



DANH SÁCH LIÊN KẾT ĐƠN



Tổ chức DSLK đơn



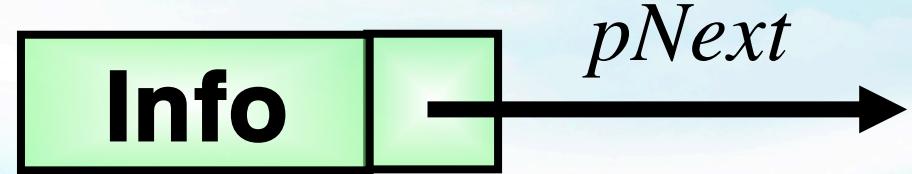
- Mỗi phần tử liên kết với phần tử đứng liền sau trong danh sách
- Cấu trúc của một phần tử gồm có hai thành phần:
 - **Thành phần dữ liệu:** Lưu trữ thông tin của phần tử
 - **Thành phần liên kết:**
 - Lưu địa chỉ phần tử đứng sau trong danh sách
 - hoặc bằng NULL nếu là phần tử cuối danh sách.



Định nghĩa CTDL DSLK đơn

- CTDL của 1 nút trong List đơn

```
typedef struct tagNode
{ Data     Info; // Lưu thông tin
  struct tagNode *pNext; // Lưu địa chỉ của Node đứng sau
}Node;
```

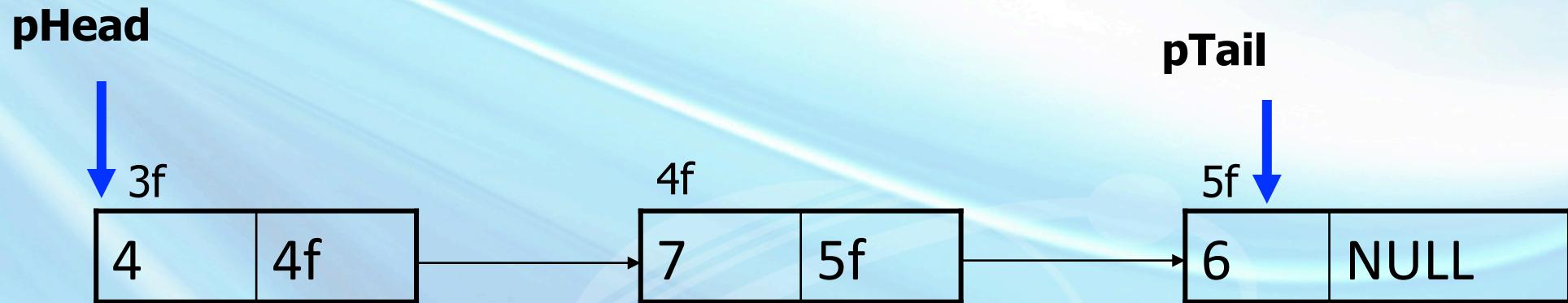


- CTDL của một DSLK đơn

```
typedef struct tagList
{ Node *pHead; // Lưu địa chỉ Node đầu tiên trong DS
  Node *pTail; // Lưu địa chỉ của Node cuối cùng trong DS
}LIST;        // kiểu danh sách liên kết đơn
```



Ví dụ DSLK đơn



Trong ví dụ trên thành phần dữ liệu là 1 số nguyên



Các thao tác cơ bản trên DSLK đơn

- Tạo 1 danh sách liên kết đơn rỗng
- Tạo 1 nút có trường Infor bằng x
- Duyệt danh sách
- Thêm một phần tử có khóa x vào danh sách
- Tìm một phần tử có Info bằng x
- Hủy một phần tử trong danh sách
- Sắp xếp danh sách liên kết đơn



➤ Địa chỉ của nút đầu tiên, địa chỉ của nút cuối cùng đều không có

```
void CreateList(LIST &L)
```

```
{
```

```
    L.pHead = NULL;
```

```
    L.pTail = NULL;
```

```
}
```



Tạo phần tử mới trên DSLK đơn

- Hàm trả về địa chỉ phần tử mới tạo

Node* CreateNode(Data x)

{ Node *p;

p = **new Node**; //*cấp phát vùng nhớ cho phần tử*

if (p == NULL) **exit(1);**

p ->Info = x; //*gán dữ liệu cho nút*

p->pNext = NULL;

return p;

}



- Duyệt danh sách là thao tác thường được thực hiện khi có nhu cầu cần xử lý các phần tử trong danh sách như:
- Đếm các phần tử trong danh sách.
 - Tìm các phần tử trong danh sách thỏa điều kiện.
 - Sắp xếp các phần tử trong danh sách.
 - Hủy toàn bộ danh sách.
 -



