

## XÂU ĐƠN HAI TRỞ QUẢN LÝ SẢN PHẨM

Viết chương trình quản lý sản phẩm, Thông tin về 1 sp gồm có :

- a. Mã sp ( int )
  - b. Tên sp ( char tensp [ 50 ] )
  - c. Số lượng ( int soluong )
  - d. Đơn giá ( float dongia)
  - e. Loại sp ( char loai ) ( nhập , xuất )
1. Viết hàm nhập xuất 1 sp
  2. Viết hàm nhập xuất nhiều sp và lưu trên DANH SÁCH LIÊN KẾT
  3. Xuất ra thông tin sản phẩm theo thứ tự tăng dần của Mãsp
  4. Tìm sp có tên " abc "
  5. Thêm 1 sp sau sp có tên "xy"
  6. Xóa sp có mã x

# Kiểu dữ liệu nhiều thành phần

```
typedef struct sp
{
    int masp;
    char tensp [ 50 ] ;
    int  soluong ;
    float dongia ;
    char loaisp ;
};
```

```

void nhapsp( sp &a)
{
    printf("nhap mã sp :");
    scanf("%d ", &a.masp) ;
    fflush();

    printf(" Nhap ten san phẩm : " ) ;    gets (a. tensp );
    printf("nhap so luong sp :");
    scanf("%d ", &a.soluong) ;

    float t;

    printf(" Nhap don gia :");    scanf("%f ", &t) ;
    a.dongia = t ;

    printf(" Nhap loai san phẩm : X/N " ) ;
    a.loaisp=getche ();
}

```

```
void nhapsp( sp &a)      —————>   loaisp
{

    char t ;
    do{
        printf( " Nhap loai san phẩm : X/N " ) ;
        t=getche();
        if( t!='X' && t!='N' )
            printf( " nhap sai, nhap lai X/N );
    } while (t!='X' && t!='N' );
    a.loaisp= t ;
}
```

```
void xuatsp( sp a)
```

```
{
```

```
    printf(" Ma sp: %d ", a.masp );
```

```
    printf(" Ten sp : %s" , a.tensp);
```

```
    printf(" So luong : %d " , a.soluong );
```

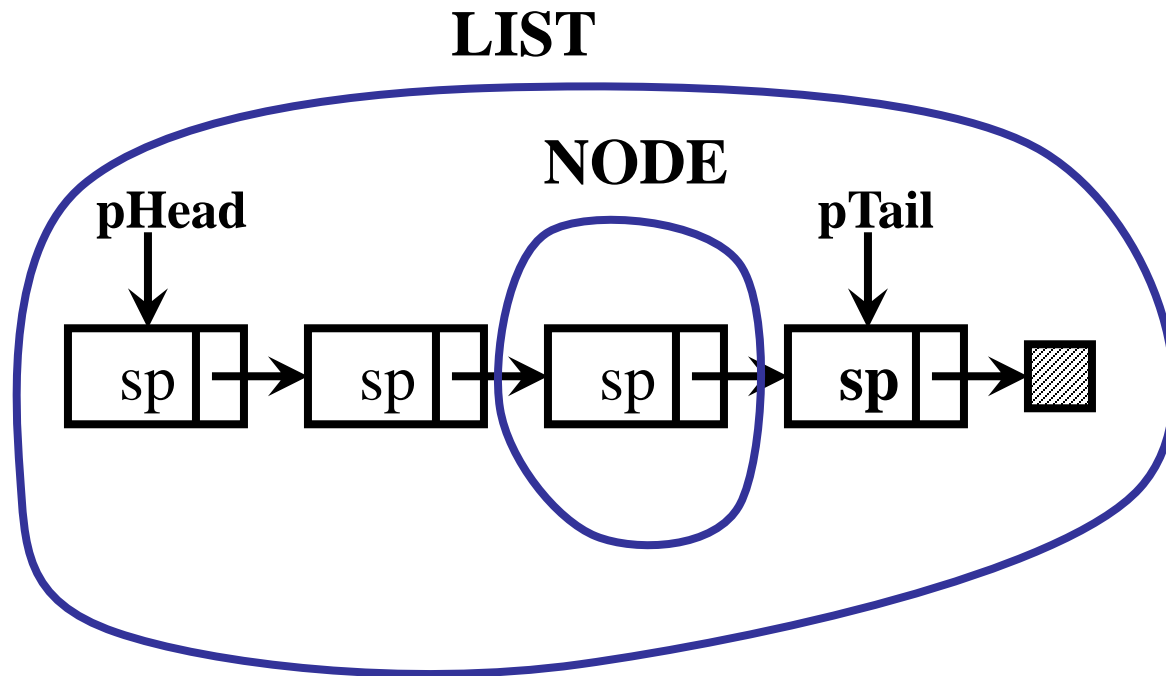
```
    printf(" Gia tien : %f " , a.dongia );
```

```
    if ( a.loai=='N') printf ( " hang nhap ");
```

```
    else printf (" hang xuat ");
```

```
}
```

# Hình ảnh cấu trúc đơn hai trỏ



# Hãy khai báo CTDL cho dslk đơn các quyển sách

1. **struct node**

2. {     sp info;

3.         struct node\*pNext;

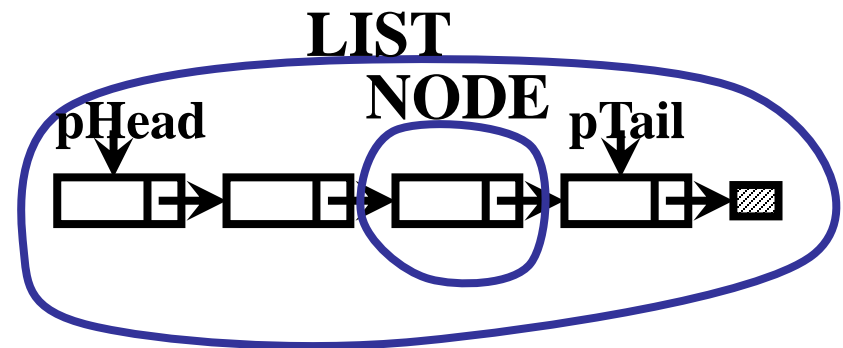
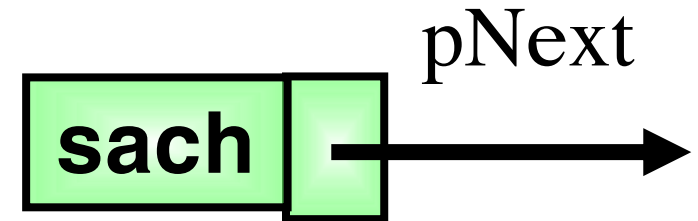
4. }; **typedef struct node NODE;**

5. **struct list**

6. {     NODE \*pHead;

7.         NODE \*pTail;

8. }; **typedef struct list LIST;**





# 3.KHỞI TẠO DANH SÁCH LIÊN KẾT ĐƠN

➤ Khái niệm: Khởi tạo danh sách liên kết đơn là tạo ra danh sách rỗng không chứa node nào hết.

