

CHAPTER 3: 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

3

- **Kiểu dữ liệu tĩnh**
 - Khái niệm: Một số đối tượng dữ liệu không thay đổi được kích thước, cấu trúc, ... trong suốt quá trình sống. Các đối tượng dữ liệu thuộc những kiểu dữ liệu gọi là 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 thực, kiểu nguyên, kiểu ký tự ... hoặc từ các cấu trúc đơn giản như mảng 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

4

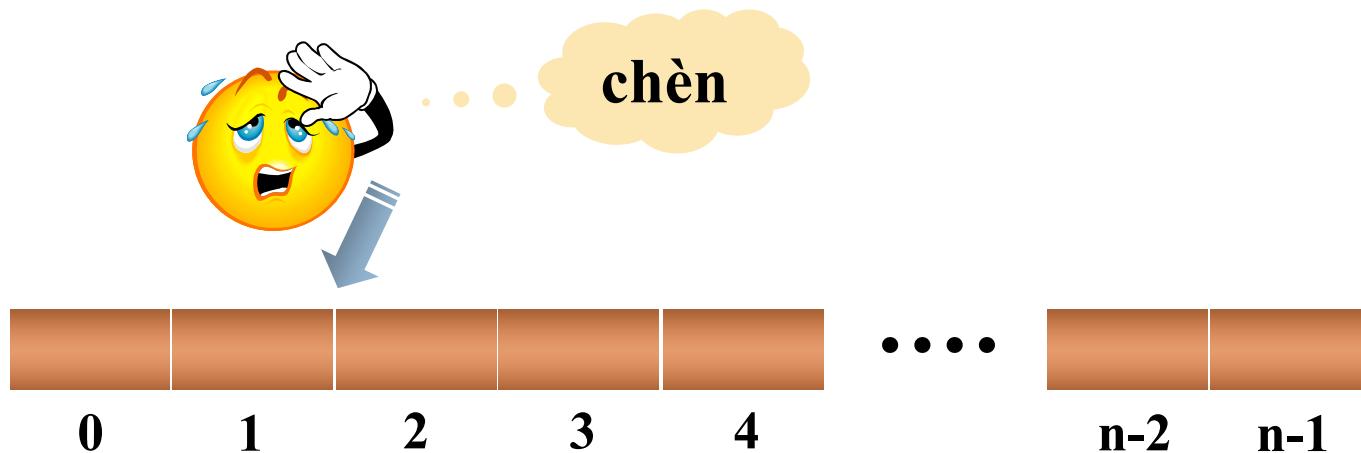
- Một số hạn chế của CTDL tinh
 - Một số đối tượng dữ liệu trong chu kỳ sống của nó có thể thay đổi về cấu trúc, độ lớn, như danh sách các học viên trong một lớp học có thể tăng thêm, giảm đi ... Nếu dùng những cấu trúc dữ liệu tinh đã biết như mảng để biểu diễn → Những thao tác phức tạp, kém tự nhiên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

5

□ Cấu trúc dữ liệu tĩnh: Ví dụ: Mảng 1 chiều

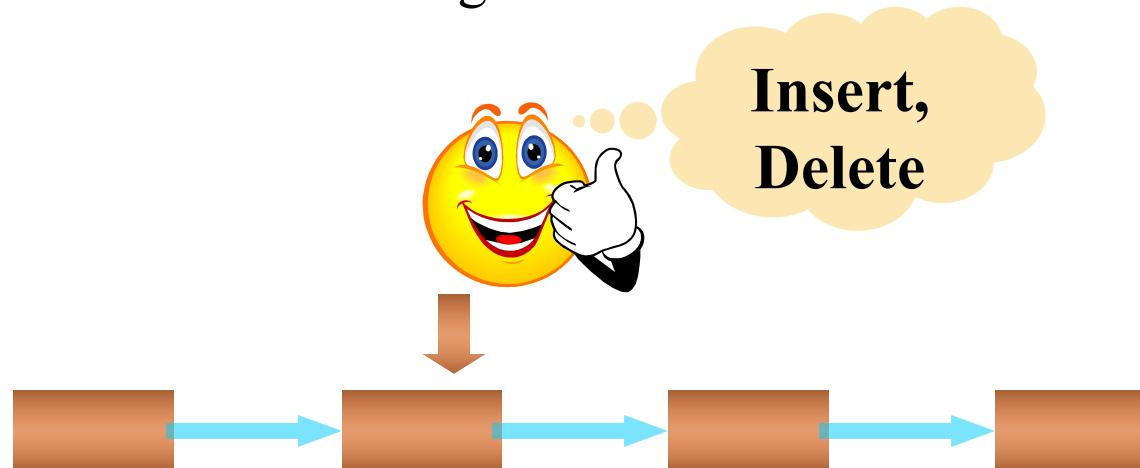
- Kích thước cố định (fixed size)
- Chèn 1 phần tử vào mảng rất khó
- Các phần tử tuân tự theo chỉ số $0 \Rightarrow n-1$
- Truy cập ngẫu nhiên (random access)



Giới thiệu

6

- **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
 - Thao tác thêm xoá đơn giản



Giới thiệu

7

- 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 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
 - ...

- 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

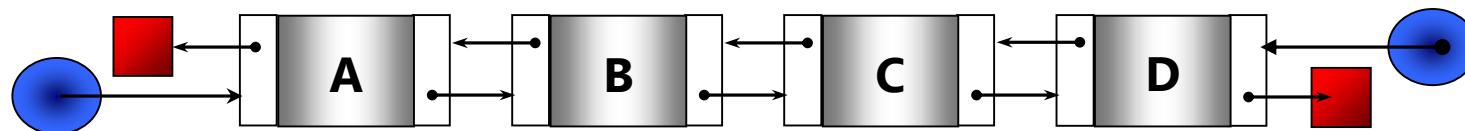
Giới thiệu

9

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



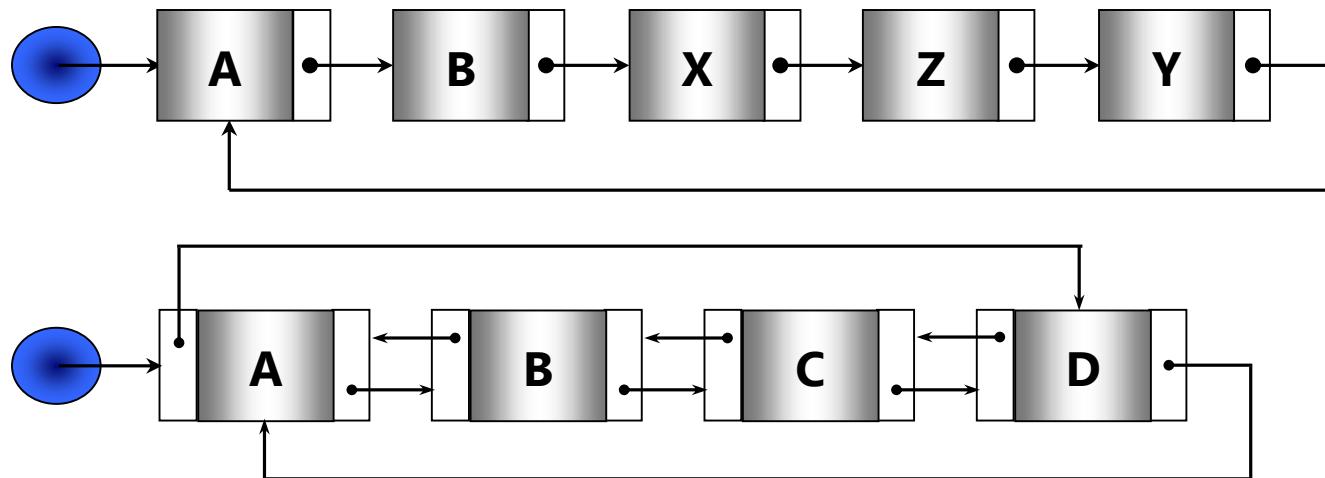
- **Danh sách liên kết đôi:** mỗi phần tử liên kết với các phần tử đứng trước và sau nó trong danh sách:



Giới thiệu

10

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



Nội dung

21

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

Danh sách liên kết đơn (DSLK đơn)

22

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

DSLK đơn – Khai báo

23

- 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*
- };



DSLK đơn – Khai báo

24

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

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

- Ví dụ 2: Định nghĩa một phần tử trong danh sách đơn lưu trữ hồ sơ **sinh viên**:

```
struct SinhVien {  
    char Ten[30];  
    int MaSV;  
};  
  
struct SVNode {  
    SinhVien data;  
    SVNode *pNext;  
};
```

