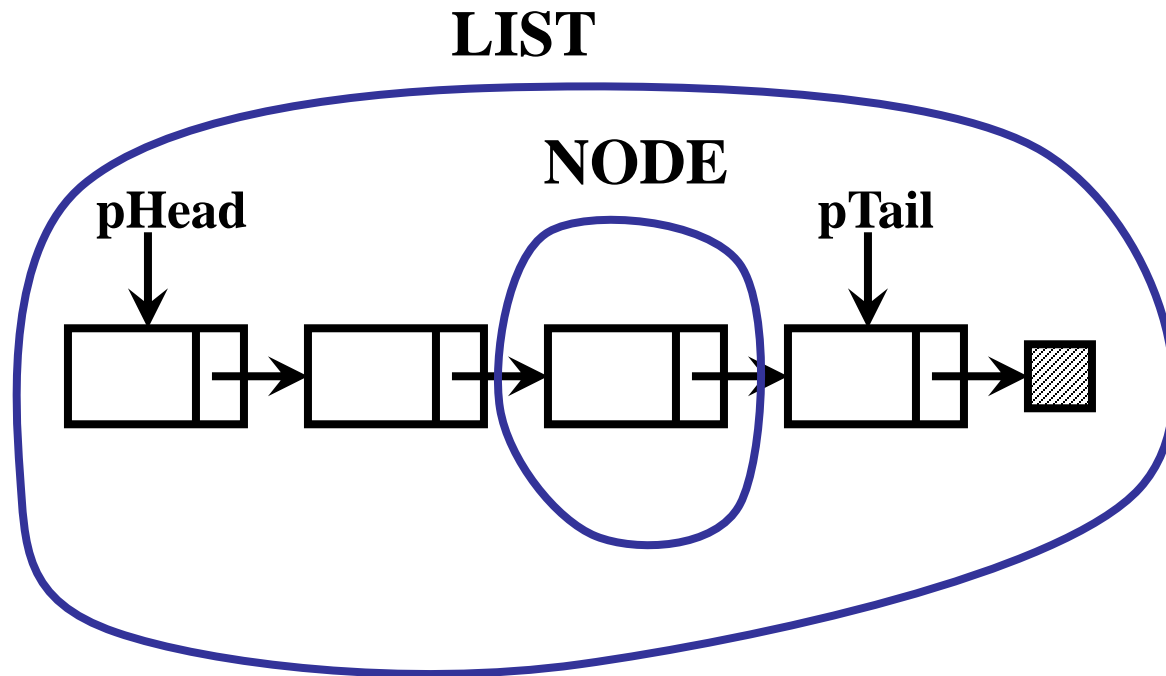


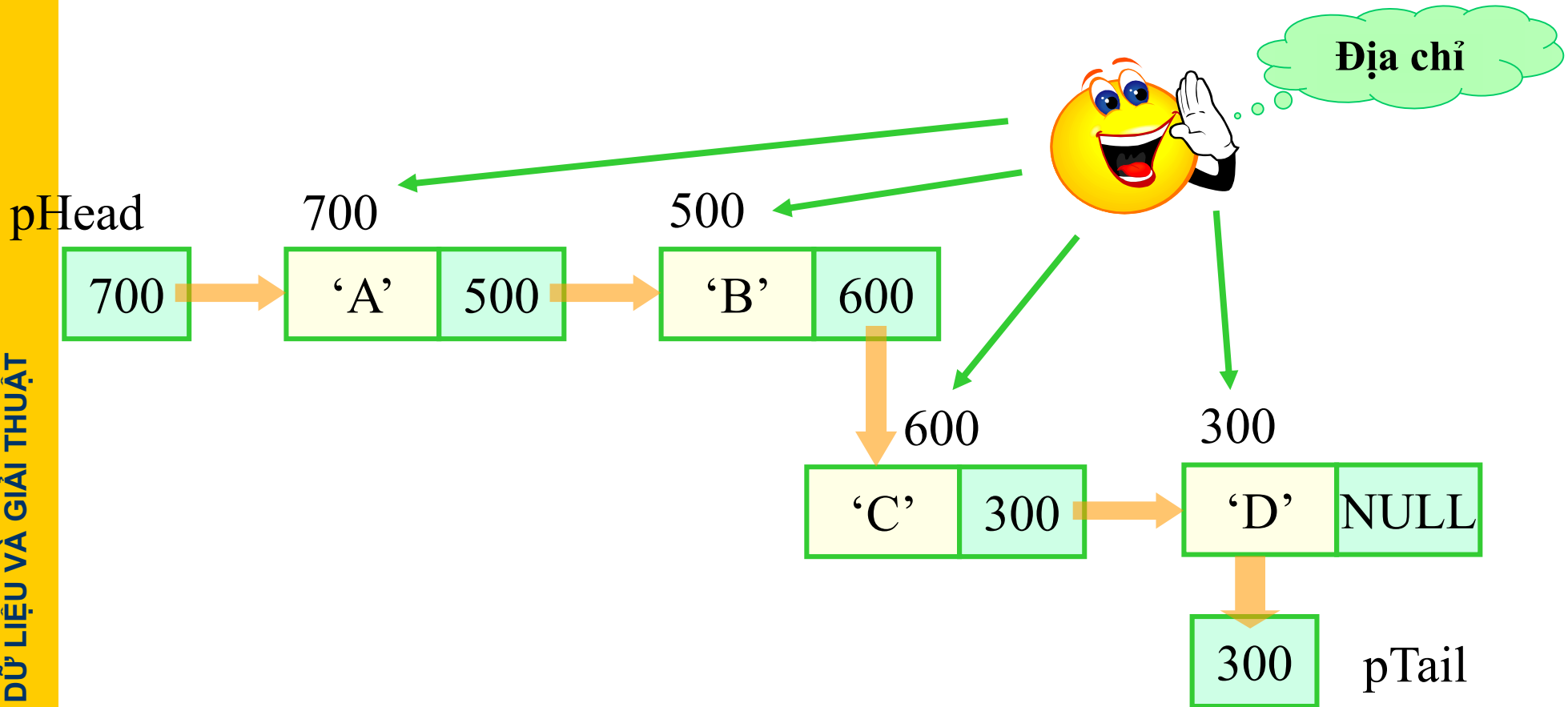
XÂU ĐƠN HAI TRỞ

Hình ảnh cấu trúc đơn hai trỏ



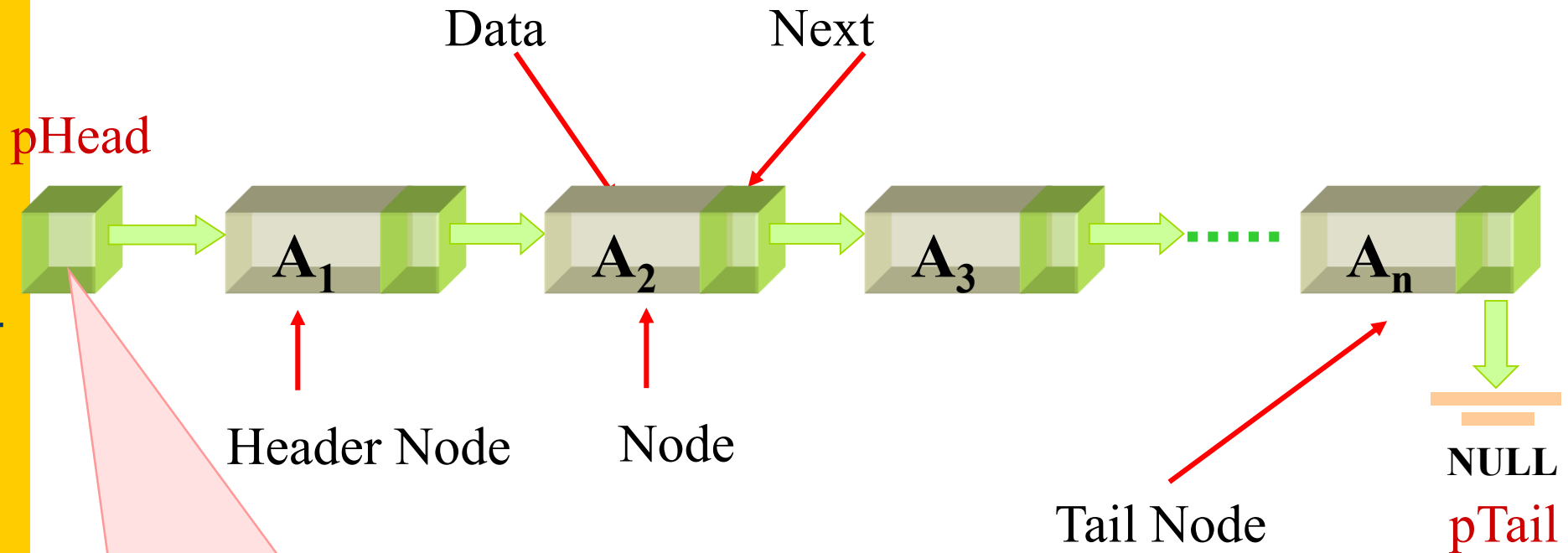
SLL – Minh họa

➤ VD:



SLL – Minh hoạ

➤ Mô tả DSLK



*Con trỏ đến node đầu tiên
(giữ địa chỉ của node đầu tiên)*

Sử dụng hai con trỏ quản lý danh sách liên kết đơn.

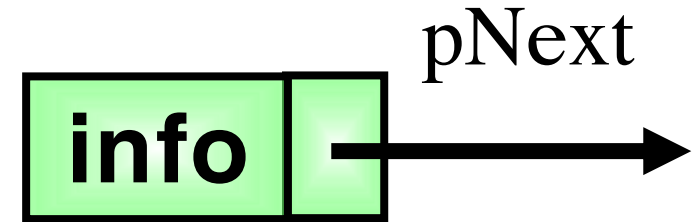
1. struct node

2. {

KDL info;

3. struct node *pNext;

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



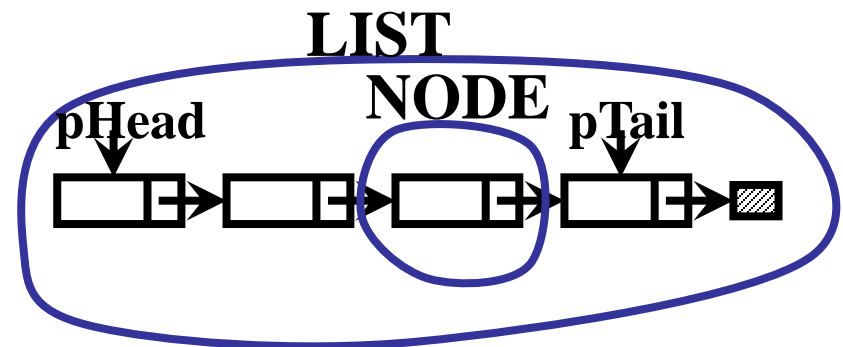
5. struct list

6. {

NODE *pHead;

7. NODE *pTail;

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



- KDL là kiểu dữ liệu của đối tượng được lưu trong danh sách liên kết đơn.

Sử dụng hai con trỏ quản lý danh sách liên kết đơn.

1. **struct node**

2. {

 KDL info;

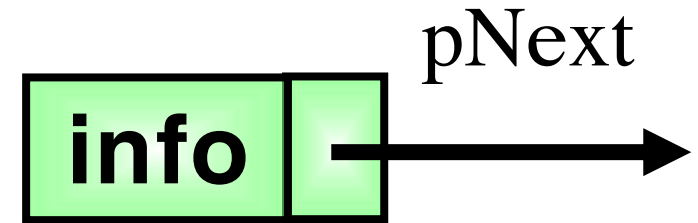
3. struct node*pNext;

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

5. Khai báo biến **NODE *a**

6. **a-> info**

a-> pNext



Sử dụng hai con trỏ quản lý danh sách liên kết đơn.