➤ Định nghĩa hàm

```
1. void Init(LIST &l)  
2. {  
3.     l.pHead = NULL;  
4.     l.pTail = NULL;  
5. }
```

## 4. KIỂM TRA DANH SÁCH LIÊN KẾT ĐƠN RỖNG

➤ Khái niệm: Kiểm tra danh sách liên kết đơn rỗng là hàm trả về giá trị 1 khi danh sách rỗng. Trong tình huống danh sách không rỗng thì hàm sẽ trả về giá trị 0.

➤ Định nghĩa hàm

```
1. int IsEmpty (LIST l)  
2. {  
3.     if (l.pHead==NULL)  
4.         return 1;  
5.     return 0;  
6. }
```

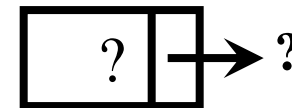
## 5. TẠO NODE CHO DANH SÁCH LIÊN KẾT ĐƠN

➤ Ví dụ 1: Định nghĩa hàm tạo một NODE cho dslk đơn các số thực để chứa thông tin đã được biết trước.

➤ Định nghĩa hàm

```
1. NODE* GetNode (sach x)
```

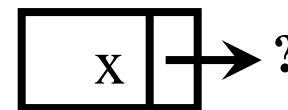
```
2. {     NODE *p = new NODE;
```



```
3.     if (p==NULL)
```

```
        return NULL;
```

```
1.     p->info = x;
```



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

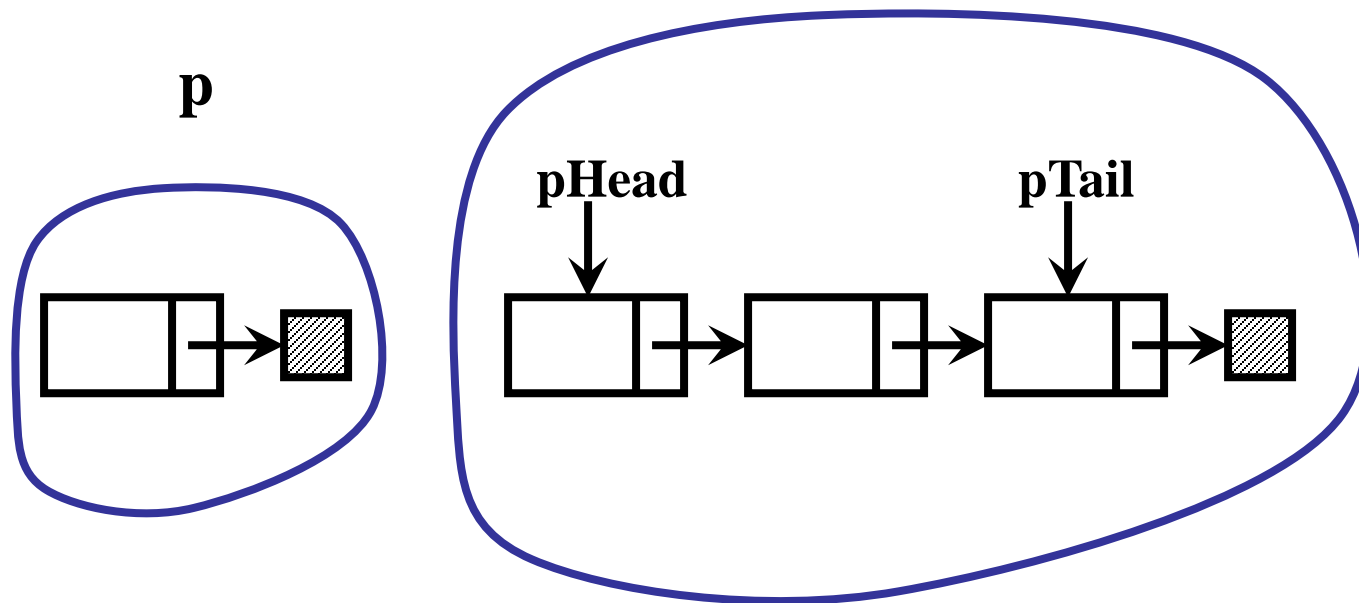


```
3.     return p;
```

```
4. }
```

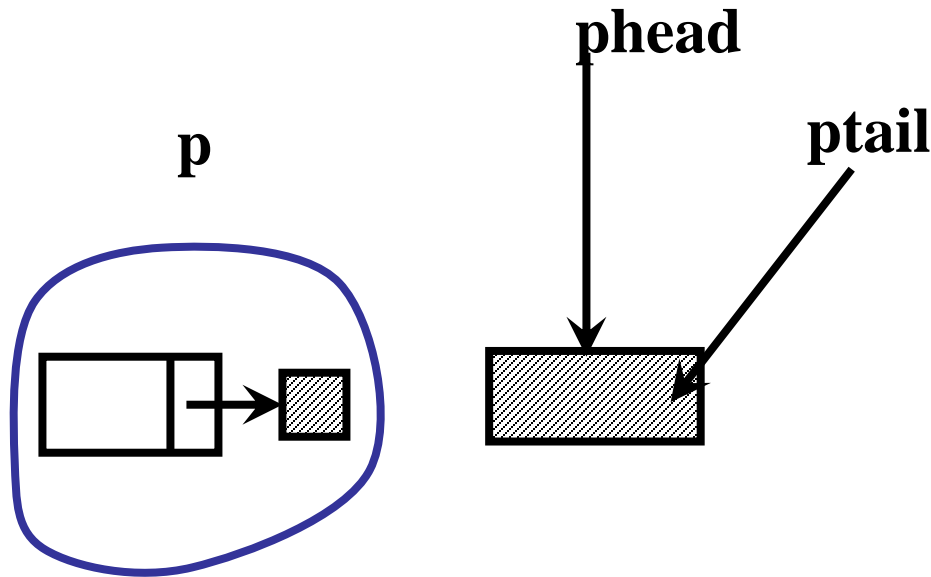
## 6. THÊM MỘT NODE VÀO ĐẦU DS LIÊN KẾT ĐƠN

- ◆ Khái niệm: Thêm một node vào đầu danh sách liên kết đơn là gắn node đó vào đầu danh sách.
- ◆ Hình vẽ