DSLK đơn – Khai báo

25

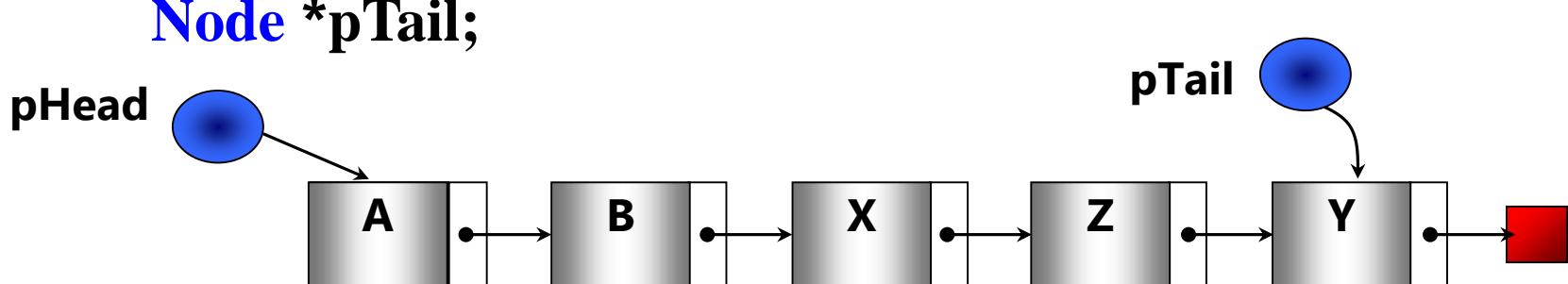
□ Tổ chức, quản lý:

- 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

26

- 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_bien;

DSLK đơn – Khai báo

27

□ Tạo một node mới

- Thủ tục **GetNode** để tạo ra một nút cho danh sách với thông tin chứa trong x

```
Node* GetNode (DataType x)
{
    Node *p;
    p = new Node; // Cấp phát vùng nhớ cho node
    if (p==NULL)
    {
        cout<<“Khong du bo nho!”; 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)

29

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

DSLK đơn

30

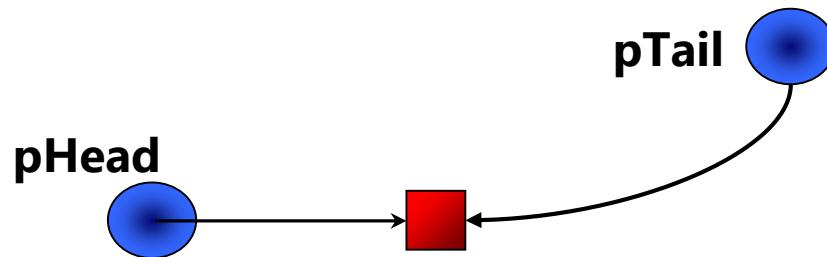
□ Các thao tác cơ bản

- 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

31

- Tạo danh sách rỗng



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

DSLK đơn

32

□ 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

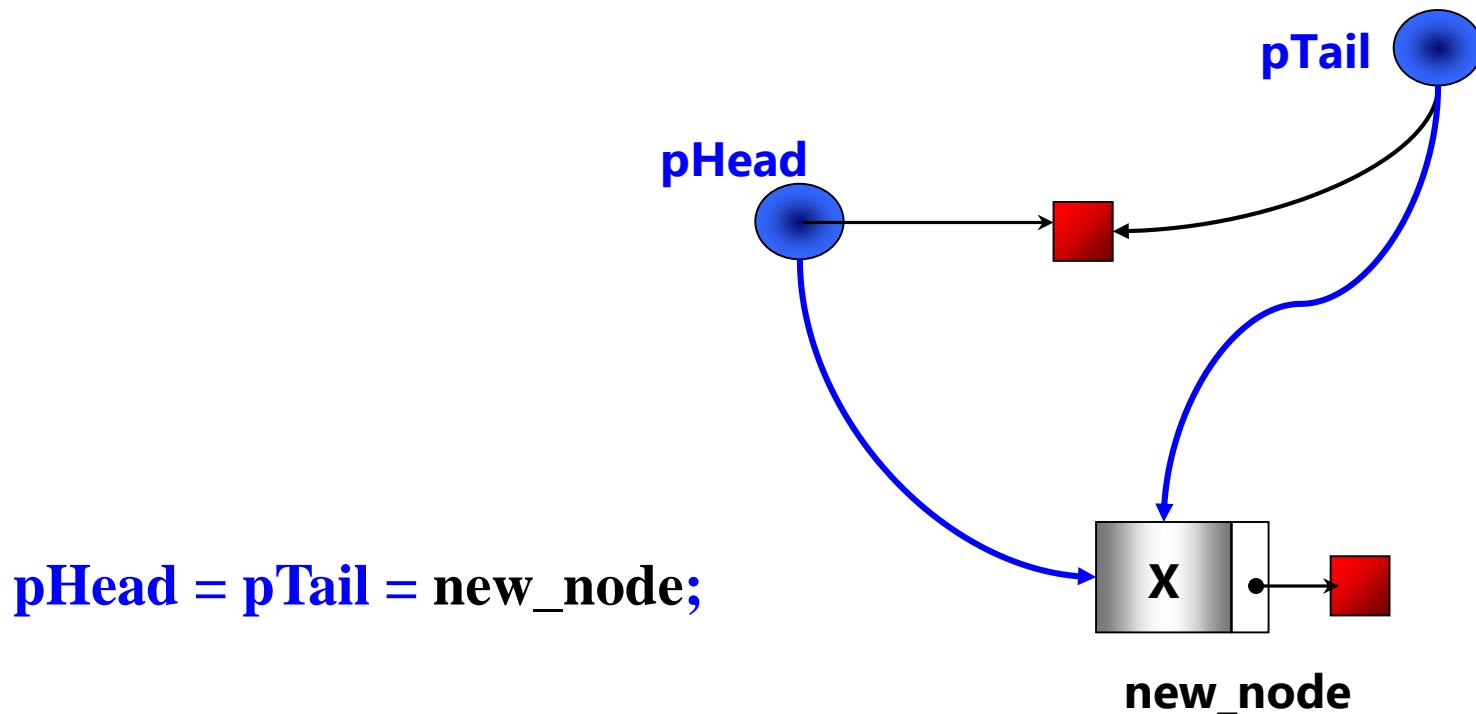
33

- **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

34

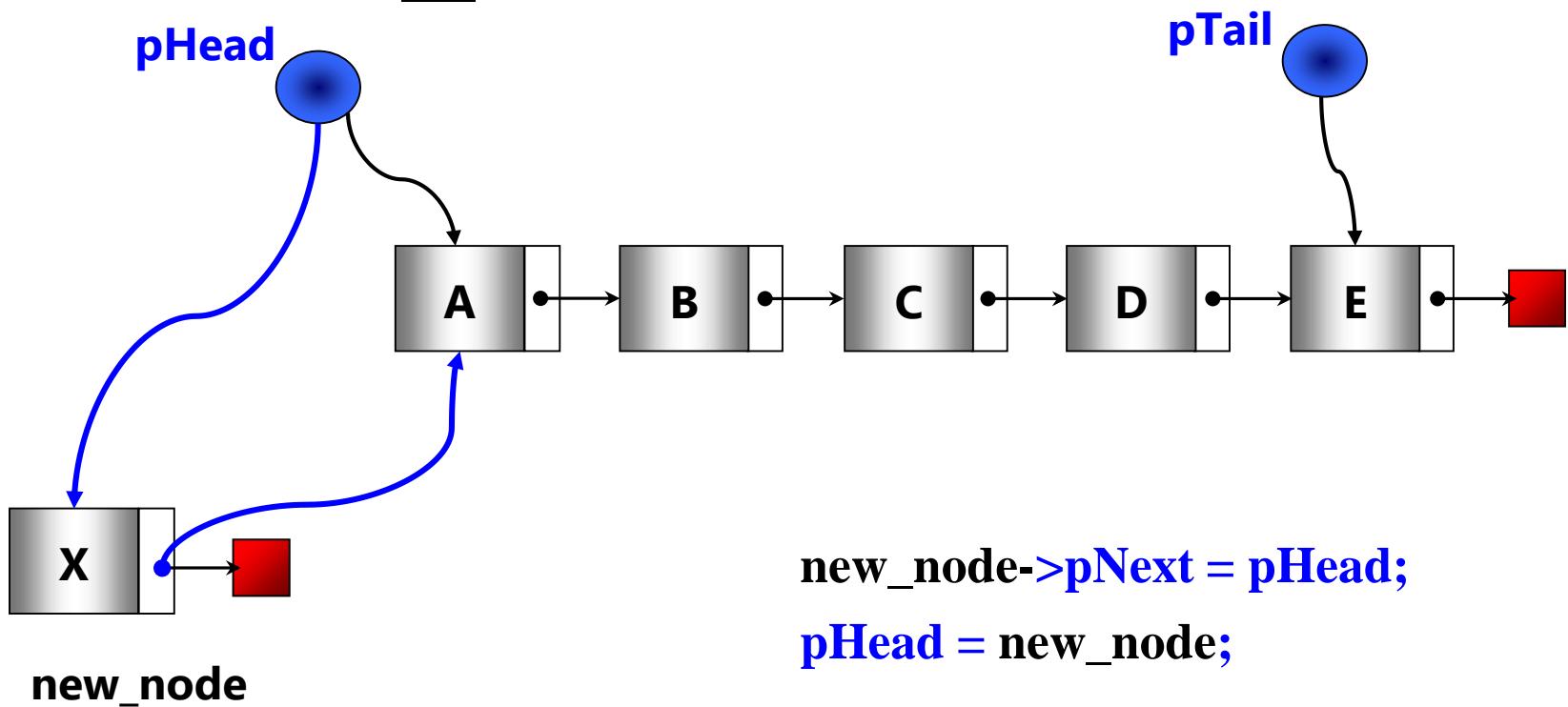
- Thêm một phần tử
 - Nếu danh sách ban đầu rỗng



DSLK đơn – Các thao tác cơ sở

35

- Thêm một phần tử
 - Nếu danh sách ban đầu không rỗng:
 - Gắn node vào đầu danh sách



DSLK đơn – Các thao tác cơ sở

36

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
 - pHad = pTail = new_node;
- Ngược lại
 - new_node->pNext = pHad;
 - pHad = new_node;

DSLK đơn – Các thao tác cơ sở

37

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

38

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

39

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(l, new_node);
```