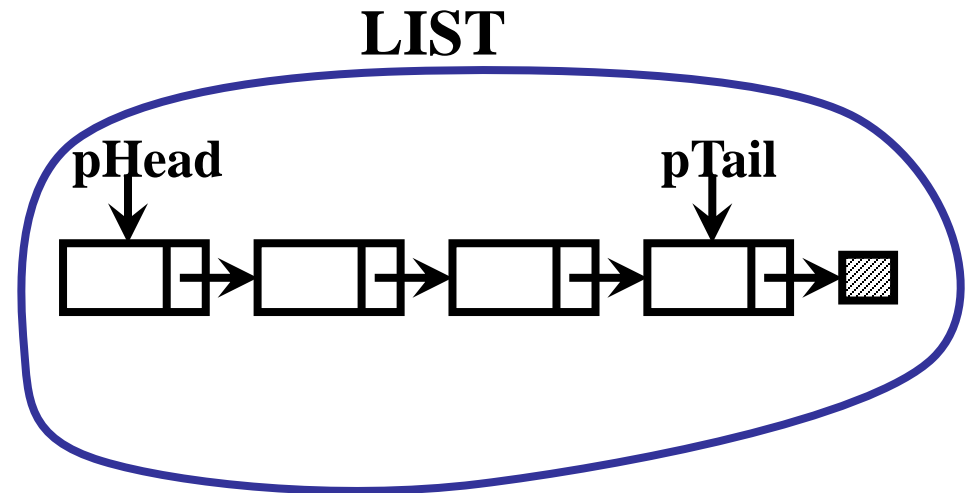
```
struct list
```

```
{
```

```
    NODE *pHead;
```

```
    NODE *pTail;
```

```
};   typedef struct list LIST;
```



- Khai báo biến **LIST l**
l.pHead
l.pTail

VD1: Hãy khai báo CTDL cho dslk đơn các số nguyên

1. **struct node**

2. { int info;

3. struct node*pNext;

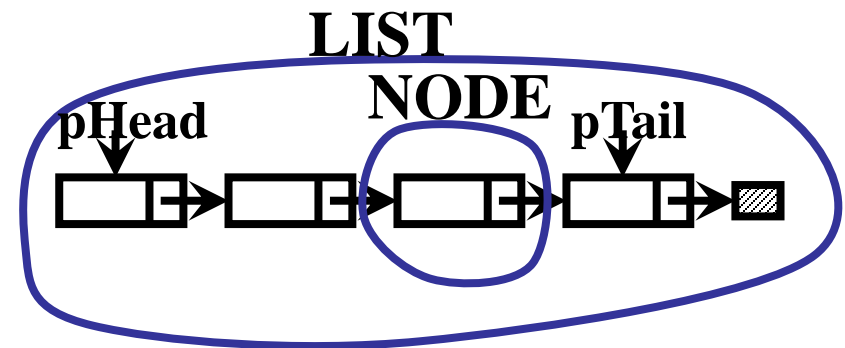
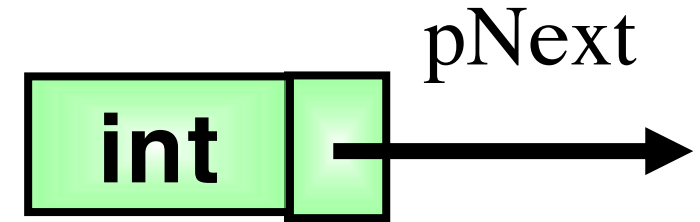
4. }; **typedef struct node NODE;**

5. **struct list**

6. { NODE *pHead;

7. NODE *pTail;

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



VD2: Hãy khai báo CTDL cho dslk đơn các số thực

1. **struct node**

2. {

float info;

3. struct node *pNext;

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

5. **struct list**

6. { NODE *pHead;

7. NODE *pTail;

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

9. **LIST l**

10. **l.pHead l.pTail**

VD2: Hãy khai báo CTDL cho dslk đơn các phân số

1. **struct phanso**

2. {

 int tu ;

3. int mau ;

4. };

5. **typedef struct phanso ps;**

VD2: Hãy khai báo CTDL cho dslk đơn các phân số

1. **struct node**

2. {

ps info;

3. struct node *pNext;

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

5. **struct list**

6. {

7. NODE *pHead;

8. NODE *pTail;

9. }; **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.

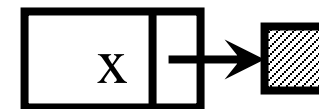
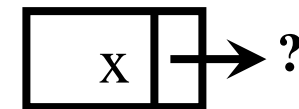
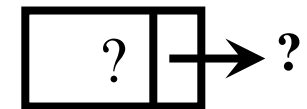
➤ Định nghĩa hàm

```
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

- **Khái niệm:** Tạo node cho danh sách liên kết đơn là xin cấp phát bộ nhớ có kích thước bằng với kích thước của kiểu dữ liệu **NODE** để chứa thông tin đã được biết trước.

```
1. NODE* GetNode (KDL x)
2. {   NODE *p=new NODE;
3.     if (p==NULL) return NULL;
4.     p->info = x;
5.     p->pNext = NULL;
6.     return p;
7. }
```

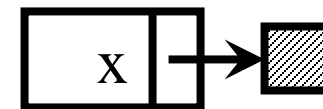
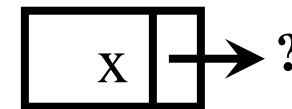


5. TẠO NODE CHO DANH SÁCH LIÊN KẾT ĐƠN

➤ 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 (float x)
2. {   NODE *p = new NODE;
3.     if (p==NULL)
4.         return NULL;
5.     p->info = x;
6.     p->pNext = NULL;
7.     return p;
8. }
```



5. TẠO NODE CHO DANH SÁCH LIÊN KẾT ĐƠN

- Ví dụ 2: Định nghĩa hàm tạo một NODE cho dslk đơn các phân số để chứa thông tin đã được biết trước.

➤ Định nghĩa hàm

```
1. NODE* GetNode (PHANSO x)
2. {
3.     NODE *p = new NODE;
4.     if (p==NULL) return NULL;
5.     p->info = x;
6.     p->pNext = NULL;
7.     return p;
8. }
```


5. TẠO NODE CHO DANH SÁCH LIÊN KẾT ĐƠN

- Ví dụ 3: Định nghĩa hàm tạo một NODE cho dslk đơn các số nguyên.

➤ Định nghĩa hàm

```
1. NODE* GetNode (int x)
2. {
3.     NODE *p = new NODE;
4.     if (p==NULL) return NULL;
5.     p->info = x;
6.     p->pNext = NULL;
7.     return p;
8. }
```

5. TẠO NODE CHO DANH SÁCH LIÊN KẾT ĐƠN

- Ví dụ 4: Định nghĩa hàm tạo một NODE cho dslk đơn tọa độ các điểm trong mặt phẳng Oxy.

➤ Định nghĩa hàm

```
1. NODE* GetNode (DIEM P)
2. {
3.     NODE *p = new NODE;
4.     if (p==NULL)     return NULL;
5.     p->info = P;
6.     p->pNext = NULL;
7.     return p;
8. }
```

5. TẠO NODE CHO DANH SÁCH LIÊN KẾT ĐƠN

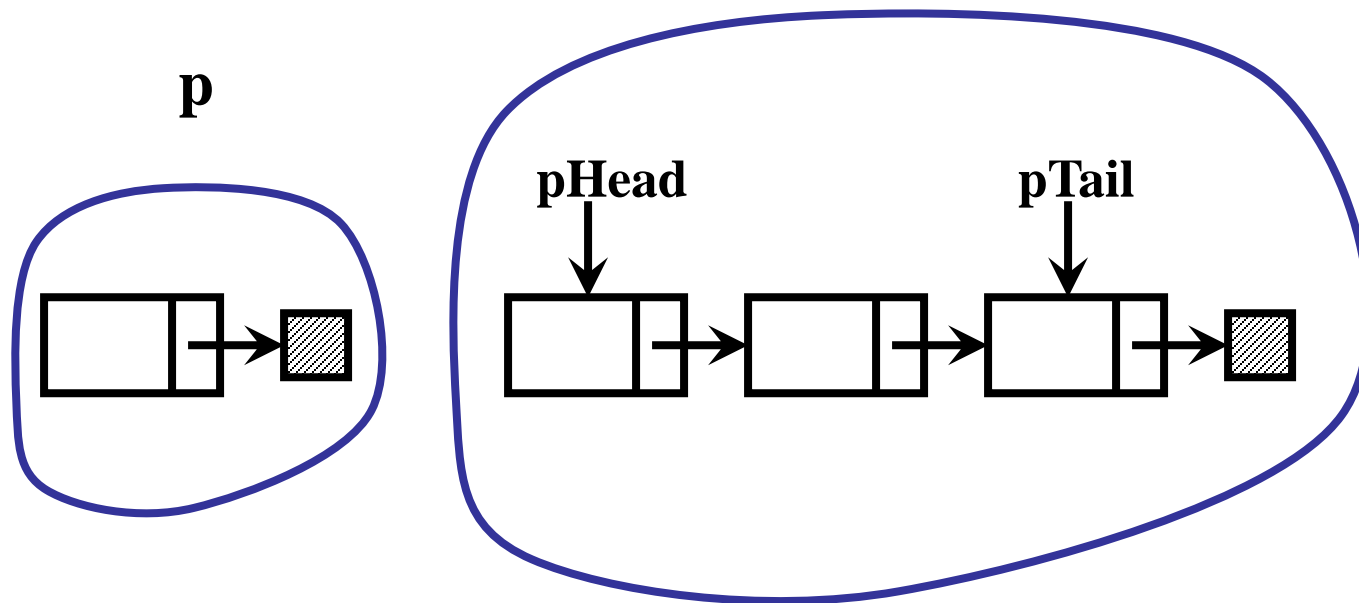
▪ Ví dụ 4: Định nghĩa hàm tạo một NODE cho dslk đơn lưu trữ sv.

➤ Định nghĩa hàm

```
1. NODE* GetNode (sv a)
2. {
3.     NODE *p = new NODE;
4.     if (p==NULL)    return NULL;
5.     p->info = a;
6.     p->pNext = NULL;
7.     return p;
8. }
```

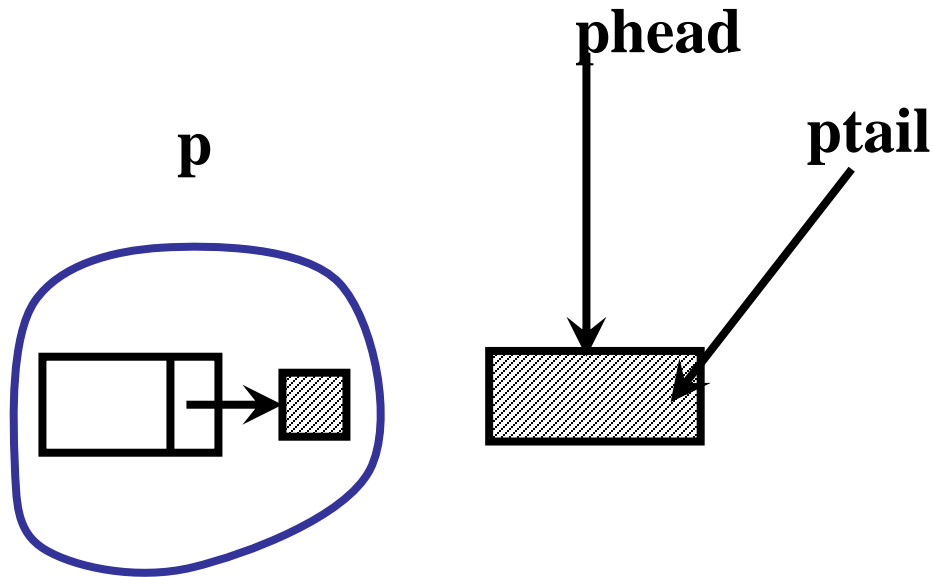
6. THÊM MỘT NODE VÀO ĐẦU DS LIÊN KẾT ĐƠN

