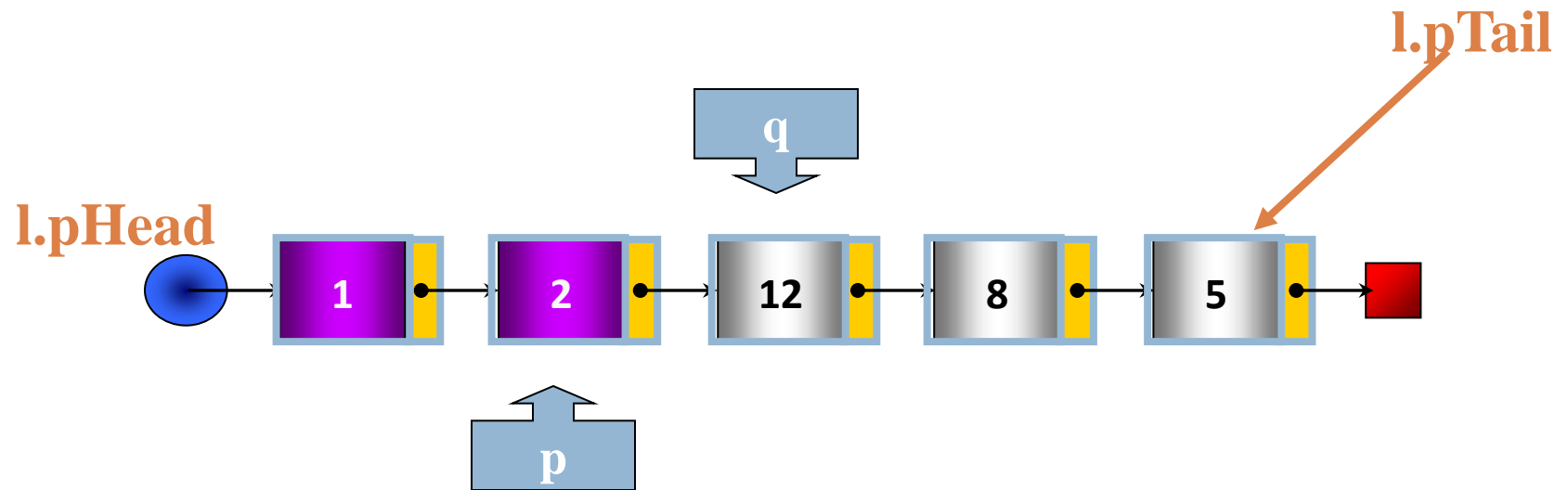
Sắp xếp trên DSLK đơn: Hoán vị data

88



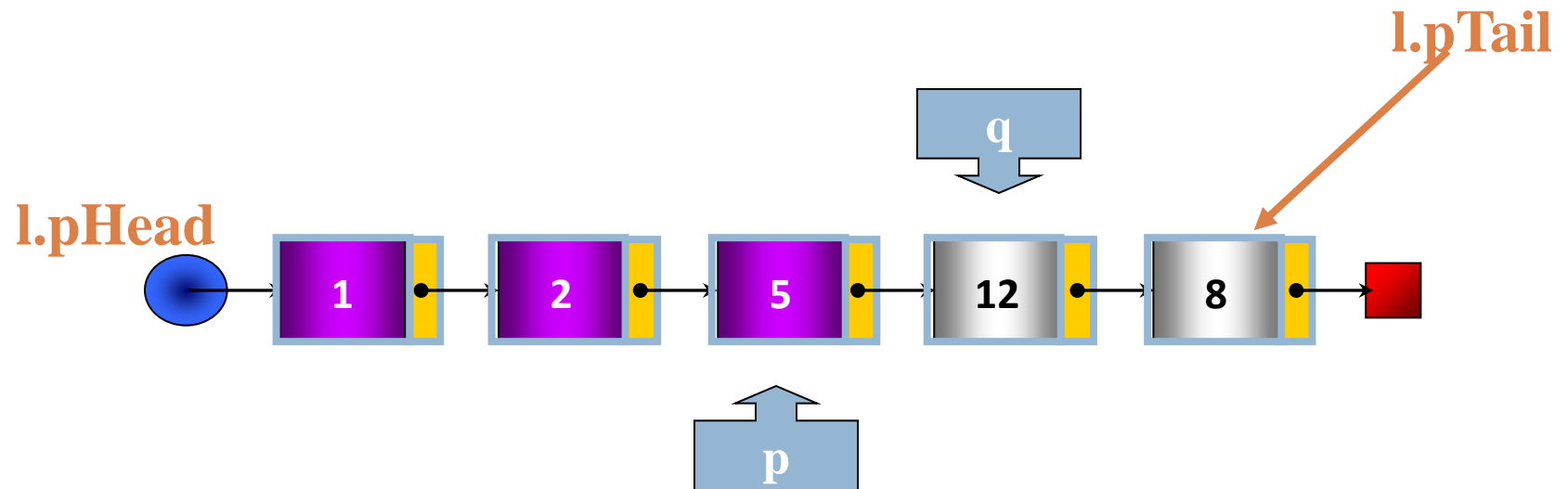
Sắp xếp trên DSLK đơn: Hoán vị data

89



Sắp xếp trên DSLK đơn: Hoán vị data

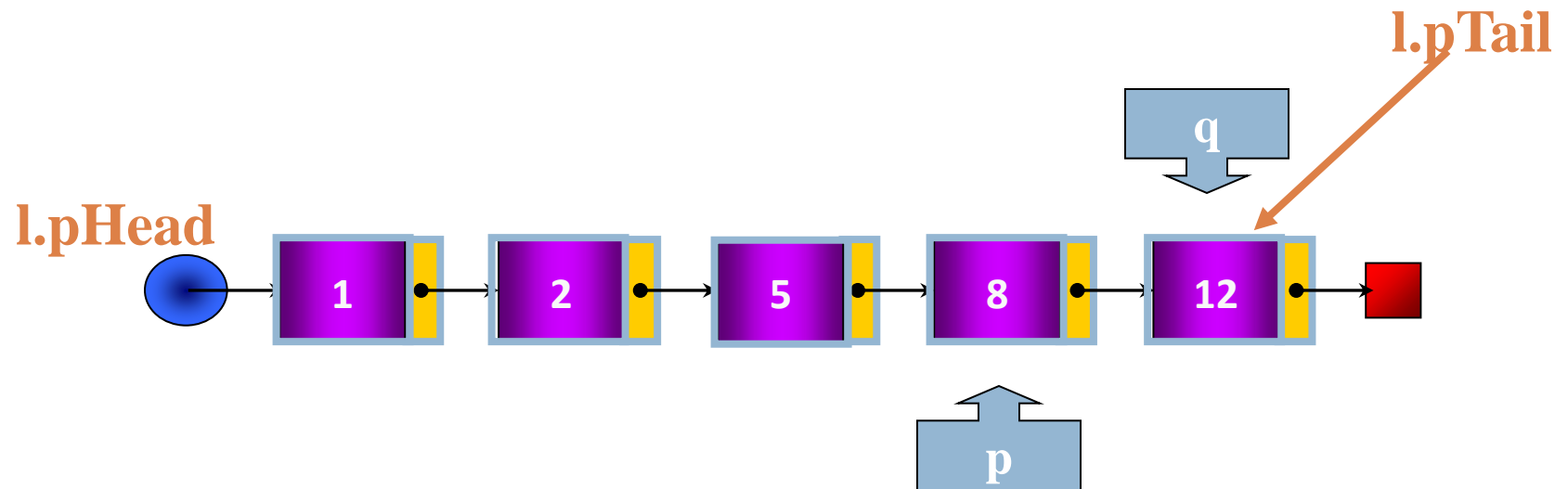
90



Sắp xếp trên DSLK đơn: Hoán vị data

91

Dừng



Sắp xếp bằng pp chọn trực tiếp

(*Selection sort*)

92

```
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

93

- 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 I;
 - ▣ Bước 3: Tách min khỏi danh sách I;
 - ▣ Bước 4: Chèn min vào cuối danh sách Result;
 - ▣ Bước 5: Lặp lại bước 2 khi chưa hết danh sách cũ I;

```

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;
        }
    }
}

```

```

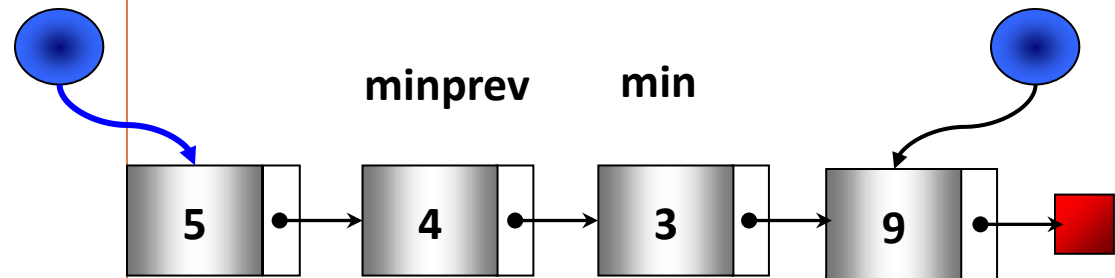
if ( minprev != NULL )
    minprev->pNext = min->pNext;
else
    l.pHead = min->pNext;

min->pNext = NULL;