DSLK đơn – Các thao tác cơ sở

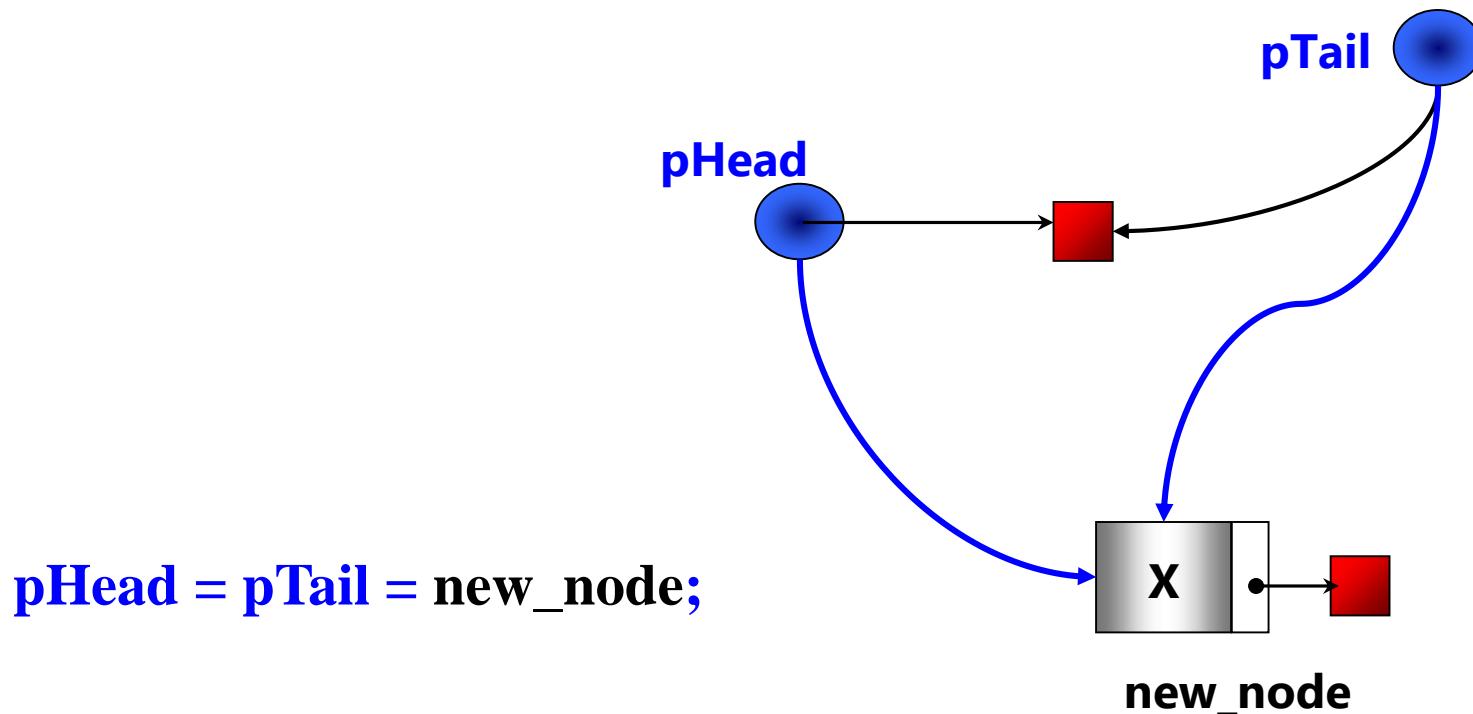
40

- **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

41

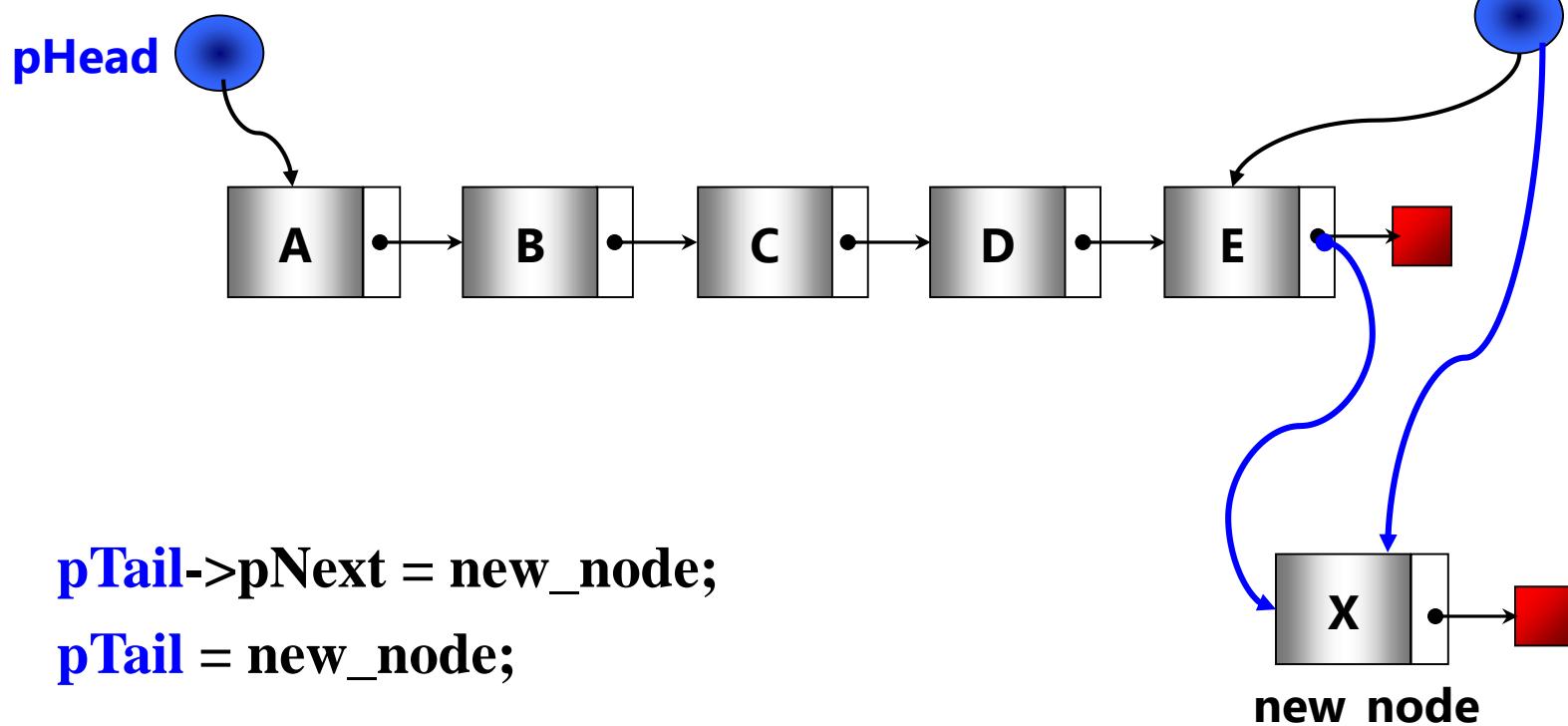
- Thêm một phần tử
 - Nếu danh sách ban đầu rỗng