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

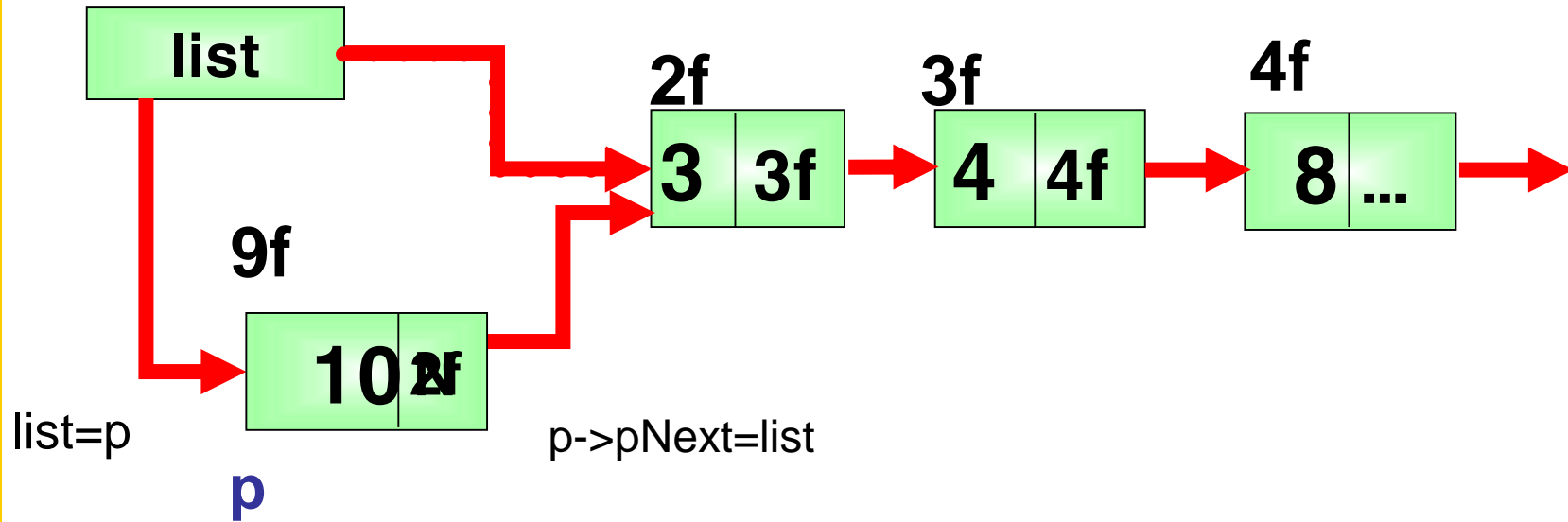


6. THÊM MỘT NODE VÀO ĐẦU DS LIÊN KẾT ĐƠN

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



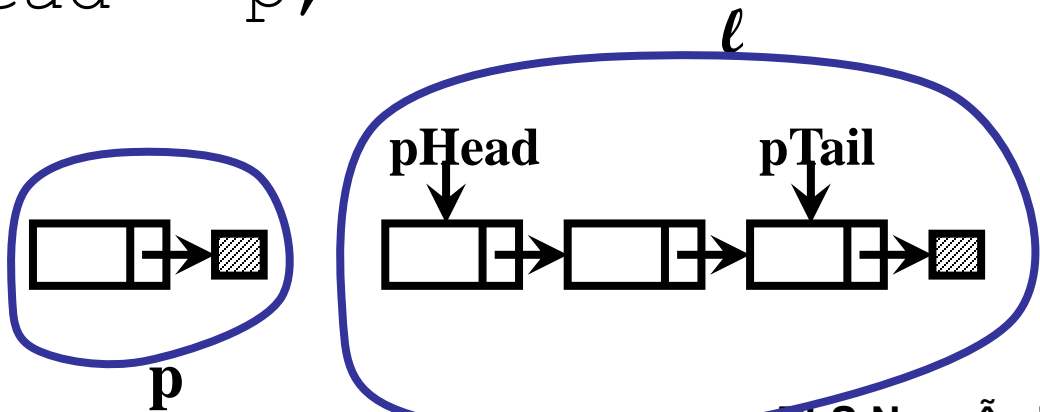
Minh họa thuật toán thêm vào đầu



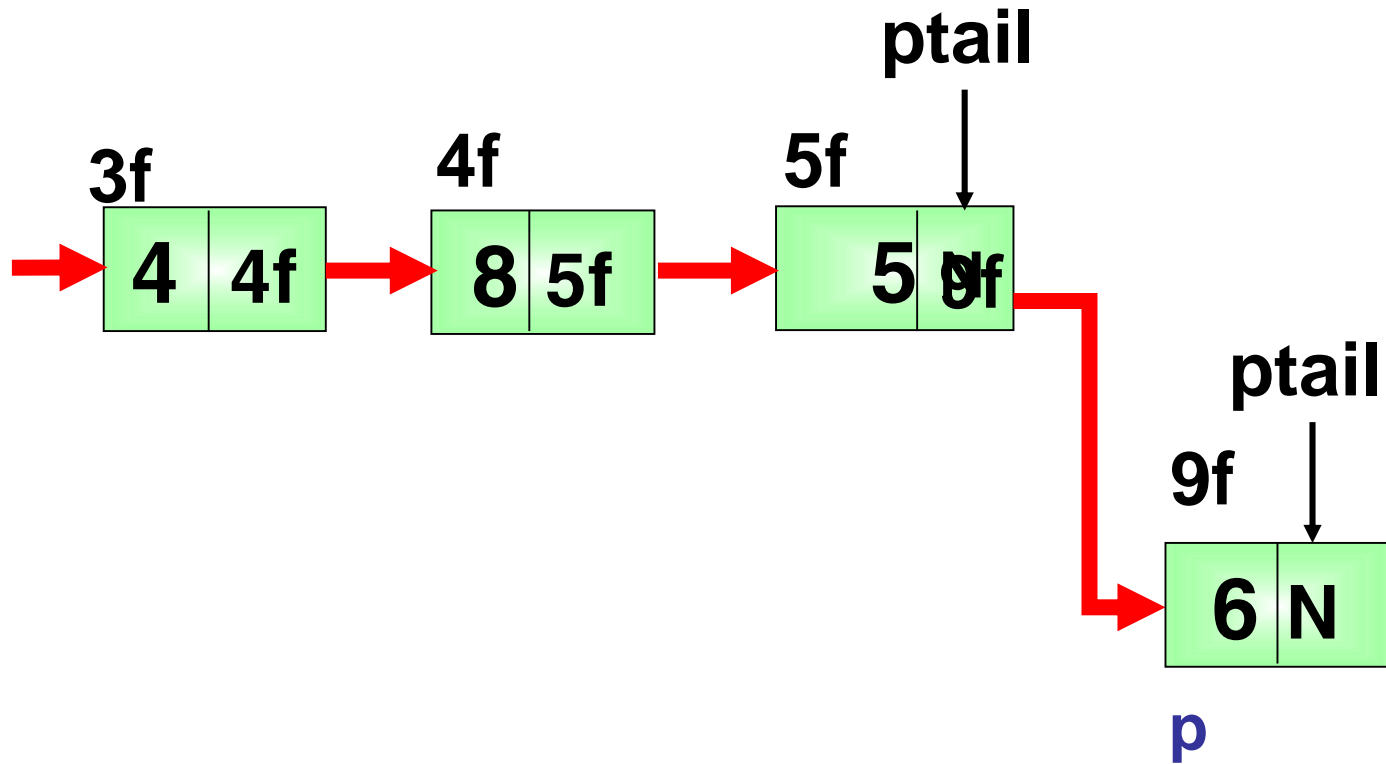
6. THÊM MỘT NODE VÀO ĐẦU DS LIÊN KẾT ĐƠN

➤ 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. }
```



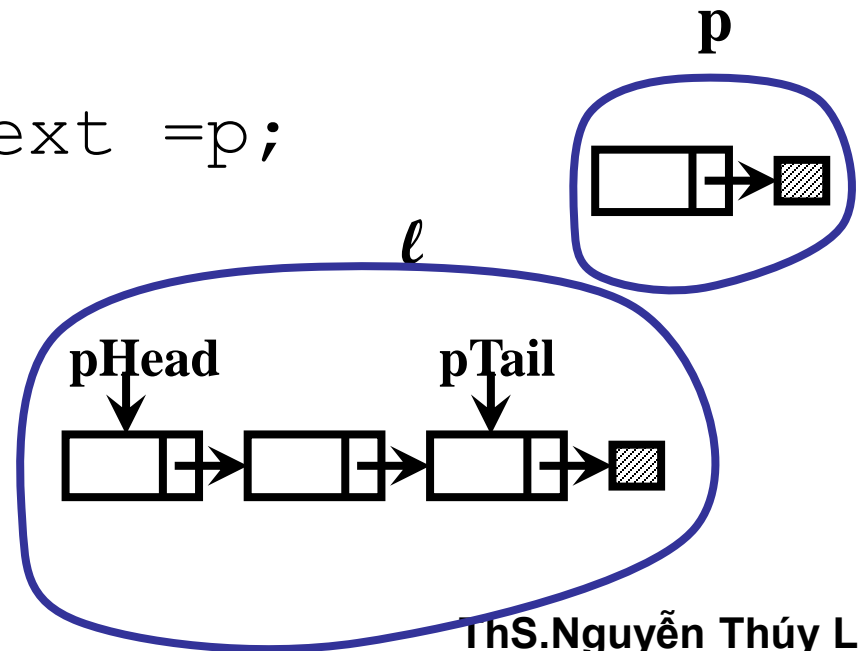
Minh họa thuật toán thêm vào cuối



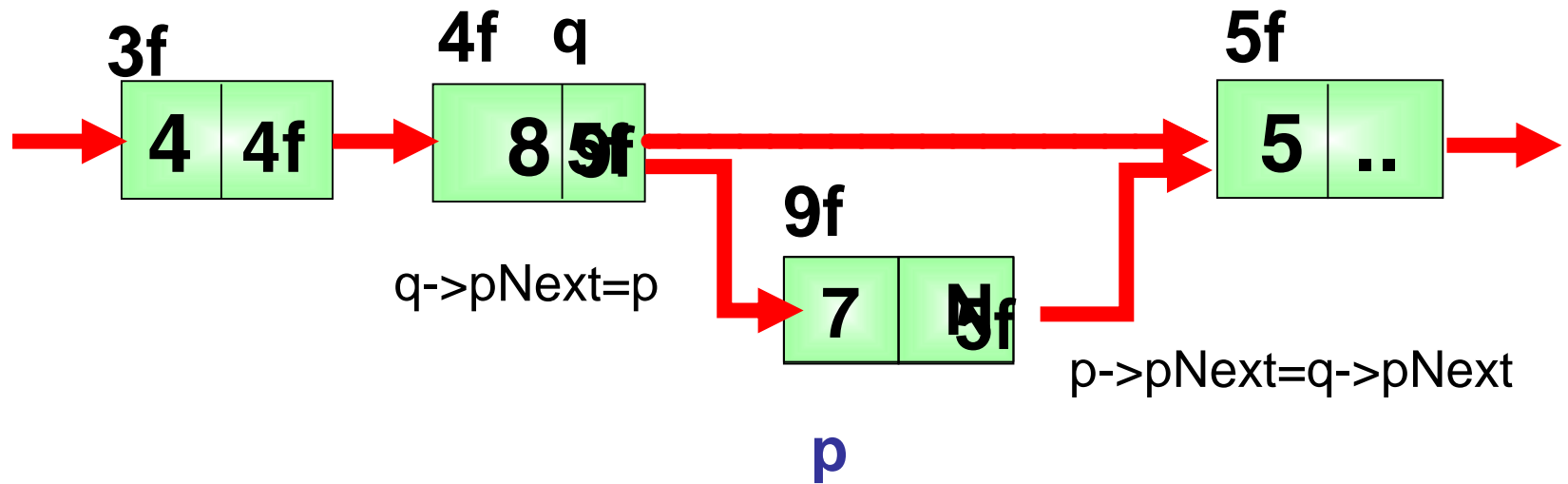
6. THÊM MỘT NODE VÀO CUỐI DS LIÊN KẾT ĐƠN

➤ 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. }
```



Thêm node p vào sau node q



Thêm node p vào sau node q

Thêm nút p vào sau nút q trong list đơn

Bắt đầu:

Nếu ($q \neq \text{NULL}$) thì

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

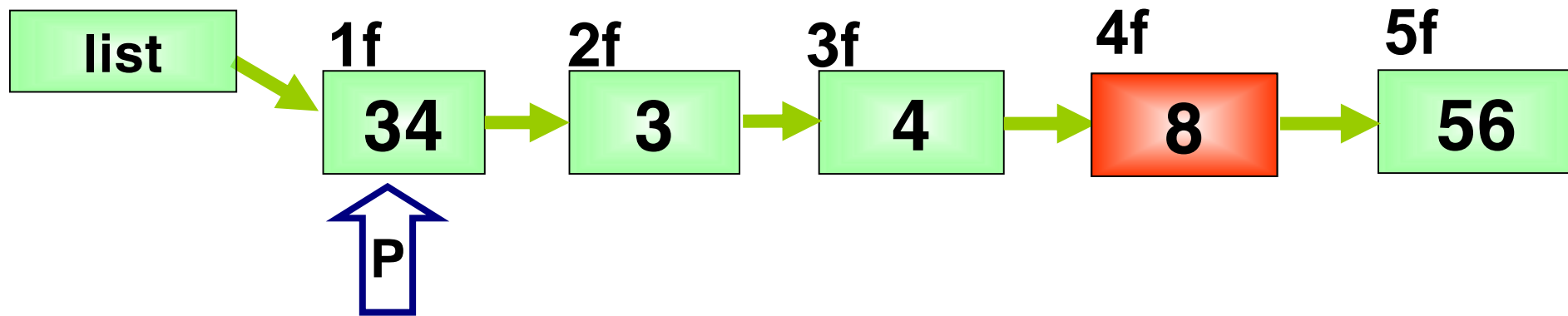
B2: $q \rightarrow \text{pNext} = p$

Ngược lại: Thêm vào đầu danh sách

Cài đặt thuật toán

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

Minh họa thuật toán tìm phần tử trong DSLK



X = 8

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

ThS. Nguyễn Thúy Loan

Tìm 1 phần tử trong DSLK đơn

Tìm tuần tự : Tìm nút có info bằng x trong list đơn

➤ Bước 1: $p = \text{list}$; // địa chỉ của phần tử đầu trong ds đơn

➤ Bước 2:

Trong khi $p \neq \text{NULL}$ và $p \rightarrow \text{info} \neq x$

$p = p \rightarrow \text{pNext}$; // xét phần tử kế

➤ Bước 3:

Nếu $p \neq \text{NULL}$ thì p lưu địa chỉ của nút có
 $\text{info} = x$

Ngược lại: Không có phần tử cần tìm

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;