- Bước 1:

`p = pHead; // p lưu địa chỉ của phần tử đầu trong DS`

- Bước 2:

Trong khi (danh sách chưa hết) thực hiện:

- + xử lý phần tử p

- + `p = p->pNext; // qua phần tử kế`



Duyệt danh sách

```
void PrintList(LIST l)
{
    Node *p;
    p = l.pHead;
    while (p != NULL)
    {
        cout<< p->Info;
        p = p->pNext;
    }
}
```



Tìm kiếm phần tử trong danh sách

- Các bước của thuật toán tìm nút có Info bằng x trong list đơn (*thuật toán tìm tuyến tính*)

Bước 1: p=pHead; // *địa chỉ của phần tử đầu trong DS*

Bước 2:

Trong khi p!=NULL và p->Info!=x

 p=p->pNext; // *xét phần tử kế*

Bước 3:

- + Nếu p!=NULL thì p lưu địa chỉ của nút có
 Info = x

- + Ngược lại : Không có phần tử cần tìm



Tìm kiếm phần tử trong danh sách

- Hàm tìm phần tử có Info = x, hàm trả về địa chỉ của nút được tìm thấy, ngược lại hàm trả về NULL.

```
Node *Search(LIST l, Data x)
```

```
{
```

```
    Node *p;
```

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

```
    while ((p!=NULL) && (p->Info != x))
```

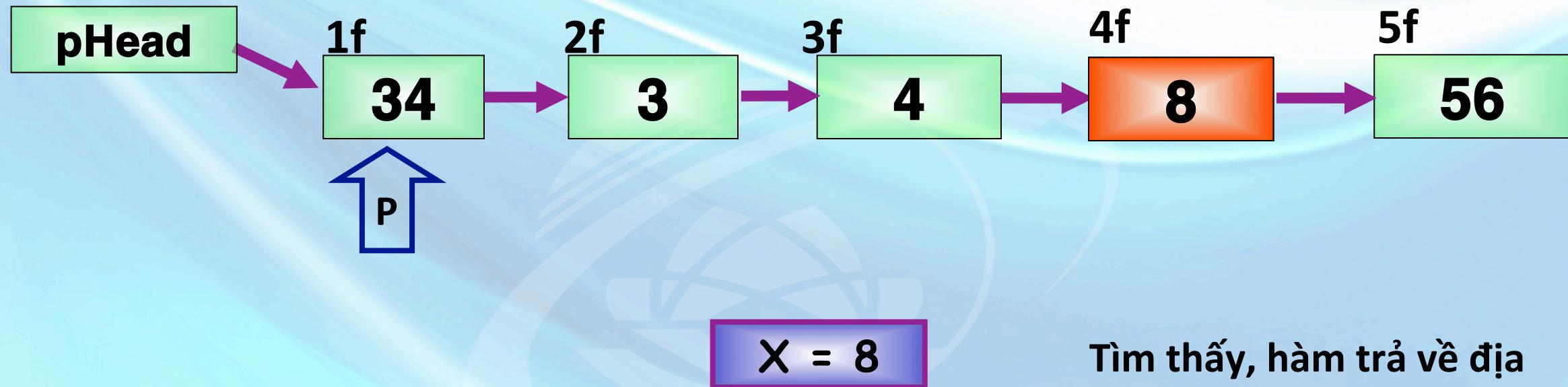
```
        p = p->pNext;
```

```
    return p;
```

```
}
```



Tìm kiếm phần tử trong danh sách





Thêm một phần tử vào trong danh sách

- **Lưu ý khi thêm:** Khi thêm 1 phần tử vào DS thì có làm cho pHead, pTail thay đổi ?
- **Các vị trí thêm 1 phần tử vào DS:**
 - Thêm vào đầu
 - Thêm vào cuối
 - Thêm vào sau 1 phần tử q cho trước



Thêm một phần tử vào đầu danh sách

- Thêm nút p vào đầu danh sách liên kết đơn

Bắt đầu:

Nếu List rỗng thì

- + pHead = p;
- + pTail = pHead;

Ngược lại

- + p->pNext = pHead;
- + pHead = p;