DSLK đơn – Các thao tác cơ sở

42

- Thêm một phần tử
 - Nếu danh sách ban đầu không rỗng:
 - Gắn node vào cuối danh sách:



DSLK đơn – Các thao tác cơ sở

43

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
 - pHHead = pTail = new_node;
- Ngược lại
 - pTail->pNext = new_node ;
 - pTail = new_node;

DSLK đơn – Các thao tác cơ sở

44

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

45

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

46

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

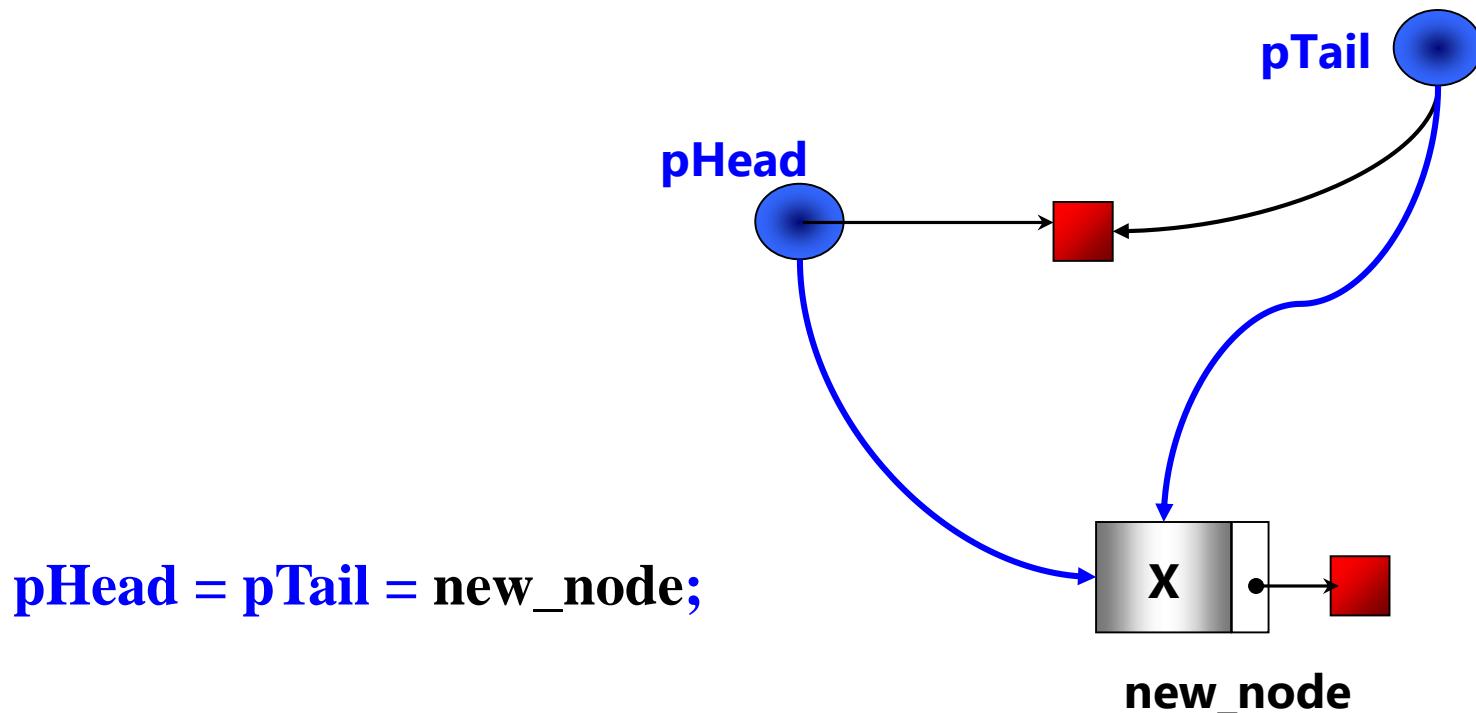
47

- **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

48

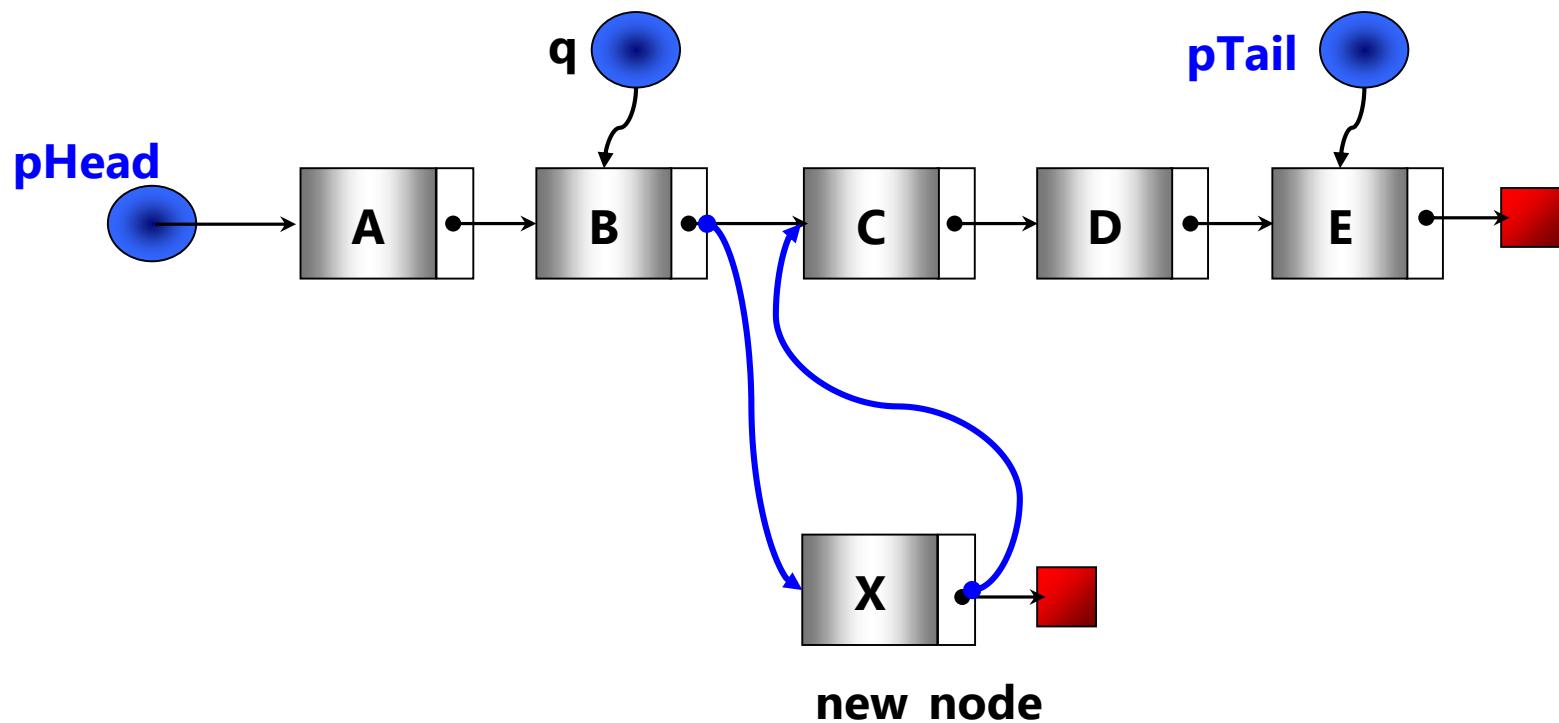
- Thêm một phần tử
 - Nếu danh sách ban đầu rỗng



DSLK đơn – Các thao tác cơ sở

49

- Thêm một phần tử
 - Nếu danh sách ban đầu rỗng
 - Chèn một phần tử sau q



DSLK đơn – Các thao tác cơ sở

50

Thuật toán: Chèn một phần tử sau q

// 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

51

Cài đặt: Chèn một phần tử sau 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

52

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

53

□ 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

54

□ Duyệt danh sách

- Là thao tác thường được thực hiện khi có nhu cầu xử lý các phần tử của danh sách theo cùng một cách thức hoặc khi cần lấy thông tin tổng hợp từ các phần tử của danh sách như:
 - Đếm các phần tử của danh sách
 - Tìm tất cả các phần tử thoả điều kiện
 - Hủy toàn bộ danh sách (và giải phóng bộ nhớ)
 - ...