addTail( lResult, min );
}

l = lResult;
}

```



Sắp xếp Thay đổi các mối liên kết

95

- 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.

Quick Sort : Thuật toán

96

//input: xâu (first, last)

//output: xâu đã được sắp tăng dần

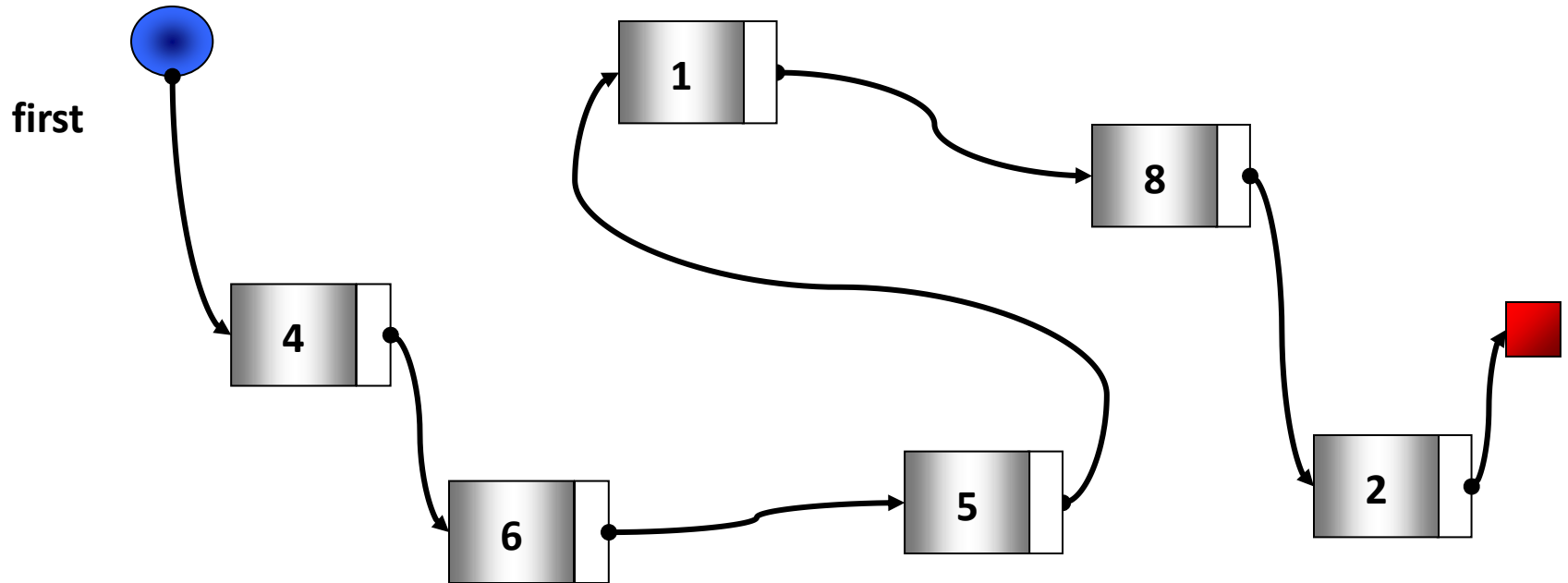
- Bước 1: Nếu xâu có ít hơn 2 phần tử

Dừng; //xâu đã có thứ tự

- Bước 2: Chọn X là phần tử đầu xâu L làm ngưỡng. Trích X ra khỏi L.
- Bước 3: Tách xâu L ra làm 2 xâu 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 xâu L đã được sắp xếp.

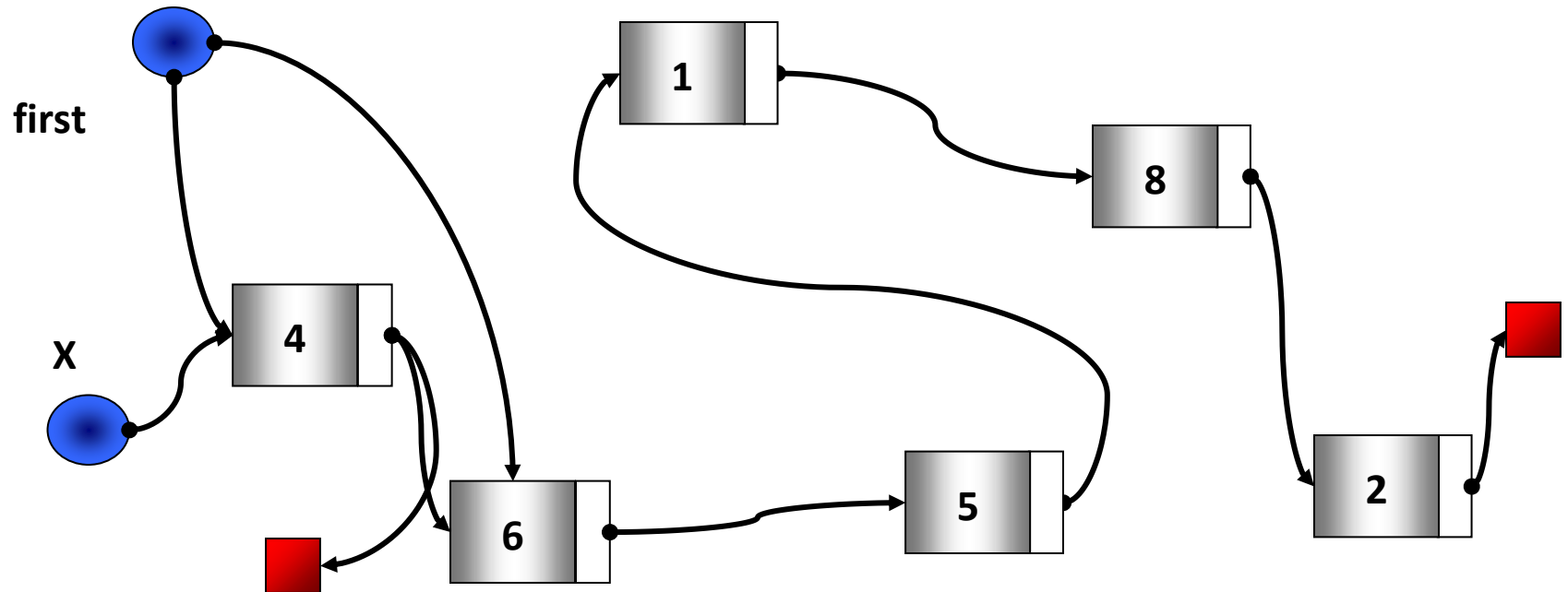
Sắp xếp quick sort

97



Quick sort : phân hoạch