## 6. THÊM MỘT NODE VÀO ĐẦU DS LIÊN KẾT ĐƠN

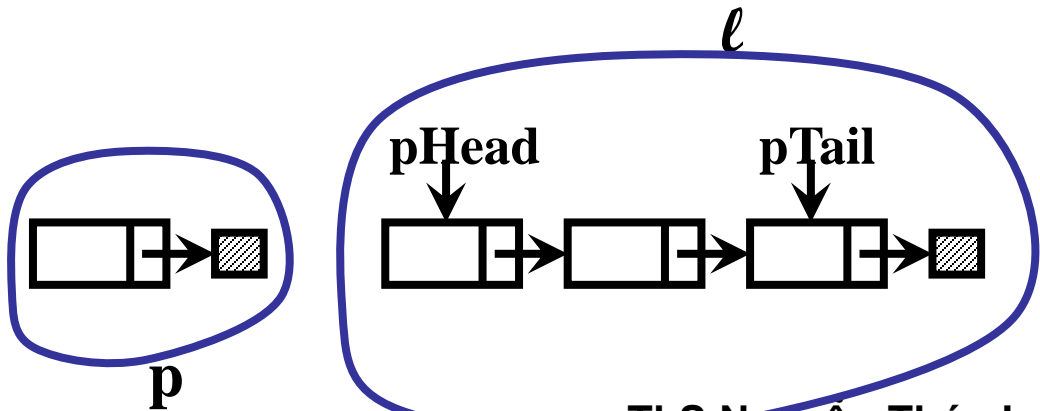
- ◆ Khái niệm: Thêm một node vào đầu danh sách liên kết đơn là gắn node đó vào đầu danh sách.



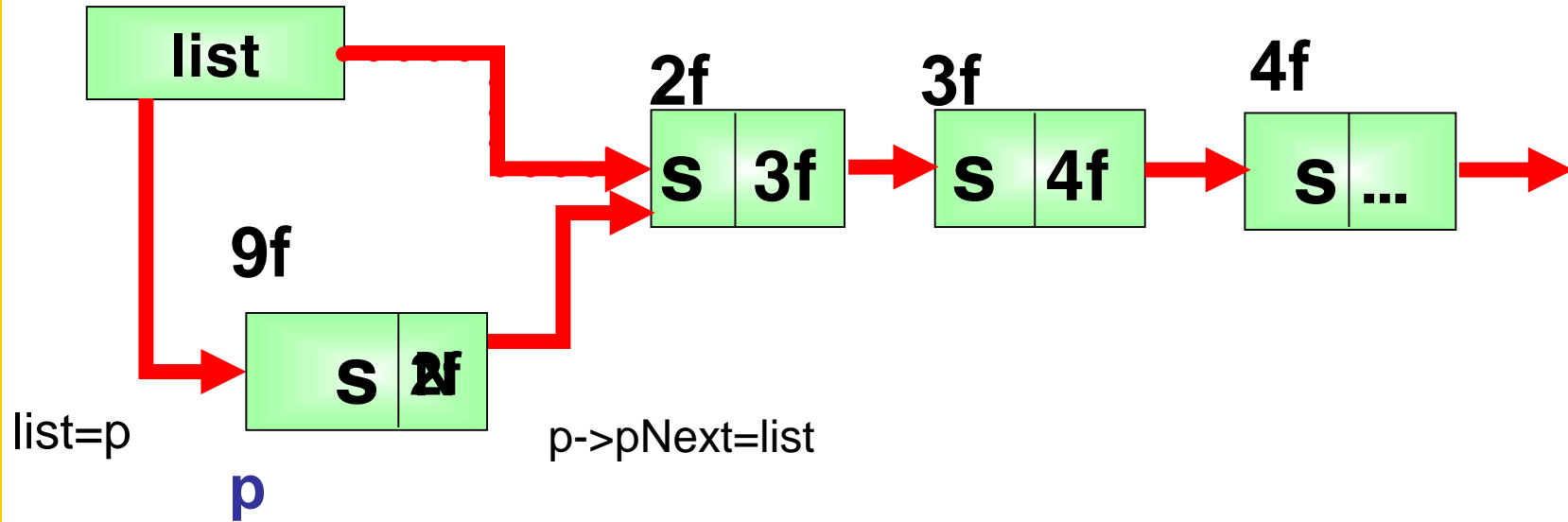
## 6. THÊM MỘT NODE VÀO ĐẦU DS LIÊN KẾT ĐƠN

➤ Định nghĩa hàm:

```
1. void AddHead (LIST&l, NODE*p)
2. {
3.     if (l.pHead==NULL)
4.         l.pHead = l.pTail = p;
5.     else
6.     {
7.
8.     }
9. }
```



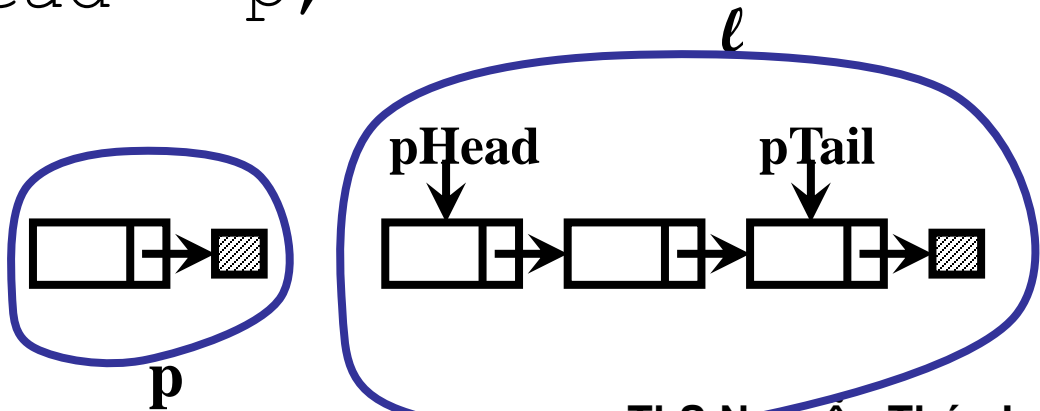
# Minh họa thuật toán thêm vào đầu



## 6. THÊM MỘT NODE VÀO ĐẦU DS LIÊN KẾT ĐƠN

➤ Định nghĩa hàm:

```
1. void AddHead (LIST&l, NODE*p)
2. {
3.     if (l.pHead==NULL)
4.         l.pHead = l.pTail = p;
5.     else
6.     {
7.         p->pNext = l.pHead;
8.         l.pHead = p;
9.     }
10. }
```