DSLK đơn – Các thao tác cơ sở

55

□ Duyệt danh sách

- Bước 1: p = pHead; //Cho p trỏ đến phần tử đầu danh sách
- Bước 2: Trong khi (Danh sách chưa hết) thực hiện:
 - B2.1 : Xử lý phần tử p
 - B2.2 : p=p->pNext; // Cho p trỏ 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;
    }
}
```

DSLK đơn – Các thao tác cơ sở

56

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

59

□ 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

60

- Tìm kiếm một phần tử có khóa x

```
Node* Search (List l, int x)
```

```
{
```

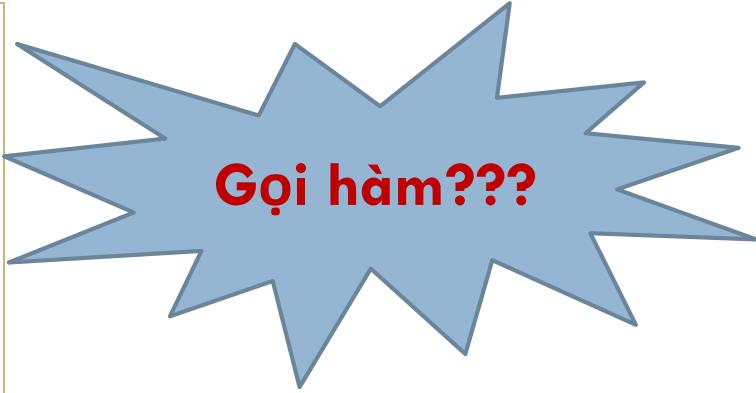
```
    Node* p = l.pHead;  
    while (p!=NULL) {  
        if (p->data==x)  
            return p;
```

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

```
}
```

```
    return NULL;
```

```
}
```



Gọi hàm???

DSLK đơn

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

- 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

63

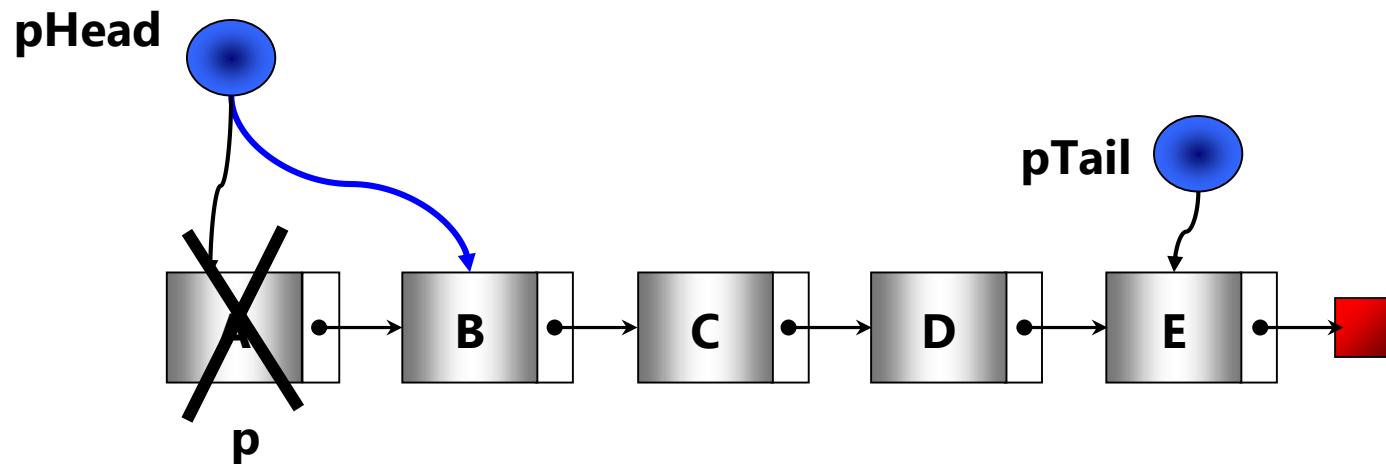
Xóa node đầu của danh sách

- ❑ Gọi p là node đầu của danh sách (pHead)
- ❑ Cho pHead trỏ vào node sau node p (là p->pNext)
- ❑ Nếu danh sách trở thành rỗng thì pTail = NULL
- ❑ Giải phóng vùng nhớ mà p trỏ tới

DSLK đơn – Các thao tác cơ sở

64

- Xóa một node của danh sách



`l.pHead = p->pNext;`

`delete p;`

DSLK đơn – Các thao tác cơ sở

65

```
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

66

- 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

67

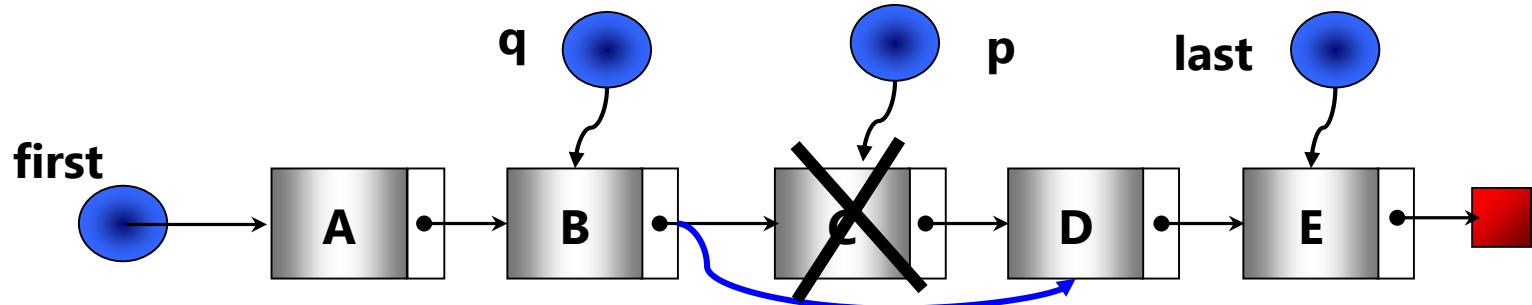
Xóa node sau node q trong danh sách

- Điều kiện để có thể xóa được node sau q là:
 - q phải khác NULL (`q !=NULL`)
 - Node sau q phải khác NULL (`q->pNext !=NULL`)
- Có các thao tác:
 - Gọi `p` là node sau `q`
 - Cho vùng `pNext` của `q` trỏ vào node đứng sau `p`
 - Nếu `p` là phần tử cuối thì `pTail` trỏ vào `q`
 - Giải phóng vùng nhớ mà `p` trỏ tới

DSLK đơn – Các thao tác cơ sở

68

Xóa node sau node q trong danh sách



$q->pNext = p->pNext;$

delete p;