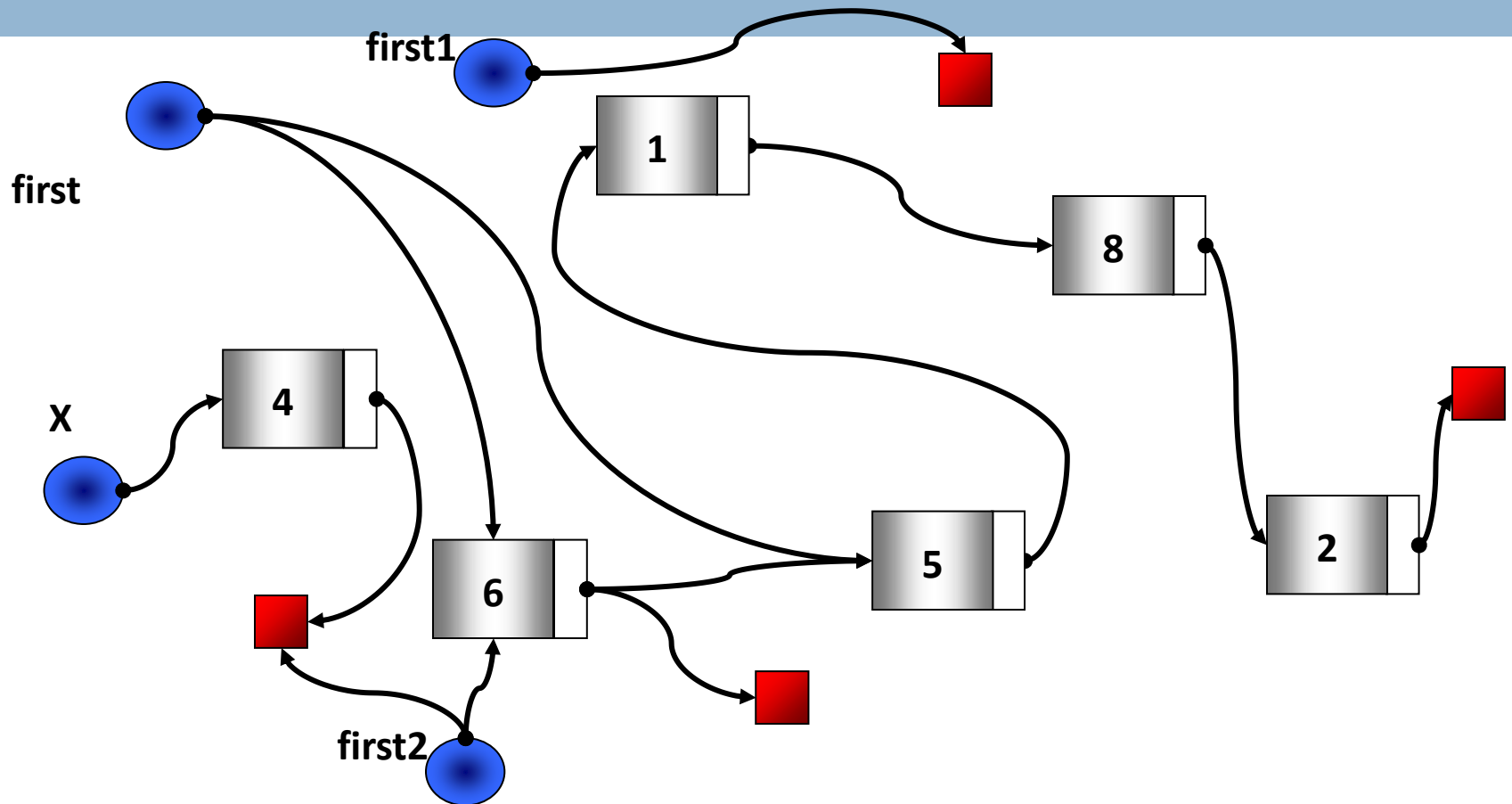
98



Chọn phần tử đầu xâu làm ngưỡng

Quick sort : phân hoạch

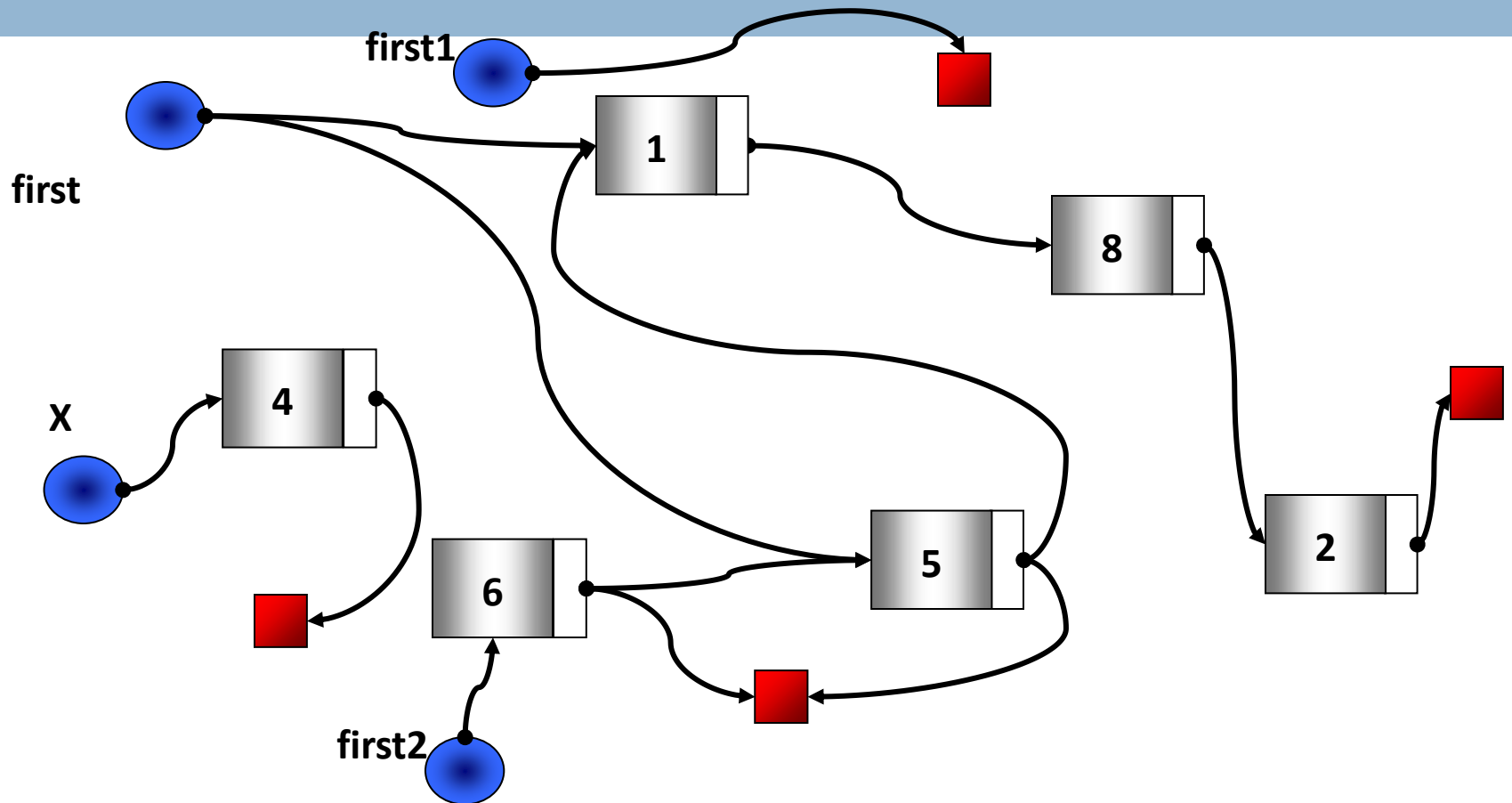
99



Tách xâu hiện hành thành 2 xâu

Quick sort : phân hoạch

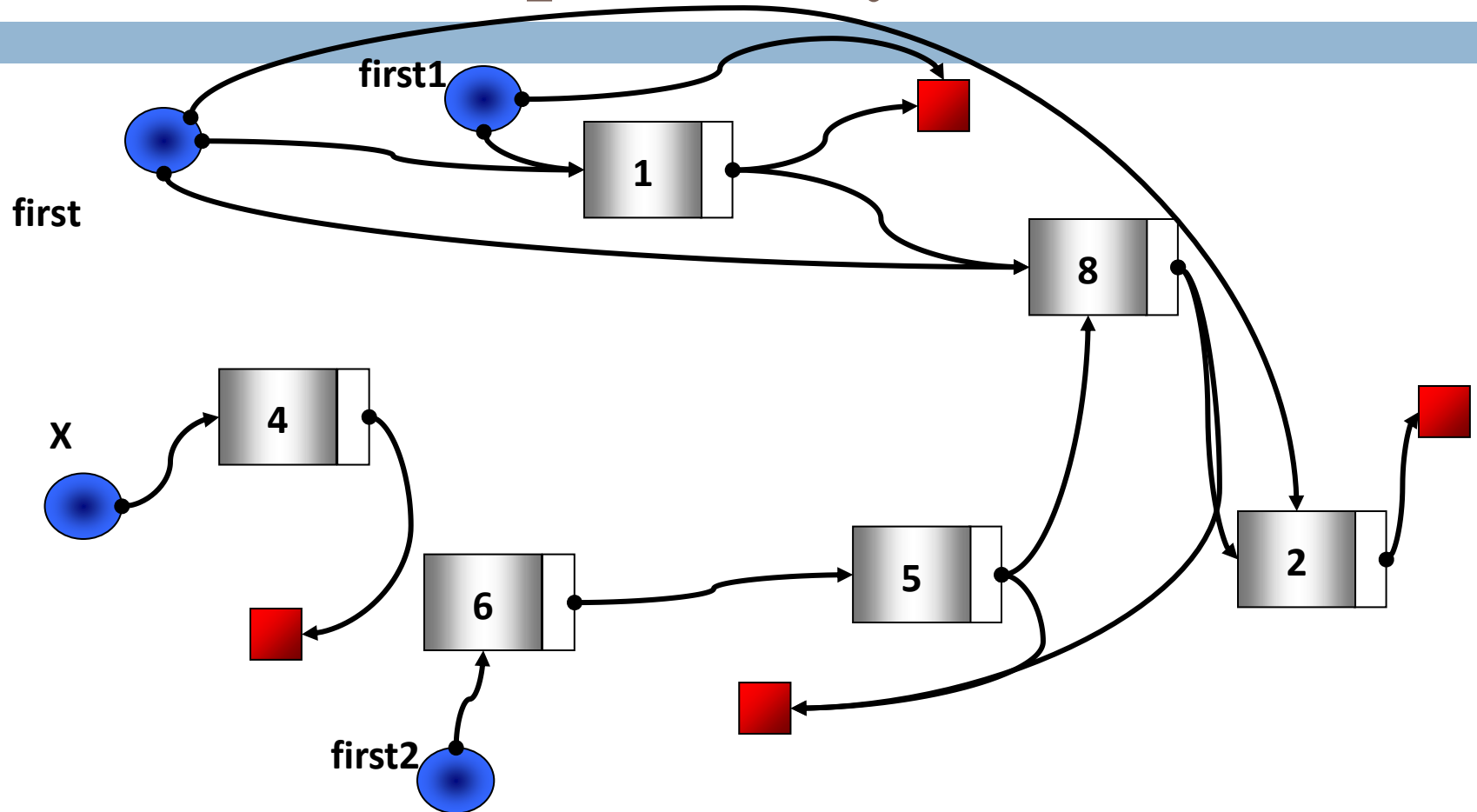
100



Tách xâu hiện hành thành 2 xâu

Quick sort : phân hoạch

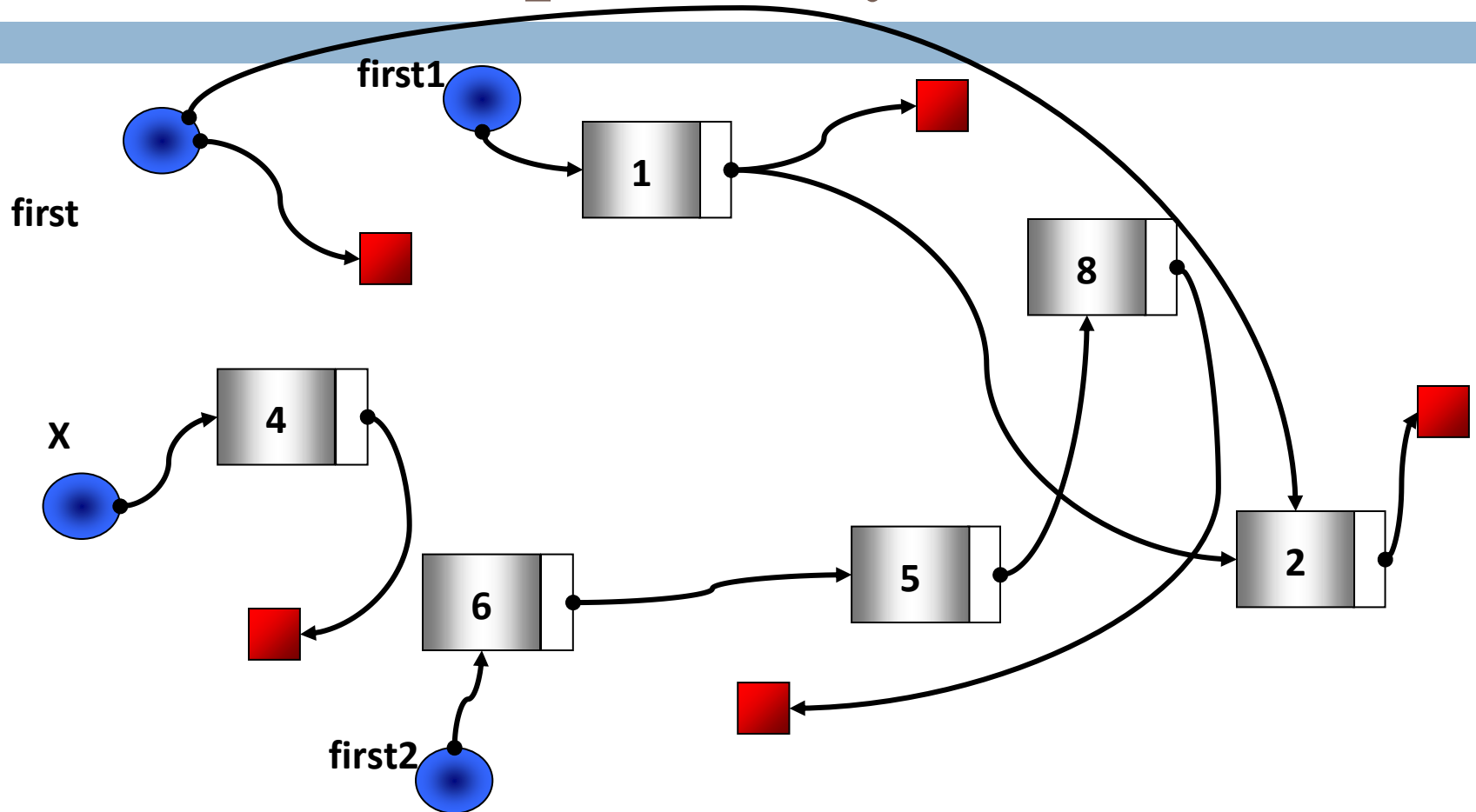
101



Tách xâu hiện hành thành 2 xâu

Quick sort : phân hoạch

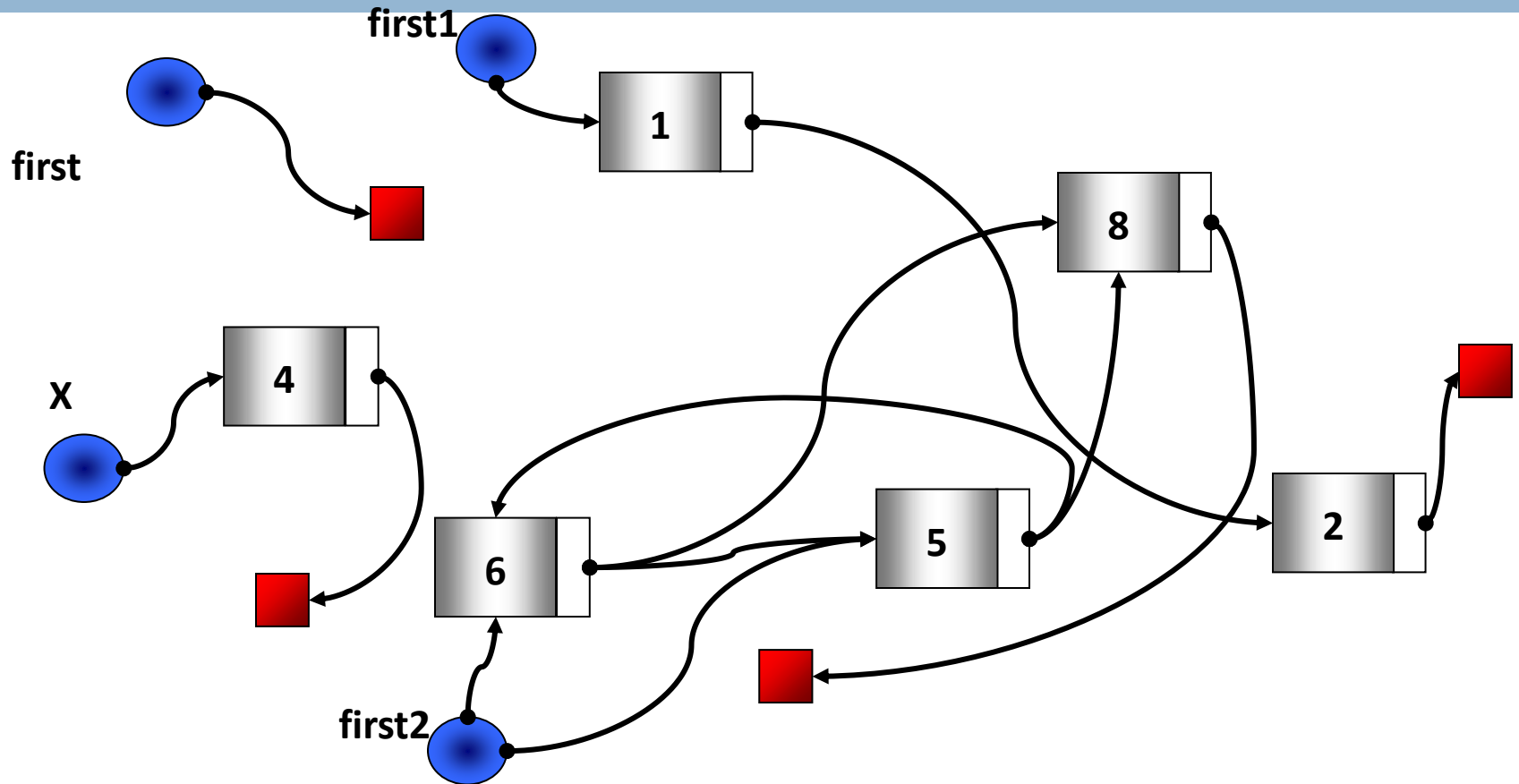
102



Tách xâu hiện hành thành 2 xâu

Quick sort

103

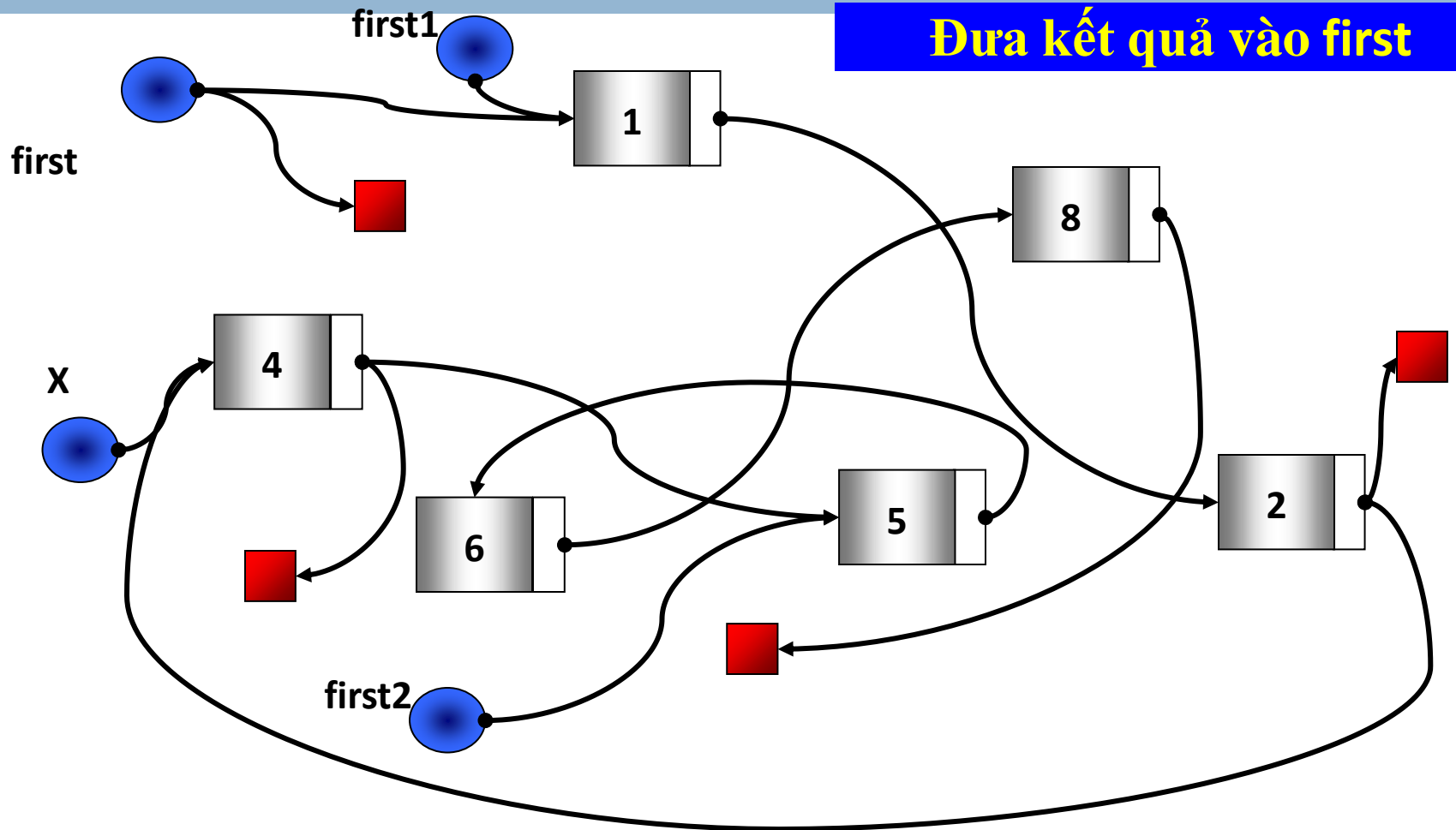


Sắp xếp các xâu l1, l2

Quick sort

104

Đưa kết quả vào first



Nối 2 danh sách

105

