

CHAPTER 6: 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**)

Giới thiệu - Cấu trúc dữ liệu tĩnh

3

- 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 ...**

Giới thiệu - Cấu trúc dữ liệu tĩnh

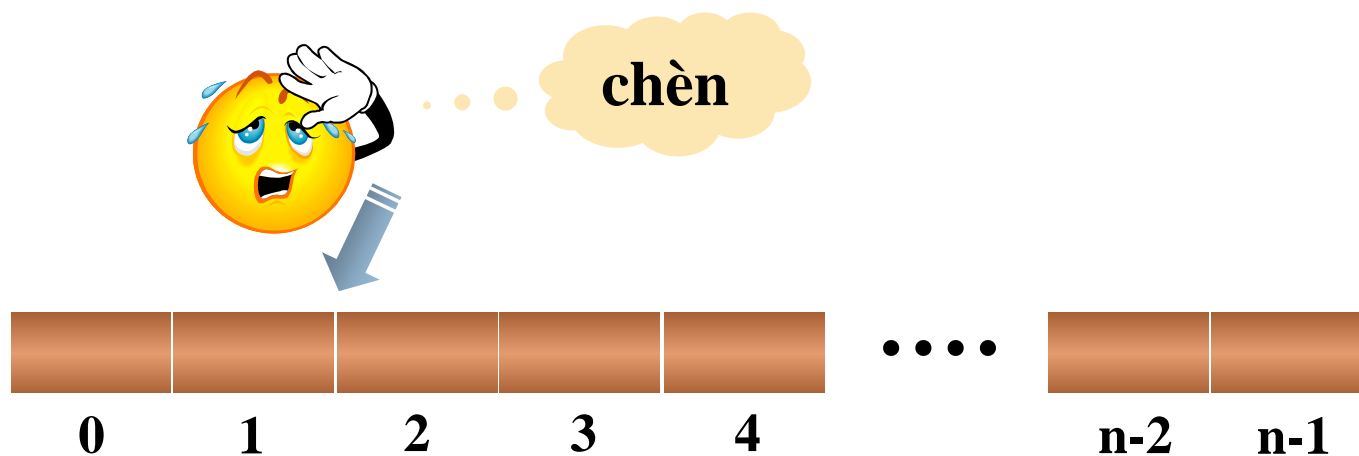
4

- 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

5

- **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 rất khó



Giới thiệu - Cấu trúc dữ liệu động

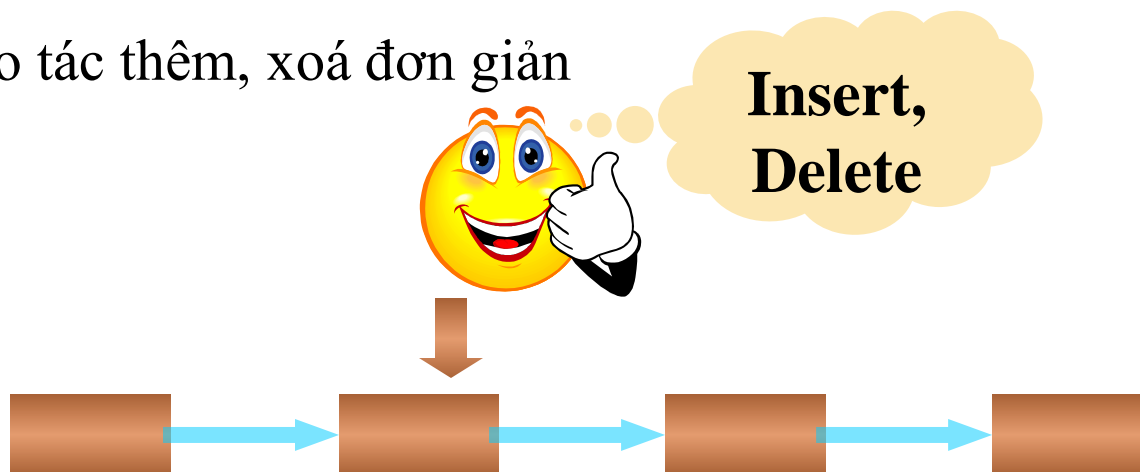
6

- 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

7

- **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)
 - Không thể truy cập ngẫu nhiên
 - Thao tác thêm, xóa đơn giản



Giới thiệu - Danh sách liên kết

8

- 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

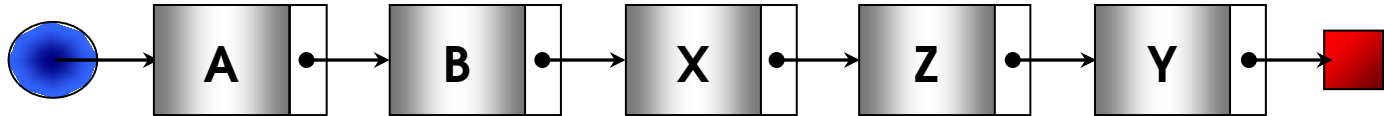
9

- 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

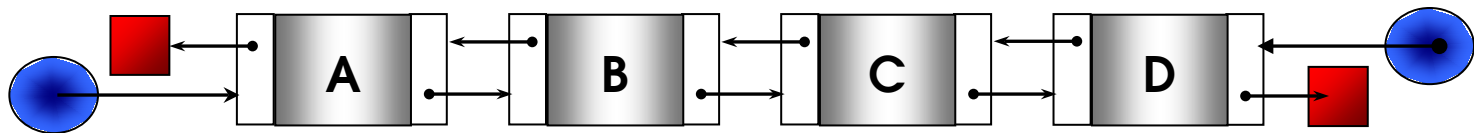
Giới thiệu - Danh sách liên kết

10

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



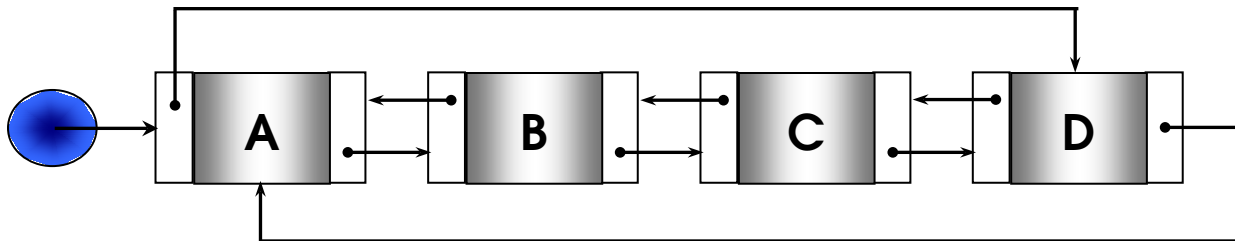
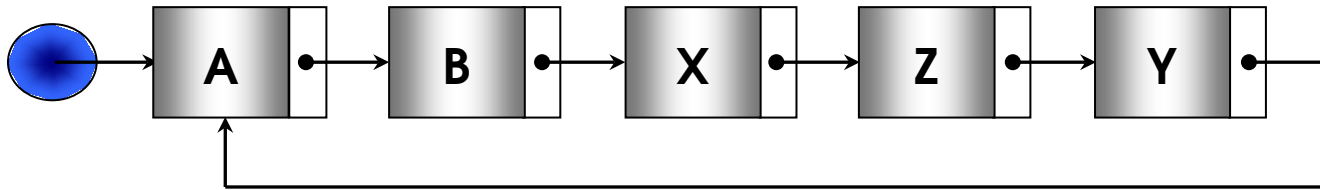
- **Danh sách liên kết kép:** mỗi phần tử liên kết với các phần tử đứng trước và sau nó trong danh sách:



Giới thiệu - Danh sách liên kết

11

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



Nội dung

12

- 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)

13

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

DSLK đơn – Khai báo

14

- 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

15

- Ví dụ 1: Khai báo node lưu số nguyên:

```
struct Node
{
    int data;
    Node *pNext;
};
```

- Ví dụ 2: Khai báo node lưu thông tin của một sinh viên:

```
struct SinhVien {
    char Ten[30];
    int MaSV;
};

struct Node {
    SinhVien data;
    Node *pNext;
};
```

DSLK đơn – Khai báo

16

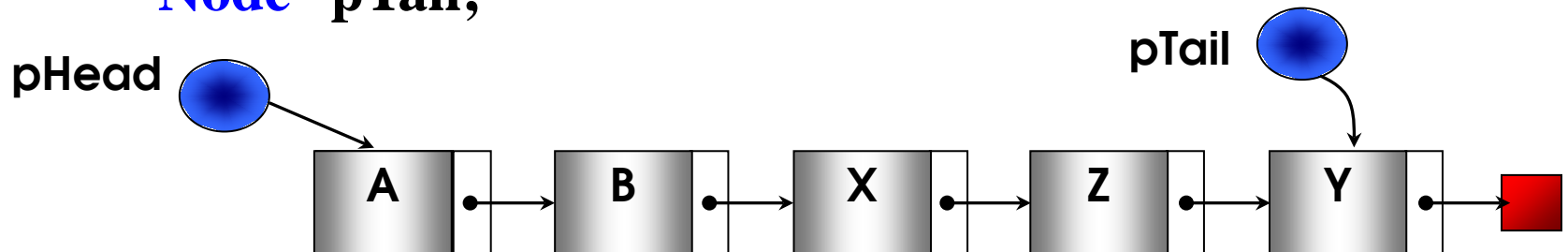
□ Tổ chức, quản lý:

- 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;



DSLK đơn – Khai báo

17

- Ví dụ: Khai báo cấu trúc 1 DSLK đơn chứa số nguyên

// kiểu của một phần tử trong danh sách

```
struct Node
{
    int    data;
    Node*  pNext;
};
```

// kiểu danh sách liên kết

```
struct List
{
    Node* pHead;
    Node* pTail;
};
```

Khai báo biến kiểu danh sách:

```
List  tên_biến;
```

DSLK đơn – Khai báo

18

□ Tạo một node mới

- Viết hàm **getNode** để tạo ra một nút cho danh sách với dữ liệu là x

```
Node* getNode (DataType x)
```

```
{
```

```
    Node *p;
```

```
    p = new Node; // Cấp phát vùng nhớ cho node
```

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

```
    {
```

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

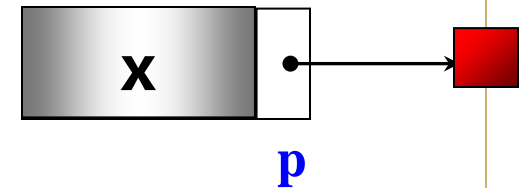
```
    }
```

```
    p->data = x; // Gán dữ liệu cho phần tử p
```

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

```
    return p;
```

```
}
```



Gọi hàm??