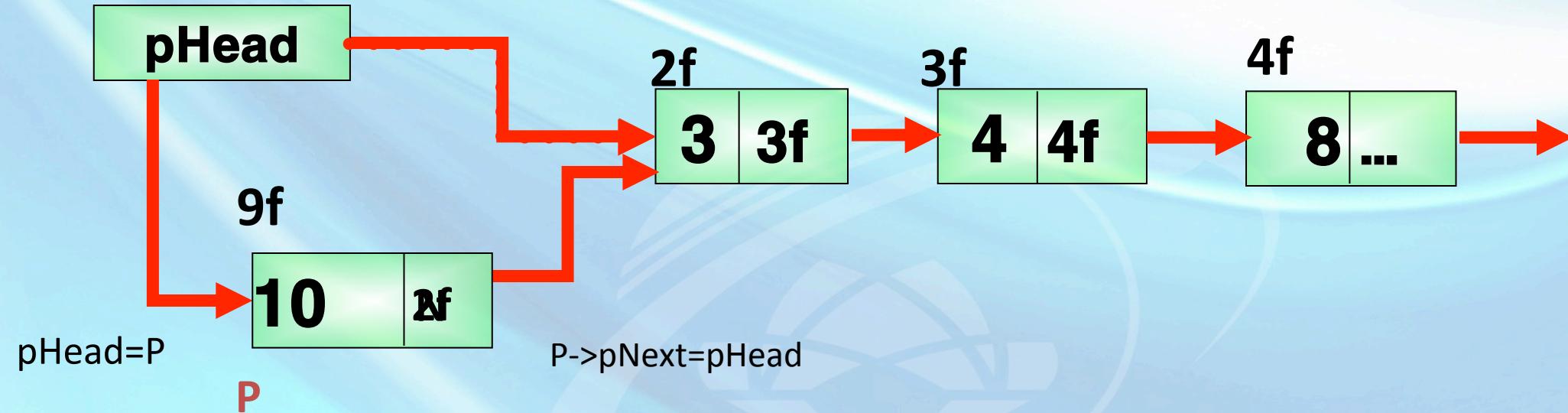


Thêm một phần tử vào đầu danh sách

```
void AddHead(LIST &l, Node* p)
{
    if (l.pHead==NULL)
    {
        l.pHead = p;
        l.pTail = l.pHead;
    }
    else
    {
        p->pNext = l.pHead;
        l.pHead = p;
    }
}
```



Thêm một phần tử vào đầu danh sách





Thêm một phần tử vào cuối danh sách

➤ Ta cần thêm nút p vào cuối DS

Bắt đầu:

Nếu List rỗng thì

+ pHead = p;

+ pTail = pHead;

Ngược lại

+ pTail->pNext = p;

+ pTail = p;