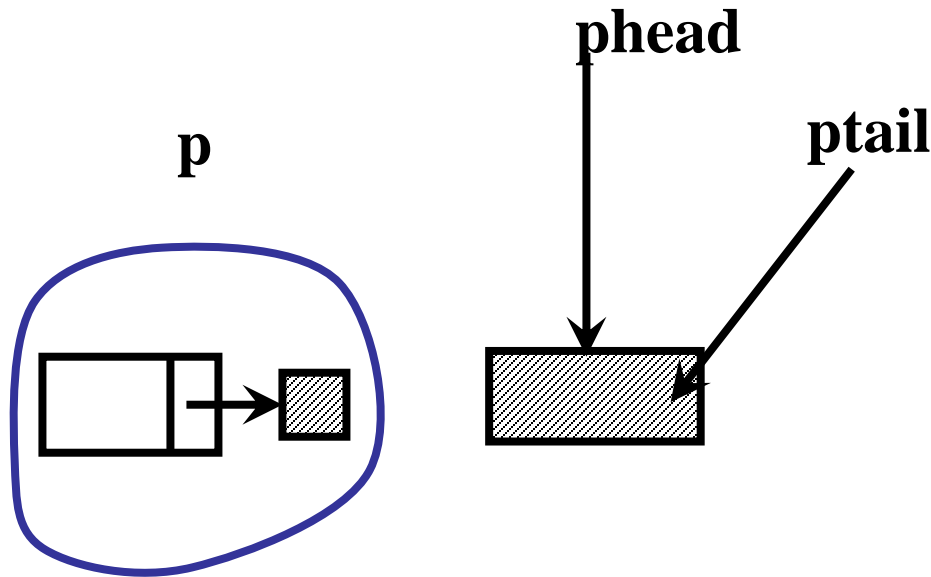


ThS. Nguyễn Thúy Loan



## 6. THÊM MỘT NODE VÀO ĐẦU DS LIÊN KẾT ĐƠN

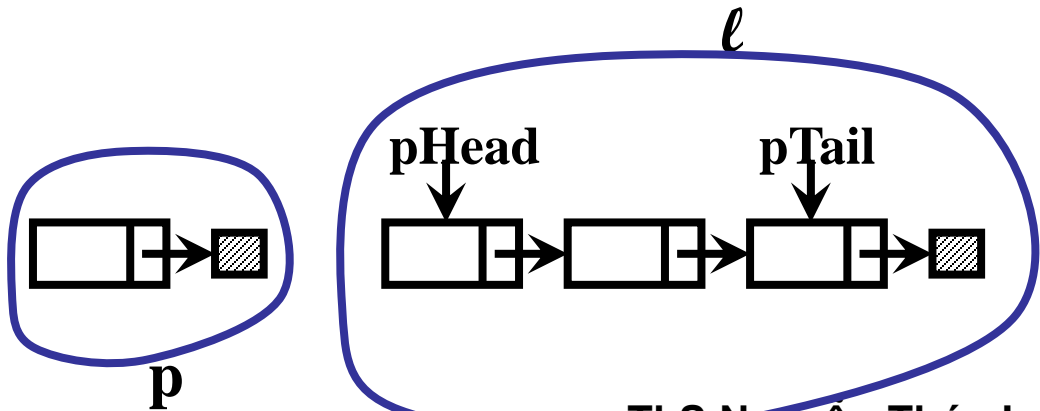
- ◆ Khái niệm: Thêm một node vào đầu danh sách liên kết đơn là gắn node đó vào đầu danh sách.



## 6. THÊM MỘT NODE VÀO ĐẦU DS LIÊN KẾT ĐƠN

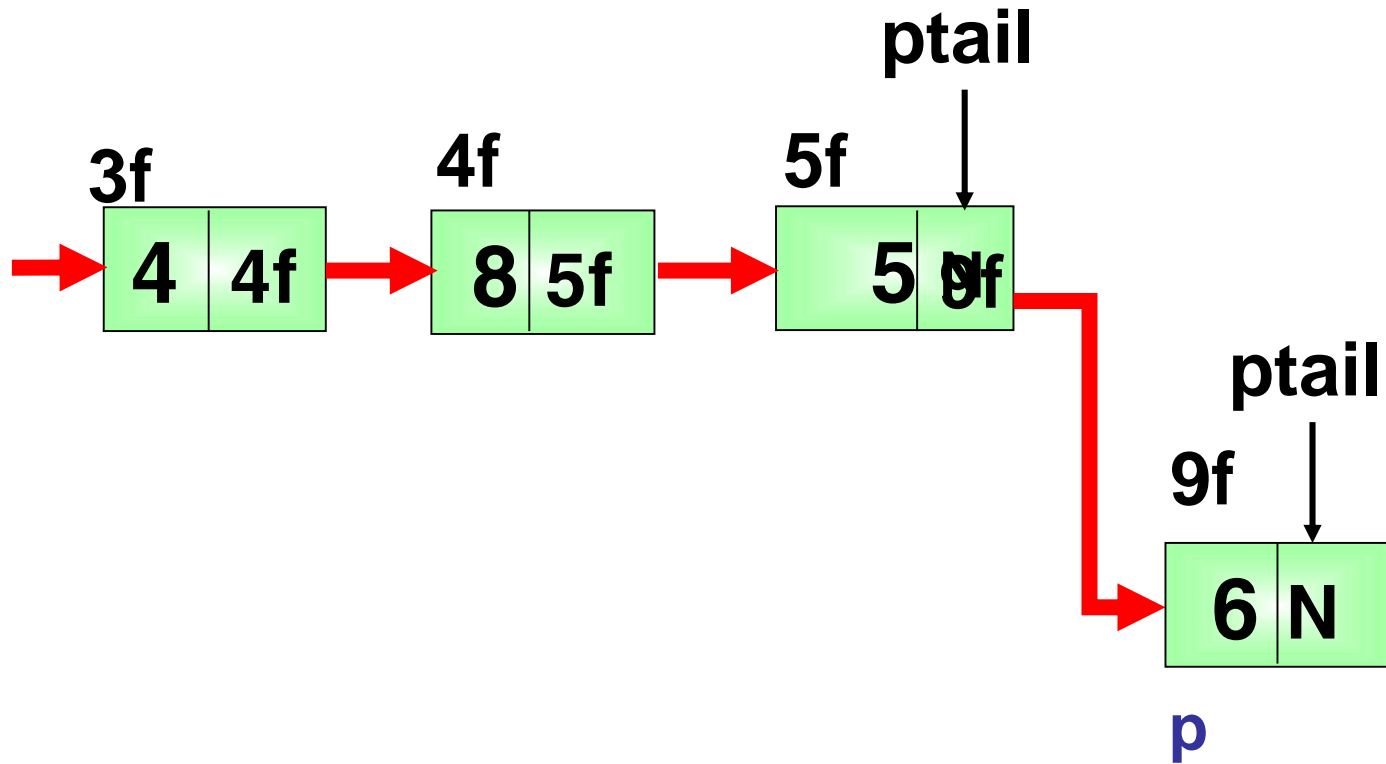
➤ Định nghĩa hàm:

```
1. void AddHead (LIST&l, NODE*p)
2. {
3.     if (l.pHead==NULL)
4.         l.pHead = l.pTail = p;
5.     else
6.     {
7.
8.     }
9. }
```