Danh sách liên kết đơn (DSLK đơn)

19

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

DSLK đơn

20

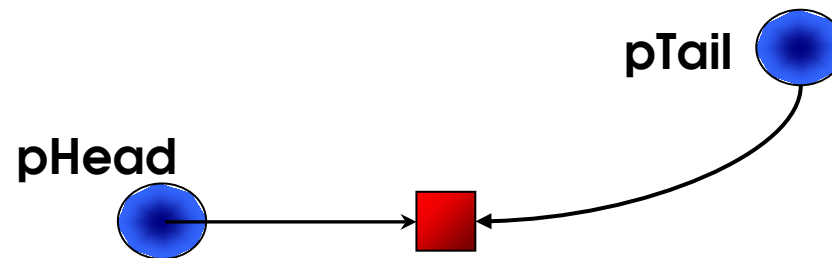
□ Các thao tác cơ bản

- 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
- ...

DSLK đơn – Các thao tác cơ sở

21

- Tạo danh sách rỗng



```
void Init(List &l)
{
    l.pHead = l.pTail = NULL;
}
```

DSLK đơn

22

- **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

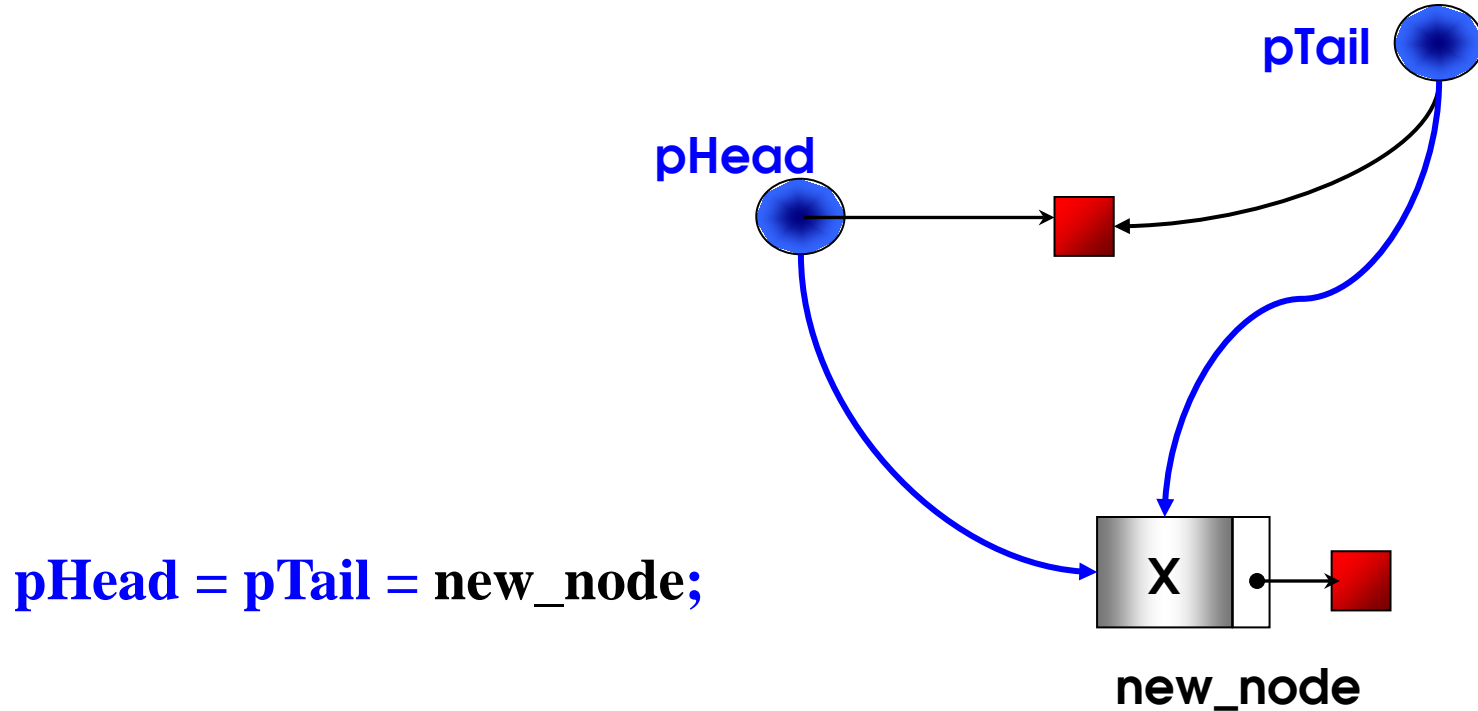
23

- **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

24

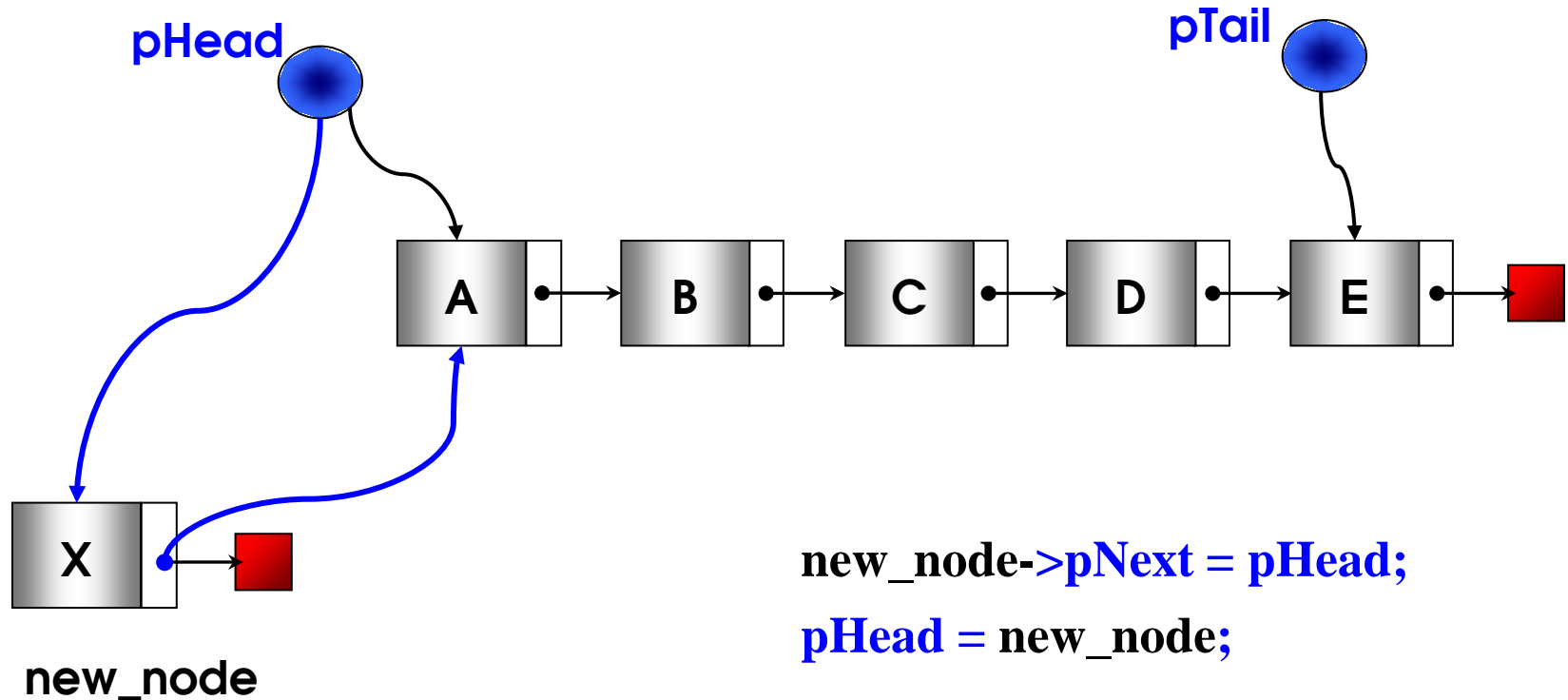
- Thêm một phần tử
 - ▣ Nếu danh sách ban đầu rỗng



DSLK đơn – Các thao tác cơ sở

25

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



DSLK đơn – Các thao tác cơ sở

26

Thuật toán: Gắn nút vào đầu DS

// input: danh sách, phần tử mới new_node

// output: danh sách với new_node ở đầu DS

- Nếu DS rỗng thì
 - $pHead = pTail = new_node;$
- Ngược lại
 - $new_node \rightarrow pNext = pHead;$
 - $pHead = new_node;$

DSLK đơn – Các thao tác cơ sở

27

Cài đặt: Gắn nút vào đầu DS

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

DSLK đơn – Các thao tác cơ sở

28

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 danh sách (???)

DSLK đơn – Các thao tác cơ sở

29

Ví dụ: Thêm một số nguyên vào đầu ds:

```
// Nhập dữ liệu cho X
int x;
cout<<"Nhập X=";
cin>>x;
// Tạo nút mới
Node* new_node = getNode(x);
// Gắn nút vào đầu ds
if (new_node != NULL)
    addHead(1, new_node);
```