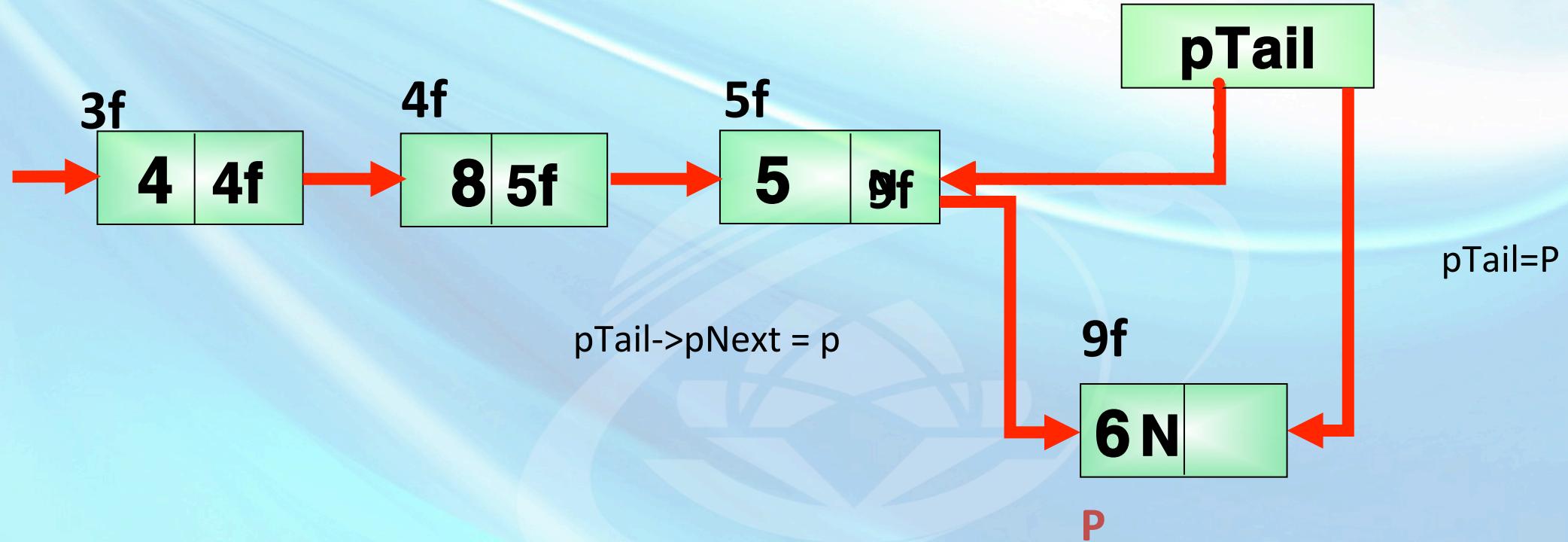


Thêm một phần tử vào cuối danh sách

```
void AddTail(LIST &l, Node *p)
{
    if (l.pHead == NULL)
    {
        l.pHead = p;
        l.pTail = l.pHead;
    }
    else
    {
        l.pTail->Next = p;
        l.pTail = p;
    }
}
```



Thêm một phần tử vào cuối danh sách





Bắt đầu:

Nếu ($q \neq \text{NULL}$) thì

B1: $p->pNext = q->pNext;$

B2:

+ $q->pNext = p ;$

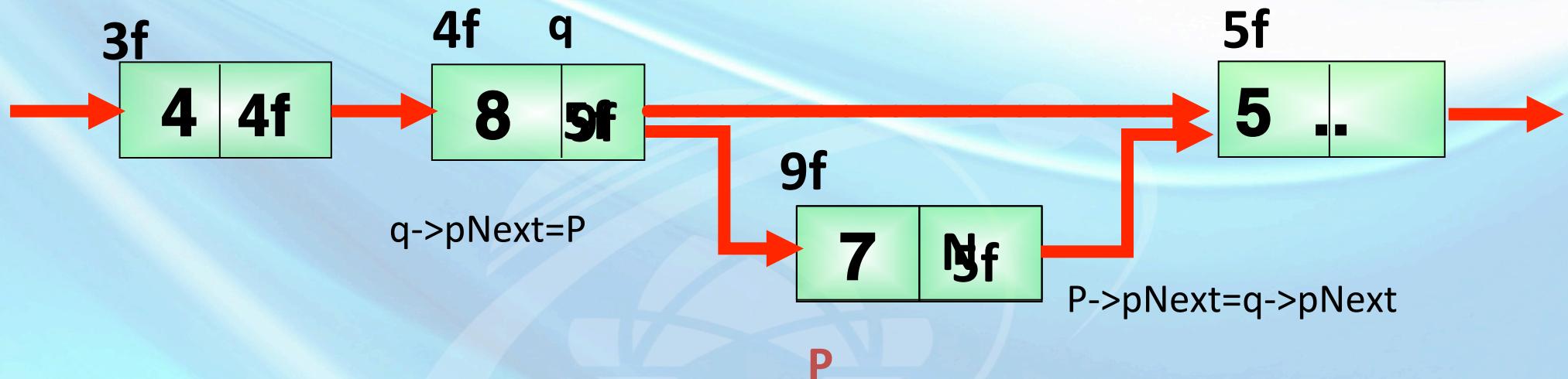
+ nếu $q = pTail$ thì $pTail=p;$

Thêm một phần tử p vào sau phần tử q trong danh sách

```
void InsertAfterQ(List &l, Node *p, Node *q)
{
    if (q != NULL)
    {
        p->pNext = q->Next;
        q->pNext = p;
        if (l.pTail == q)          l.Tail=p;
    }
    else
        AddHead(l,q); // hàm thêm q vào đầu DS
}
```



Thêm một phần tử p vào sau phần tử q trong danh sách





Huỷ một phần tử trong danh sách

- **Nguyên tắc:** Phải “*cô lập*” phần tử cần hủy trước khi hủy.
- Các vị trí có thể hủy:
 - Hủy phần tử đứng đầu DS
 - Hủy phần tử đứng cuối DS
 - Hủy phần tử có khoá bằng x
 - Huỷ phần tử đứng sau q trong DS



Huỷ phần tử đứng đầu trong danh sách

➤ Bắt đầu:

- Nếu (`pHead != NULL`) thì
 - B1: `p = pHead;`
 - B2:
 - + `pHead = pHead->pNext ;`
 - + `delete (p);`
 - B3:
Nếu (`pHead == NULL`) thì `pTail=NULL;`