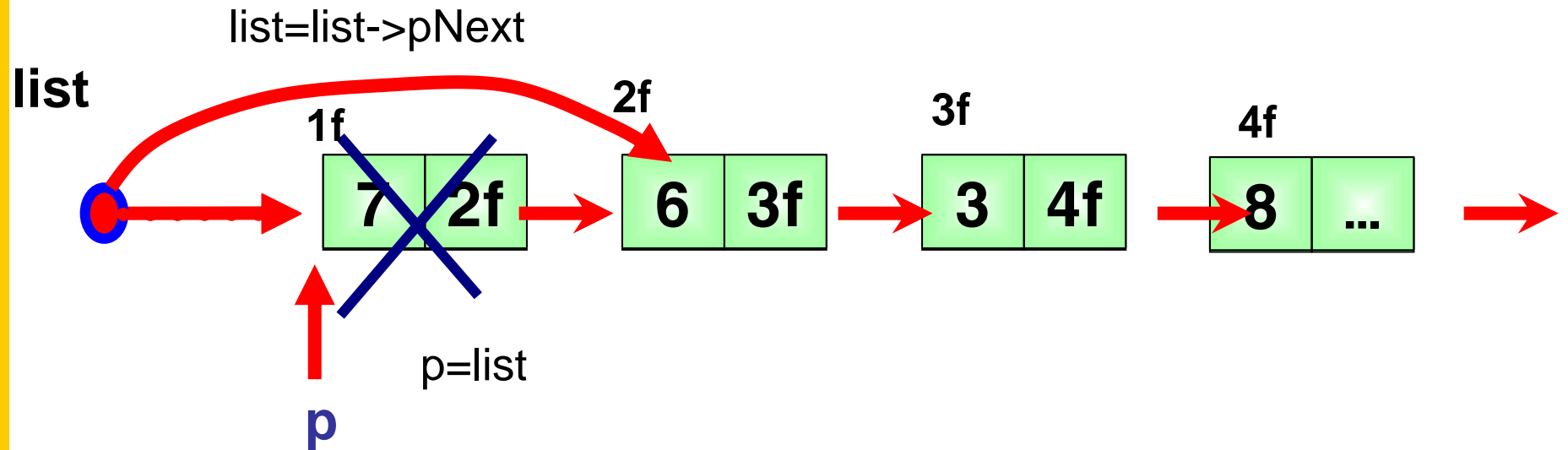
 while(p!=NULL && p->info!=x)

 p=p->pnext;

 return p;

}

Thuật toán hủy phần tử đầu trong DSLK



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

➤ Bắt đầu:

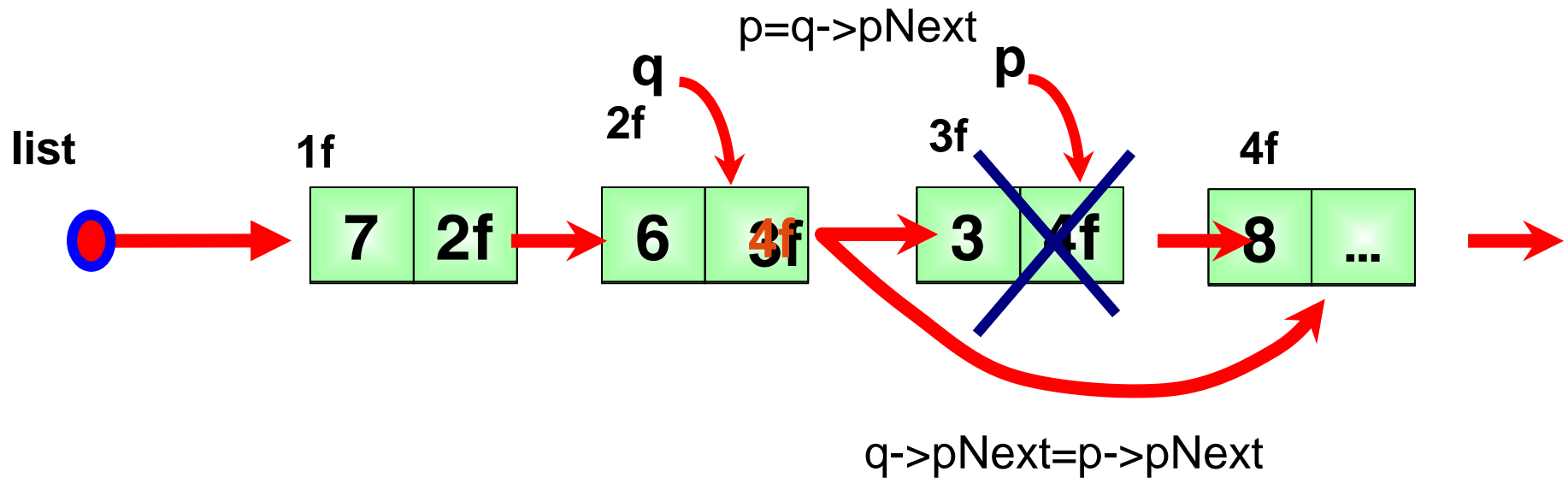
↪ Nếu ($l \neq \text{NULL}$) thì

- 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;
        p->pnext =NULL;
        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

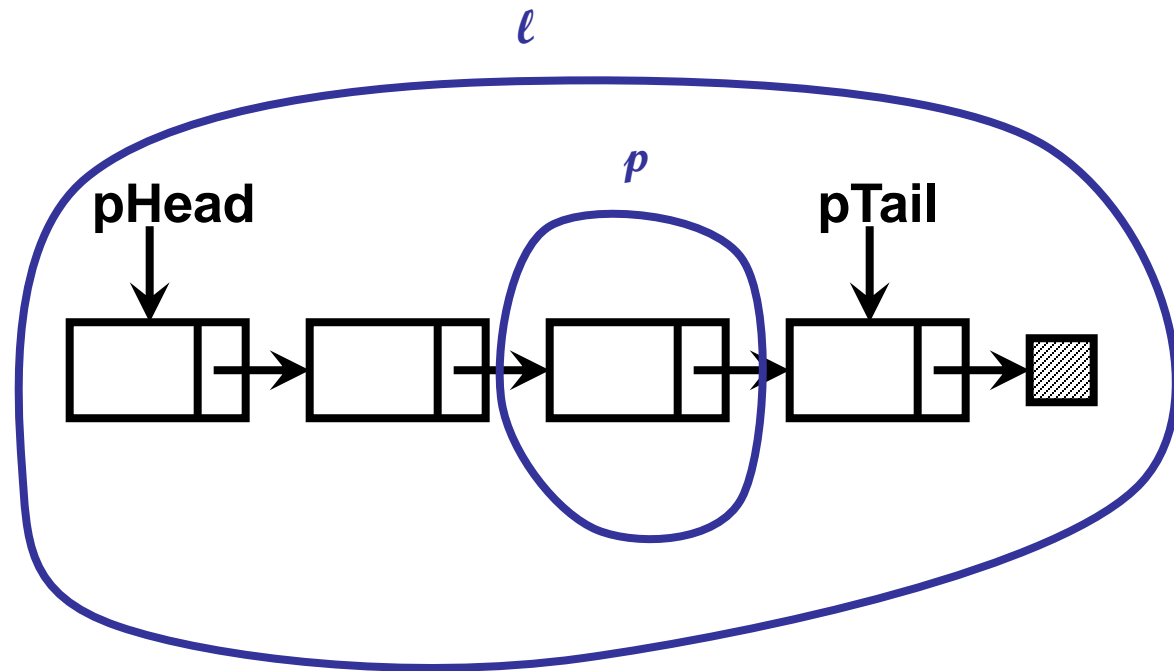
Hủy p ra khỏi danh sách bằng cách
hủy phần tử đứng sau q

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!=NULL)
        lc=lc->pnext;
    return lc;
}
```

Hủy phần tử x trong List

```
void delete x ( list &l , int x )
{
    node *q, *p ;
    p=l.phead ;
    if ( p->info == x )
    {
        l.phead = l.phead->pnext ;
        delete p;
    }
    else
    {

```

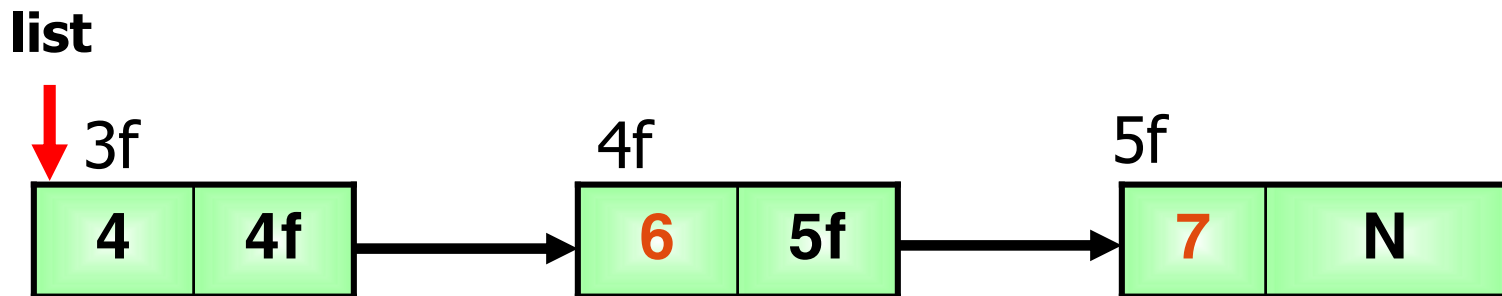
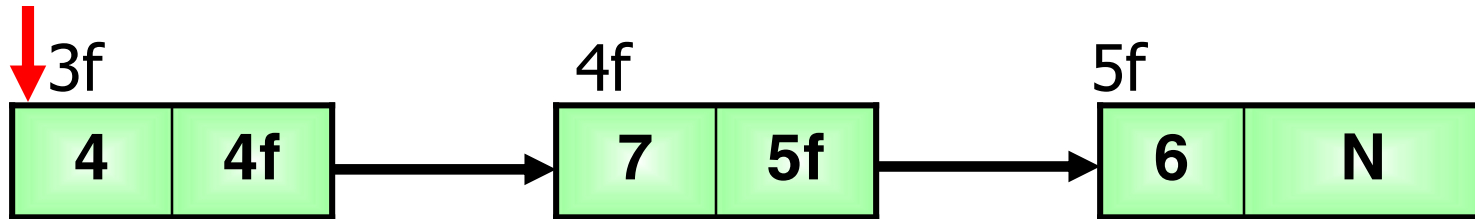
Hủy phần tử x trong List

else

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

Sắp xếp danh sách

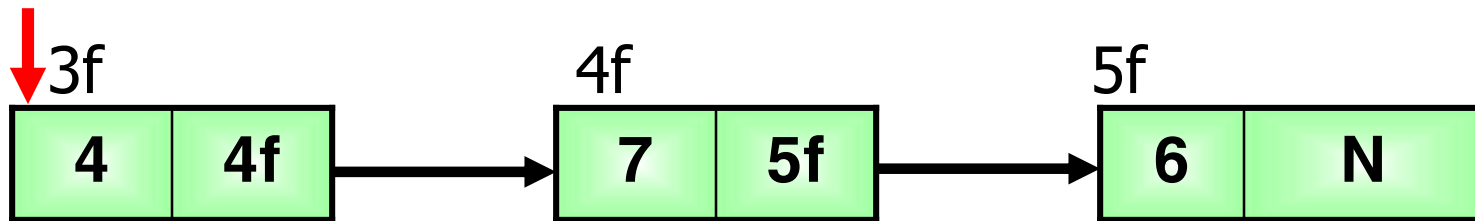
- Có hai cách tiếp cận
- **Cách 1:** Thay đổi thành phần info list



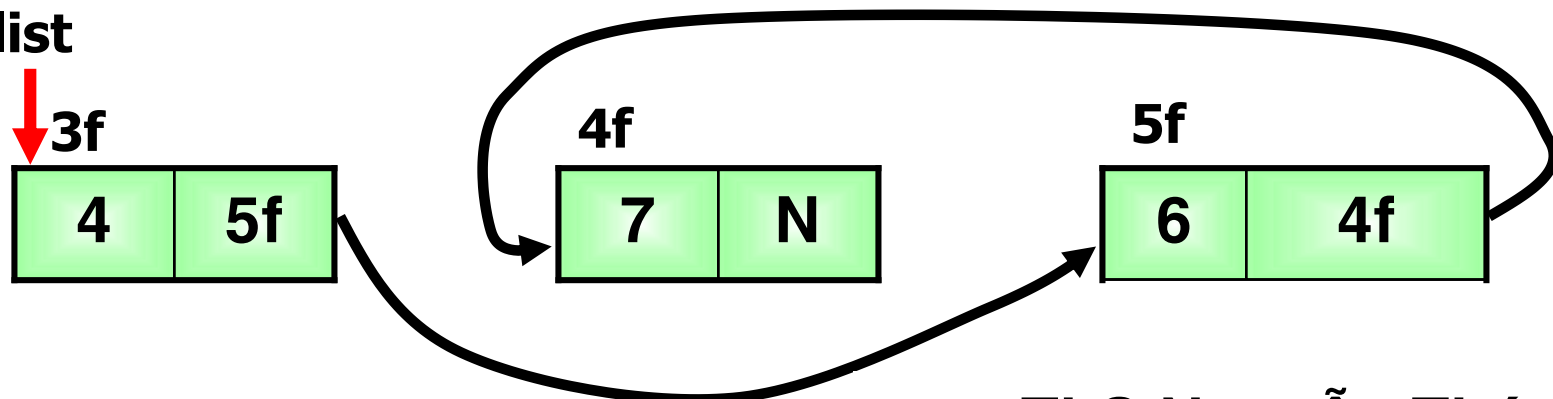
Sắp xếp danh sách

- **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:

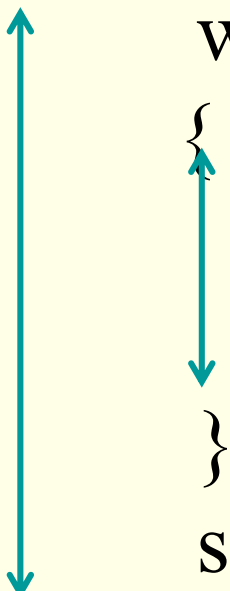
Kích thước của trường này không thay đổi, do đó không phụ thuộc vào kích thước bản chất dữ liệu lưu tại mỗi nút.

Thao tác sắp xếp nhanh

Nhược: Cài đặt phức tạp

SLL- 10. Sắp xếp