DSLK đơn – Các thao tác cơ sở

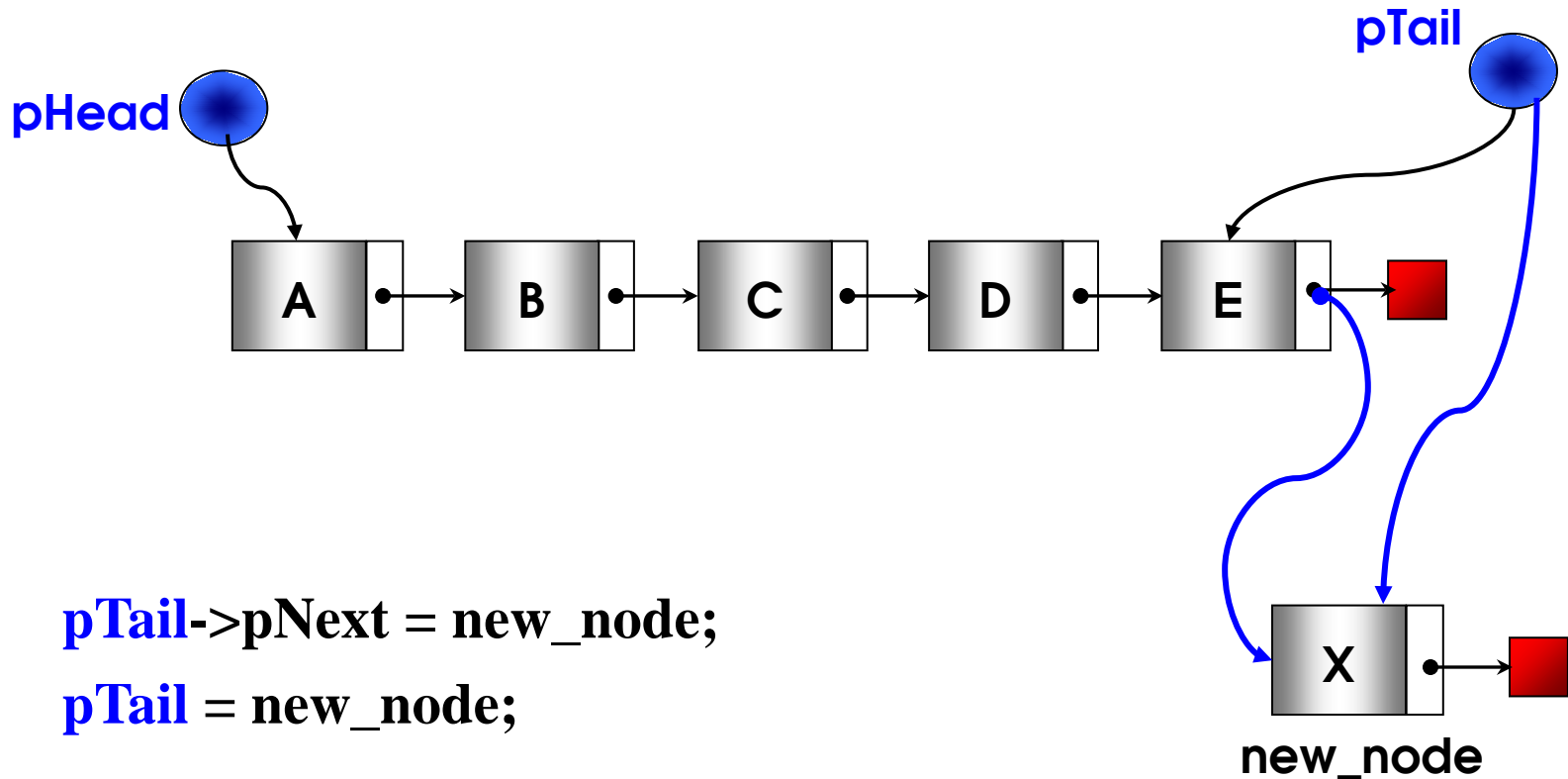
30

- **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

31

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



DSLK đơn – Các thao tác cơ sở

32

Thuật toán: Thêm một phần tử vào cuối DS

// input: danh sách, phần tử mới new_node

// output: danh sách với new_node ở cuối DS

- Nếu DS rỗng thì
 - $pHead = pTail = new_node;$
- Ngược lại
 - $pTail \rightarrow pNext = new_node ;$
 - $pTail = new_node;$

DSLK đơn – Các thao tác cơ sở

33

Cài đặt: Gắn nút vào cuối DS

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

DSLK đơn – Các thao tác cơ sở

34

Thuật toán: Thêm một thành phần dữ liệu vào cuối ds

// input: danh sách thành phần dữ liệu X

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

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

DSLK đơn – Các thao tác cơ sở

35

Ví dụ: Thêm một số nguyên vào cuối ds:

```
// Nhập dữ liệu cho X
int x;
cout<<"Nhập X=";
cin>>x;
// Tạo nút mới
Node* p = getNode(x);
// Gắn nút vào cuối DS
if (p != NULL)
    addTail(l, p);
```

DSLK đơn – Các thao tác cơ sở

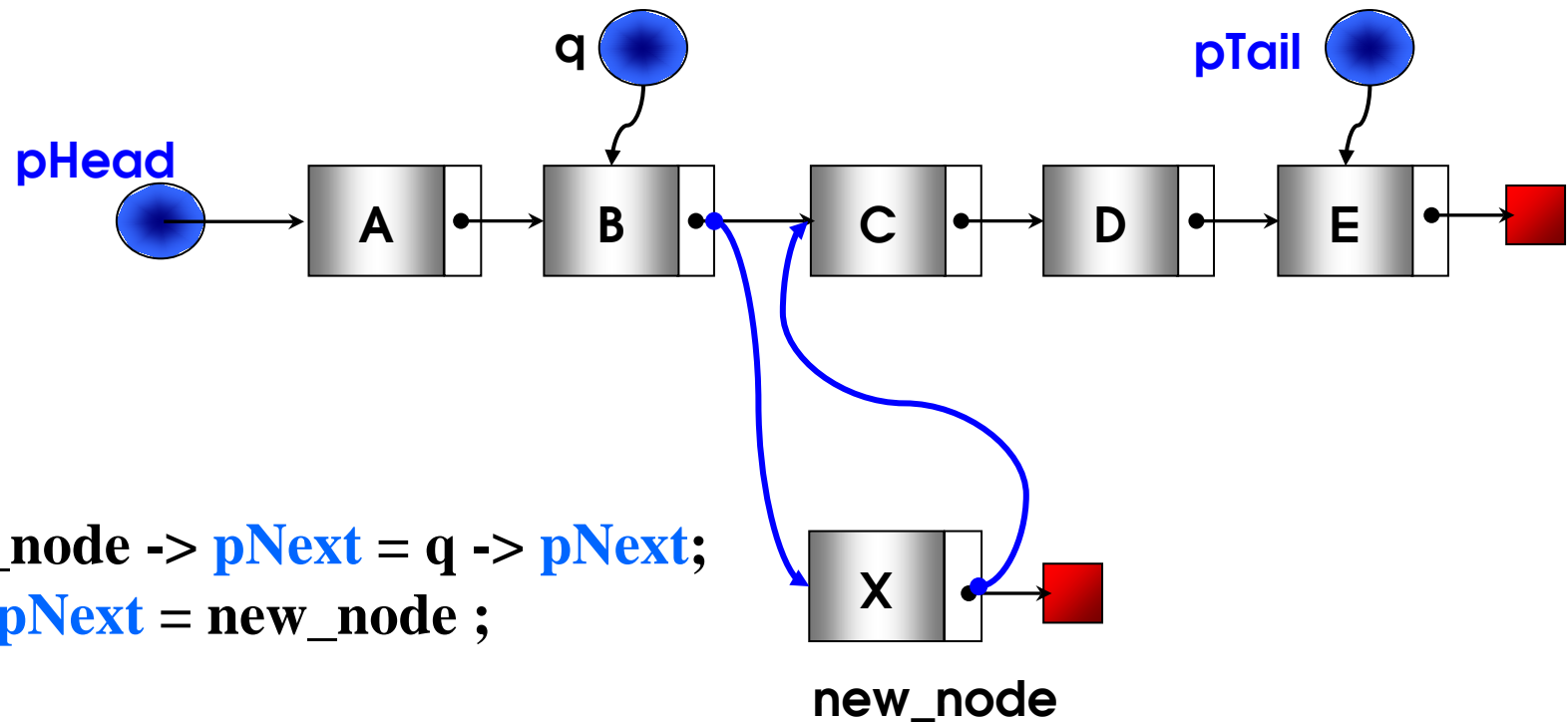
36

- **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

37

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



DSLK đơn – Các thao tác cơ sở

38

Thuật toán: Chèn một phần tử vào sau nút q (addAfter)

// input: danh sách l, q, phần tử mới new_node

// output: danh sách với new_node ở sau q

□ Nếu (q != NULL) thì:

new_node -> pNext = q -> pNext;

q -> pNext = new_node ;

Nếu (q == l.pTail) thì

l.pTail = new_node;

□ Ngược lại

Thêm new_node vào đầu danh sách

DSLK đơn – Các thao tác cơ sở

39

Cài đặt: Chèn một phần tử vào sau nút q

```
void addAfter (List &l, Node *q, Node* new_node)
{
    if (q!=NULL)
    {
        new_node->pNext = q->pNext;
        q->pNext = new_node;
        if (q==l.pTail)
            l.pTail = new_node;
    }
}
```

DSLK đơn – Các thao tác cơ sở

40

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

41

- **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

42

□ 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

43

□ 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*

```
void processList (List l)
```

```
{
```

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

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


```
    {
```

```
        // xử lý cụ thể p tùy ứng dụng
```

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

```
    }
```

```
}
```



**Chuyển
thành vòng
lặp for??**

DSLK đơn – Các thao tác cơ sở

44

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

DSLK đơn – Các thao tác cơ sở

45

Ví dụ: In các phần tử trong danh sách

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

DSLK đơn – Các thao tác cơ sở

46

- 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

47

- **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

48

- 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

49

- **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

50

- 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

51

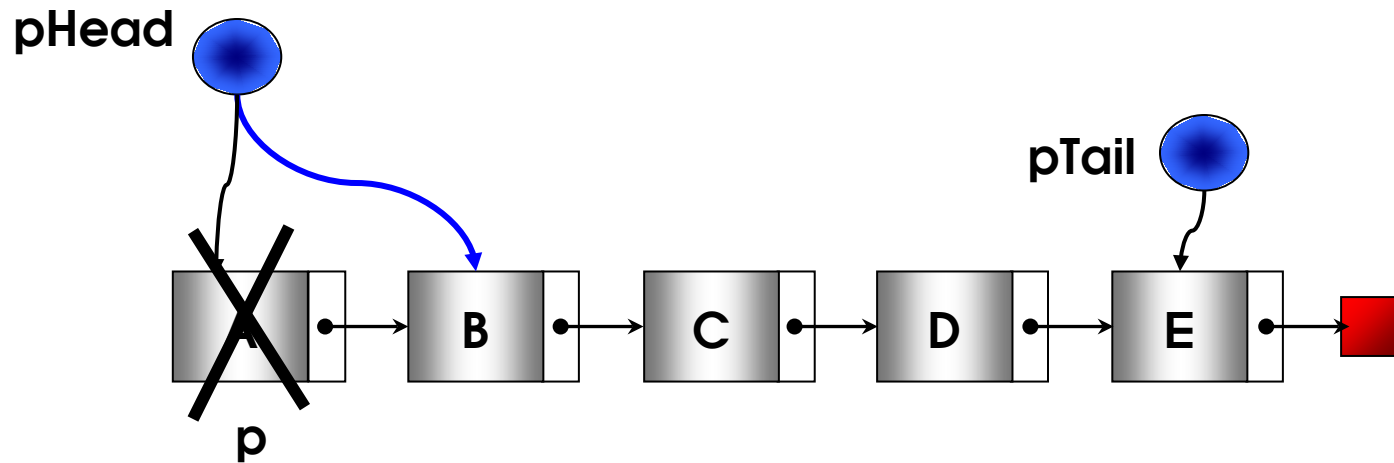
Thuật toán: Xóa node đầu danh sách

- ❑ 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

52

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



pHead = p->pNext;

delete p;