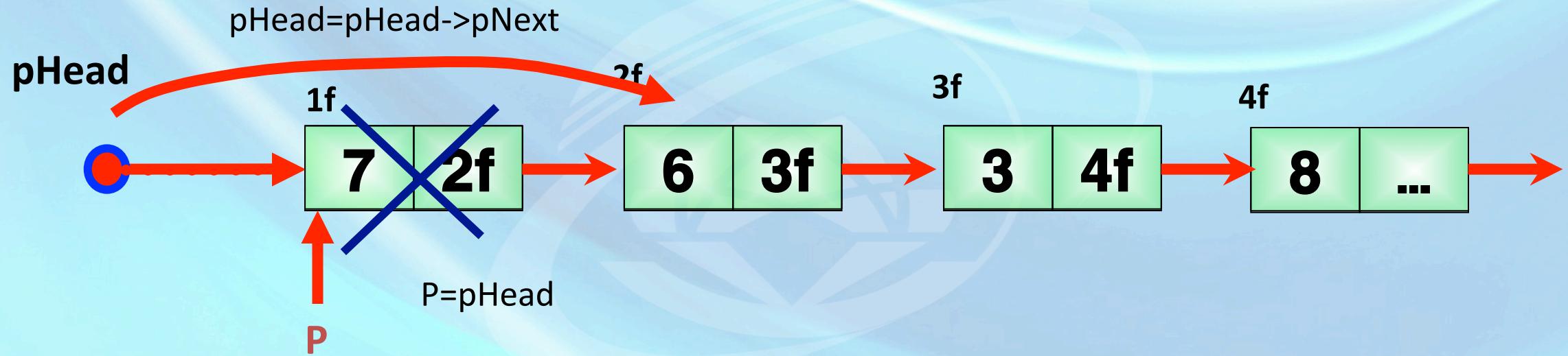
Huỷ phần tử đứng đầu trong danh sách

- Hàm trả về 1 nếu hủy được, ngược lại, hàm trả về 0.

```
int RemoveHead(List &l, int &x)
{
    Node *p;
    if (l.pHead != NULL)
    {
        p = l.pHead;
        x = p->Info; // lưu dữ liệu của nút cần hủy
        l.pHead = l.pHead->pNext;
        delete p;
        if (l.pHead == NULL) l.pTail=NULL;
        return 1;
    }
    return 0;
}
```



Huỷ phần tử đứng đầu trong danh sách





➤ Bắt đầu

Nếu ($q \neq \text{NULL}$) thì

- B1: $p = q->pNext;$
- B2: Nếu ($p \neq \text{NULL}$) thì
 - + $q->pNext = p->pNext;$
 - + nếu ($p == pTail$) $pTail = q;$
 - + $\text{delete } p;$

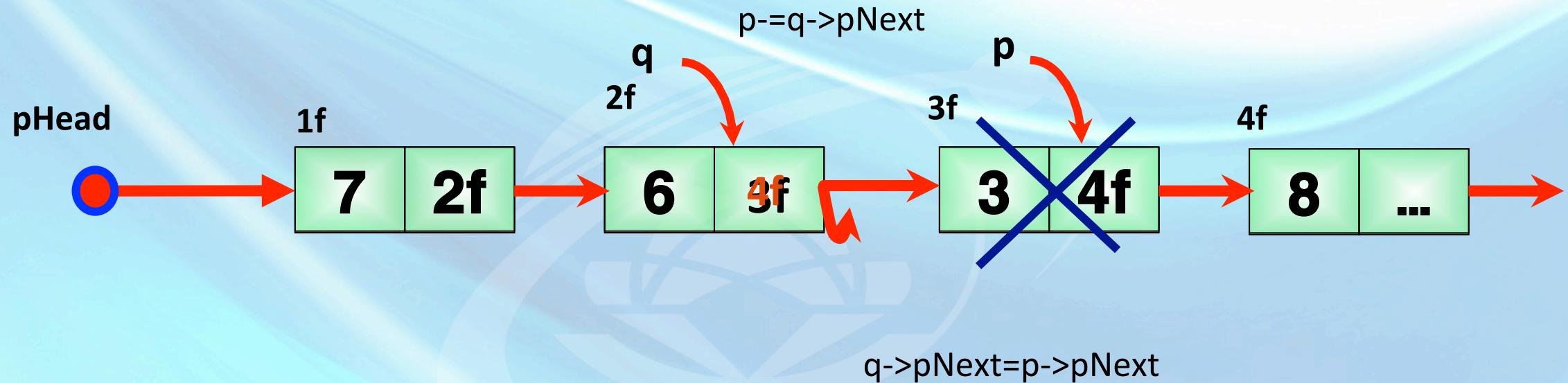


Huỷ phần tử đứng sau phần tử q trong danh sách

```
int RemoveAfterQ(LIST &l, Node *q, int &x)
{
    Node *p;
    if(q != NULL) // q tồn tại trong DS
    {
        p = q->pNext; // p là nút cần hủy
        if (p != NULL) // q không phải là nút cuối
        {
            if (p == l.pTail) // nút cần hủy là nút cuối cùng
                l.pTail = q; // cập nhật lại pTail
            q->pNext = p->pNext; // tách p ra khỏi xâu
            x = p->Info; // lưu dữ liệu nút hủy
            delete p; // hủy nút p
        }
        return 1;
    }
    else return 0;
}
```