```
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

107

Nhận xét:

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

Bài tập

108

- 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

109

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)

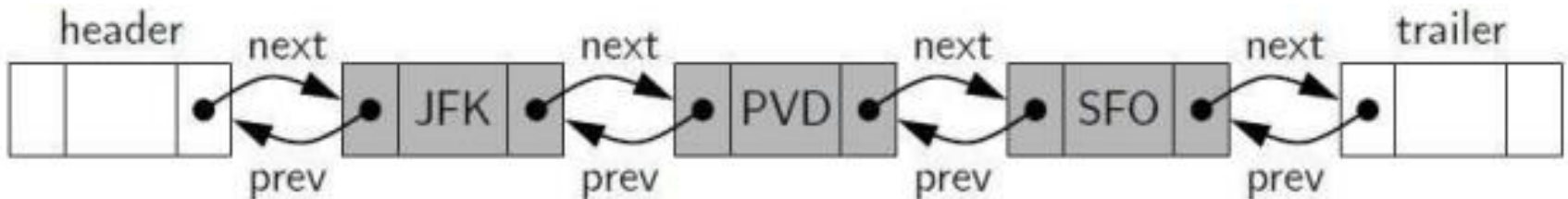
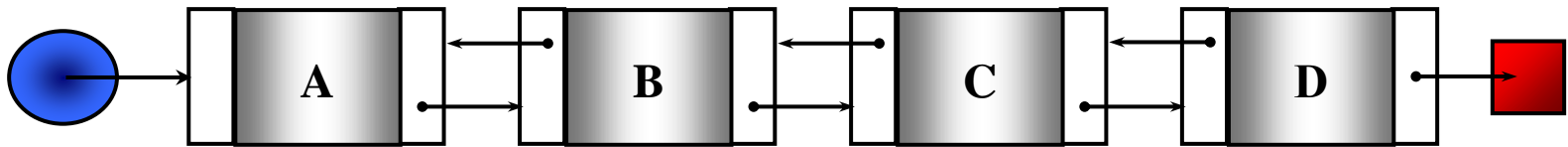
110

- 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)

111

- 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

112

□ 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

113

- 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??

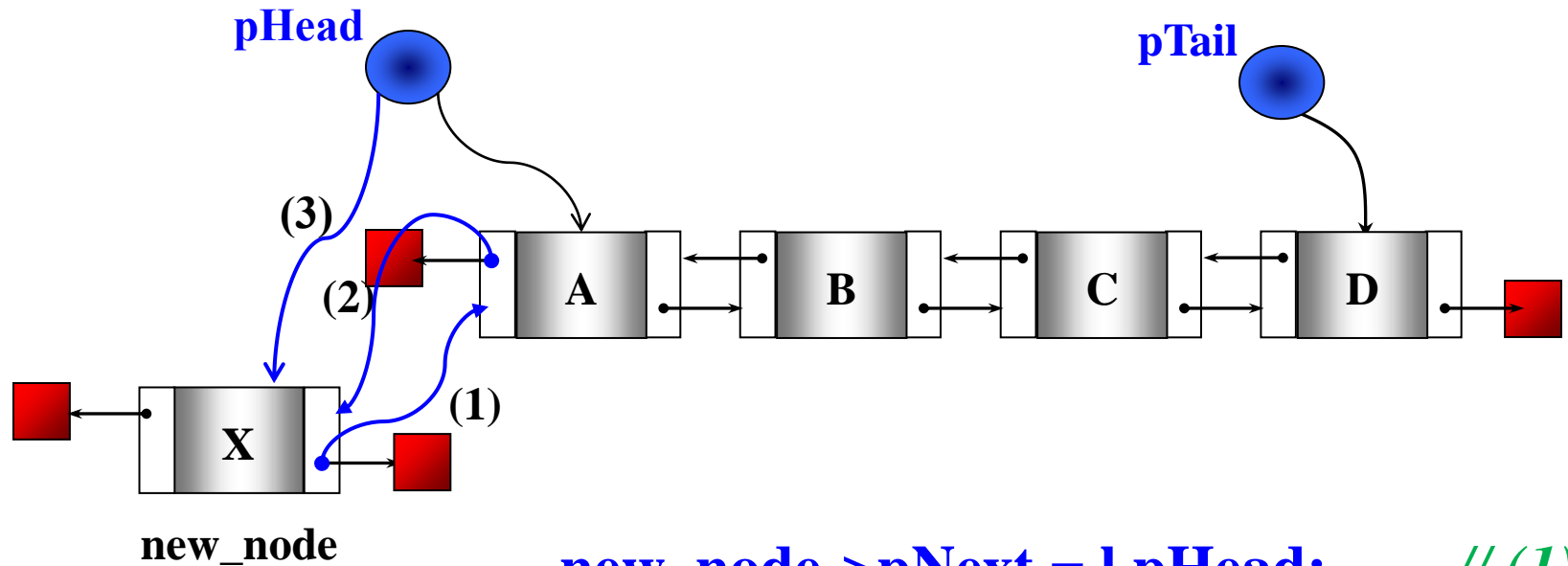
DSLK đôi – Thêm 1 nút vào ds

114

- 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

115



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

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

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

Cài đặt: Thêm vào đầu ds

116

```
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)

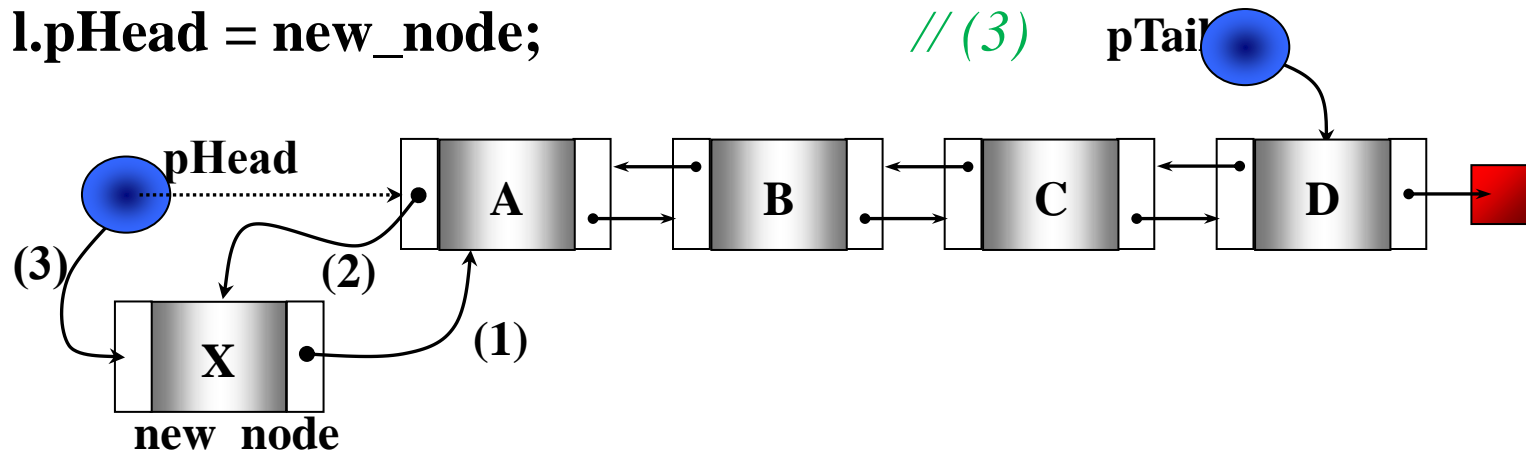
```
        l.pHead = new_node;
```

// (3)

```
    }
```

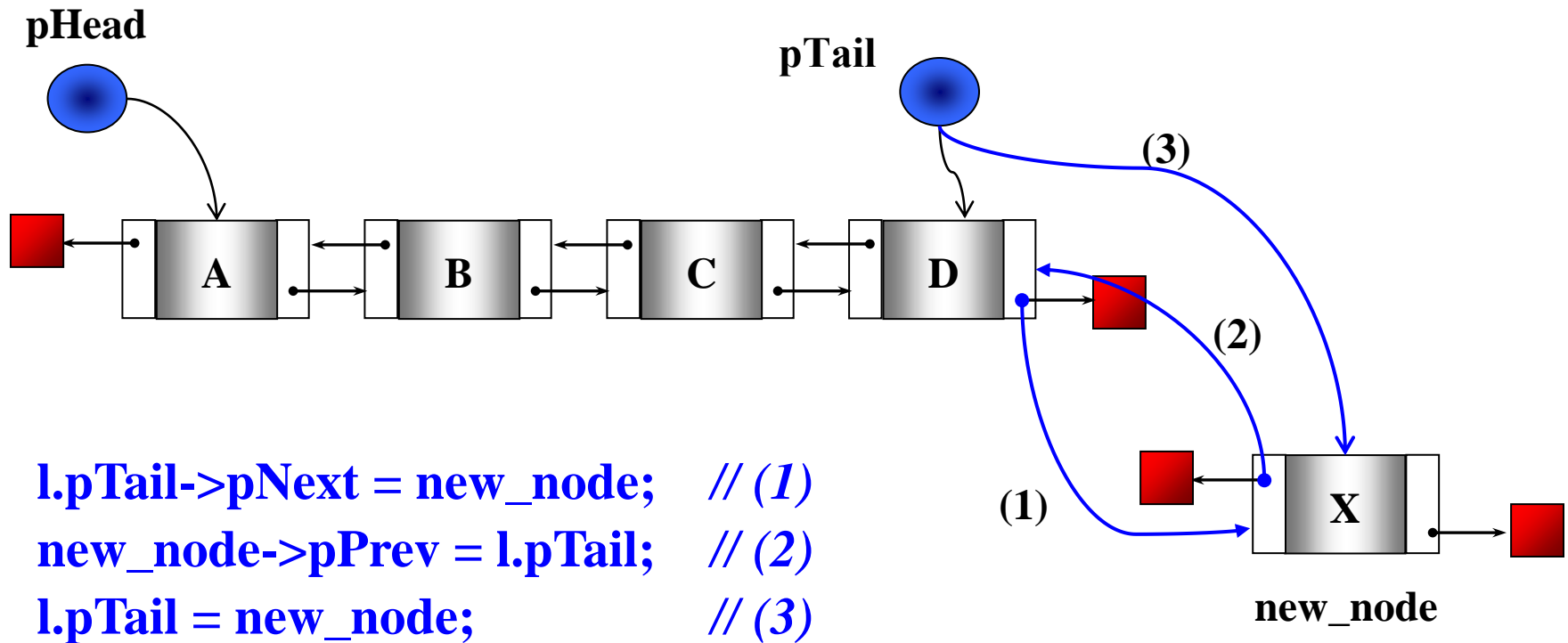
```
}
```

Gọi hàm??



Minh họa: Thêm vào cuối ds

117