DSLK đơn – Các thao tác cơ sở

53

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

54

- 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

55

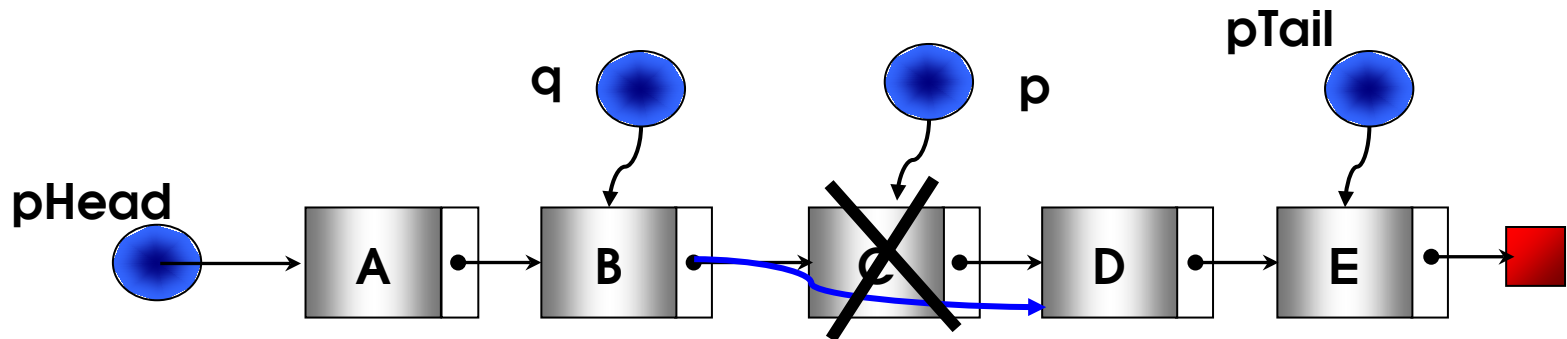
Thuật toán: Xóa node sau node **q** trong danh sách:

- ▣ 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 **pNext** của **q** trở vào node đứng sau **p**
 - Bước 3: Nếu **p** là phần tử cuối thì **pTail** trở vào **q**
 - Bước 4: Giải phóng vùng nhớ mà **p** trở tới

DSLK đơn – Các thao tác cơ sở

56

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

57

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

58

- 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

59

Thuật toán: Xóa 1 node có khoá k

□ Bước 1:

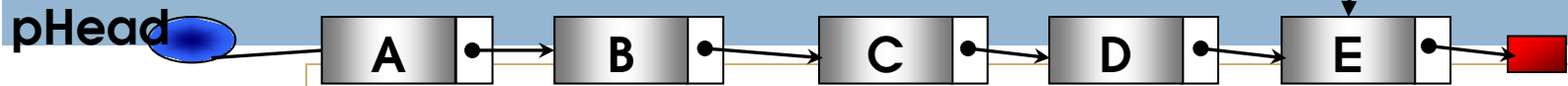
- Tìm node có khóa k (gọi là p) và node đứng trước nó (gọi là q)

□ 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

60



- 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

61

- **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

62

□ 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

63

- 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???

Danh sách liên kết đơn (DSLK đơn)

65

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

Sắp xếp trên DSLK đơn

66

- Cài đặt lại trên danh sách liên kết một trong những thuật toán sắp xếp đã biết trên mảng
- Các cách tiếp cận:
 - ▣ **Phương án 1:** Hoán vị nội dung các phần tử trong danh sách (thao tác trên vùng **data**)
 - ▣ **Phương án 2:** Thay đổi các mối liên kết (thao tác trên vùng **link**)

Sắp xếp trên DSLK đơn – PA 1

67

- Do thực hiện hoán vị nội dung của các phần tử nên đòi hỏi sử dụng thêm vùng nhớ trung gian \Rightarrow chỉ thích hợp với các dạng danh sách mà phần data có kích thước nhỏ
- Khi kích thước của phần data lớn, việc hoán vị giá trị của hai phần tử sẽ chiếm chi phí đáng kể
- Chú ý **cách thức truy xuất** đến các phần tử trên danh sách liên kết: truy xuất thông qua liên kết

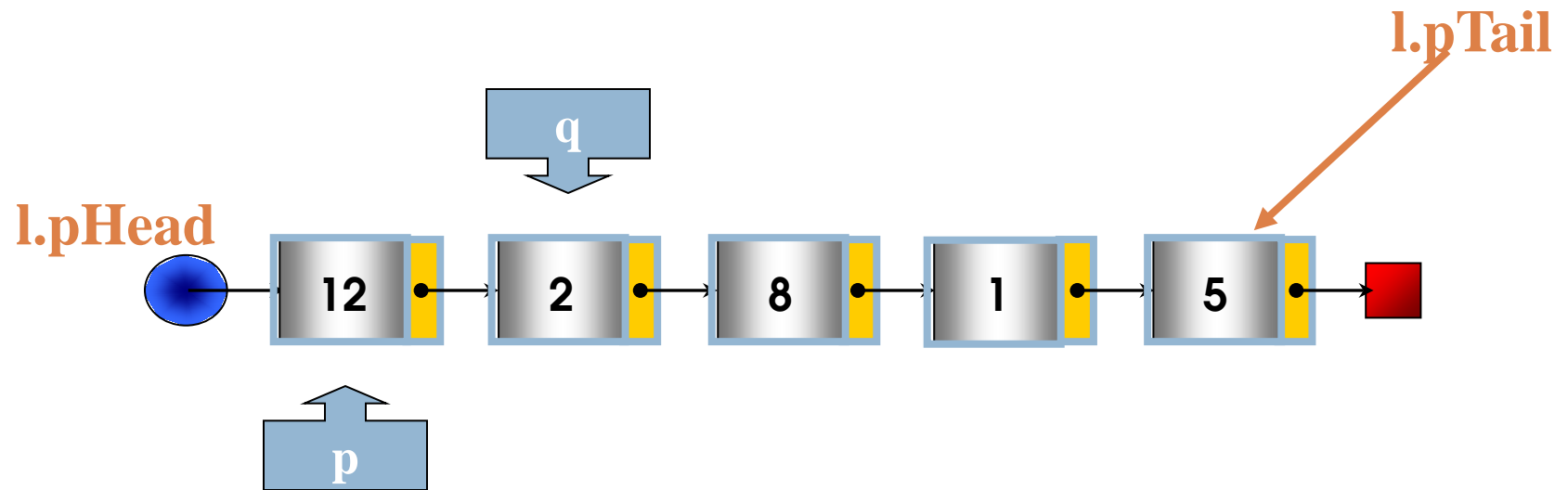
Sắp xếp bằng phương pháp đổi chỗ trực tiếp (*Interchange Sort*)

68

```
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);
}
```

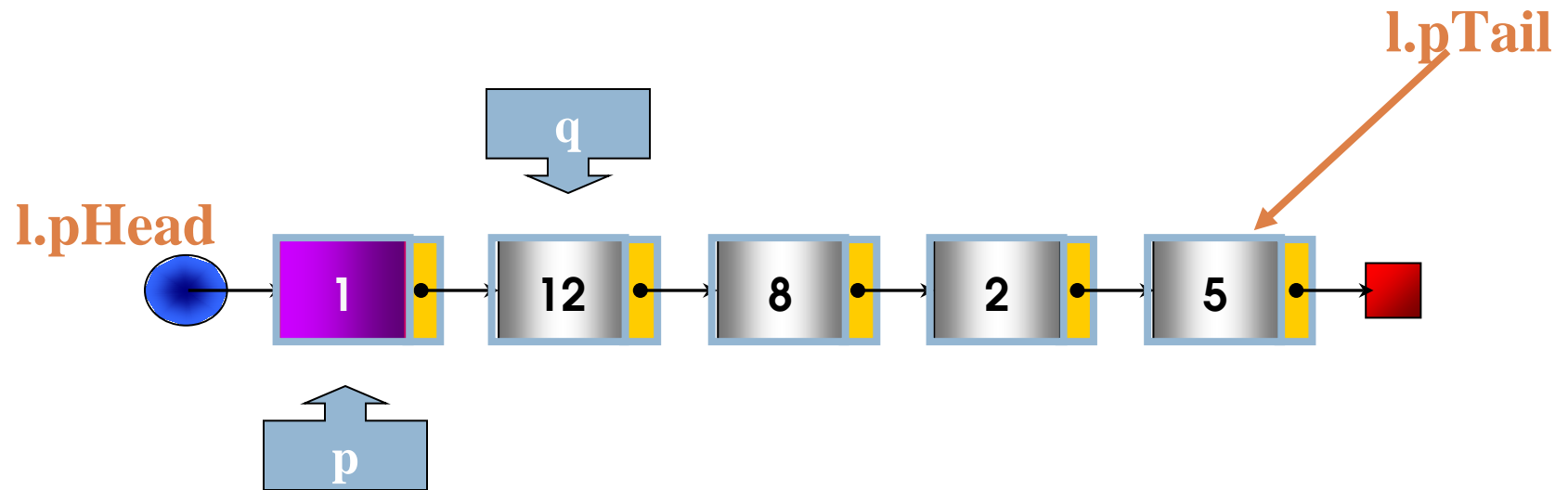
Sắp xếp đổi chỗ trực tiếp (*Interchange Sort*)

69



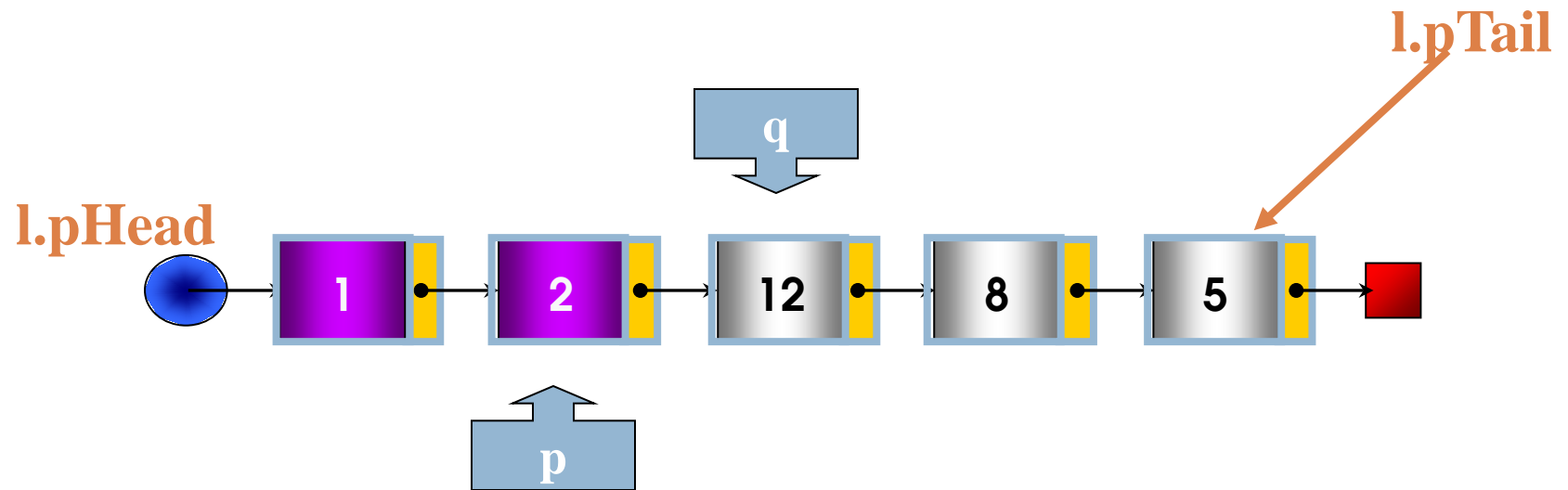
Sắp xếp đổi chỗ trực tiếp (*Interchange Sort*)