DSLK đơn – Các thao tác cơ sở

69

Xóa node sau node q trong danh sách

```
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

70

- 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

71

- **Thuật toán: Hủy 1 phần tử có khoá k**
 - Bước 1:
 - Tìm phần tử **p** có khóa k và phần tử **q** đứng trước nó
 - Bước 2:
 - Nếu $(p \neq \text{NULL})$ thì // tìm thấy k
 - Hủy **p** ra khỏi ds; tương tự hủy phần tử sau q;
 - Ngược lại
 - Báo không có k

DSLK đơn – Các thao tác cơ sở

72

- Cài đặt:
Hủy 1
phần tử
có khoá
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

73

□ 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

74

□ 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 (Danh sách chưa hết) thực hiện:

- B1.1:

- $p = pHead;$
 - $pHead = pHead ->pNext;$ // Cho *p* trỏ tới phần tử kế

- B1.2:

- Hủy *p*;

- Bước 2:

- $pTail = NULL;$ // Bảo đảm tính nhất quán khi xâu rỗng

DSLK đơn – Các thao tác cơ sở

75

- Hủy toàn bộ danh sách: cài đặt

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



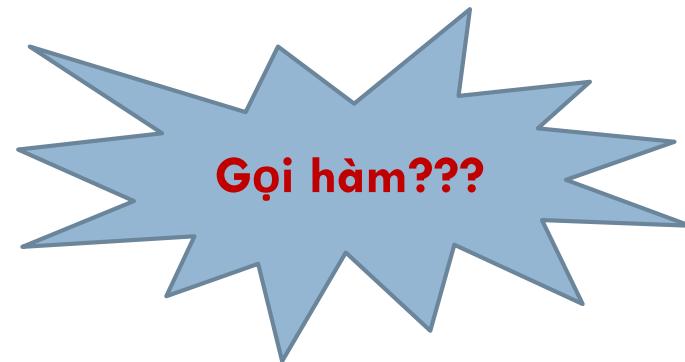
Gọi hàm???

DSLK đơn – Các thao tác cơ sở

76

- 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;
}
```



DSLK đơn – Các thao tác cơ sở

77

- Trích phần tử đầu danh sách

```
Node* PickHead (List &l)
```

```
{
```

```
    Node *p = NULL;
```

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

```
        p = l.pHead;
```

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

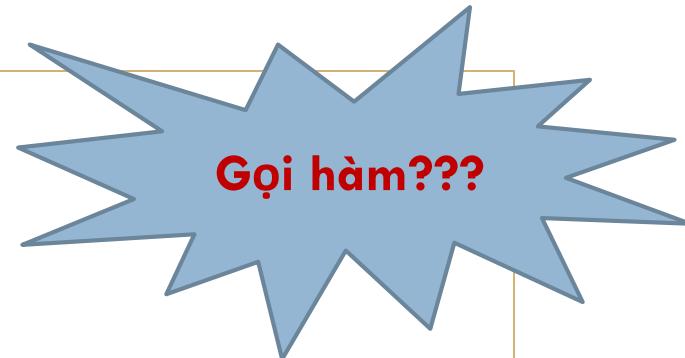
```
        p->pNext = NULL;
```

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

```
}
```

```
    return p;
```

```
}
```



Exercise

78

- Write a program for buiding single linked list (Display menu)
 - Add one node at first
 - Add one node at last
 - Add many node at first
 - Add many node at last
 - Add one node after select node
 - Display List
 - Find one node
 - Select and display n(th) node
 - Display node count
 - Remove one node
 - Remove List
 - Get sum of all nodes
 - Inserting a new node in a sorted list

Danh sách liên kết đơn (DSLK đơn)

79

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

Nội dung

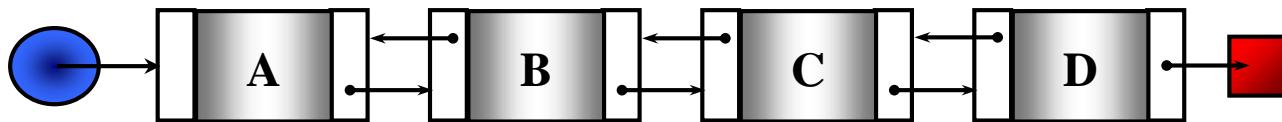
114

- 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)

115

- ☐ Là danh sách mà mỗi phần tử trong danh sách có kết nối với 1 phần tử đứng trước và 1 phần tử đứng sau nó



DSLK đôi – Khai báo cấu trúc

116

- Dùng hai con trỏ:
 - pPrev liên kết với phần tử đứng trước
 - pNext liên kết với phần tử đứng sau

```
struct DNode
{
    DataType data;
    DNode* pPre;           // 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

117

- Hàm tạo nút:

DNode* getNode (**DataType** x)

{

DNode *p;

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

if (p==**NULL**) {

cout<<“Khong du bo nho”; **return** **NULL**;

}

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

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

return p;

}



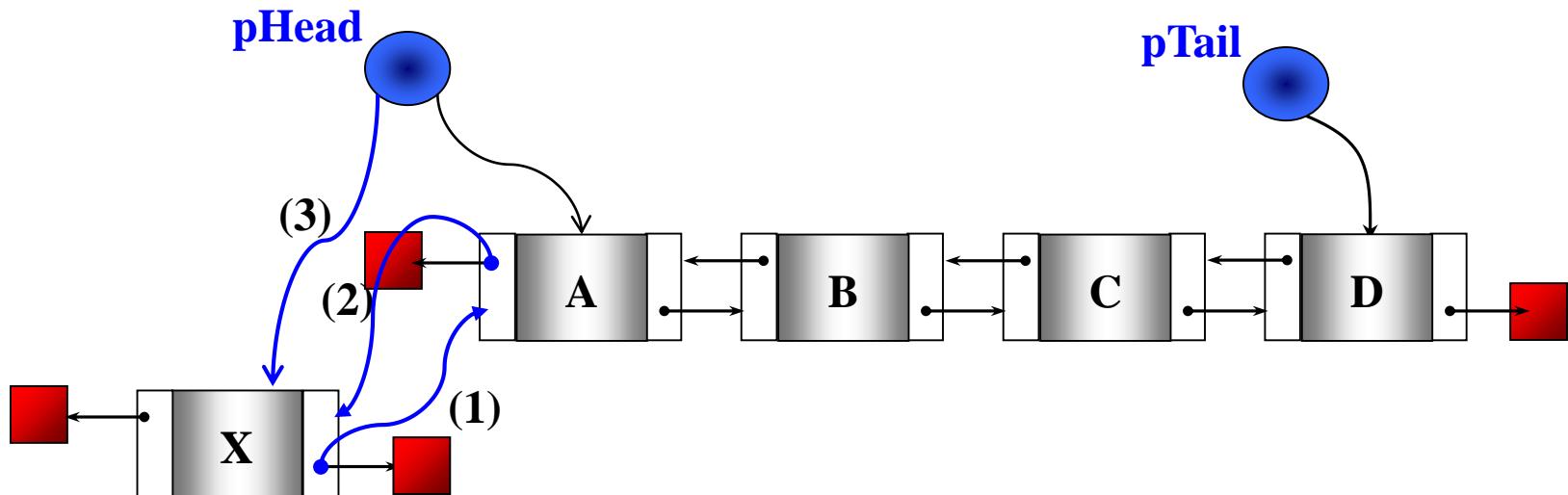
DSLK đôi – Thêm 1 nút vào ds

118

- Có 4 loại thao tác chèn new_node vào danh sách:
 - Cách 1: Chèn vào đầu danh sách
 - Cách 2: Chèn vào cuối danh sách
 - Cách 3 : Chèn vào danh sách sau một phần tử q
 - Cách 4 : Chèn vào danh sách trước một phần tử q

DSLK đôi – Thêm vào đầu ds

119



new_node

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

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

l.pHead = new_node; // (3)

DSLK đôi – Thêm vào đầu ds

120

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

```
{
```

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

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