```
void saptang ( list &l )
{
    NODE * q, *min, *p = l.phead ;
    while (p!=NULL)
    {
        min = p; q = p;
        while (q!=NULL)
        {
            if ( q->info < min->info )
                min = q;
            q = q->pnext;
        }
        swap ( p->info , min->info ) ;
        p = p->pnext;
    }
}
```



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

```
1. void Input (LIST &l)
2. {   int x;
3.     Init(l );
4.     do{
5.         printf("Nhap gia tri x: ");
6.         scanf("%d", &x);
7.         if( x!=0)
8.         {
9.             NODE*p=GetNode(x) ;
10.            if (p!=NULL)
11.                Addtail(l,p) ;
12.        }
13.    }while (x!=0)
14. }
```

7. NHẬP TỪ BÀN PHÍM DS LIÊN KẾT ĐƠN

➤ Khái niệm: Nhập từ bàn phím dslk đơn là lần lượt nhập các thông tin của từng node trong danh sách.

➤ Định nghĩa hàm trừu tượng

```
1. void Input (LIST&ℓ)
2. {   int n;
3.     printf("Nhap n: ");
4.     scanf("%d", &n);
5.     Init(ℓ);
6.     for(int i=1; i<=n; i++)
7.     {   KDL x;
8.         Nhap(x);
9.         NODE*p = GetNode(x);
10.        if (p!=NULL)
11.            AddHead(ℓ, p);
12.    }
13. }
```

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

```
1. void Input (LIST&l)
2. {
    1. float x
    2. Init (l) ;
3.     do
4.         {   printf ("Nhap so thuc: ");
5.             scanf ("%f", &x) ;
6.                 1. if (x!=0)
7.                     2. {
8.                         NODE*p=GetNode (x) ;
9.                         if (p!=NULL)
10.                            Addtail (l,p) ;
11.                     }
12.             1. } while (x!=0)
13. }
```

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

```
1. void Input (LIST & l)
2. {
3.     int n;
4.     printf("Nhap n: ");
5.     scanf("%d", &n);
6.     Init(l) ;
7.     for(int i=1; i<=n; i++)
8.     {
9.         int x;
10.        printf("Nhap so nguyen:");
11.        scanf("%d", &x);
12.        NODE*p = GetNode(x) ;
13.        if (p!=NULL)
14.            AddHead(l,p) ;
15.    }
16. }
```

Ví dụ: Nhập danh sách liên kết đơn các phân số

```
1. void Input (LIST&l)
2. {
3.     printf("Nhap n: ");
4.     scanf("%d", &n);
5.     Init(l);
6.     for(int i=1; i<=n; i++)
7.     {
8.         PHANSO x;
9.         printf("Nhap phan so: ");
10.        Nhap(x);
11.        NODE*p=GetNode(x);
12.        if (p!=NULL)
13.            AddHead(l,p);
14.    }
15. }
```

Ví dụ: Nhập danh sách liên kết đơn các phân số

```
1. void Nhap (PHANSO &x)
2. {
3.     printf("Nhap tu: ");
4.     scanf("%d", &x.tu);
5.     printf("Nhap mau: ");
6.     scanf("%d", &x.mau);
7. }
```

VD:Tính tổng các số lẻ trong dslk đơn các số nguyên.

```
10.int  TongLe (LIST l)
11.{
12.    int s = 0;
13.    NODE*p = l.pHead;
14.    while (p!=NULL)
15.    {
16.        if (p->info%2!=0)
17.            s = s + p->info;
18.        p = p->pNext;
19.    }
20.    return s;
21.}
```

8. DUYỆT TUẦN TỰ DANH SÁCH LIÊN KẾT ĐƠN

➤ **Khái niệm:** duyệt danh sách liên kết đơn là thăm qua tất cả các node mỗi node một lần.

➤ Định nghĩa hàm trừu tượng

```
11.KDL <Tên Hàm> (LIST l)
```

```
12. {      ...
```

```
13.     NODE*p = l.pHead;
```

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

```
15.     {
```

```
16.         ...
```

```
17.         p = p->pNext;
```

```
18.     }
```

```
19.     ...
```

```
20. }
```


9. CHƯƠNG TRÌNH ĐẦU TIÊN DSLK ĐƠN

Viết chương trình thực hiện các yêu cầu sau:

- Nhập dslk đơn các số nguyên.
- Tính tổng các giá trị trong dslk đơn.
- Xuất dslk đơn.

```
1. #include "stdio.h"
2. #include "conio.h"
3. #include "math.h"
4. #include "string.h"
5. struct node
6. {
7.     int info;
8.     struct node *pNext;
9. }; typedef struct node NODE;
```

9. CHƯƠNG TRÌNH ĐẦU TIÊN DSLK ĐƠN

```
struct list
{
    NODE*pHead;
    NODE*pTail;
};
typedef struct list LIST;
// Khai báo hàm
void Init(LIST&);
NODE* GetNode(int);
void AddHead(LIST&,NODE*);
void Input(LIST&);
void Output(LIST);
int Tong(LIST);
```

9. CHƯƠNG TRÌNH ĐẦU TIÊN DSLK ĐƠN

```
// Hàm main
void main()
{
    LIST lst;
    Input(lst);
    Output(lst);
    int kq = Tong(lst);
    printf("Tong la %d",kq);
}
```

9. CHƯƠNG TRÌNH ĐẦU TIÊN DSLK ĐƠN

➤ Định nghĩa hàm

```
1. NODE* GetNode (int x)
2. {
3.     NODE *p = new NODE;
4.     if (p==NULL)
5.         return NULL;
6.     p->info = x;
7.     p->pNext = NULL;
8.     return p;
9. }
```

9. CHƯƠNG TRÌNH ĐẦU TIÊN DSLK ĐƠN

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

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

```
1. void Input (NODE * &phead)
2. {   int x;
3.   Init (phead) ;
4.   do {
5.       printf ("Nhap gia tri x: ");
6.       scanf ("%d", &x) ;
7.       if ( x!=0)
8.       {
9.           NODE*p=GetNode (x) ;
10.          if (p!=NULL)
11.              AddHead (phead,p) ;
12.      }
13.  } while (x!=0)
14. }
```

9. CHƯƠNG TRÌNH ĐẦU TIÊN DSLK ĐƠN

```
11. void AddHead (LIST&l, NODE*p)
12. {
13.     if (l.pHead==NULL)
14.         l.pHead=l.pTail = p;
15.     else
16.     {
17.         p->pNext = l.pHead;
18.         l.pHead = p;
19.     }
20. }
```

9. CHƯƠNG TRÌNH ĐẦU TIÊN DSLK ĐƠN

```
10. void Input (LIST&ℓ)
11. {   int n;
12.     printf("Nhap số sinh viên : ");
13.     scanf ("%d", &n) ;
14.     Init (ℓ) ;
15.     for (int i=1; i<=n; i++)
16.     {   sv x;
17.         printf ("Nhap 1 sv:");
18.         nhap1sv (x) ;
19.         NODE*p=GetNode (x) ;
20.         if (p!=NULL)
21.             AddHead (ℓ, p) ;
22.     }
23. }
```


9. CHƯƠNG TRÌNH ĐẦU TIÊN DSLK ĐƠN

```
1. void Output (LIST l)
2. {
3.     NODE*p = l.pHead;
4.     while (p!=NULL)
5.     {
6.         printf ("%4d", p->info) ;
7.         p = p->pNext;
8.     }
9. }
```

9. CHƯƠNG TRÌNH ĐẦU TIÊN DSLK ĐƠN

➤ Định nghĩa hàm

```
10.int  Tong(LIST l)
11.{
12.    int s = 0;
13.    NODE*p = l.pHead;
14.    while (p!=NULL)
15.    {
16.        s = s + p->info;
17.        p = p->pNext;
18.    }
19.    return s;
20.}
```

Bài toán: dùng DS liên kết đơn 1 trở, viết chương trình

1. Nhập vào một dãy số nguyên , nhập cho đến khi gặp số 0 thì dừng.
2. Xuất ra dãy số vừa nhập.
3. Tính tổng các phần tử có trong dãy số
4. Tìm phần tử lớn nhất có trong dãy.
5. Thêm phần tử có giá trị là 3 vào sau phần tử có giá trị là 2 đầu tiên ở trong dãy số. Nếu không tồn tại phần tử có giá trị là 2 thì thêm vào đầu dãy
6. Xóa phần tử âm đầu tiên có trong dãy.
7. Xóa phần tử có giá trị là 5 cuối cùng có trong dãy số

VD1: Hãy khai báo CTDL cho dslk đơn các số nguyên

1. **struct node**

2. { int info;

3. struct node*pNext;

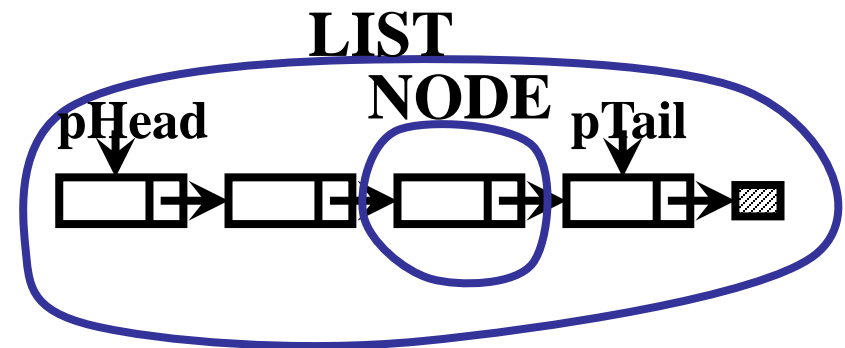
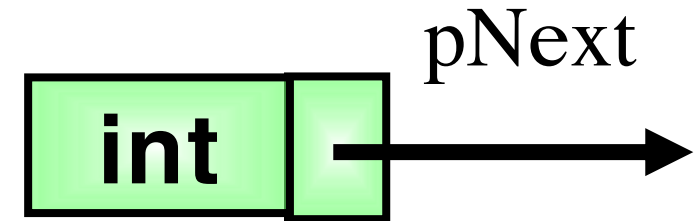
4. }; **typedef struct node NODE;**

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. }
```

10. CHƯƠNG TRÌNH THỨ HAI DSLK ĐƠN

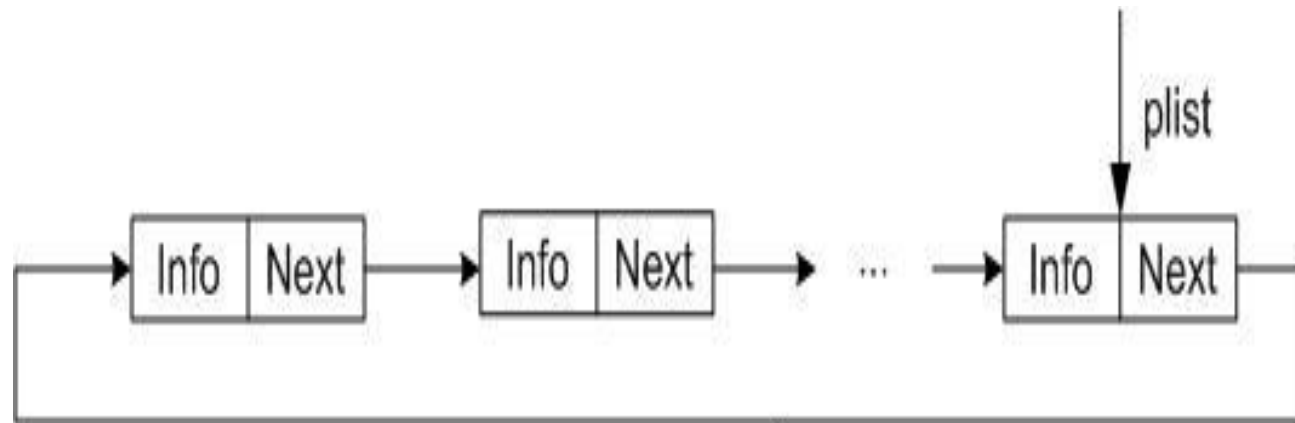
- Bài toán: Viết chương trình thực hiện các yêu cầu sau:
 - ↪ Nhập dslk đơn các số thực.
 - ↪ Xuất dslk đơn.
 - ↪ Đếm số lượng giá trị âm trong dslk đơn.
 - ↪ Tìm địa chỉ node lớn nhất trong dslk đơn.