Huỷ phần tử đứng sau phần tử q trong danh sách





Huỷ phần tử có khoá x trong danh sách

Bước 1:

Tìm phần tử p có khoá bằng x, và q đứng trước p

Bước 2:

Nếu ($p \neq \text{NULL}$) thì // *tìm thấy phần tử có khoá bằng x*

Hủy p ra khỏi DS bằng cách hủy phần tử

đứng sau q

Ngược lại

Báo không tìm thấy phần tử có khoá bằng x



Huỷ phần tử có khoá x trong danh sách

```
int RemoveX(LIST &l, int x)
{ Node *p,*q = NULL; p=l.Head;
  while((p != NULL) && (p->Info != x)) // tìm x
  {
    q = p;
    p = p->Next;
  }
  if (p == NULL) //không tìm thấy phần tử có khoá bằng x
    return 0;
  if (q != NULL) //tìm thấy phần tử có khoá bằng x
    DeleteAfterQ(l,q,x);
  else RemoveHead(l,x); //phần tử cần hủy nằm đầu DS
  return 1;
}
```



- Bước 1:

Trong khi (danh sách chưa hết) thực hiện

- B1.1:

$p = pHead;$

$pHead = pHead->pNext; // cập nhật pHead$

- B1.2:

Hủy p

- Bước 2:

$pTail = NULL; // bảo toàn tính nhất quán khi DS rỗng$

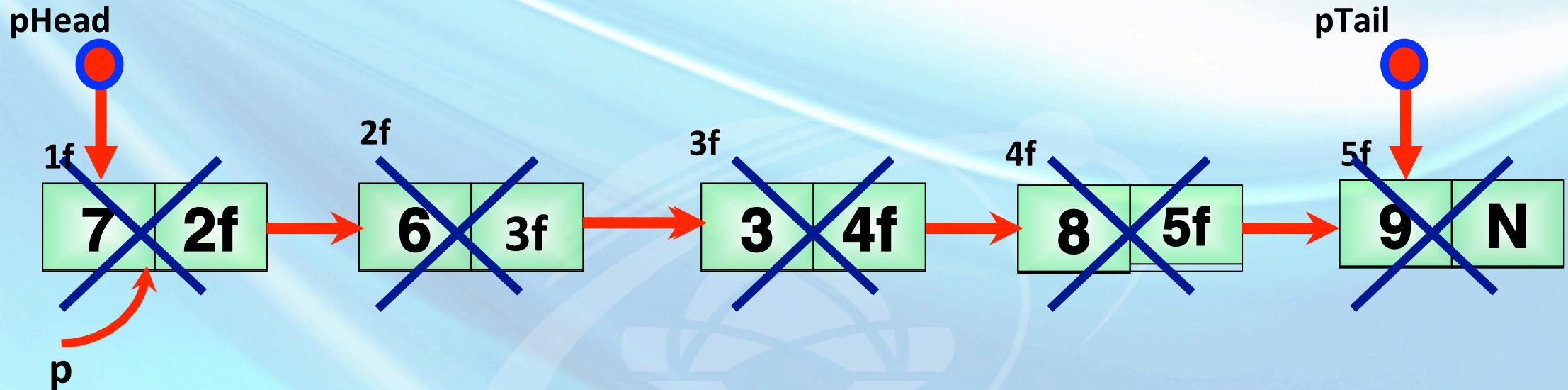


Huỷ danh sách

```
void RemoveList(LIST &l)
{
    Node *p;
    while (l.pHead!=NULL) //còn phần tử trong List
    {
        p = l.pHead;
        l.pHead = p->pNext;
        delete p;
    }
}
```