```
else
```

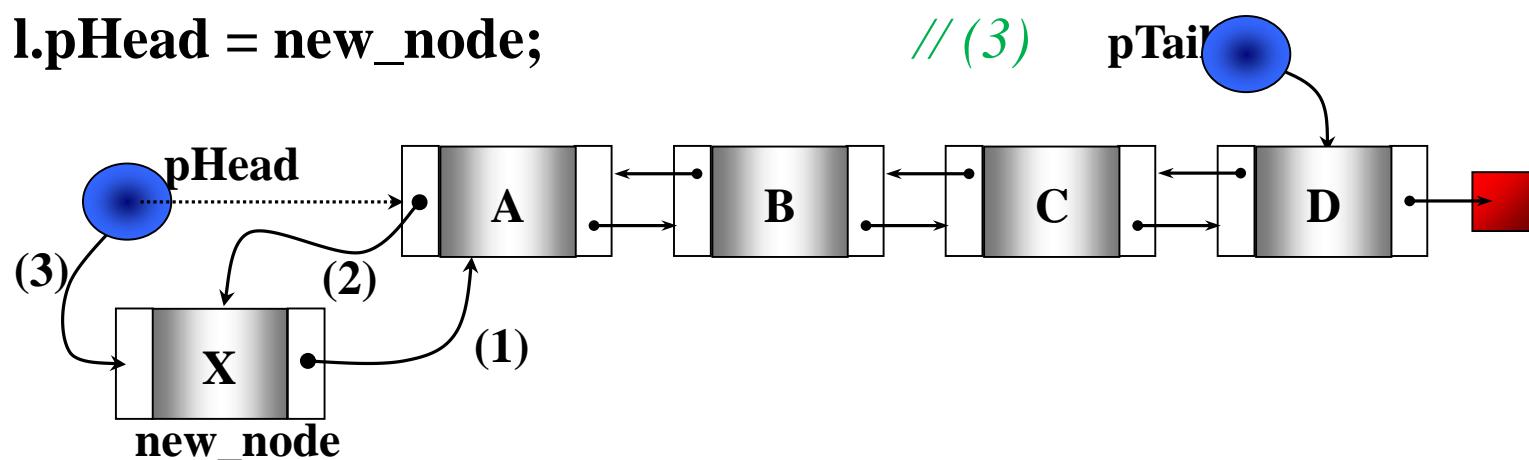
```
    { new_node->pNext = l.pHead; // (1)
```

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

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

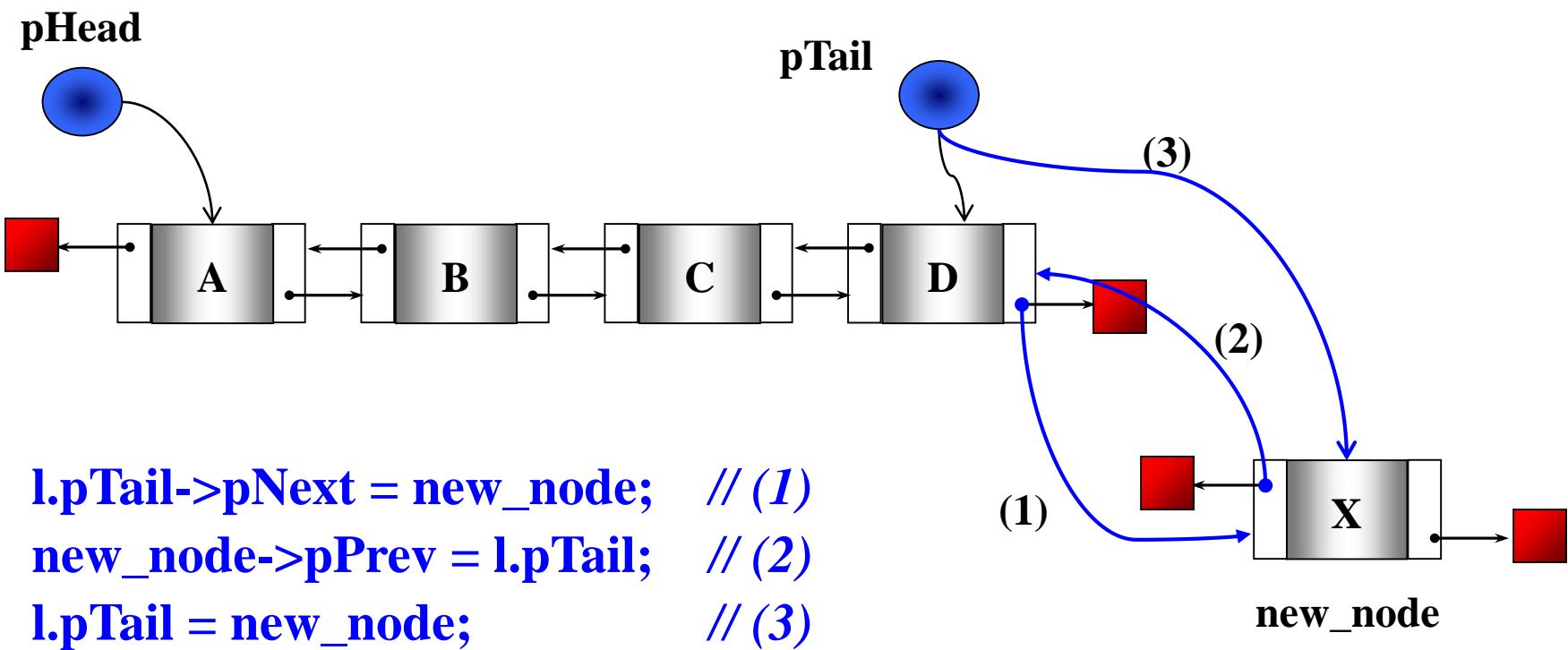
```
}
```

```
}
```



DSLK đôi – Thêm vào cuối ds

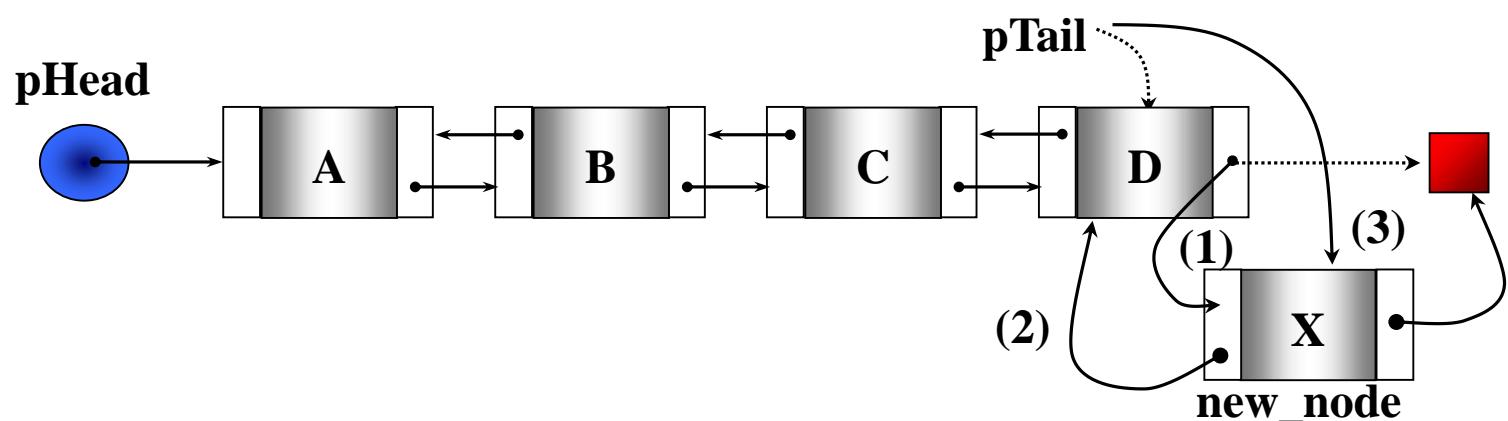
122



DSLK đôi – Thêm vào cuối ds

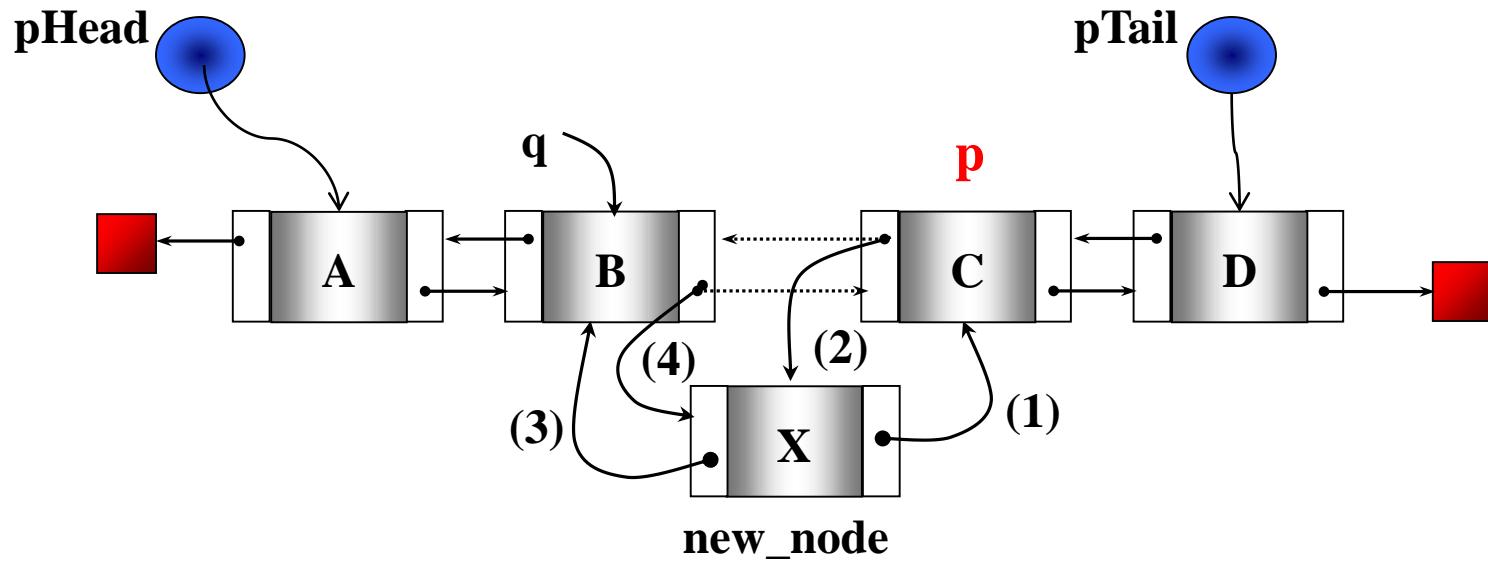
123