ThS. Nguyễn Thúy Loan

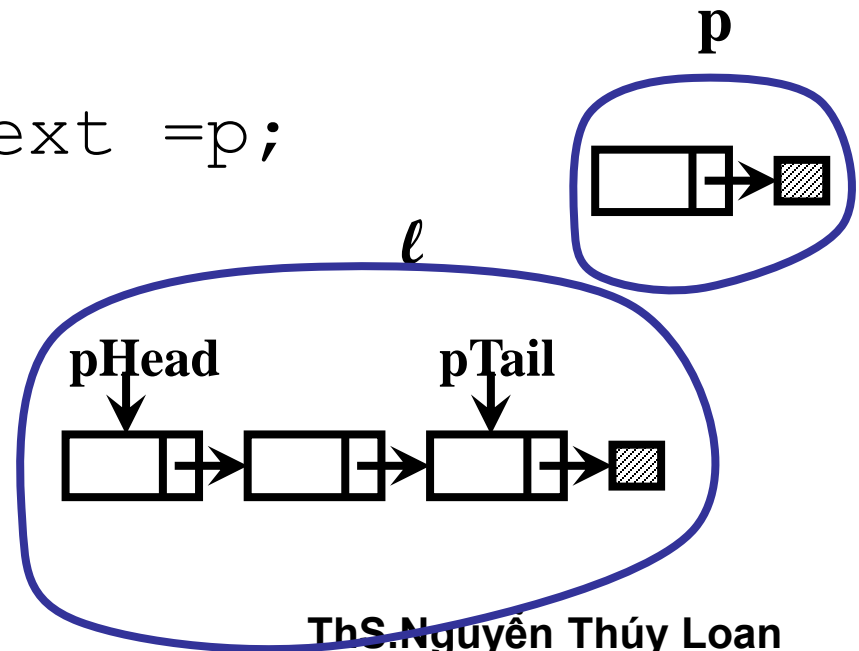
# Minh họa thuật toán thêm vào cuối



## 6. THÊM MỘT NODE VÀO CUỐI DS LIÊN KẾT ĐƠN

➤ Định nghĩa hàm:

```
1. void Addtail (LIST&l, NODE*p)
2. {
3.     if (l.pHead==NULL)
4.         l.pHead = l.pTail = p;
5.     else
6.     {
7.         l.ptail->pNext = p;
8.         l.ptail = p;
9.     }
10. }
```



## Ví dụ: Nhập danh sách liên kết đơn các số nguyên.

```
void Input(LIST &l)
{
    sach x;
    char ch;
    Init(l ) ;
    do{
        printf("Nhap 1 qs: ");
        nhapqs(x);
        NODE*p=GetNode(x) ;
        if (p!=NULL)
            Addtail(l,p) ;
        printf( " nhan N thoat ");
        ch=getche() ;
    }while( ch!='n' && ch!='N' )
}
```

## 8. DUYỆT TUẦN TỰ DANH SÁCH LIÊN KẾT ĐƠN

```
1. void Output( LIST l )
2. {
3.     for (NODE*p=l.phead;p!=NULL;p=p->pnext)
4.
5.         xuat1qs (p->info );
6.
7.
8. }
```

## 8. DUYỆT TUẦN TỰ DANH SÁCH LIÊN KẾT ĐƠN

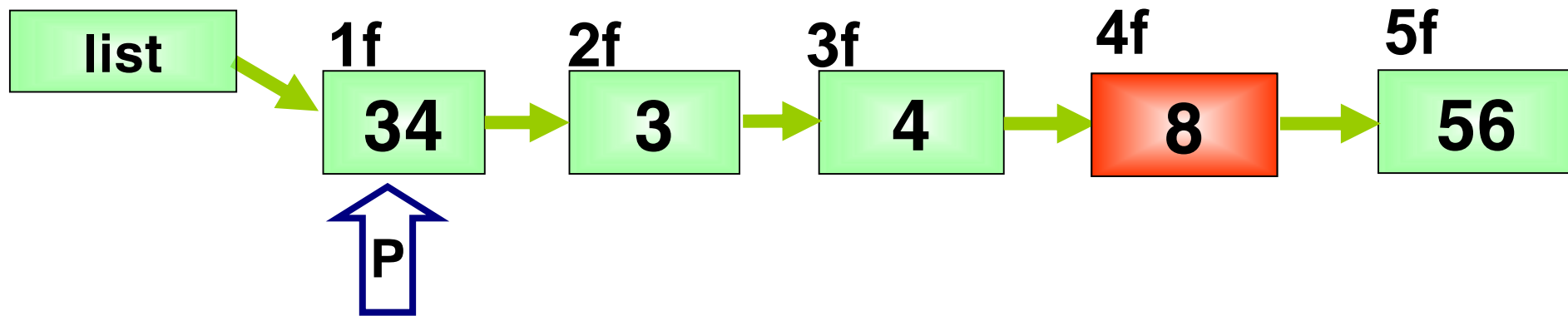
```
void xuatsach>10( LIST l )
{
    for (NODE*p=l.phead;p!=NULL;p=p->pnext)

        if (p->info.soluong>10)

            xuat1qs (p->info ) ;

}
```

# Minh họa thuật toán tìm phần tử trong DSLK



**X = 8**

Tìm thấy, hàm trả  
về địa chỉ của  
nút tìm thấy là 4f



## Tìm 1 phần tử trong DSLK đơn

Hàm tìm phần tử có info = x, hàm trả về địa chỉ của nút có info = x, ngược lại hàm trả về NULL

**NODE \* search( list l, int x)**

{

    NODE \*p = l.phead;