```
l.pTail->pNext = new_node; // (1)  
new_node->pPrev = l.pTail; // (2)  
l.pTail = new_node;       // (3)
```

Cài đặt – Thêm vào cuối ds

118

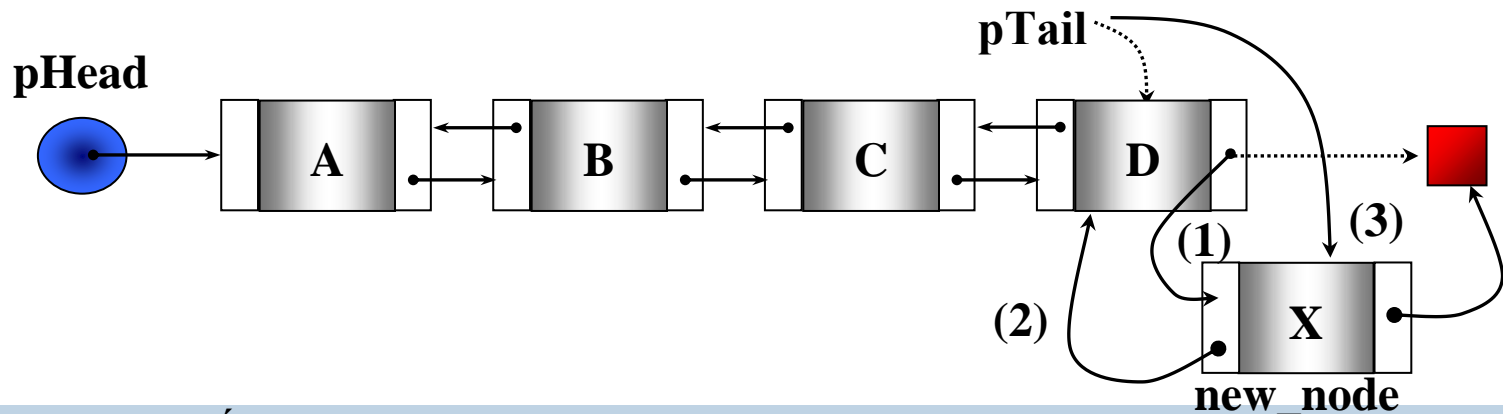
```
void addTail ( DList &l, DNode *new_node )  
{  
    if ( l.pHead==NULL )  
        l.pHead = l.pTail = new_node;  
    else  
    {  
        l.pTail->pNext = new_node;  
        new_node->pPrev = l.pTail;  
        l.pTail = new_node;  
    }  
}
```

Gọi hàm??

// (1)

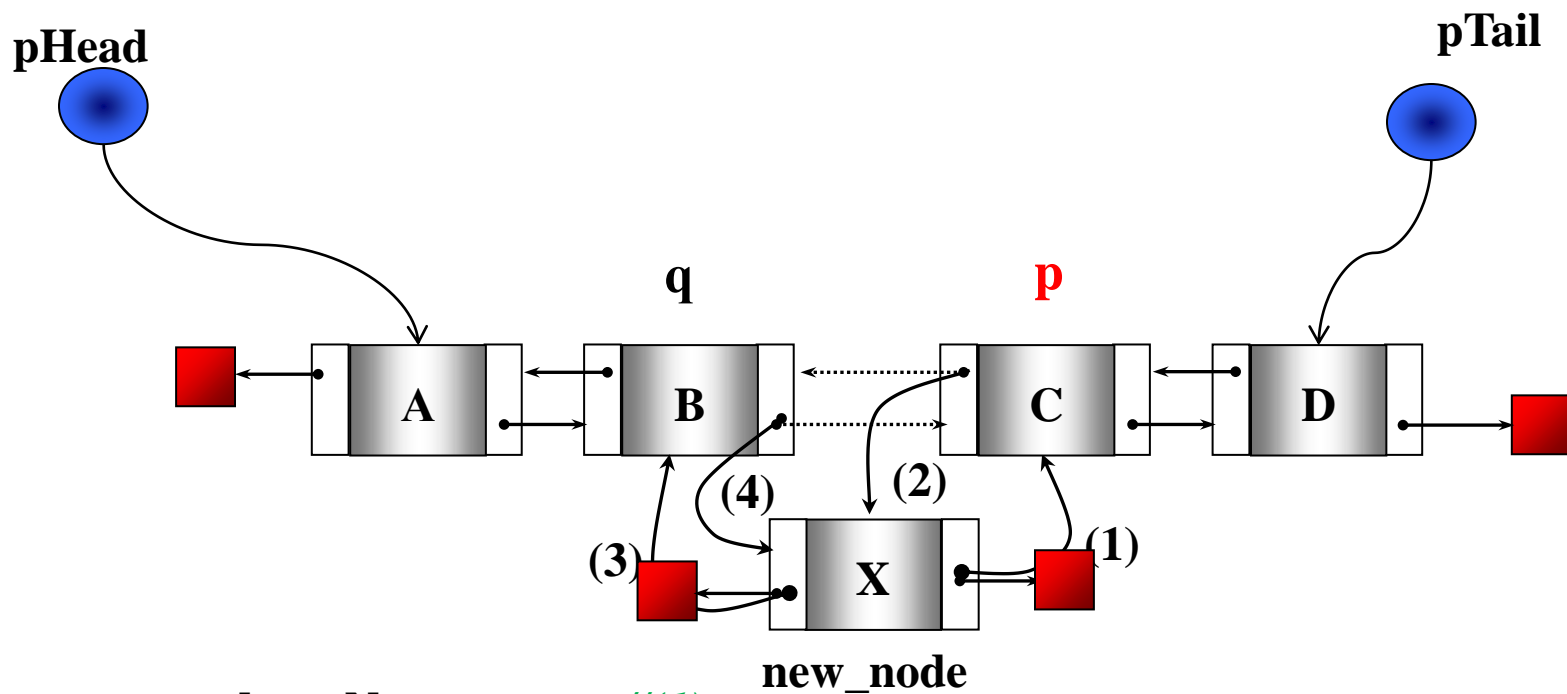
// (2)

// (3)



Minh họa: Chèn vào sau q

119



```
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

120

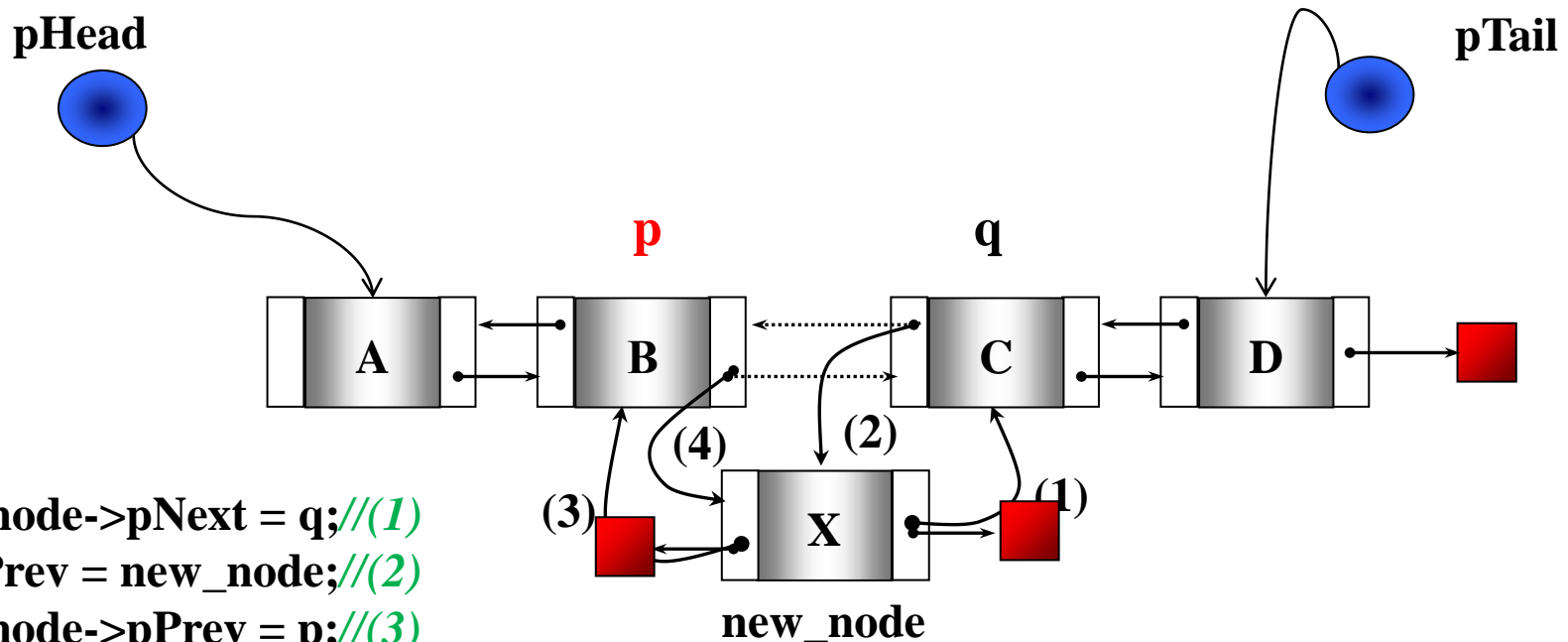
```
void addAfter (DList &l, DNode *q, DNode *new_node)
{
    DNode *p = q->pNext;
    if (q!=NULL) {
        new_node->pNext = p;           //(1)
        if ( p != NULL ) p->pPrev = new_node;   //(2)
        new_node->pPrev = q;           //(3)
        q->pNext = new_node;          //(4)
        if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



Gọi hàm??

Minh họa: Chèn vào trước q

121



`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;`

Cài đặt: Chèn vào trước q

122