Huỷ danh sách





Sắp xếp trên danh sách

Có hai cách tiếp cận

- **Cách 1:** Thay đổi thành phần Info

pHead



pTail

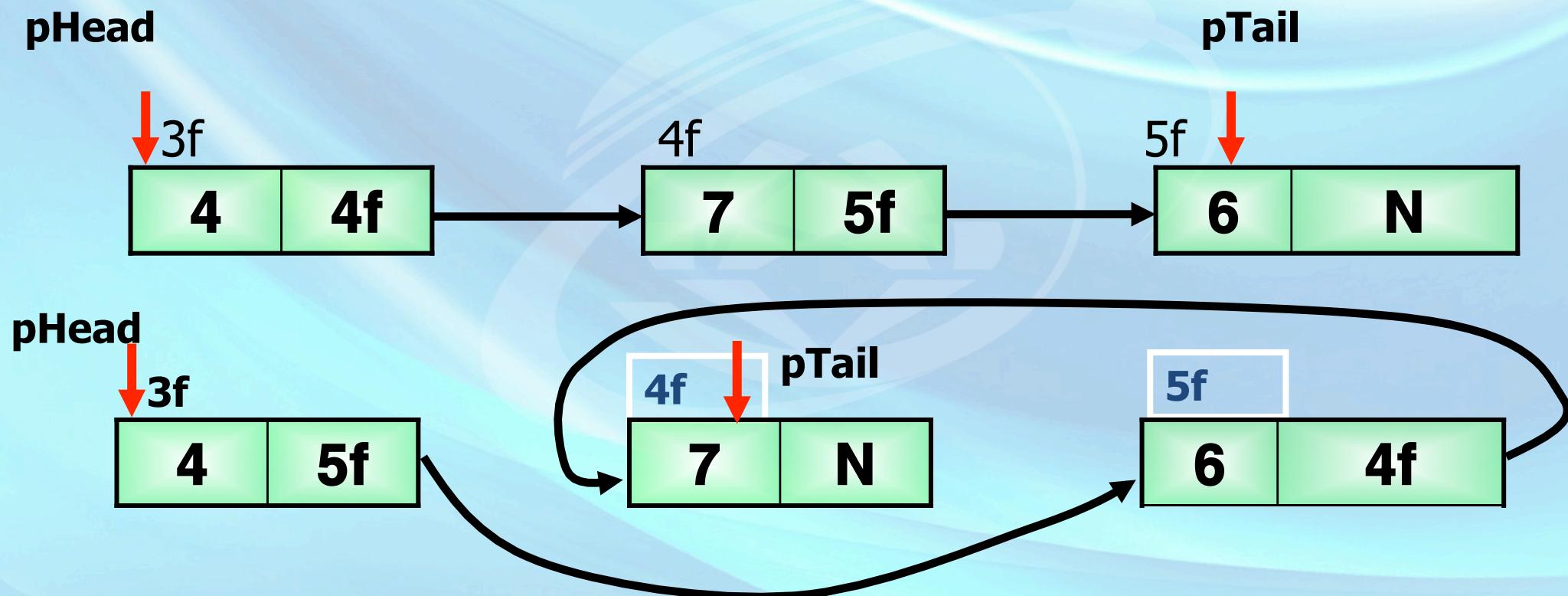
pHead





Sắp xếp trên danh sách

➤ **Cách 2:** Thay đổi thành phần pNext (thay đổi trình tự liên kết của các phần tử sao cho tạo lập nên được thứ tự mong muốn)





➤ Thay đổi thành phần Info (dữ liệu)

- **Ưu:** Cài đặt đơn giản, tương tự như sắp xếp mảng
- **Nhược:**
 - Đòi hỏi thêm vùng nhớ khi hoán vị nội dung của 2 phần tử -> chỉ phù hợp với những DS có kích thước Info nhỏ
 - Khi kích thước Info (dữ liệu) lớn chi phí cho việc hoán vị thành phần Info lớn
 - ✓ Làm cho thao tác sắp xếp chậm

➤ Thay đổi thành phần pNext

- **Ưu:**
 - Kích thước của trường này không thay đổi, do đó không phụ thuộc vào kích thước bản chất dữ liệu lưu tại mỗi nút.
 - ✓ Thao tác sắp xếp nhanh
- **Nhược:** Cài đặt phức tạp

```
void SelectionSort(LIST &l)
```

```
{
```

```
    Node *p, *q, *min;
```

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

```
    while (p != l.pTail)
```

```
{
```

```
    min = p;
```

```
    q = p->Next;
```

```
        while (q != NULL)
```

```
{
```

```
            if (q->Info < p->Info)
```

```
                min = q;
```

```
                q = q->Next;
```

```
}
```

```
        Swap(min->Info, p->Info);
```

```
        p = p->Next;
```

```
}
```



Yêu cầu: Viết chương trình tạo một DSLK đơn, trong đó thành phần dữ liệu của mỗi nút là số nguyên dương.