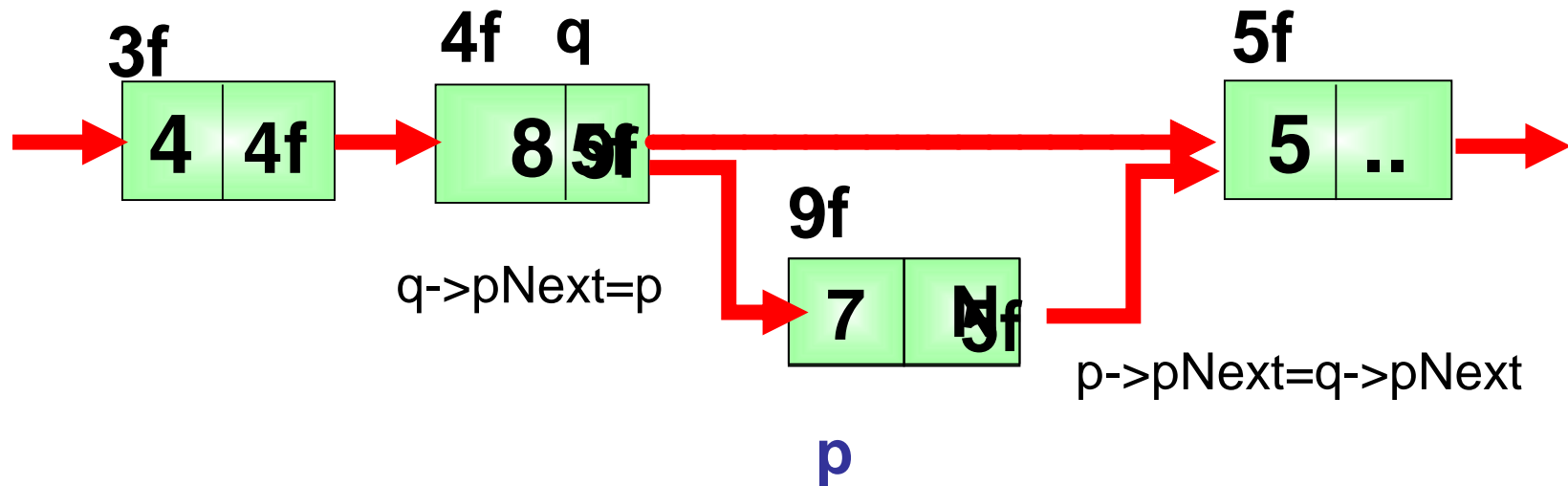
    while(p!=NULL && p->info!=x)

        p=p->pnext;

    return p;

}

# Thêm phần tử p vào sau phần tử q



# Thêm phần tử p vào sau phần tử q

Ta cần thêm nút p vào sau nút q trong list đơn

## Bắt đầu:

Nếu ( $q \neq \text{NULL}$ ) thì

B1:  $p \rightarrow \text{pNext} = q \rightarrow \text{pNext}$

B2:  $q \rightarrow \text{pNext} = p$

Ngược lại: Thêm vào đầu danh sách

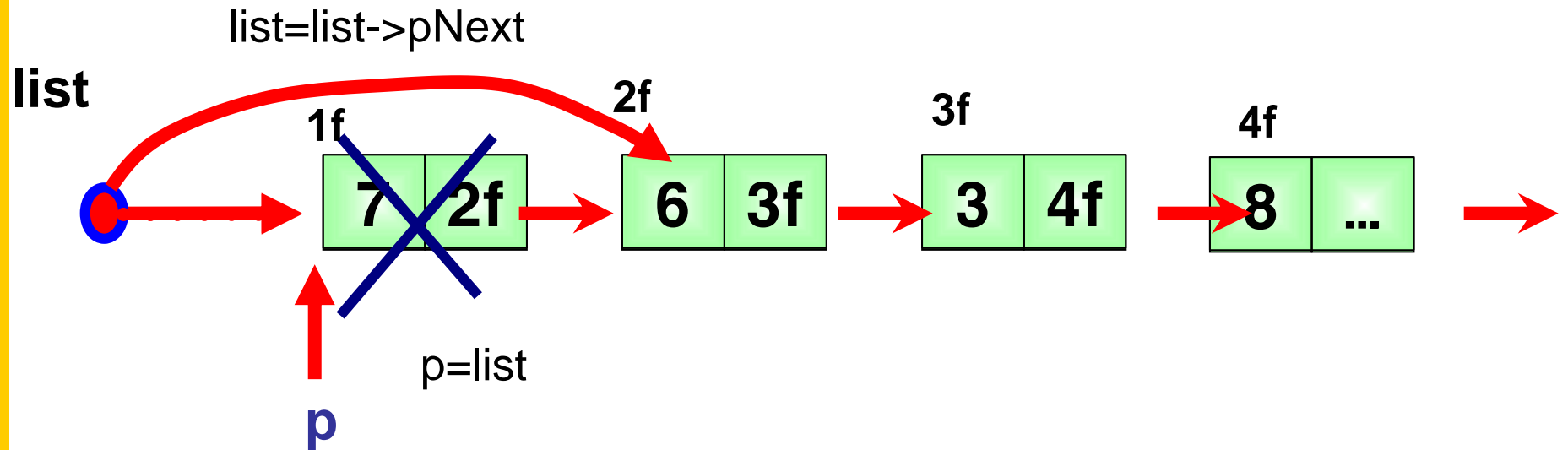
# Cài đặt thuật toán

```
➤ void addAfterq(LIST &l, NODE *p, NODE *q)
➤ {
➤     if(q!=NULL)
➤     {
➤         p->pNext=q->pNext;
➤         q->pNext=p;
➤     }
➤     else
➤         AddHead(l,q); // thêm q vào đầu list
➤ }
```

# Hủy phần tử trong DSLK đơn

- **Nguyên tắc:** Phải cô lập phần tử cần hủy trước khi hủy.
- Các vị trí cần hủy
  - ↪ Hủy phần tử đứng đầu danh sách
  - ↪ Hủy phần tử có khoá bằng x
  - ↪ Hủy phần tử đứng sau q trong danh sách liên kết đơn
- Ở phần trên, các phần tử trong DSLK đơn được cấp phát vùng nhớ động bằng hàm new, thì sẽ được giải phóng vùng nhớ bằng hàm delete.