70



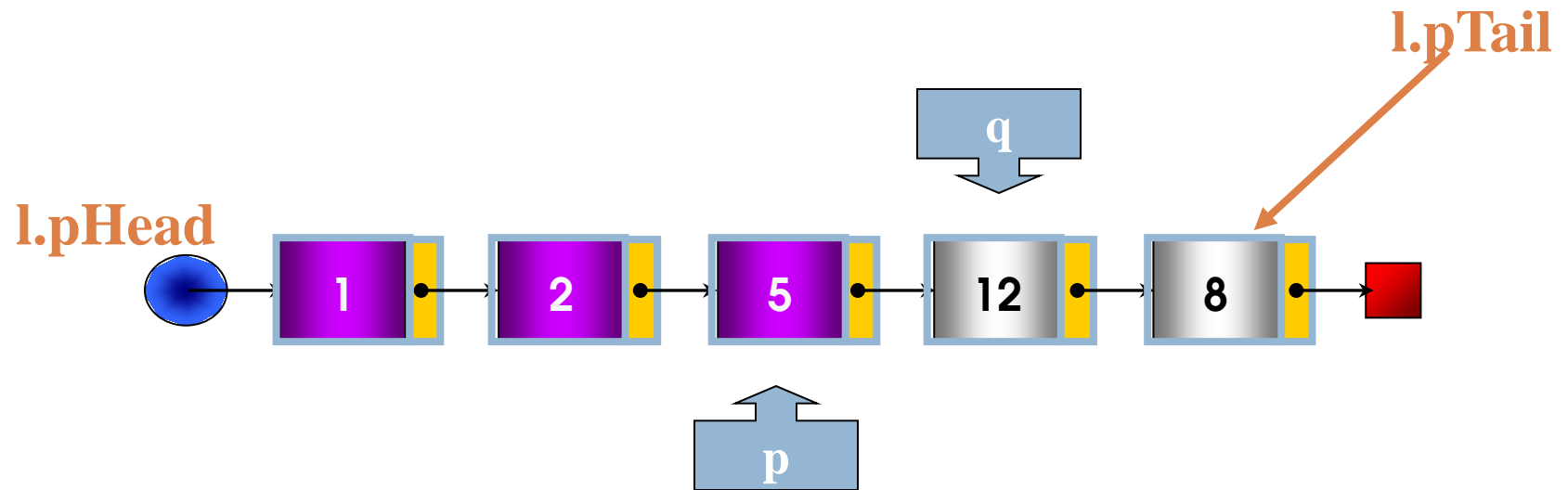
Sắp xếp đổi chỗ trực tiếp (*Interchange Sort*)

71



Sắp xếp đổi chỗ trực tiếp (*Interchange Sort*)

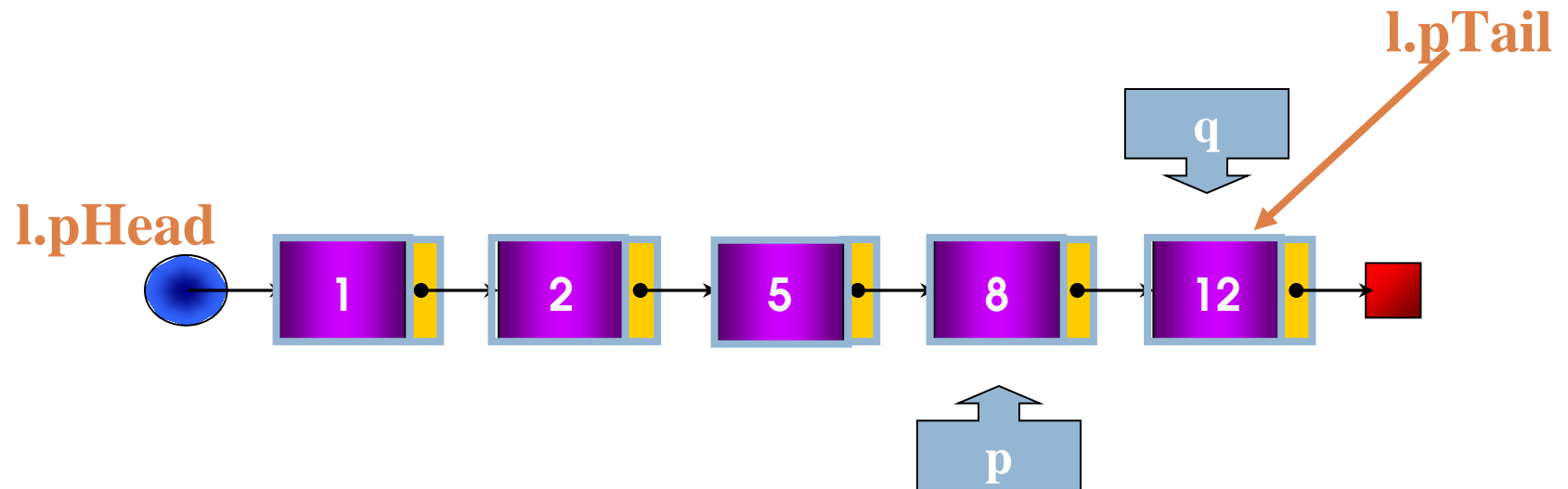
72



Sắp xếp đổi chỗ trực tiếp (*Interchange Sort*)

73

Dừng



Nội dung

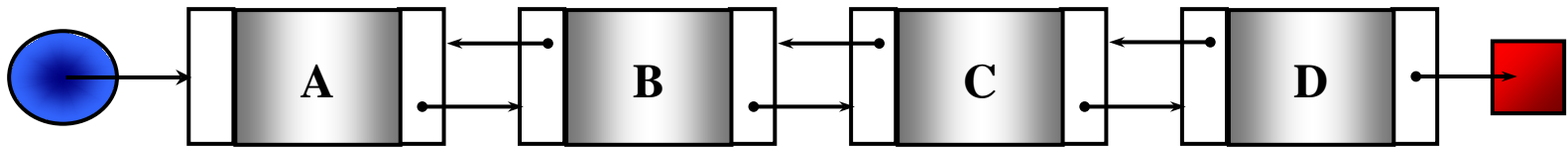
100

- 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)

101

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



DSLK đôi – Khai báo cấu trúc

102

- 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

103

- 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??

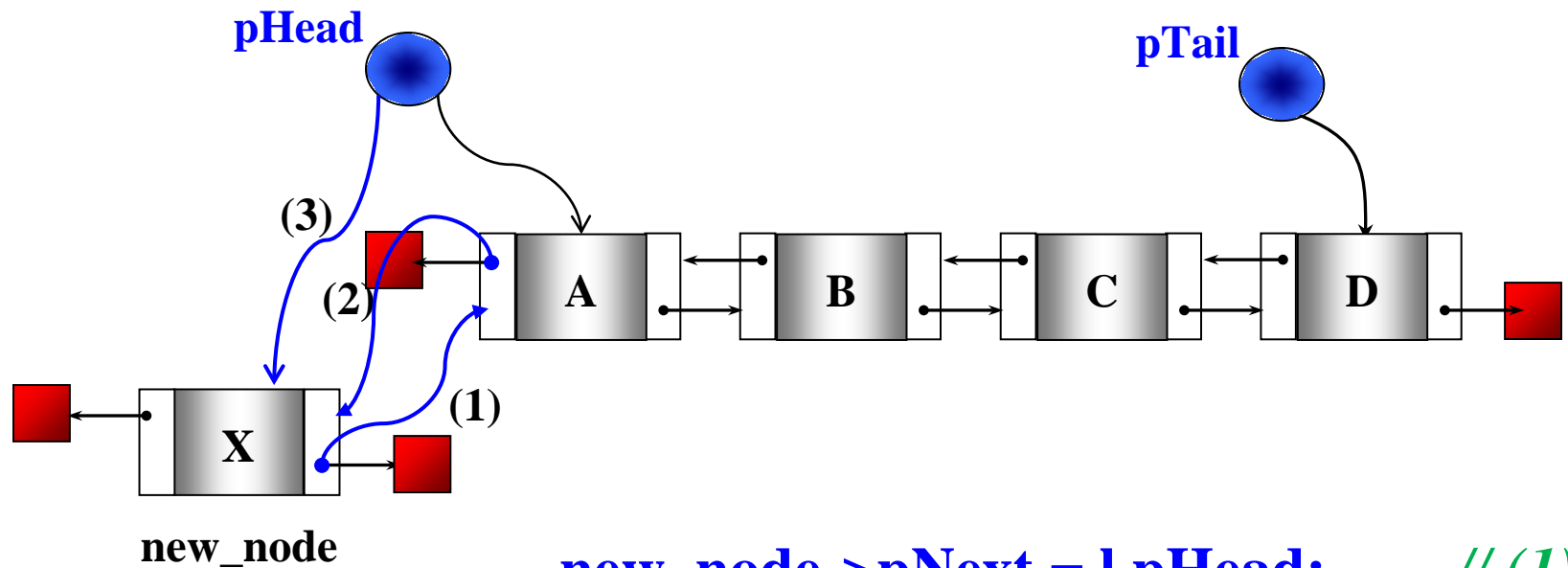
DSLK đôi – Thêm 1 nút vào ds

104

- 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

105



`new_node->pNext = l.pHead;` // (1)

`l.pHead->pPrev = new_node;` // (2)

`l.pHead = new_node;` // (3)