```
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

123

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

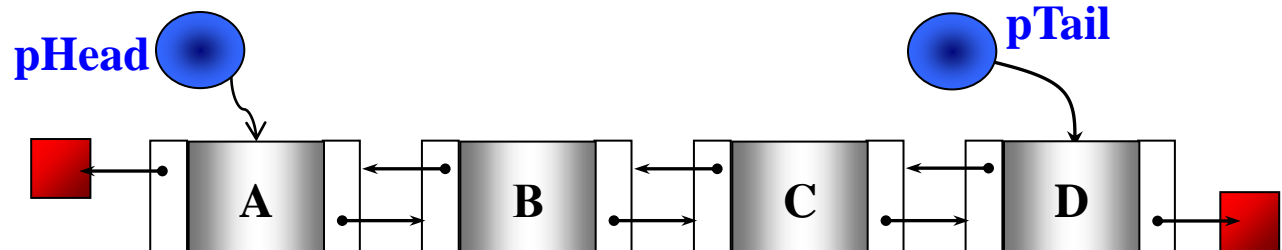
124

- 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

125

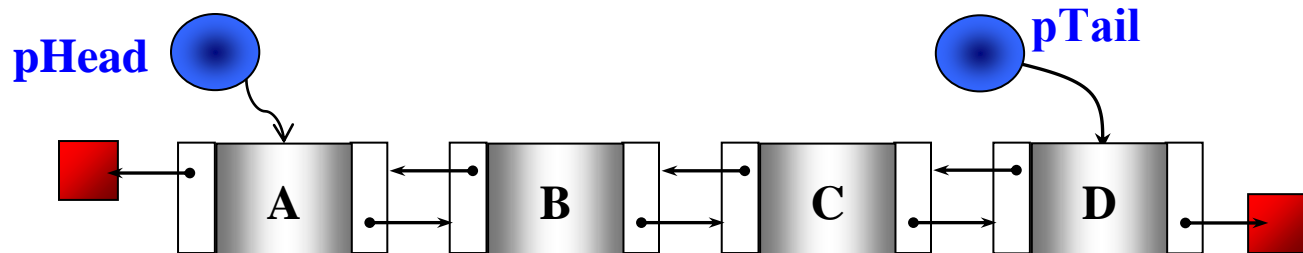
```
int removeHead ( DList &l )  
{  
    if ( l.pHead == NULL ) return 0;  
    DNode *p = l.pHead;  
    l.pHead = l.pHead->pNext;  
    delete p;  
    if ( l.pHead != NULL )    l.pHead->pPrev = NULL;  
    else                    l.pTail = NULL;  
    return 1;  
}
```



DSLK đôi – Hủy cuối ds

126

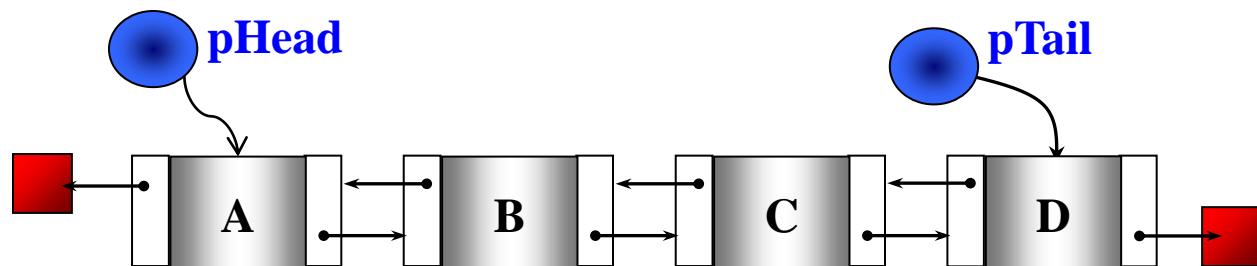
```
int removeTail ( DList &l )
{
    if ( l.pTail == NULL ) return 0;
    DNode *p = l.pTail;
    l.pTail = l.pTail->pPrev;
    delete p;
    if ( l.pTail != NULL ) l.pTail->pNext = NULL;
    else l.pHead = NULL;
    return 1;
}
```



DSLK đôi – Hủy phần tử sau q

127

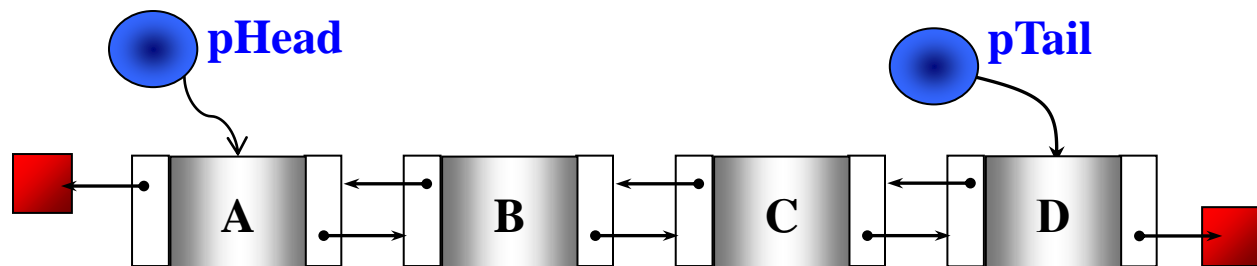
```
int removeAfter ( DList &l, DNode *q )
{
    if ( q == NULL ) return 0;
    DNode *p = q ->pNext ;
    if ( p != NULL )
    {
        q->pNext = p->pNext;
        if ( p == l.pTail )    l.pTail = q;
        else    p->pNext->pPrev = q;
        delete p;
        return 1;
    }
    else return 0;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



DSLK đôi – Hủy phần tử trước q

128

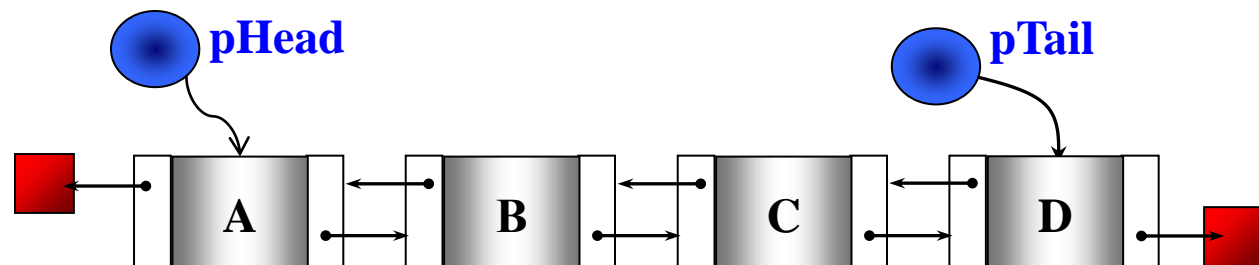
```
int removeBefore ( DList &l, DNode *q )
{
    if ( q == NULL ) return 0;
    DNode *p = q ->pPrev;
    if ( p != NULL )
    {
        q->pPrev = p->pPrev;
        if ( p == l.pHead )    l.pHead = q;
        else    p->pPrev->pNext = q;
        delete p;
        return 1;
    }
    else return 0;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



DSLK đôi – Hủy phần tử có khóa k

129

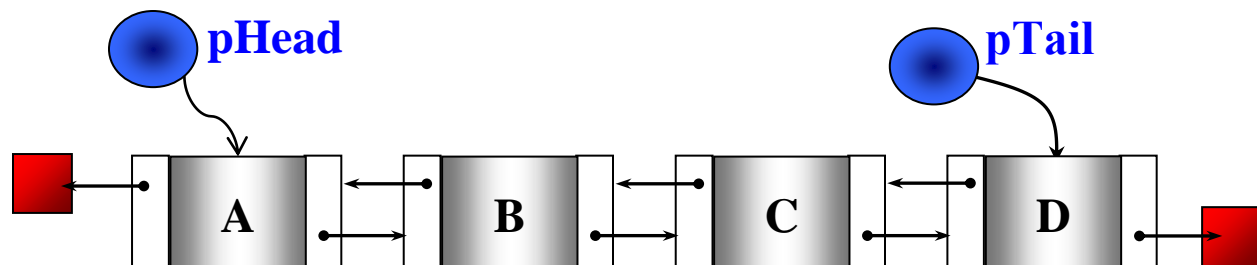
```
int removeNode ( DList &l, int k )  
{  
    DNode *p = l.pHead;  
    while ( p != NULL )  
    {  
        if ( p->data == k ) {  
  
            break;}  
        p = p->pNext;  
    }  
}
```



DSLK đôi – Hủy phần tử có khóa k

130

```
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 );
}
```



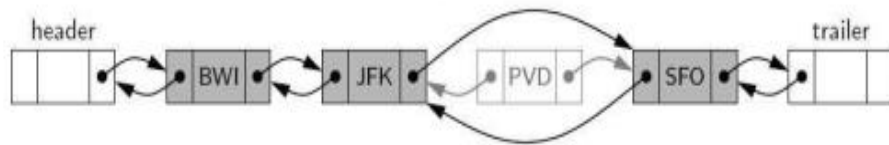
Cài đặt: Chèn và Duyệt danh sách bằng Python

131

Case I: Inserting at an arbitrary position in a non-empty list.



(a)



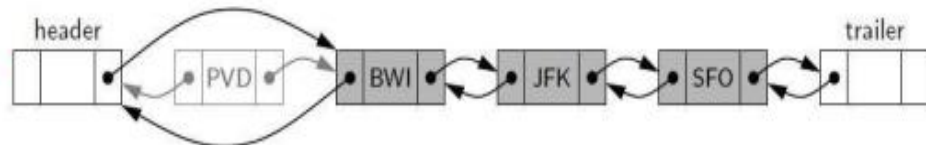
(b)



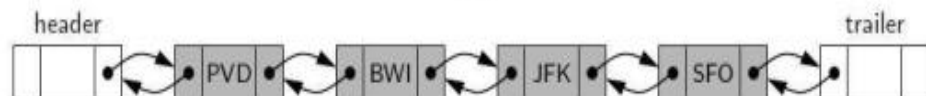
(c)



(a)



(b)



(c)

Case II: Inserting at the beginning of the list

Cài đặt: Chèn và Duyệt danh sách bằng Python

```
1 class _DoublyLinkedBase:
2     '''A base class providing a doubly linked list representation.'''
3
4     #-----
5     class _Node:
6         '''Lightweight, nonpublic class for storing a doubly linked node.'''
7         __slots__ = '_element', '_prev', '_next'      # streamline memory
8
9         def __init__(self, element, prev, next): # initialize node's field
10             self._element = element          # element to be stored
11             self._prev = prev                 # Previous node reference
12             self._next = next                 # next node reference
13     #-----
14     def __init__(self):
15         '''Create an empty list.'''
16         self._header = self._Node(None, None, None)
17         self._trailer = self._Node(None, None, None)
18         self._header._next = self._trailer      # trailer is after header
19         self._trailer._prev = self._header      # header is before trailer
20         self._size = 0                          # Number of elements
21
22     def __len__(self):
23         '''Return the number of elements in the list.'''
24         return self._size
25
26     def is_empty(self):
27         '''Return True if list is empty.'''
28         return self._size == 0
```

Base Class declaration

We maintain two references: `_prev` & `_next`

Cài đặt: Chèn và Duyệt danh sách bằng Python

133

```
30 def _insert_between(self, e, predecessor, successor):
31     '''Add element e between two existing nodes and return new node.'''
32     newest = self._Node(e, predecessor, successor) # linked to neighbors
33     predecessor._next = newest
34     successor._prev = newest
35     self._size += 1
36     return newest
37
38 def _delete_node(self, node):
39     '''Delete nonsentinel node from the list and return its element.'''
40     predecessor = node._prev
41     successor = node._next
42     predecessor._next = successor
43     successor._prev = predecessor
44     self._size -= 1 # record deleted element
45     element = node._element
46     node._prev = node._next = node._element = None # deprecate node
47     return element # return deleted element
```


Cài đặt: Chèn và Duyệt danh sách bằng Python

```
1 class LinkedDeque(_DoublyLinkedBase):           # note the use of inheritance
2     '''Double-ended queue implementation based on a doubly linked list.'''
3
4     def first(self):
5         '''Return (but do not remove) the element at the front of the deque.'''
6         if self.is_empty():
7             raise Empty("Deque is empty")
8         return self._header._next._element       # real item just after header
9
10    def last(self):
11        '''Return (but do not remove) the element at the back of the deque.'''
12        if self.is_empty():
13            raise Empty("Deque is empty")
14        return self._trailer._prev._element       # real item just before trailer
15
16    def insert_first(self, e):
17        '''Add an element to the front of the deque.'''
18        self._insert_between(e, self._header, self._header._next) # after header
19
20    def insert_last(self, e):
21        '''Add an element to the back of the deque.'''
22        self._insert_between(e, self._trailer._prev, self._trailer) # before trailer
23
```