# Thuật toán hủy phần tử đầu trong DSLK



# Thuật toán hủy phần tử đầu trong DSLK

## ➤ Bắt đầu:

↪ Nếu ( $l \neq \text{NULL}$ ) thì

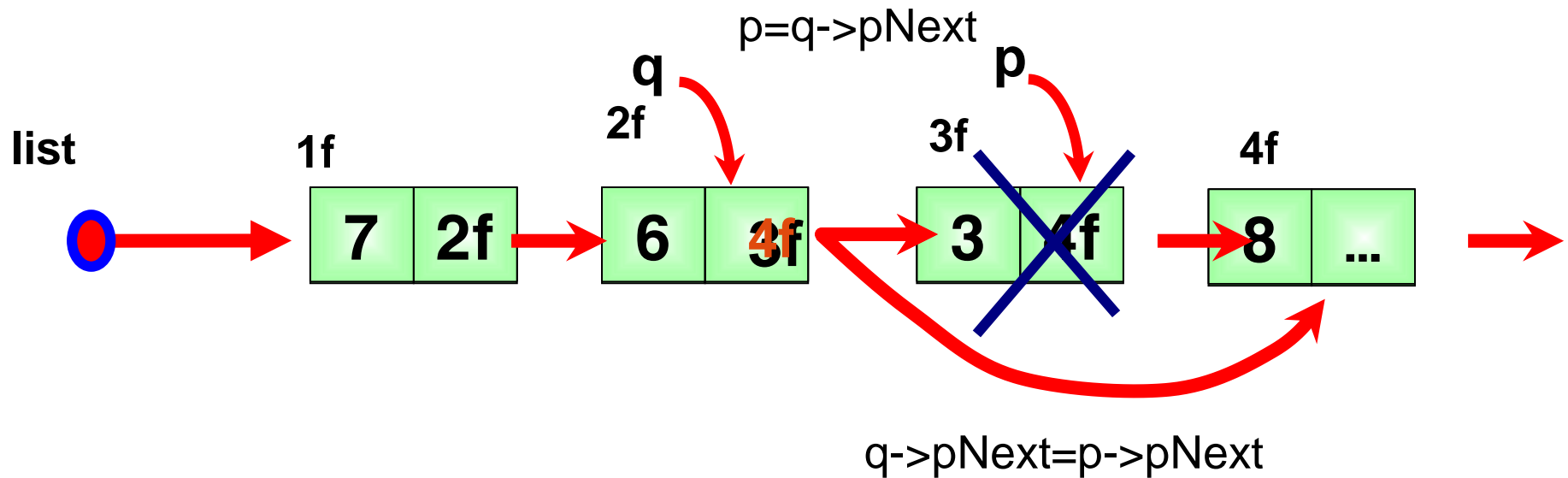
- B1:  $p = \text{pHead}$  //cho p bằng phần tử cần xóa
- B2:  $l = l \rightarrow \text{pNext}$
- $\text{delete}(p)$

# Thuật toán hủy phần tử đầu trong DSLK

```
void deletehead ( list &l )
{
    NODE *p;
    if(l.phead !=NULL)
    {
        p=l.phead;
        l.phead = l.phead->pnext;
        delete(p);
    }
}
```



# Hủy phần tử sau phần tử q trong List



# Hủy phần tử sau phần tử q trong List

## Bắt đầu

Nếu ( $q \neq \text{NULL}$ ) thì // *q tồn tại trong List*

B1:  $p = q \rightarrow \text{pNext};$  // *p là phần tử cần hủy*

B2: Nếu ( $p \neq \text{NULL}$ ) thì // *q không phải là phần tử cuối*

+  $q \rightarrow \text{pNext} = p \rightarrow \text{pNext};$  // *tách p ra khỏi xâu*

+  $\text{delete } p;$  // *hủy p*

# Thuật toán hủy phần tử có khoá x

## 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

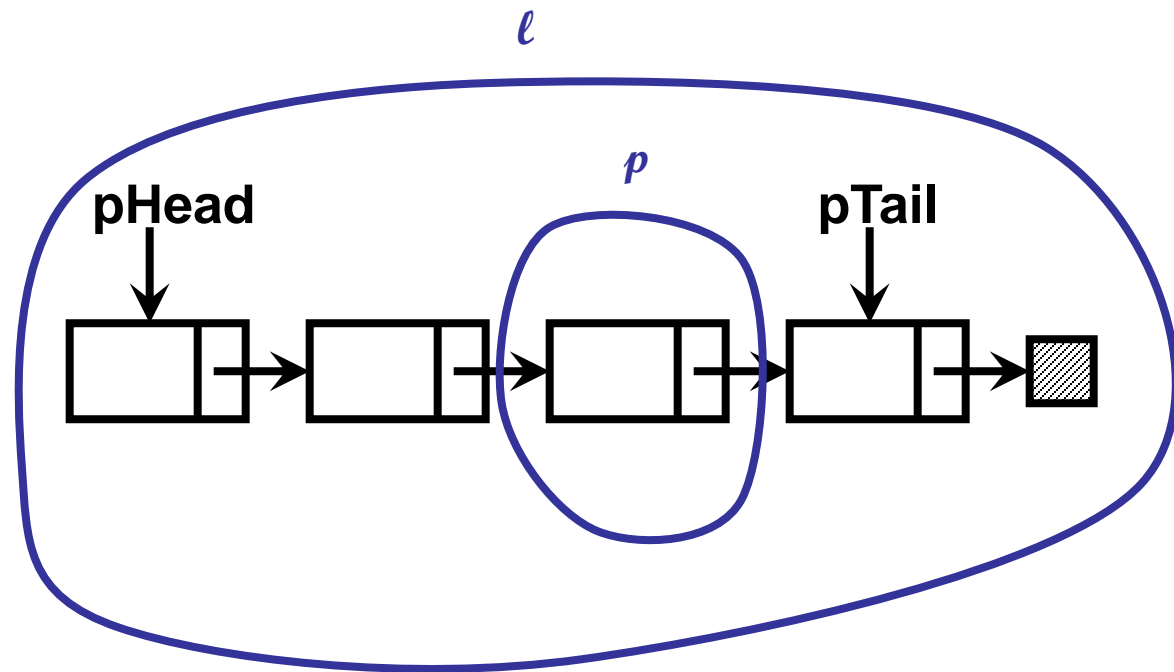
Hủy p ra khỏi danh sách bằng cách  
hủy phần tử đứng sau q

Ngược lại

Báo không tìm thấy phần tử có khoá

# Xóa 1 phần tử trong DSLK đơn

- Bài tập 012: Định nghĩa hàm tách node  $p$  trong danh sách liên kết đơn ra khỏi danh sách.
- Ý tưởng:



## Xóa 1 phần tử trong DSLK đơn

```
int xoa1pt( list &l , NODE *p)
{
    if(l.phead==NULL) return NULL;
    if(l.phead==p)  xoahead(l);
    NODE *q= Before(l,p);
    q->pnext=p->pnext;
    p->pnext=NULL;
    delete p;
    return 0;
}
```

## Tìm 1 node trong danh sách liên kết đơn.

```
NODE* Before( list l, NODE *p )
{
    if(l.phead==NULL) return NULL;
    if(l.phead==p) return NULL;
    NODE *lc=l.phead;
    while(lc->pnext!=p)
        lc=lc->pnext;
    return lc;
}
```

# xóa 1 node trong danh sách liên kết đơn.

```
void remove_anynode(list &l)
{
    node *p,*q;
    int msv;
    printf("\tNhap mssv can xoa: ");
    scanf("%d",&msv);
    p=l.phead;
    if(p->info.mssv == msv)
    {
        l.phead = l.phead->pnext;
        delete p;
    }
    else
```