1. Liệt kê tất cả thành phần dữ liệu của các nút.
2. Tìm 1 phần tử có khoá bằng x ?
3. Xoá 1 phần tử đầu ?
4. Xoá 1 phần tử có khoá bằng x ?
5. Sắp xếp tăng dần theo thành phần dữ liệu (Info).
6. Chèn 1 phần tử vào DSLK, sao cho sau khi chèn DSLK vẫn tăng dần theo thành phần dữ liệu.



Ví dụ

```
void main()
{
    LIST l1; Node *p; int x;
    CreateList(l1);
    do {
        cout<<" Nhập x= "; cin>>x;
        if (x>0)
        {
            p = CreateNode(x);
            AddHead(l1,x);
        }
    } while (x>0);
    cout<<"Danh sách mới tạo là:";
    PrintList(l1);
    cout<<"Nhập số cần tìm x ="; cin>>x;
```



Ví dụ

```
p = Search(l1,x);  
if (p==NULL) cout<<“không tìm thấy”;  
else      cout<<“tìm thấy”;  
RemoveHead(l1,x);  
cout<<“danh sách sau khi xóa: ”;  
PrintList(l1);  
cout<<“Nhập khoá cần xoá: ”;  
cin>>x;  
RemoveX(l1,x);
```



```
cout<<“Danh sách sau khi xoá:”;  
PrintList(l1);  
SelectionSort(l1);  
cout<<“Danh sách sau khi sắp xếp:”;  
PrintList(l1);  
RemoveList(l1);  
}
```



Ví dụ ứng dụng DSLK

- Quản lý danh sách các học viên trong lớp học.
- Quản lý danh sách nhân viên trong một công ty, trong cơ quan.
- Quản lý các cuốn sách trong thư viện.
- Quản lý các băng đĩa trong tiệm cho thuê đĩa.
-



Ví dụ Quản lý sinh viên dùng DSLK

Yêu cầu: Thông tin của một sinh viên gồm: mã số sinh viên MSSV, tên sinh viên TEN, điểm trung bình DTB .

1. Hãy khai báo cấu trúc dữ liệu dạng DSLK đơn để lưu danh sách sinh viên nói trên.
2. Nhập danh sách các sinh viên, và thêm từng sinh viên vào đầu danh sách (việc nhập kết thúc khi tên của một sinh viên bằng khoảng trắng)
3. Tìm một sinh viên có trong lớp học hay không ?
4. Xoá một sinh viên có MSSV bằng x (x nhập từ bàn phím).
5. Liệt kê thông tin của các sinh viên có điểm trung bình lớn hơn hay bằng 5.



Ví dụ Quản lý sinh viên dùng DSLK

6. Xếp loại và in ra thông tin của từng sinh viên, biết rằng cách xếp loại như sau:

ĐTB ≤ 3.6 : Loại yếu

ĐTB ≥ 5.0 và ĐTB < 6.5 : Loại trung bình

ĐTB ≥ 6.5 và ĐTB < 7.0 : Loại trung bình khá

ĐTB ≥ 7.0 và ĐTB < 8.0 : Loại khá

ĐTB ≥ 8.0 và ĐTB < 9.0 : Loại giỏi.

ĐTB ≥ 9.0 : Loại xuất sắc

7. Sắp xếp và in ra danh sách sinh viên tăng theo điểm trung bình.

8. Chèn một sinh viên vào danh sách sinh viên tăng theo điểm trung bình nói trên, sao cho sau khi chèn danh sách sinh viên vẫn tăng theo điểm trung bình



- Cấu trúc dữ liệu của một sinh viên

```
typedef struct
{
    char TEN[40];
    char MSSV[40];
    float DTB;
}SV
```

- Cấu trúc dữ liệu của 1 nút trong DSLK

```
typedef struct tagNode
{
    SV Info;
    struct tagNode *pNext;
}Node;
```



Câu hỏi và Bài tập

1. Nêu các bước để thêm một nút vào đầu, giữa và cuối danh sách liên kết đơn.
2. Nêu các bước để xóa một nút ở đầu, giữa và cuối danh sách liên kết đơn.
3. Viết thủ tục để in ra tất cả các phần tử của 1 danh sách liên kết đơn.
4. Viết chương trình thực hiện việc sắp xếp 1 danh sách liên kết đơn bao gồm các phần tử là số nguyên.
5. Viết chương trình cộng 2 đa thức được biểu diễn thông qua danh sách liên kết đơn.