```
void addTail (DList &l, DNode *new_node)
{
    if (l.pHead==NULL)
        l.pHead = l.pTail = new_node;
    else
    {
        l.pTail->pNext = new_node;           // (1)
        new_node->pPrev = l.pTail;          // (2)
        l.pTail = new_node;                 // (3)
    }
}
```



DSLK đôi – Chèn vào sau q

125



DSLK đôi – Chèn vào sau q

126

```
void addAfter (DList &l, DNode *q, DNode *new_node)
```

```
{
```

```
    if (q!=NULL) {
```

```
        DNode *p = q->pNext;
```

```
        new_node->pNext = p; // (1)
```

```
        new_node->pPrev = q; // (3)
```

```
        q->pNext = new_node; // (4)
```

```
        if (p != NULL) p->pPrev = new_node; // (2)
```

```
        else l.pTail = new_node;// if (q == l.pTail)
```

```
}
```

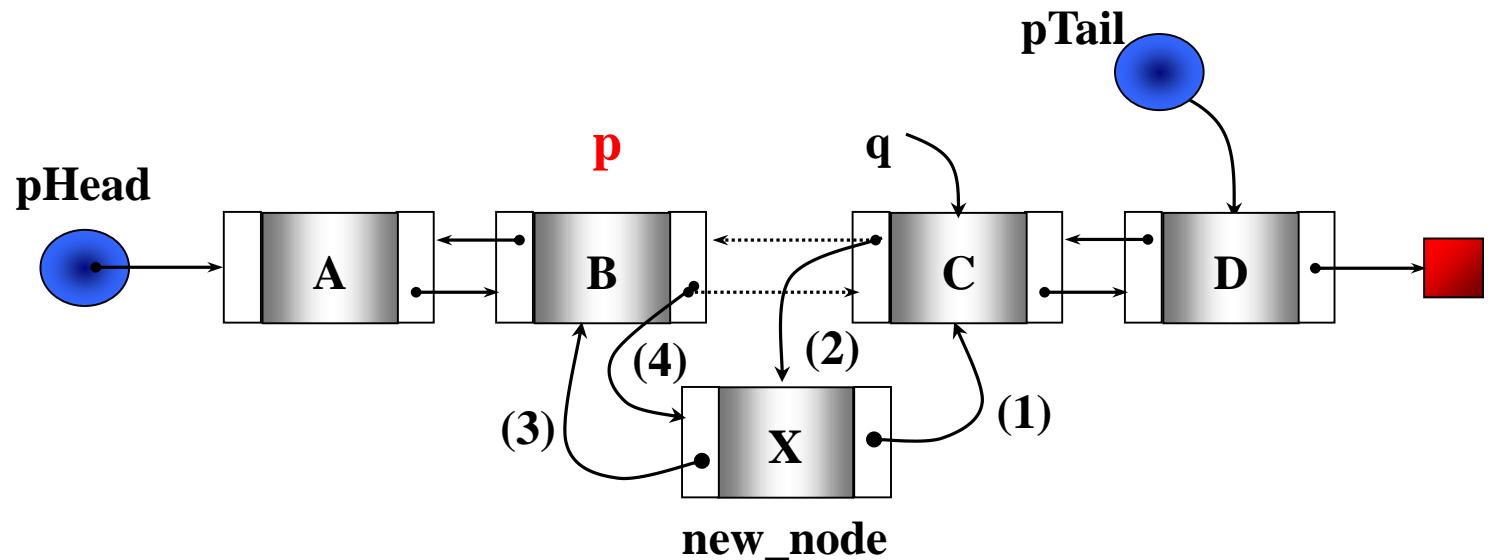
```
}
```



Gọi hàm??

DSLK đôi – Chèn vào trước q

128



DSLK đôi – Chèn vào trước q

129

```
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)
        else l.pHead = new_node; // (q == l.pHead)
    }
}
```



}

DSLK đôi – Hủy phần tử

131

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

DSLK đôi – Hủy đầu ds

132

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

DSLK đôi – Hủy cuối ds

133

```
int removeTail (DList &l)
{
    if (l.pTail == NULL) return 0;
    DNode *p = l.pTail;
    l.pTail = l.pTail->pPrev;

    if (l.pTail == NULL) l.pHead = NULL;
    else l.pTail->pNext = NULL;
    delete p;
    return 1;
}
```

DSLK đôi – Hủy phần tử sau q

134

```
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

135

```
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

136

```
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

137

```
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

138

- 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 $O(n)$

DSLK đôi – Nhận xét

139

- ❑ 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

Nội dung

140

- 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)

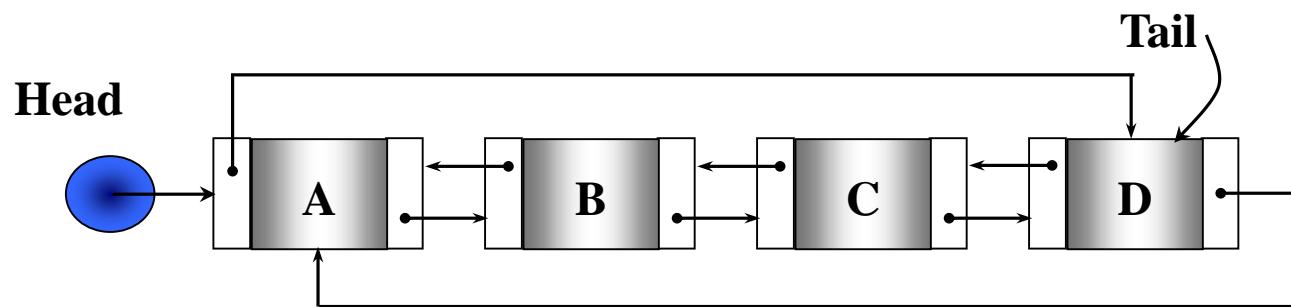
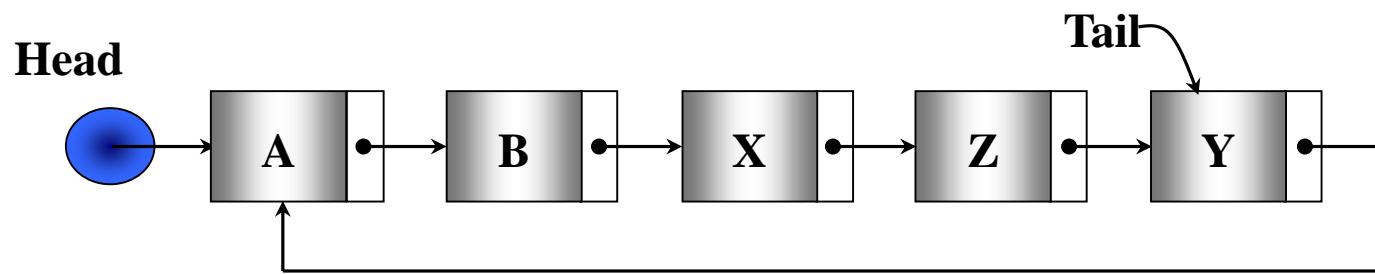
141

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

142

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



DSLK vòng – Tìm kiếm

143

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

DSLK vòng – Tìm kiếm

144

Node* Search (List l, int x)

{

Node *p = l.pHead;

do{

if (p->data== x) return p;

p = p->pNext;

} while (p != l.pHead); // chưa đi giáp vòng

return NULL;

}

DSLK vòng – Thêm vào đầu ds

145

```
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

146

```
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

147

```
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

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;
}
```