Cài đặt: Chèn và Duyệt danh sách bằng Python

```
25 def delete_first(self):
26     '''
27     Remove and return the element from the front of the deque.
28     Raise Empty exception if the deque is empty.
29     '''
30     if self.is_empty():
31         raise Empty("Deque is empty")
32     return self._delete_node(self._header._next) # use inherited method
33
34 def delete_last(self):
35     '''
36     Remove and return the element from the back of the deque.
37     Raise Empty exception if the deque is empty.
38     '''
39     if self.is_empty():
40         raise Empty("Deque is empty")
41     return self._delete_node(self._trailer._prev) # use inherited method
42
43 def __str__(self):
44     ''' String representation of deque '''
45
46     arr = ''
47     start = self._header._next
48     for i in range(self._size):
49         arr += str(start._element) + ', '
50         start = start._next
51     return '<' + arr + '<'
52
53
54 #####
55
56 D = LinkedDeque()
57 D.insert_first(10)
58 D.insert_first(15)
59 D.insert_last(5)
60 D.insert_last(-1)
61 D.insert_first(20)
62
63 print('Length of Deque: ', len(D))
64 print('D: ', D)
65
66 print('Delete from head: ', D.delete_first())
67 print('Delete from tail ', D.delete_last())
68
69
70 print('Length of Deque: ', len(D))
71 print('D: ', D)
72
```

Length of Deque: 5
D: <20, 15, 10, 5, -1, <
Delete from head: 20
Delete from tail -1
Length of Deque: 3
D: <15, 10, 5, <

Cài đặt: Chèn và Duyệt danh sách liên kết đôi bằng Python

136

```
3 class PositionList(_DoublyLinkedBase):
4     '''A sequential container of elements allowing positional access'''
5
6     # ----- nested Position Class -----
7     class Position:
8         '''An abstraction representing the location of a single element'''
9
10        def __init__(self, container, node):
11            '''Constructor should not be invoked by user'''
12            self._container = container
13            self._node = node
14
15        def element(self):
16            '''Return the element stored at this Position'''
17            return self._node._element
18
19        def __eq__(self, other):
20            '''Return True if other is a Position representing the same location'''
21            return type(other) is type(self) and other._node is self._node
22
23        def __ne__(self, other):
24            '''Return True if other does not represent the same location'''
25            return not (self == other)
26
```


Cài đặt: Chèn và Duyệt danh sách liên kết đôi bằng Python

137

```
28
29 def _validate(self, p):
30     '''Return position's node or raise appropriate error if invalid'''
31     if not isinstance(p, self.Position):
32         raise TypeError('p must be proper Position type')
33     if p._container is not self:
34         raise ValueError('p does not belong to this container')
35     if p._node._next is None: # convention for deprecated nodes
36         raise ValueError('p is no longer valid')
37     return p._node
38
39 def _make_position(self, node):
40     ''' Return Position instance for a given node(or None if sentinel)'''
41     if node is self._header or node is self._trailer:
42         return None #boundary violation - sentinel node
43     else:
44         return self.Position(self, node) # legitimate position
45
```

```

46 # ----- Accessors -----
47 def first(self):
48     ''' Return the first Position in the list (or None if list is empty)'''
49     return self._make_position(self._header._next)
50
51 def last(self):
52     '''Return the last Position in the list (or None if list is empty)'''
53     return self._make_position(self._trailer._prev)
54
55 def before(self, p):
56     '''Return the Position just before Position p (or None if p is first)'''
57     node = self._validate(p)
58     return self._make_position(node._prev)
59
60 def after(self, p):
61     '''Return the Position just after Position p (or None if p is last)'''
62     node = self._validate(p)
63     return self._make_position(node._next)
64
65 def __iter__(self):
66     '''Generate a forward iteration of the elements of the list'''
67     cursor = self.first()
68     while cursor is not None:
69         yield cursor.element()
70         cursor = self.after(cursor)
71
72 def __str__(self):
73     ''' Generates a string representation of the list'''
74     arr = ''
75     cursor = self.first()
76     while cursor is not None:
77         arr += str(cursor.element()) + ', '
78         cursor = self.after(cursor)
79     return '<' + arr + '>'

```

Cài đặt: Chèn và Duyệt danh sách liên kết đôi

```
81 #----- Mutators -----
82 # override inherited version to return Position, rather than Node
83 def insert_between(self, e, predecessor, successor):
84     '''Add element between existing nodes and return new Position'''
85     node = super()._insert_between(e, predecessor, successor)
86     return self._make_position(node)
87
88 def add_first(self, e):
89     '''Insert element e at the front of the list and return new Position'''
90     return self._insert_between(e, self._header, self._header._next)
91
92 def add_last(self, e):
93     '''Insert element e at the back of the list and return new Position'''
94     return self._insert_between(e, self._trailer._prev, self._trailer)
95
96 def add_before(self, p, e):
97     '''Insert element e into list before Position p and return new Position'''
98     original = self._validate(p)
99     return self._insert_between(e, original._prev, original)
100
101 def add_after(self, p, e):
102     '''Insert element e into list after Position p and return new Position'''
103     original = self._validate(p)
104     return self._insert_between(e, original, original._next)
105
106 def delete(self, p):
107     '''Remove and return the element at Position p'''
108     original = self._validate(p)
109     return self._delete_node(original) # inherited method returns element
110
111 def replace(self, p, e):
112     '''
113     Replace the element at Position p with e.
114     Return the element formerly at Position p.
115     '''
116     original = self._validate(p)
117     old_value = original._element
118     original._element = e
119     return old_value
120
```

```
121 L = PositionList()
124 L.add_first(5)
125 L.add_last(10)
126 L.add_first(7)
127 L.add_first(-2)
128 L.add_last(3)
129
130 print('Length of List: ', len(L))
131 print('Positional List: ', L)
132
133 print('Delete: ', L.delete(L.first()))
134 print('Delete: ', L.delete(L.last()))
135
136 print('Length of List: ', len(L))
137 print('Positional List: ', L)
138
139 L.add_before(L.last(), 11)
140 L.add_after(L.first(), 100) # Second position
141
142 print('Length of List: ', len(L))
143 print('Positional List: ', L)
144
145 L.replace(L.last(), 1)
146 L.replace(L.after(L.first()), 100) # Second position
147
148 print('Length of List: ', len(L))
149 print('Positional List: ', L)
150
151 print("Using Iterator to print elements:")
152 for e in L:
153     print(e, end=' ')
154
155
156
157
```

```
Length of List: 5
Positional List: <-2, 7, 5, 10, 3, >
Delete: -2
Delete: 3
Length of List: 3
Positional List: <7, 5, 10, >
Length of List: 5
Positional List: <7, 15, 5, 11, 10, >
Length of List: 5
Positional List: <7, 100, 5, 11, 1, >
Using Iterator to print elements:
7 100 5 11 1
```


Cài đặt: Chèn và Duyệt danh sách liên kết đôi bằng Python

140

```
1 def insertion_sort(L):
2     '''Sort PositionalList of comparable elements into nondecreasing order'''
3
4     if len(L) > 1:                # otherwise, no need to sort it
5         marker = L.first()
6         while marker != L.last():
7             pivot = L.after(marker)    # next item to the current position
8             value = pivot.element()
9             if value > marker.element(): # pivot is already sorted
10                 marker = pivot        # pivot becomes the new marker
11             else:                    # must relocate the pivote
12                 walk = marker        # find the leftmost item greater than value
13                 while walk != L.first() and L.before(walk).element() > value:
14                     walk = L.before(walk)    # keep moving left
15                 L.delete(pivot)
16                 L.add_before(walk, value)    # reinsert value before walk
17
18 #####
19 print(['Original List L: ', L])
20 insertion_sort(L)
21 print('Sorted List:', L)
```

Original List L: <7, 100, 5, 11, 1, >
Sorted List: <1, 5, 7, 11, 100, >

DSLK đôi – Nhận xét

141

- 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

142

- 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)

143

- 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

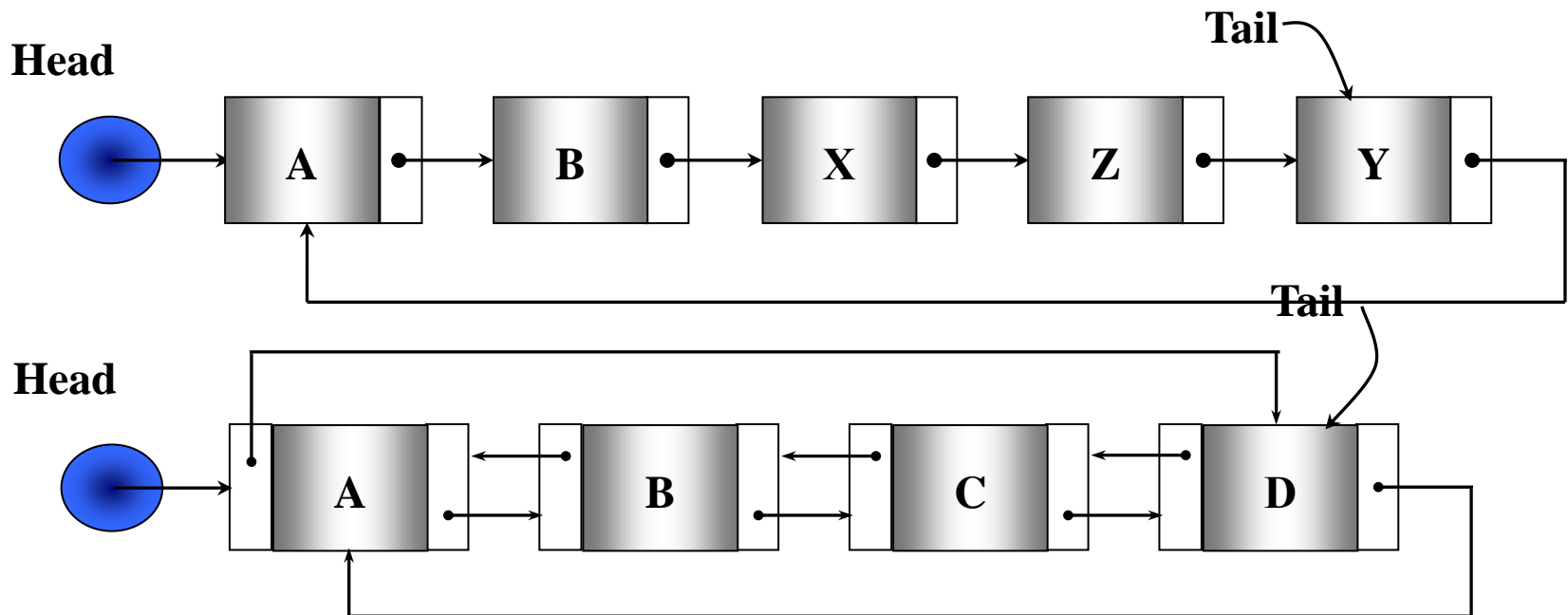
144

- 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)

145

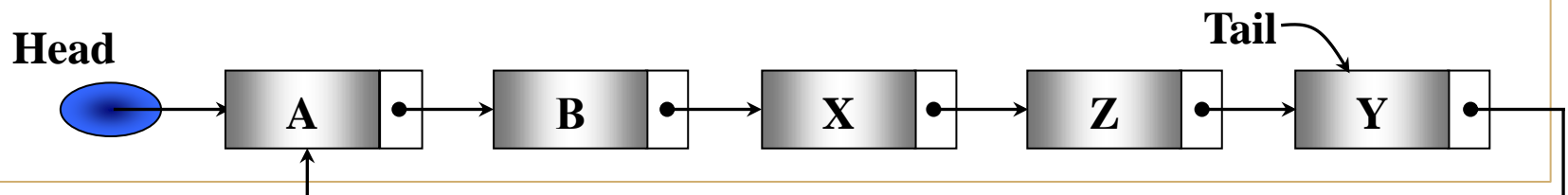
- Là một danh sách liên kết đơn (hoặc đôi) mà nút cuối danh sách, thay vì trỏ đến **NULL**, sẽ trỏ tới nút đầu danh sách
- Đối với danh sách vòng, có thể xuất phát từ một phần tử bất kỳ để duyệt toàn bộ danh sách