SLL– Bài tập bổ sung

1. Thêm vào cuối danh sách
2. Sắp xếp danh sách
3. Xoá 1 phần tử có khoá là x
4. Thêm phần tử x vào ds đã có thứ tự (tăng) sao cho sau khi thêm vẫn có thứ tự (tăng).
5. Xác định vị trí của node x trong danh sách
6. Xác định kích thước của danh sách (số phần tử)
7. Chèn một phần tử có khoá x vào vị trí pos trong ds
8. Xoá các phần tử trùng nhau trong danh sách, chỉ giữ lại duy nhất một phần tử (*)
9. Trộn hai danh sách có thứ tự tăng thành một danh sách cũng có thứ tự tăng. (*)

ThS.Nguyễn Thúy Loan

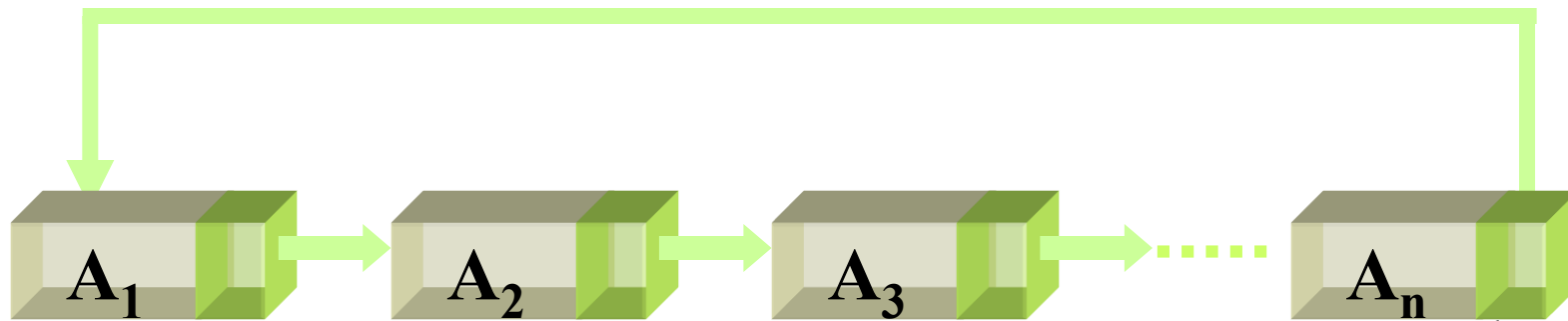
DANH SÁCH LIÊN KẾT VÒNG



Hình: Danh sách liên kết vòng

Circular Linked List

- Tương tự như danh sách liên kết đơn.
- Trường next của nút cuối chỉ đến đầu danh sách

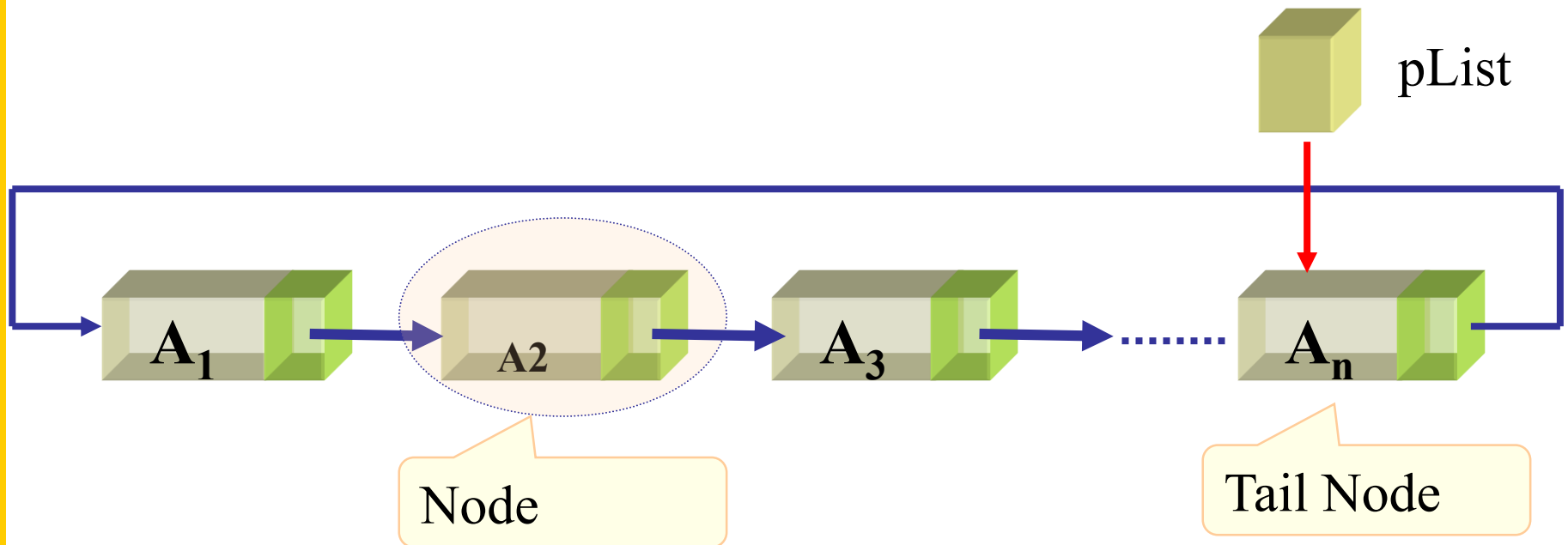


Trường Next của nút cuối ko còn trở đến NULL, mà trở đến nút đầu

Circular Linked List

➤ Mô tả CLL

↪ Sử dụng pList trỏ đến phần tử cuối của danh sách



CTDL DANH SÁCH LIÊN KẾT ĐƠN VÒNG

```
1.struct node
```

```
2.{
```

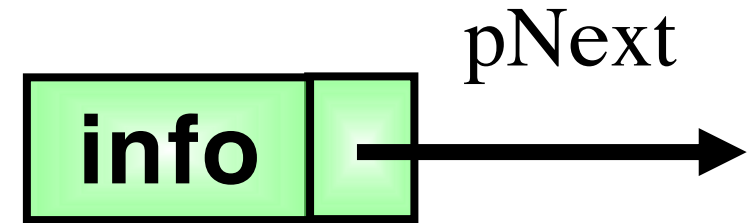
```
3.    KDL info;
```

```
4.    struct node*pNext;
```

```
5.};
```

```
6.typedef struct node NODE;
```

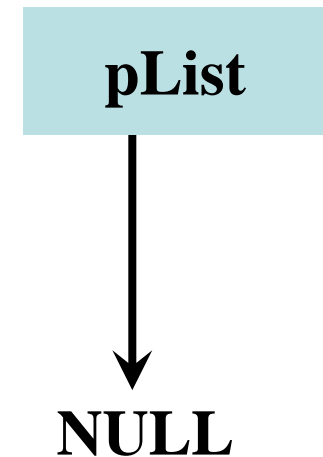
```
7.typedef NODE *Clist;
```



1.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(Clist &pList)  
2. {  
3.     pList = NULL;  
4. }
```

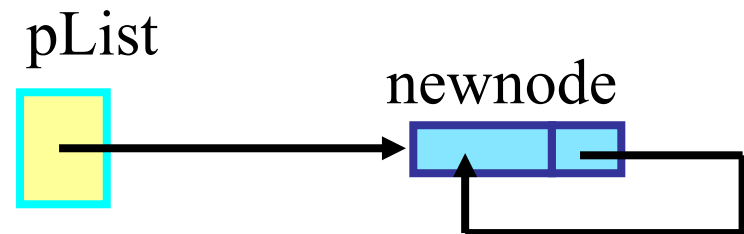
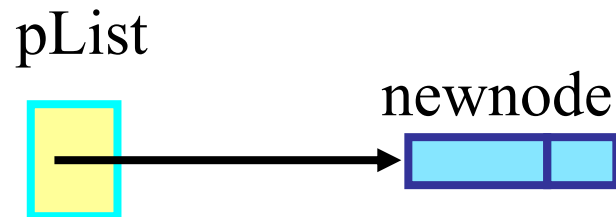
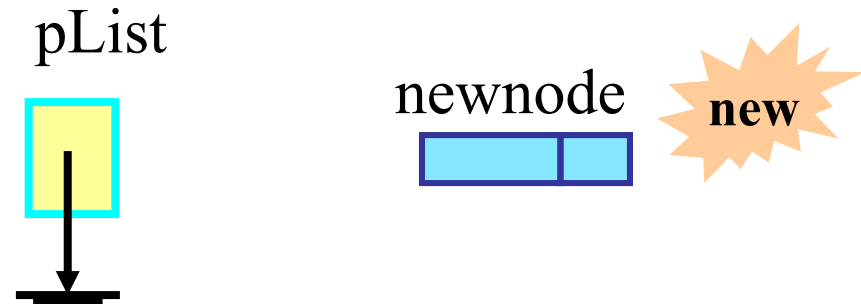
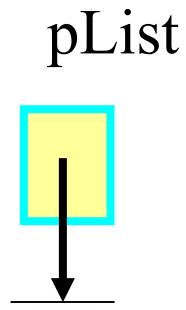


Circular Linked List

- Các thao tác
 1. InsertFirst
 2. InsertLast
 3. DeleteFirst
 4. DeleteLast
 5. ShowList
 6. Search
 7. AddList

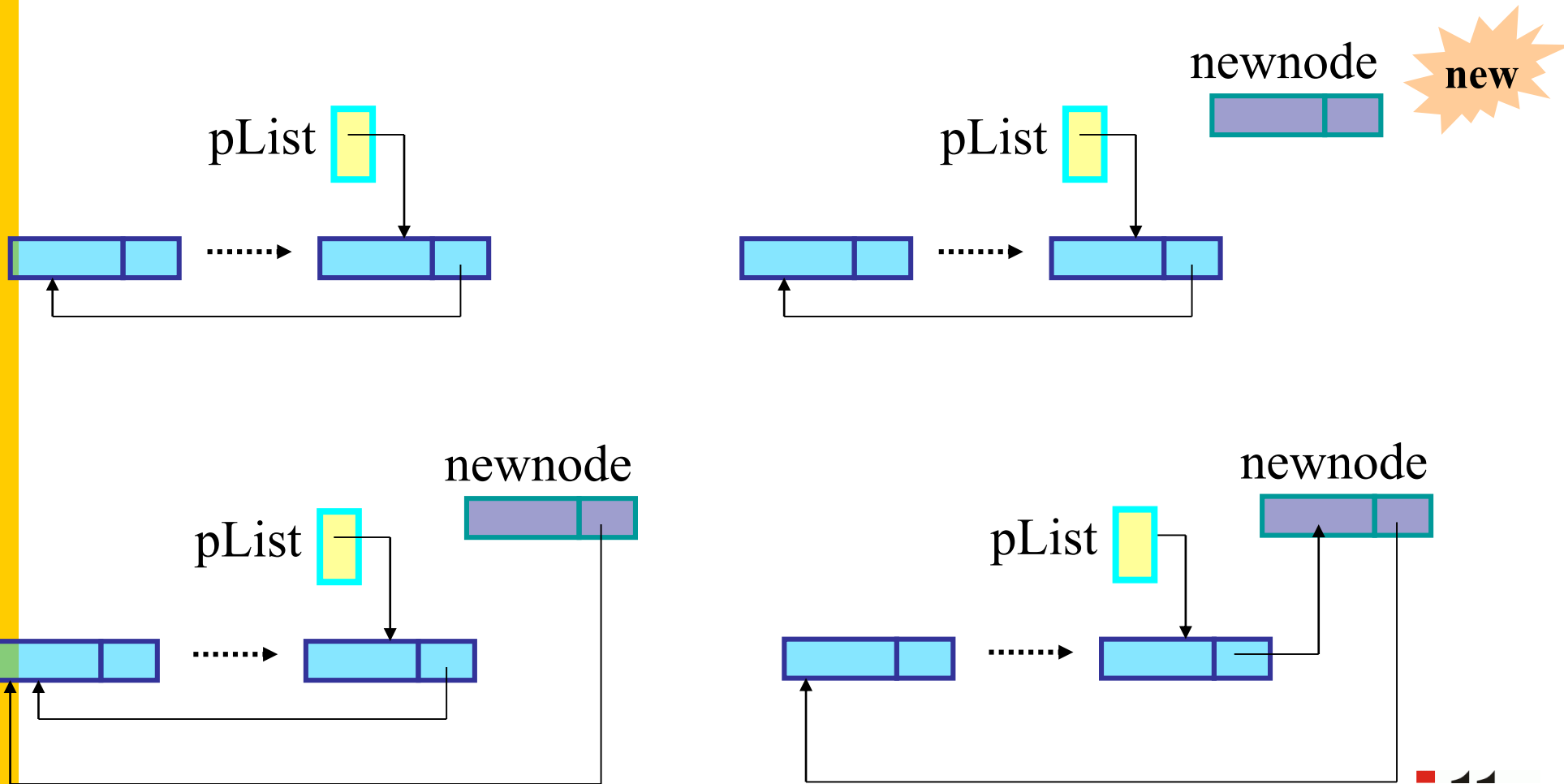
CLL- 1. InsertFirst

➤ Chèn vào đầu – $pList = NULL$



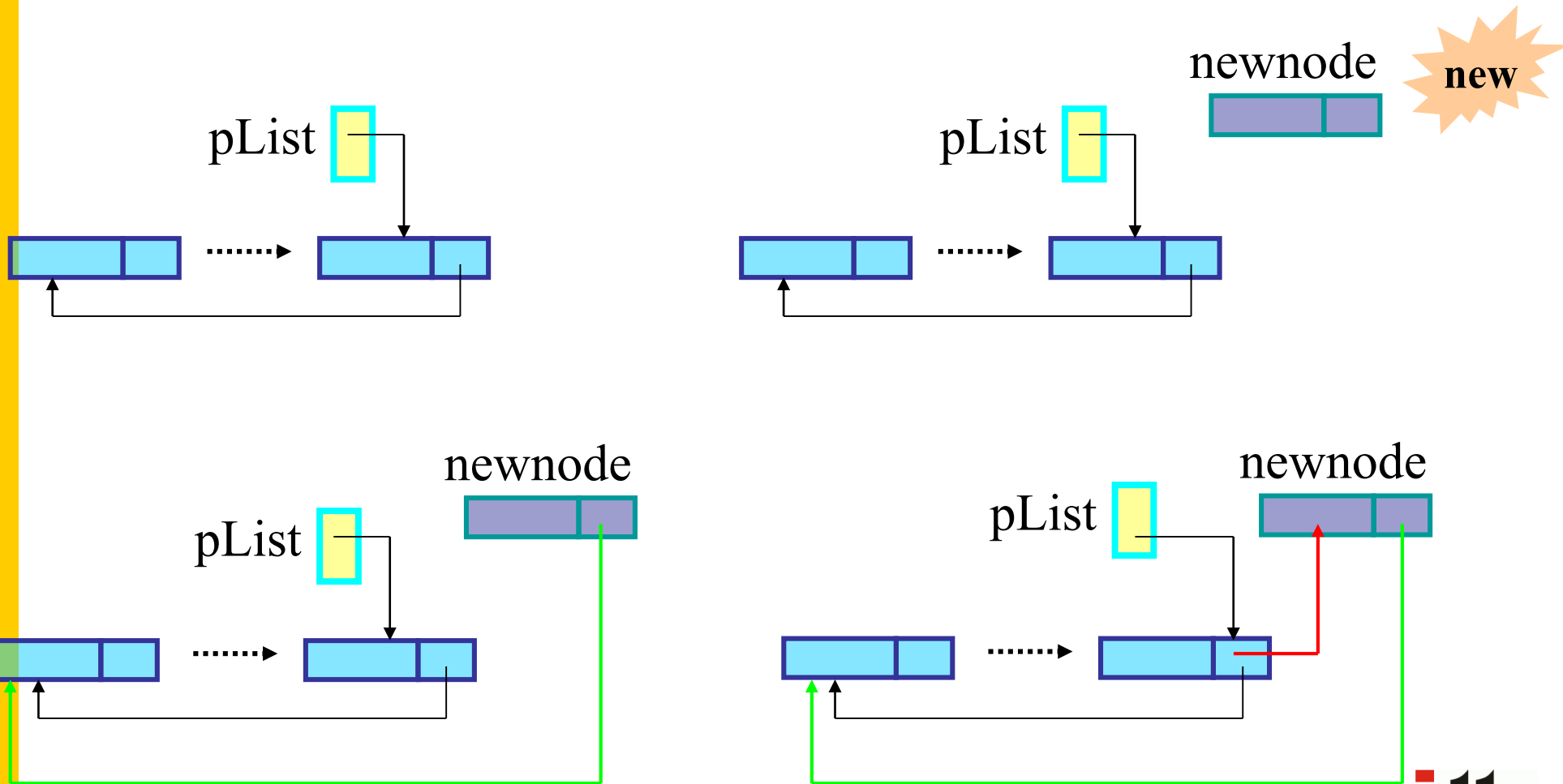
CLL- 1. InsertFirst

➤ Chèn vào đầu – $pList \neq NULL$



CLL- 1. InsertFirst

- Chèn vào đầu – $pList \neq NULL$



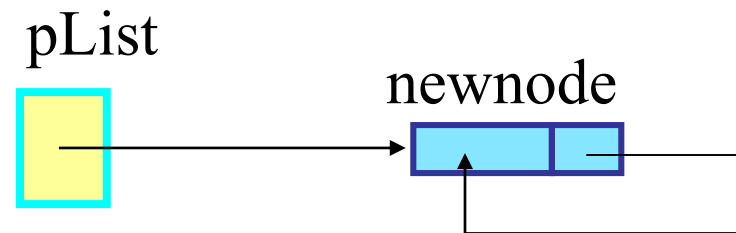
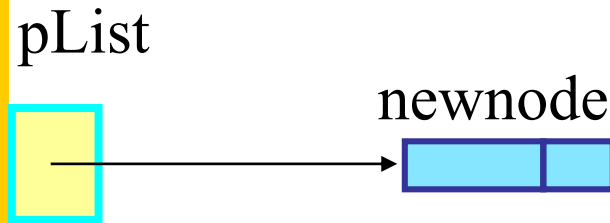
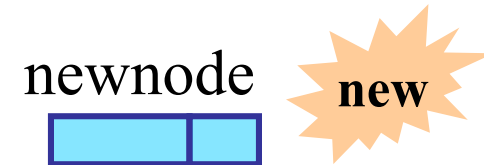
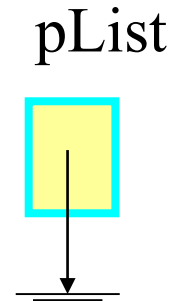
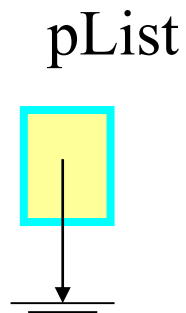
CLL- 1. InsertFirst