Cài đặt: Thêm vào đầu ds

106

```
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)

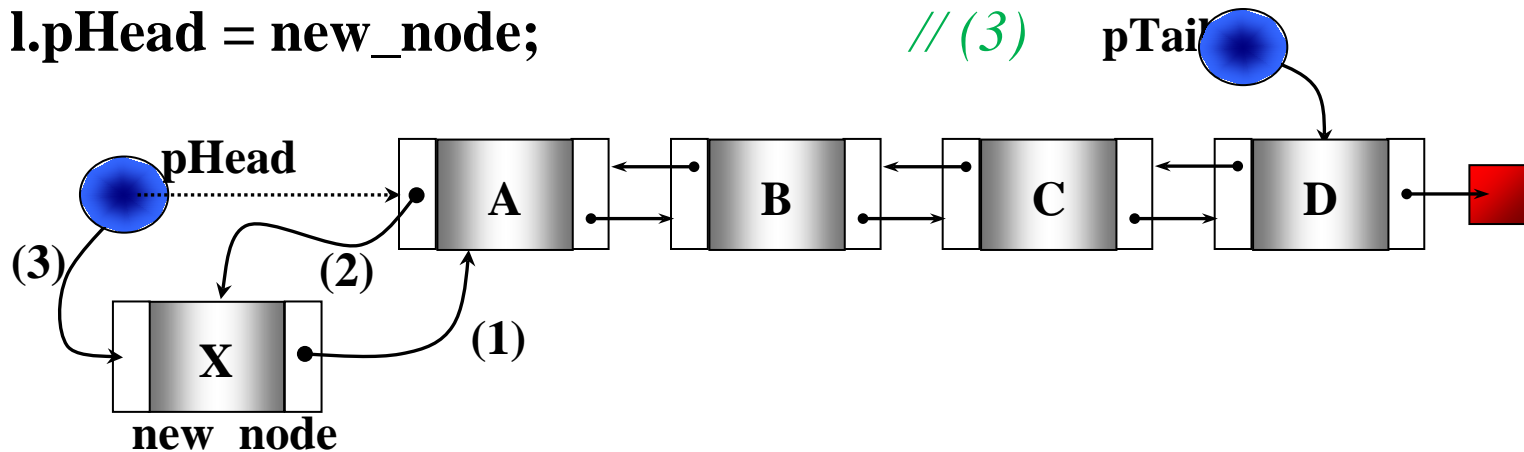
```
        l.pHead = new_node;
```

// (3)

```
    }
```

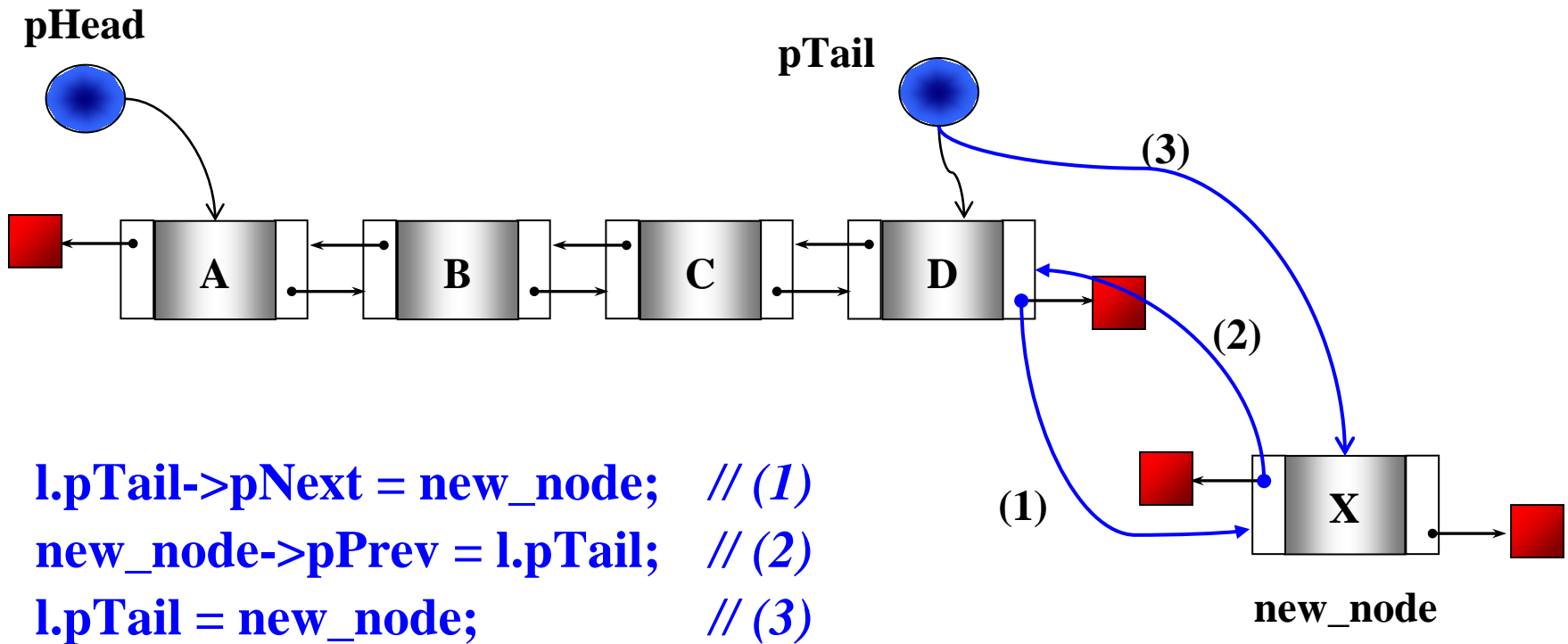
```
}
```

Gọi hàm??



Minh họa: Thêm vào cuối ds

108



Cài đặt – Thêm vào cuối ds

109

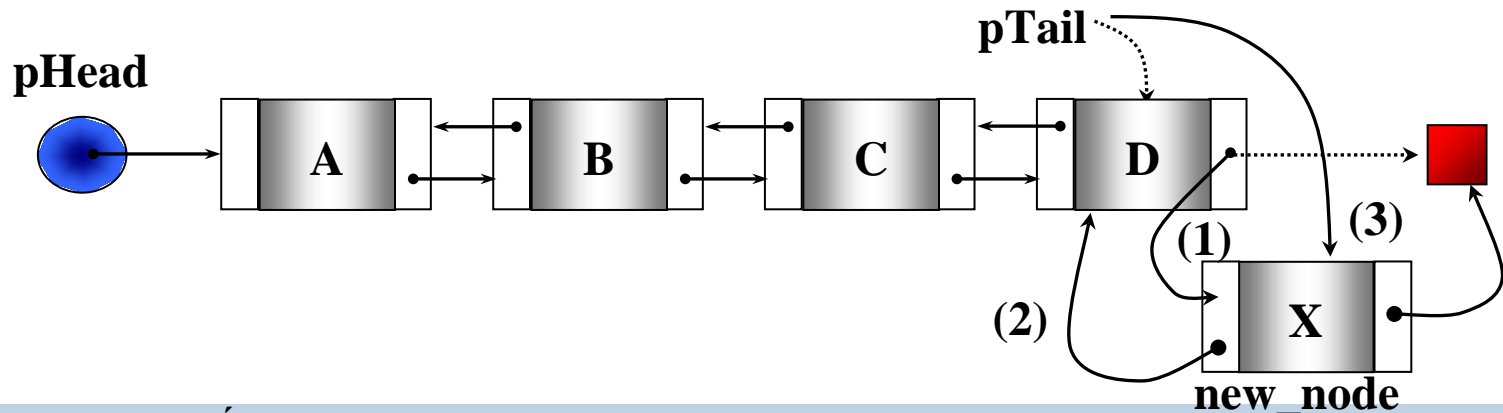
```
void addTail (DList &l, DNode *new_node)
{
    if (l.pHead==NULL)
        l.pHead = l.pTail = new_node;
    else
    {
        l.pTail->pNext = new_node;
        new_node->pPrev = l.pTail;
        l.pTail = new_node;
    }
}
```

Gọi hàm??

// (1)

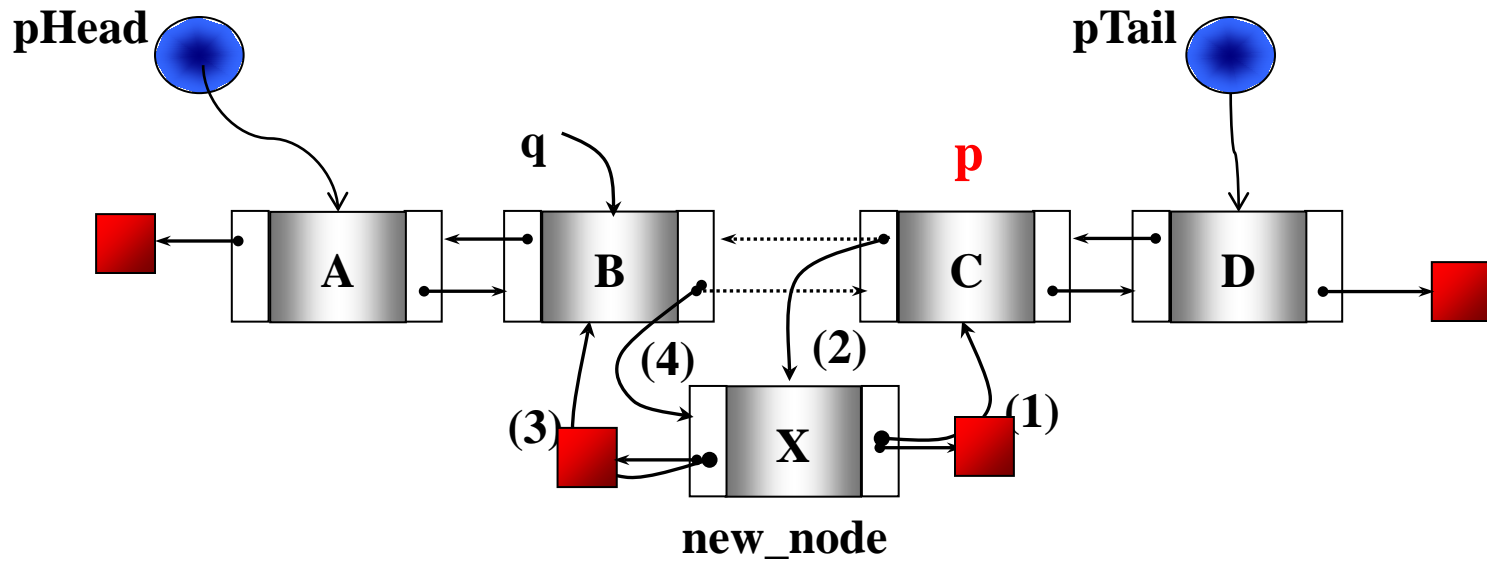
// (2)

// (3)



Minh họa: Chèn vào sau q

111



Cài đặt: Chèn vào sau q

112