DSLK vòng – Thêm vào đầu ds

146

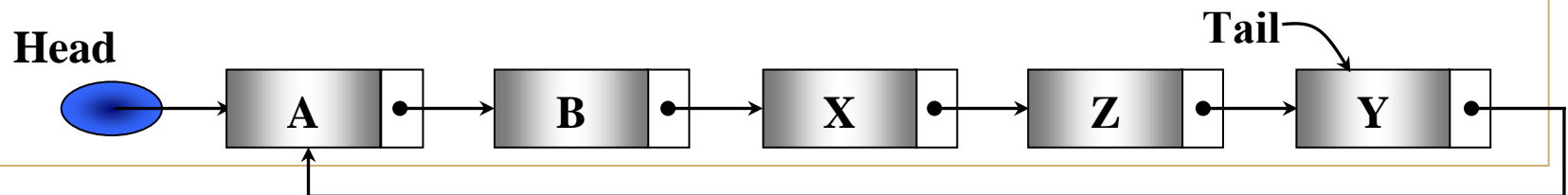
```
void addHead (List &l, Node *new_node)
{
    if (l.pHead == NULL) {
        l.pHead = l.pTail = new_node;
        l.pTail->pNext = l.pHead;
    }
    else{
        new_node->pNext = l.pHead;
        l.pTail->pNext = new_node;
        l.pHead = new_node;
    }
}
```



DSLK vòng – Thêm vào cuối ds

147

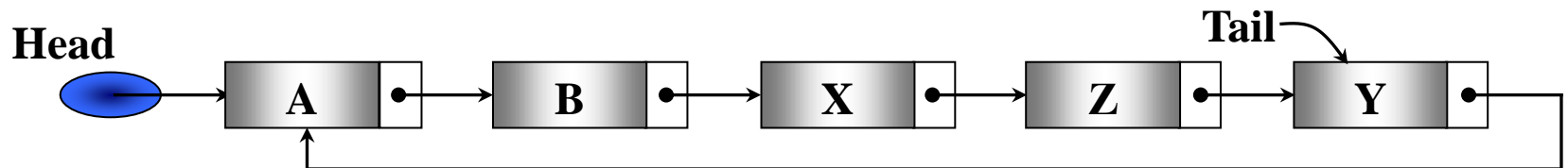
```
void addTail (List &l, Node *new_node)
{
    if (l.pHead == NULL) {
        l.pHead = l.pTail = new_node;
        l.pTail->pNext = l.pHead;
    }
    else{
        new_node->pNext = l.pHead;
        l.pTail->pNext = new_node;
        l.pTail = new_node;
    }
}
```



DSLK vòng – Hủy nút đầu ds

148

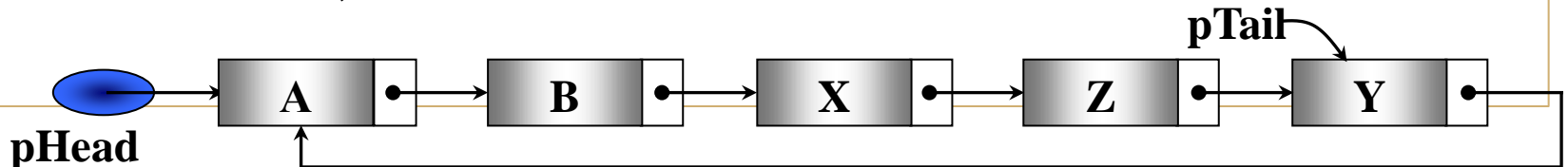
```
int removeHead (List &l){  
    Node *p = l.pHead;  
    if ( p == NULL ) return 0;  
    if ( l.pHead == l.pTail )  
        l.pHead = l.pTail = NULL;  
    else  
        { l.pHead = p->pNext; l.pTail->pNext = l.pHead; }  
    delete p;  
    return 1;  
}
```



DSLK vòng – Hủy phần tử sau q

149

```
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

150

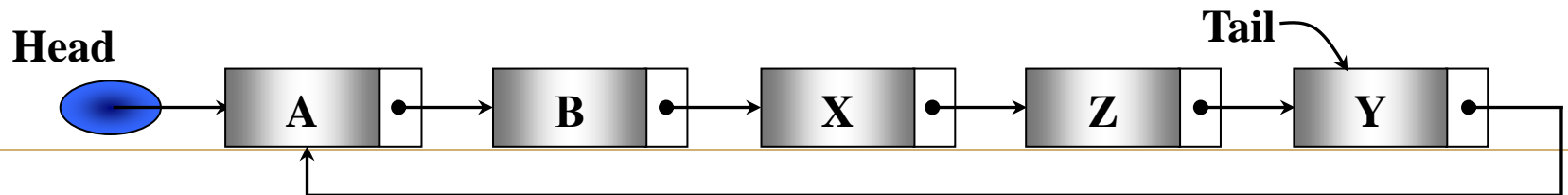
- Danh sách vòng không có phần tử đầu danh sách rõ rệt, nhưng ta có thể đánh dấu một phần tử bất kỳ trên danh sách xem như phần tử đầu xâu để kiểm tra việc duyệt đã qua hết các phần tử của danh sách hay chưa

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

Ví dụ: Tìm kiếm

151

```
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;
}
```



Case Study: Maintaining Access Frequencies

152

Favorites List ADT

- For maintaining a collection of elements while keeping track of the number of times each element is accessed.
- Examples:
 - Web browser that keeps track of a user's most accessed URLs.
 - A music collection that maintains a list of the most frequently played songs for a user.
- Favorites List ADT should provide the following methods:
 - `access(e)`: Access the element `e`, incrementing its access count, and adding it to the favorites list if it is not already present.
 - `remove(e)`: Remove element `e` from the favorites list, if present.
 - `top(k)`: Return an iteration of the `k` most accessed elements.

The favourite list is sorted.

- Every time an item is accessed, its count is incremented and its position is adjusted within the list using a technique similar to insertion-sort algorithm.

Case Study: Maintaining Access Frequencies

153

```
4 class FavoritesList:
5     '''List of elements ordered from most frequently accessed to least.'''
6
7     #----- nested _Item Class -----
8     class _Item:
9         __slots__ = '_value', '_count'          # streamline memory usage
10
11         def __init__(self, e):
12             self._value = e                    # user's element
13             self._count = 0                    # access count initially 0
14
15     #----- non-public utilities -----
16     def _find_position(self, e):
17         '''Search for element e and return its Position (or None if not found)'''
18         walk = self._data.first()
19         while walk is not None and walk.element()._value != e:
20             walk = self._data.after(walk) # move forward
21         return walk
22
23     def _move_up(self, p):
24         '''Move item at Position p earlier in the list based on access count'''
25         if p != self._data.first(): # otherwise, already at top, can't move up
26             cnt = p.element()._count
27             walk = self._data.before(p) # move up
28             if cnt > walk.element()._count:
29                 while (walk != self._data.first() and
30                        cnt > self._data.before(walk).element()._count):
31                     walk = self._data.before(walk) # keep moving upward
32             self._data.add_before(walk, self._data.delete(p)) # delete / reinsert
```

Case Study: Maintaining Access Frequencies

```
35 # ----- Public Methods -----
36 def __init__(self):
37     ''' Create an empty list of favourites'''
38     self._data = Positionallist() # will be a list of _Item instances
39
40 def __len__(self):
41     '''Return the number of entries on the favourite list'''
42     return len(self._data)
43
44 def is_empty(self):
45     '''Return True if list is empty.'''
46     return len(self._data) == 0
47
48 def access(self, e):
49     '''Access element e, thereby increasing its access count'''
50     p = self._find_position(e) # try to locate the existing element
51     if p is None:
52         p = self._data.add_last(self._Item(e)) # if new, place at end
53         p.element().count += 1 # increment its access count
54         self._move_up(p) # consider moving forward
55
56 def remove(self, e):
57     ''' Remove element e from the list of favourites'''
58     p = self._find_position(e) # locate existing element
59     if p is not None:
60         self._data.delete(p) # delete if found
61
62 def top(self, k):
63     '''Generate a sequence of top k elements in terms of access count.'''
64     if not 1 <= k <= len(self):
65         raise ValueError('Illegal value for k')
66     walk = self._data.first()
67     for j in range(k):
68         item = walk.element() # element of list is _Item
69         yield item._value # report user's element
70         walk = self._data.after(walk) # move forward
```

```
76 F = Favoriteslist()
77 F.access('BackStreetBoys')
78 F.access('KatyPerry')
79 F.access('Eminem')
80 F.access('MichaelJackson')
81 F.access('ImagineDragons')
82 F.access('BritneySpears')
83 F.access('BackStreetBoys')
84 F.access('ImagineDragons')
85 F.access('ImagineDragons')
86 F.access('ImagineDragons')
87 F.access('KatyPerry')
88 F.access('Eminem')
89
90
91 for k in F.top(5):
92     print(k)
93
94
95 F.access('BritneySpears')
96 F.access('BritneySpears')
97 F.access('BritneySpears')
98 F.access('BritneySpears')
99 F.access('BritneySpears')
100
101 print('\nUpdated Favourites \n')
102 for k in F.top(5):
103     print(k)

```

ImagineDragons
BackStreetBoys
KatyPerry
Eminem
MichaelJackson

Updated Favourites

BritneySpears
ImagineDragons
BackStreetBoys
KatyPerry

Case Study: Maintaining Access Frequencies

```
6 class FavoriteslistMTF(Favoriteslist):
7     '''List of elements ordered with move-to-front heuristic'''
8
9
10    # we override move up to provide move-to-front semantics
11    def _move_up(self, p):
12        '''Move accessed item at Position p to front of list.'''
13        if p != self._data.first():
14            self._data.add_first(self._data.delete(p)) # delete / reinsert
15
16
17    # we override top because list is no longer sorted
18    def top(self, k):
19        '''Generate sequence of top k elements in terms of access count'''
20        if not 1 <= k <= len(self):
21            raise ValueError('Illegal value for k')
22
23        # we begin by making a copy of the original list
24        temp = PositionalList()
25        for item in self._data:
26            temp.add_last(item) # positional lists support iteration
27
28        # we repeatedly find, report, and remove element with largest count
29        for j in range(k):
30            # find and report next highest from temp
31            highPos = temp.first()
32            walk = temp.after(highPos)
33            while walk is not None:
34                if walk.element()._count > highPos.element()._count:
35                    highPos = walk
36            walk = temp.after(walk)
37            # We have found the element with highest count
38            yield highPos.element()._value # report element to user
39            temp.delete(highPos) # remove from temp list
40
41
42    # to support print operations
43    def __str__(self):
44        ''' provides a string representation of the list'''
45        arr = ''
46        cursor = self._data.first()
47        while cursor is not None:
48            arr += str((cursor.element()._value, cursor.element()._count)) + '\n'
49            cursor = self._data.after(cursor)
50        return '<' + arr + '>'
51
```

- We modify `_move_up()` method to implement move-to-front heuristic
- The `top()` method is also modified for sorting the list to be displayed

```
56 M = FavoriteslistMTF()
57 M.access('BackStreetBoys')
58 M.access('KatyPerry')
59 M.access('Eminem')
60 M.access('MichaelJackson')
61 M.access('ImagineDragons')
62 M.access('BritneySpears')
63 M.access('BackStreetBoys')
64 M.access('ImagineDragons')
65 M.access('ImagineDragons')
66 M.access('ImagineDragons')
67 M.access('KatyPerry')
68 M.access('Eminem')
69 M.access('BritneySpears')
70 M.access('BritneySpears')
71 M.access('BritneySpears')
72 M.access('BritneySpears')
73 M.access('BritneySpears')
74 M.access('Eminem')
75 M.access('Eminem')
76 M.access('Eminem')
77
78 print('Length of list M:', len(M))
79 print('List M: ', M)
80
81 print('\nTop Favourites:')
82 for k in M.top(5):
83     print(k)
84
```

Length of list M: 6
List M: <('Eminem', 5), ('BritneySpears', 6), ('KatyPerry', 2), ('ImagineDragons', 4), ('BackStreetBoys', 2), ('MichaelJackson', 1)>
>

Top Favourites:
BritneySpears
Eminem
ImagineDragons
KatyPerry
BackStreetBoys