```
void addhead (Clist &pList, int x)
{
    NODE * newNode ;
    newNode = GetNode(x);
    if (pList == NULL)
    {
        pList = newNode;
        pList->pNext = pList
    }
    else
    {
        newNode->pNext = pList->pNext;
        pList->pNext = newNode;
    }
}
```

ThS.Nguyễn Thúy Loan

CLL- 2. InsertLast

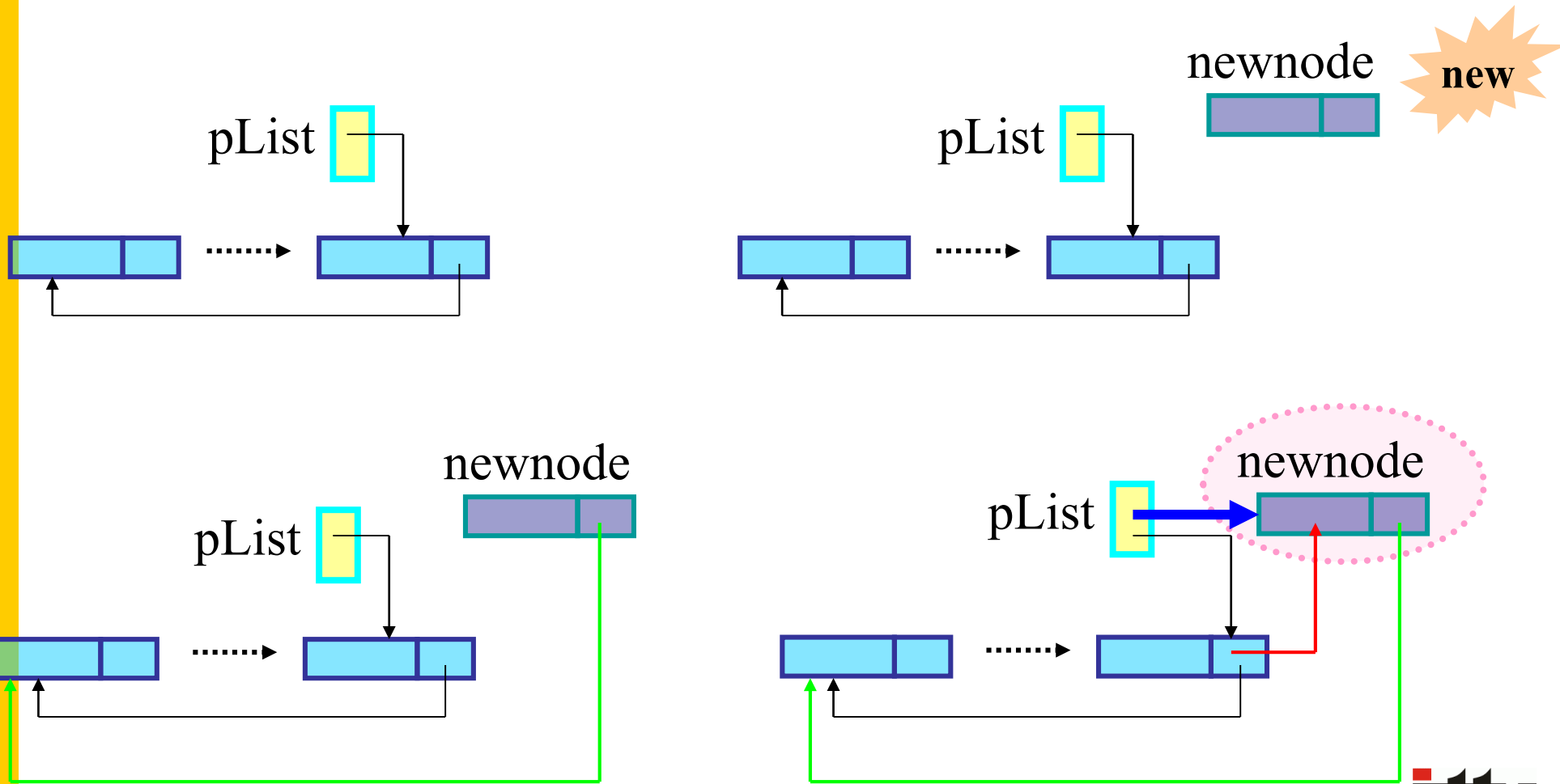
➤ Chèn vào cuối – $pList = \text{NULL}$



ThS.Nguyễn Thúy Loan

CLL- 2. InsertLast

➤ Chèn vào cuối – $pList \neq NULL$



CLL- 2. InsertLast

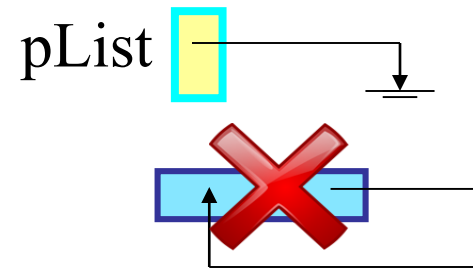
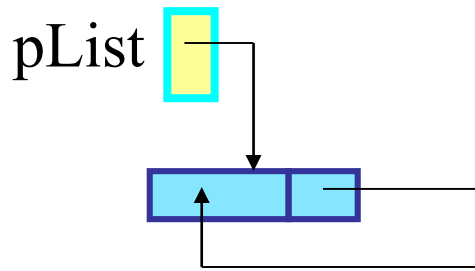
```
void addLast(Clist &pList, int x)
{
    NODE * newNode;
    newNode = GetNode(x);
    if (pList == NULL)
    {
        pList = newNode;
        pList->pNext = pList;
    }
    else {
        newNode->pNext = pList->pNext;
        pList->pNext = newNode;
        pList = newNode;
    }
}
```

ThS.Nguyễn Thúy Loan

CLL- 3. DeleteFirst

➤ Xóa nút đầu:

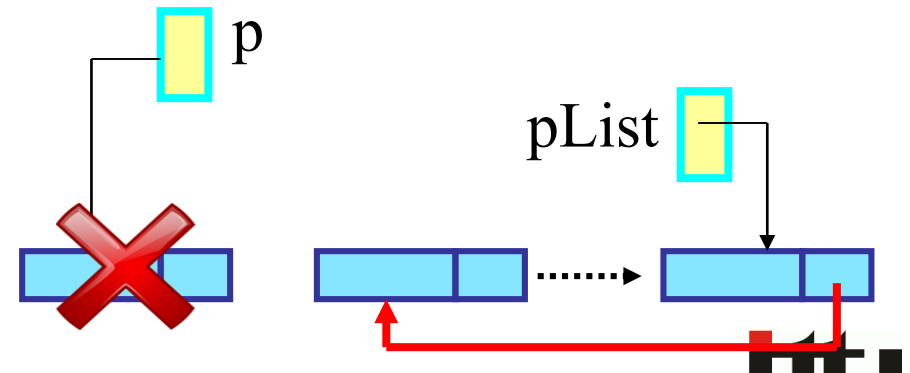
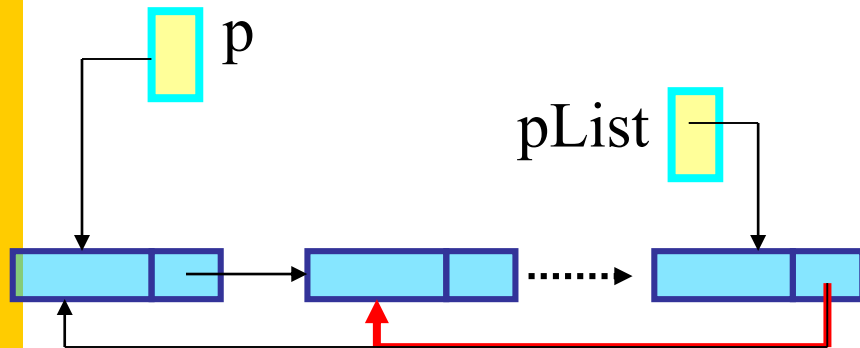
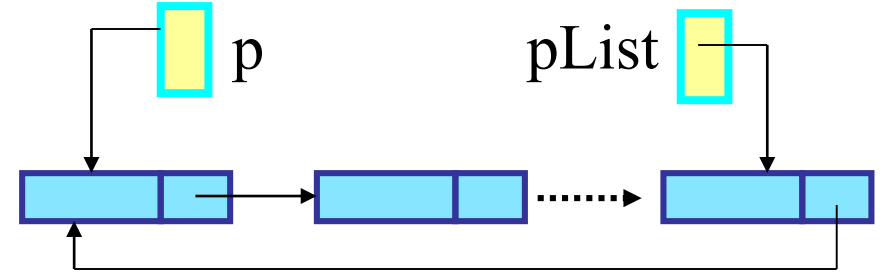
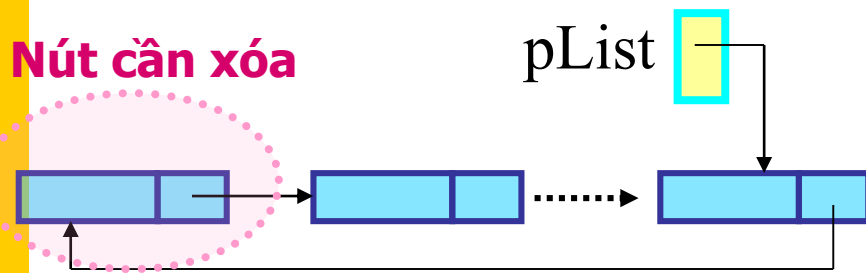
↪ Danh sách chỉ có 1 nút ($pList \rightarrow next == pList$)



CLL- 3. DeleteFirst

➤ Xóa nút đầu

↪ Danh sách có nhiều nút ($pList \rightarrow next \neq pList$)



CLL- 3. DeleteFirst

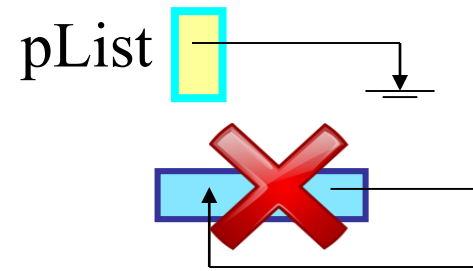
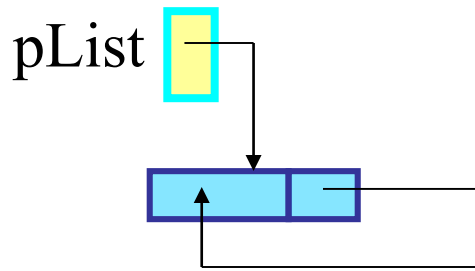
```
void Deletehead (Clist &pList) {  
    NODE * p;  
    if (pList == NULL) return;  
    else  
        if (pList == pList->pNext)           //ds chỉ có 1 phần tử  
        {  
            p= pList;  
            pList = NULL;  
            delete p;  
        }  
        else {                               //ds có 2 phần tử trở lên  
            p = pList->pNext;  
            pList->pNext = p->pNext;  
            delete p;  
        }  
}
```

ThS.Nguyễn Thúy Loan

CLL- 4. DeleteLast

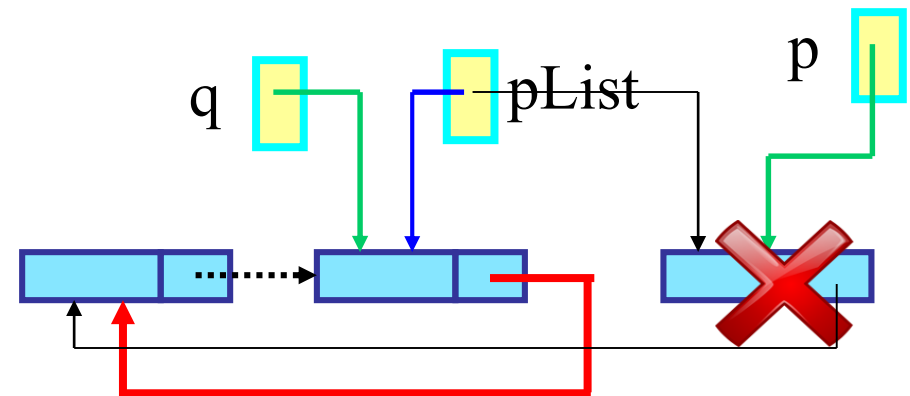
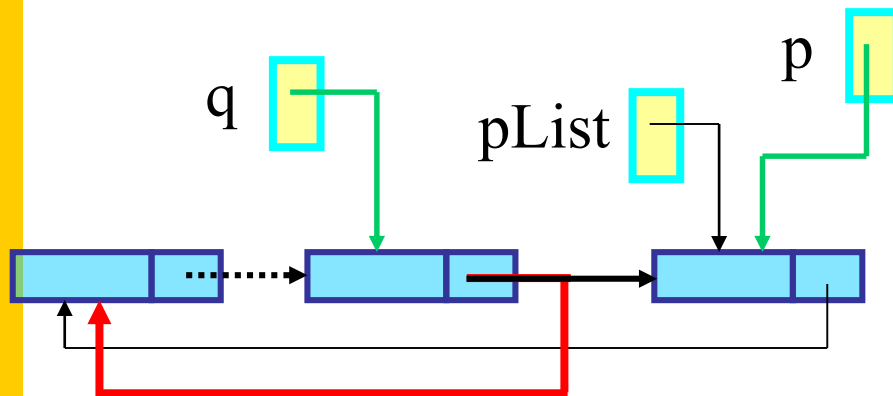
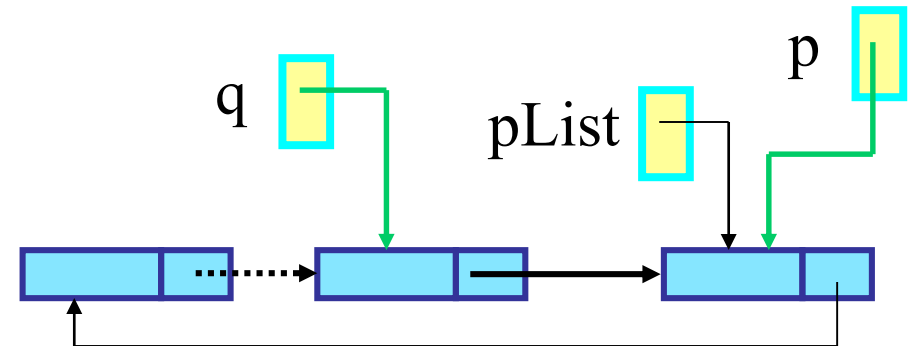
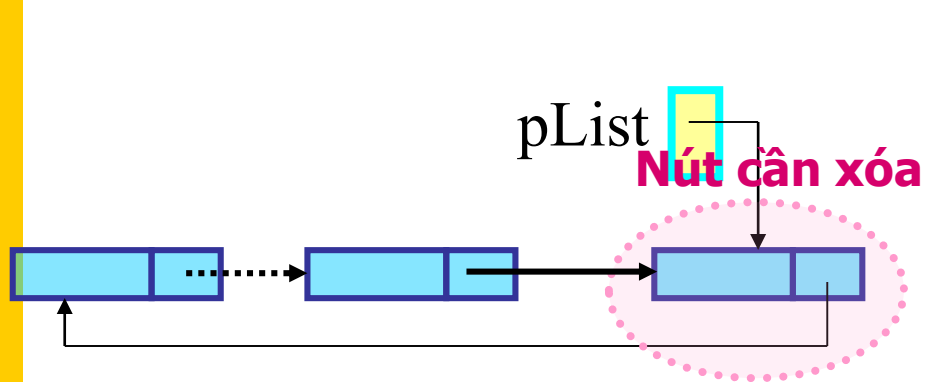
➤ Xóa nút cuối:

↪ Danh sách chỉ có 1 nút ($pList \rightarrow next == pList$)



CLL- 4. DeleteLast

- Xóa nút cuối – $pList \rightarrow pNext \neq pList$



ThS. Nguyễn Thúy Loan

CLL- 4. DeleteLast