```
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;
    }
    else
        addFirst (l, new_node); // chèn vào đầu ds
}
```



Gọi hàm??

//(1)

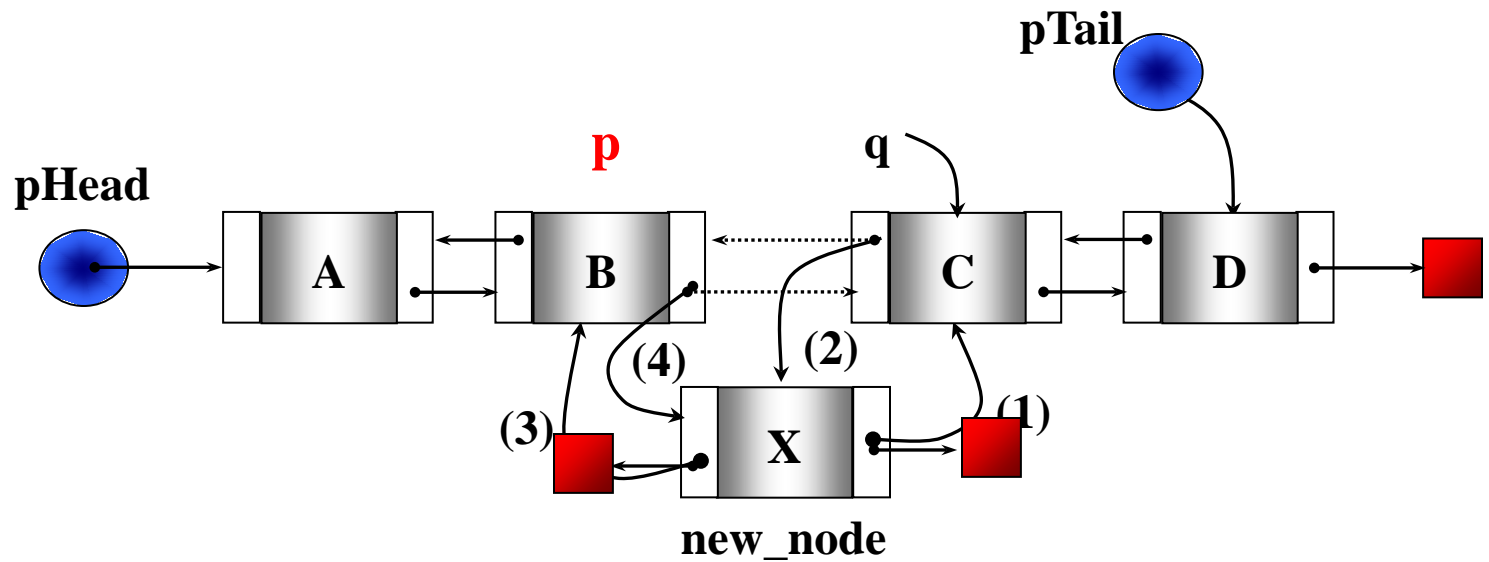
//(2)

//(3)

//(4)

Minh họa: Chèn vào trước q

114



Cài đặt: Chèn vào trước q

115

```
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;
    }
    else
        addTail (l, new_node); // chèn vào cuối ds
}
```



Gọi hàm??

//(1)

//(2)

//(3)

//(4)

addTail (l, new_node); // chèn vào cuối ds

DSLK đôi – Hủy phần tử

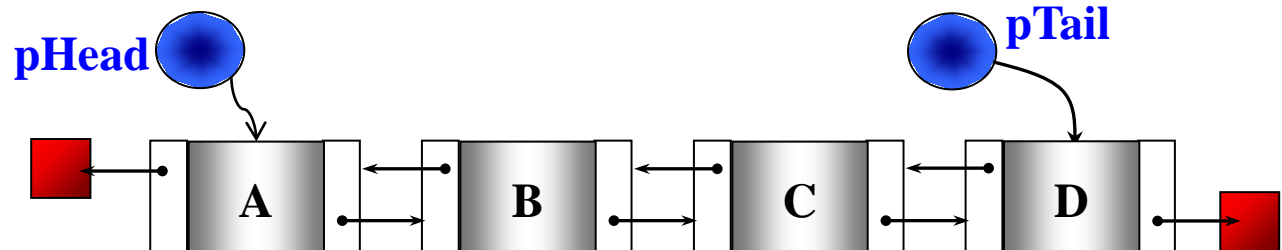
117

- Có 5 loại 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

118

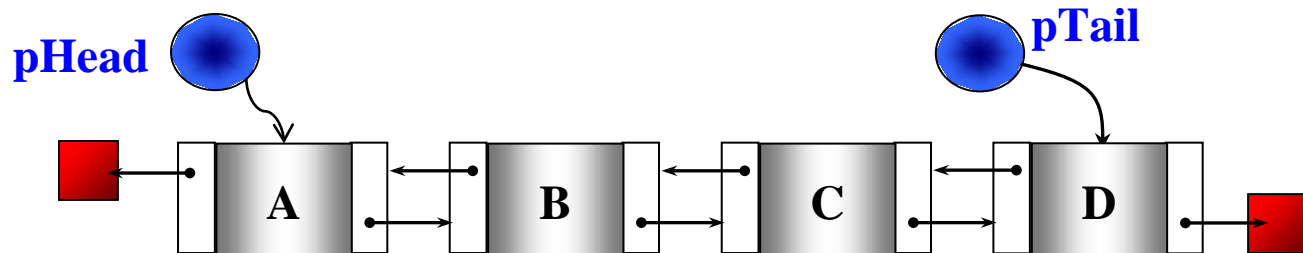
```
int removeHead (DList &l)
{
    if ( l.pHead == NULL)    return 0;
    DNode *p = l.pHead;
    l.pHead = l.pHead->pNext;
    delete p;
    if (l.pHead != NULL)      l.pHead->pPrev = NULL;
    else                      l.pTail = NULL;
    return 1;
}
```



DSLK đôi – Hủy cuối ds

119

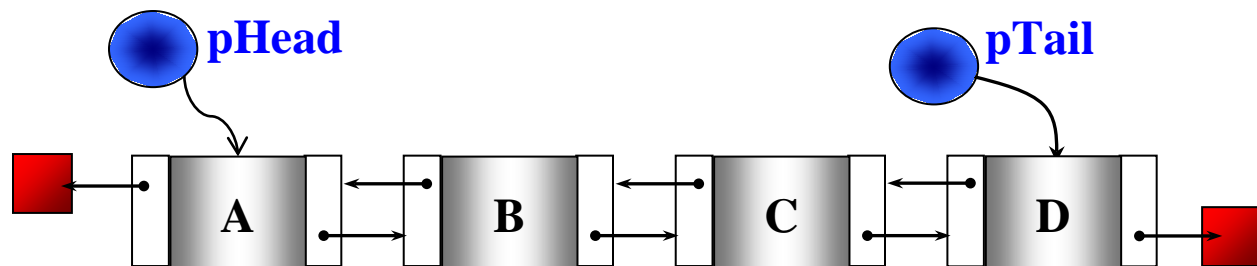
```
int removeTail (DList &l)
{
    if (l.pTail == NULL) return 0;
    DNode *p = l.pTail;
    l.pTail = l.pTail->pPrev;
    delete p;
    if (l.pTail != NULL) l.pTail->pNext = NULL;
    else        l.pHead = NULL;
    return 1;
}
```



DSLK đôi – Hủy phần tử sau q

120

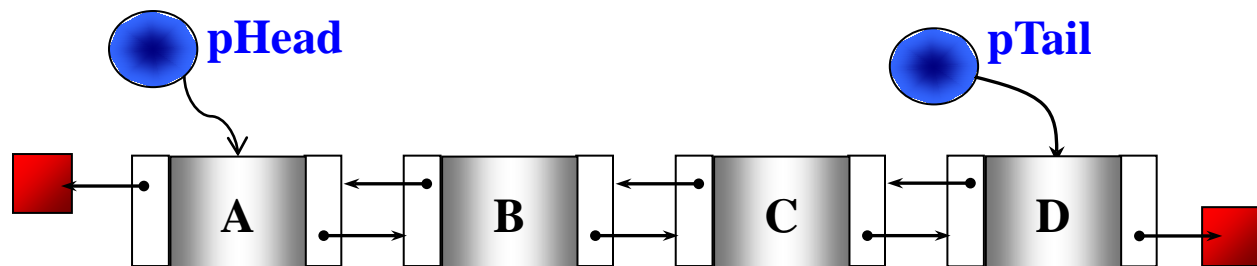
```
int removeAfter (DList &l, DNode *q)
{
    if (q == NULL) return 0;
    DNode *p = q ->pNext ;
    if (p != NULL)
    {
        q->pNext = p->pNext;
        if (p != l.pTail) p->pNext->pPrev = q;
        else
            l.pTail = q;
        delete p;
        return 1;
    }
    else return 0;
}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



DSLK đôi – Hủy phần tử trước q

121

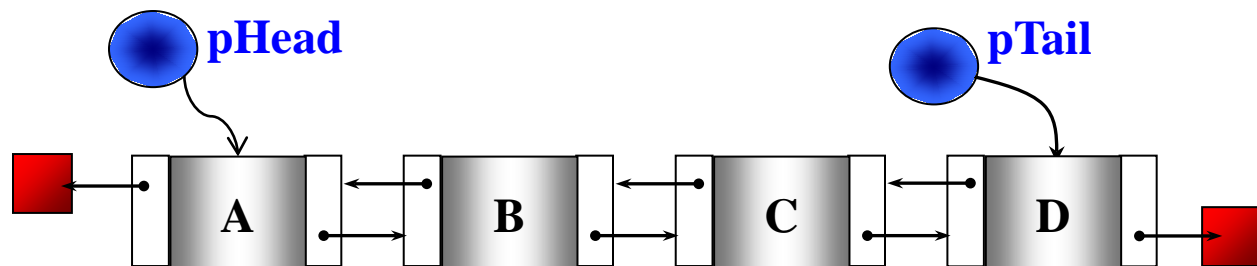
```
int removeBefore (DList &l, DNode *q)
{
    if (q == NULL) return 0;
    DNode *p = q ->pPrev;
    if (p != NULL)
    {
        q->pPrev = p->pPrev;
        if (p != l.pHead) p->pPrev->pNext = q;
        else l.pHead = q;
        delete p;
        return 1;
    }
    else return 0;
}
```