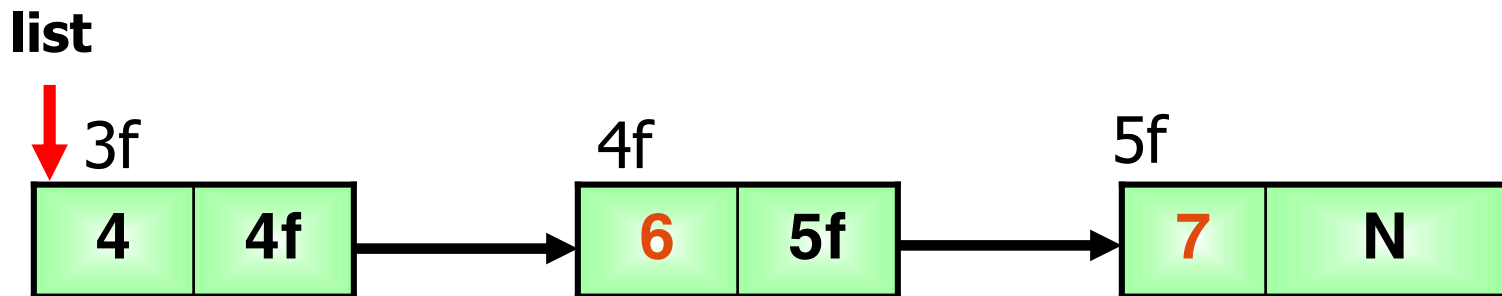
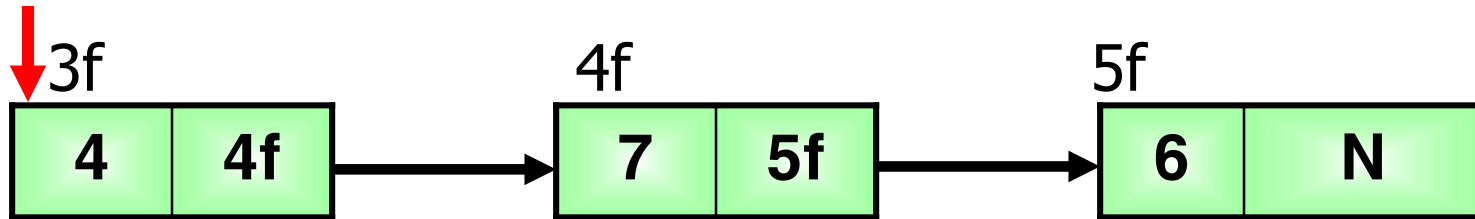
# Tìm 1 node trong danh sách liên kết đơn.

```
{
    q = p->pnext;
    while(q!=NULL)
    {
        if(q->info.mssv==msv)
        {
            p->pnext = q->pnext;
            q->pnext = NULL;
            delete q;
        }
        p = q;
        q = q->pnext;
    } } }
```



# Sắp xếp danh sách

- Có hai cách tiếp cận
- **Cách 1:** Thay đổi thành phần info list



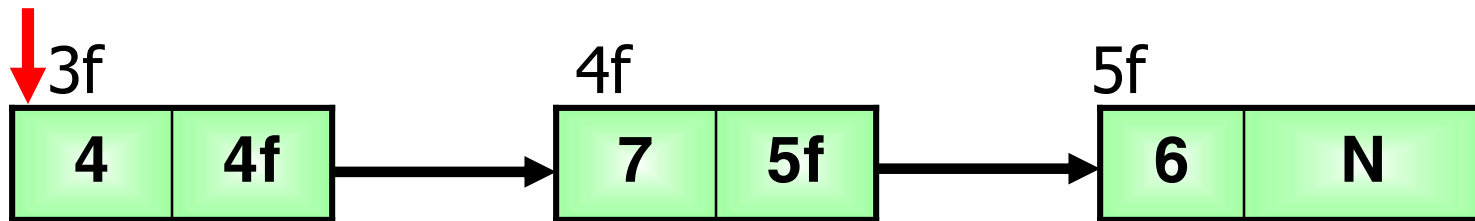
# Tìm 1 node trong danh sách liên kết đơn.

Định nghĩa hàm tìm địa chỉ của node nằm trước node  $q$  có trong danh sách liên kết đơn.

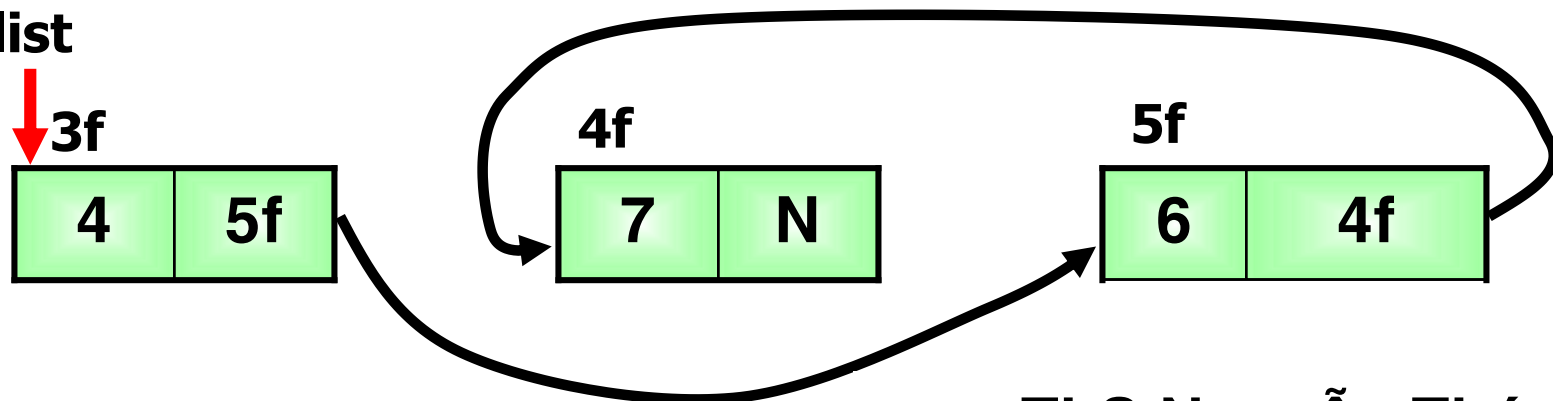
## Sắp xếp danh sách

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

list



list



ThS.Nguyễn Thúy Loan

# Ưu, nhược điểm của 2 cách tiếp cận

## 1. 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 cấu trúc 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

# Ưu, nhược điểm của 2 cách tiếp cận

## 2. 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

# Dùng thuật toán SX SelectionSort để SX List

```
void SelectionSort(NodePtr &list)
```

```
{  
    NodePtr p,q,min;  
    p=list;  
    while(p->pNext!=NULL)  
    {  
        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;  
    }  
}
```

# Các thuật toán sắp xếp hiệu quả trên List

- Các thuật toán sắp xếp xâu (List) bằng các thay đổi thành phần pNext (thành phần liên kết) có hiệu quả cao như:
  - ↪ Thuật toán sắp xếp Quick Sort
  - ↪ Thuật toán sắp xếp Merge Sort
  - ↪ Thuật toán sắp xếp Radix Sort