DSLK đôi – Hủy phần tử có khóa k

122

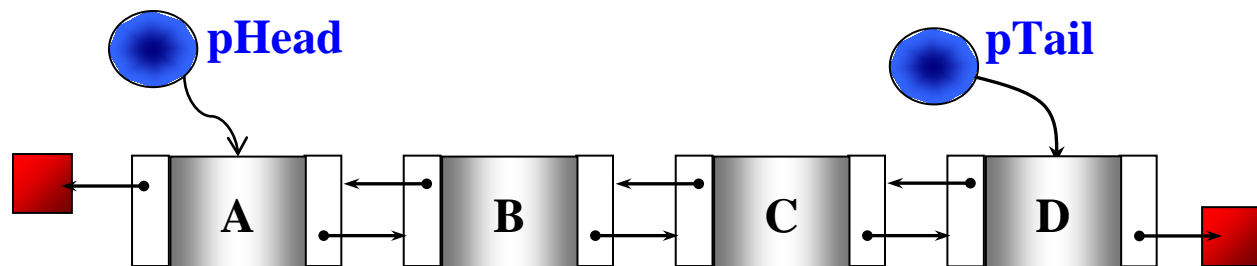
```
int removeNode (DList &l, int k)
{
    DNode *p = l.pHead;
    while (p != NULL)
    {
        if (p->data == k) break;
        p = p->pNext;
    }
}
```



DSLK đôi – Hủy phần tử có khóa k

123

```
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

124

- 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 mối 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

125

- 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)

126

- 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

127

- 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)

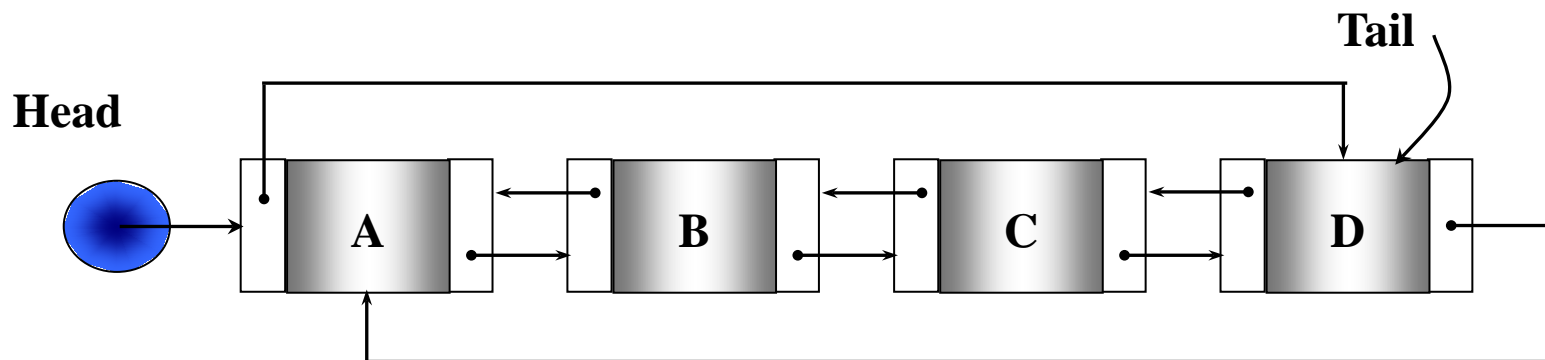
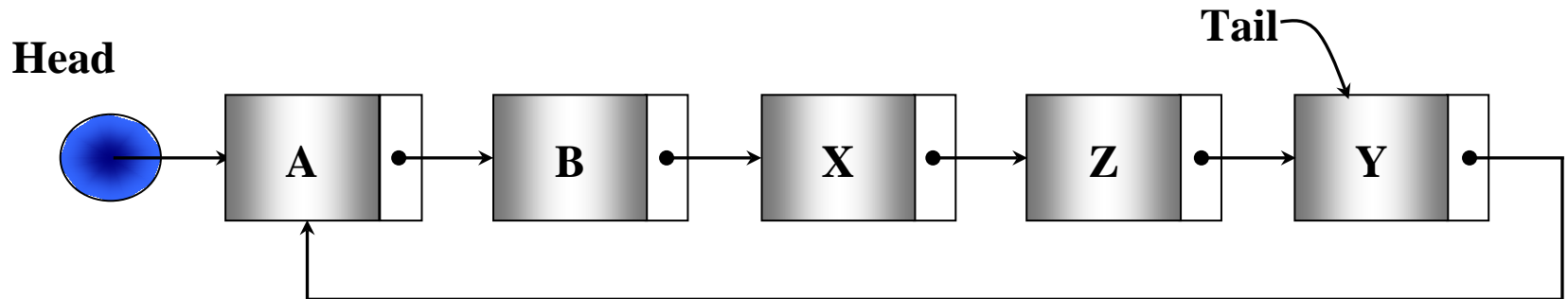
128

- Là một danh sách liên kết đơn (hoặc đôi) mà phần tử cuối danh sách, thay vì mang giá trị **NULL**, trở tới phần tử 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

129

- Để biểu diễn, có thể sử dụng các kỹ thuật biểu diễn như danh sách đơn (hoặc đôi)



DSLK vòng – Tìm kiếm

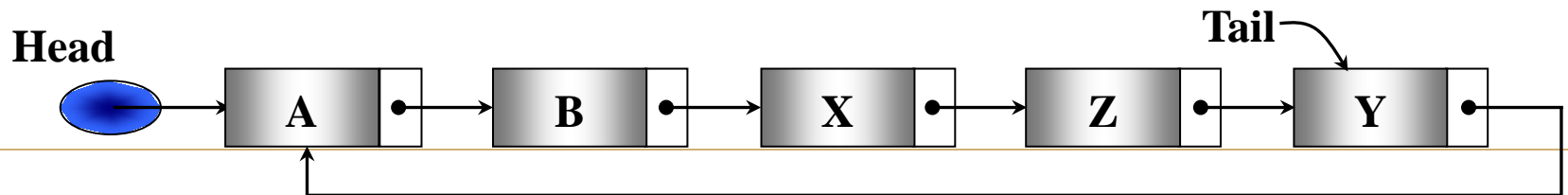
130

- 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

DSLK vòng – Tìm kiếm

131

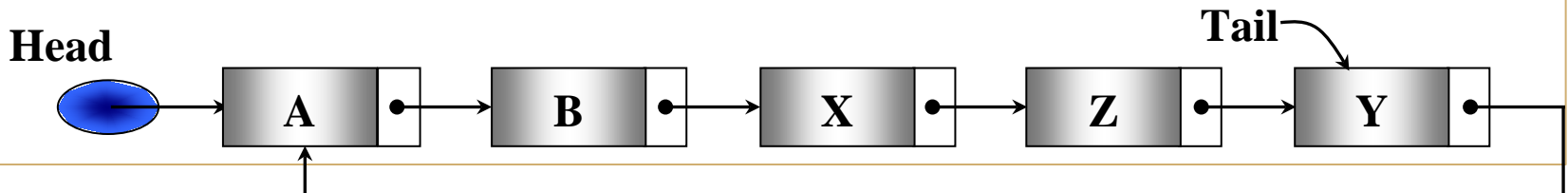
```
Node* Search (List &l, int x)
{
    Node *p = l.pHead;
    do{
        if (p->data== x) return p;
        p = p->pNext;
    } while (p != l.pHead);    // chưa đi giáp vòng
    return p;
}
```



DSLK vòng – Thêm vào đầu ds

132

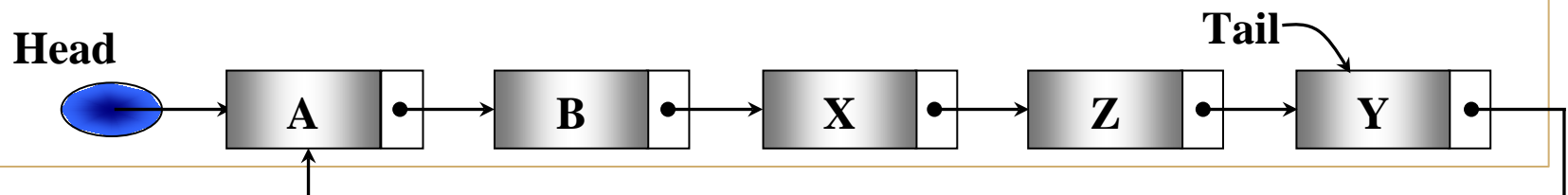
```
void addHead (List &l, Node *new_node)
{
    if (l.pHead == NULL) {
        l.pHead = l.pTail = new_node;
        l.pTail->pNext = l.pHead;
    }
    else{
        new_node->pNext = l.pHead;
        l.pTail->pNext = new_node;
        l.pHead = new_node;
    }
}
```



DSLK vòng – Thêm vào cuối ds

133

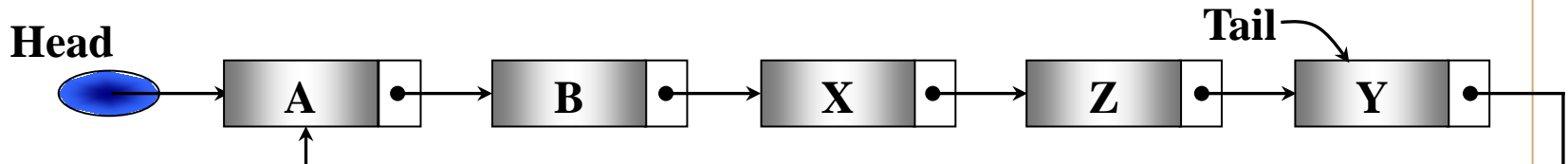
```
void addTail (List &l, Node *new_node)
{
    if (l.pHead == NULL) {
        l.pHead = l.pTail = new_node;
        l.pTail->pNext = l.pHead;
    }
    else{
        new_node->pNext = l.pHead;
        l.pTail->pNext = new_node;
        l.pTail = new_node;
    }
}
```



DSLK vòng – Thêm sau nút q

134

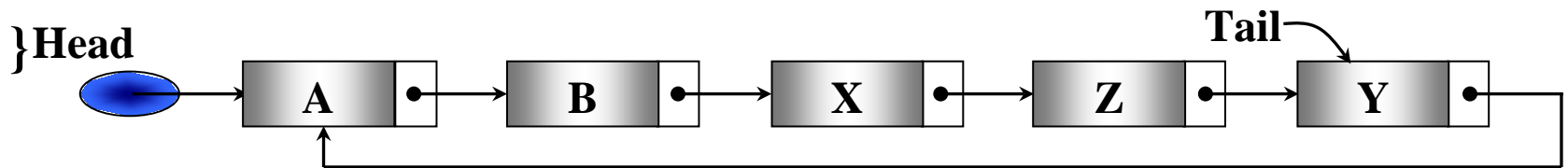
```
void addAfter (List &l, Node *q, Node *new_node)
{
    if (l.pHead == NULL) {
        l.pHead = l.pTail = new_node;
        l.pTail->pNext = l.pHead;
    }
    else{
        new_node->pNext = q->pNext;
        q->pNext = new_node;
        if (q == l.pTail)
            l.pTail = new_node;
    }
}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



DSLK vòng – Hủy nút đầu ds

135

```
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

136

```
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;
}
```