```
void DeleteLast( Clist &pList) {
```

```
}
```

CLL- 5. ShowList

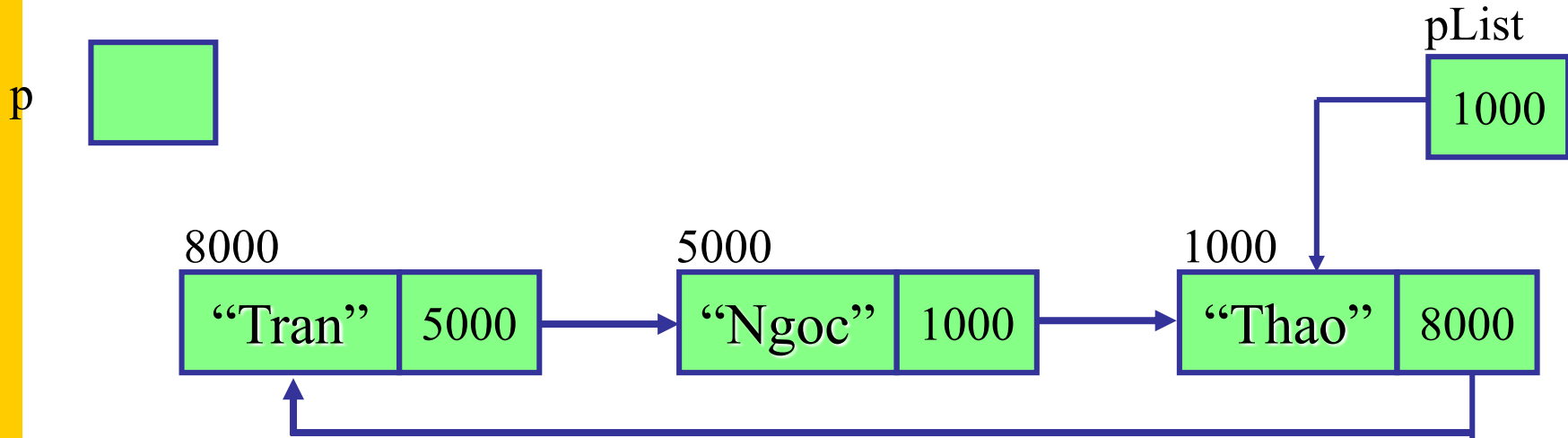
➤ ShowList:

↪ Duyệt từ đầu danh sách

↪ Đến khi nào quay lại phần tử đầu thì dừng

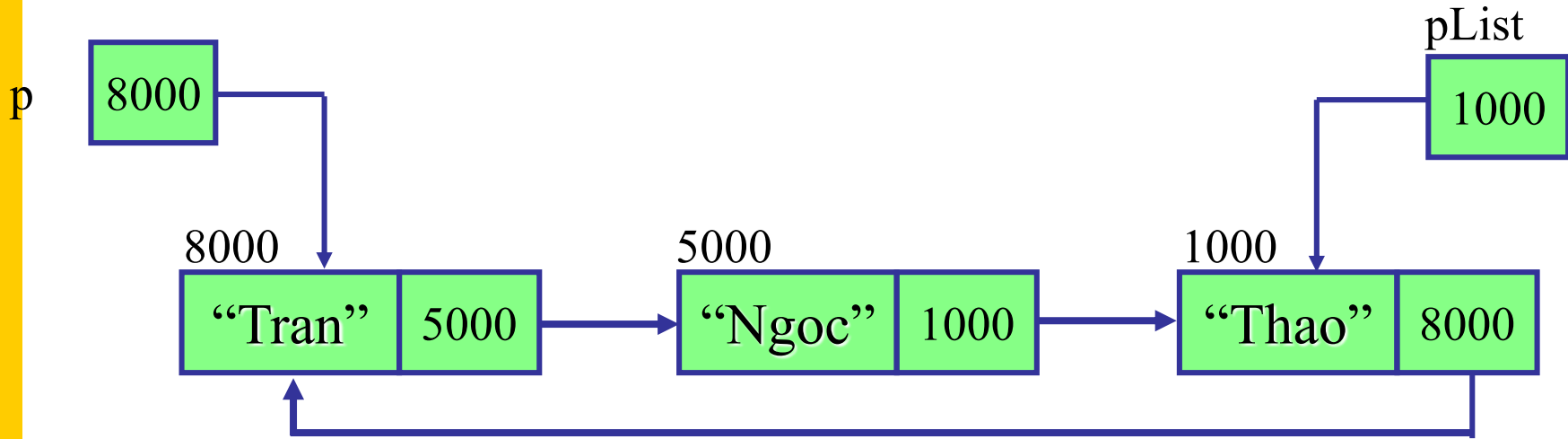
```
void ShowList(Clist pList)
{
    NODE * p;
    if (pList == NULL ) return;
    p = pList->pNext;
    do {
        ShowNode(p);
        p = p->pNext;
    } while (p!=pList->pNext);
}
```

CLL- 5. ShowList



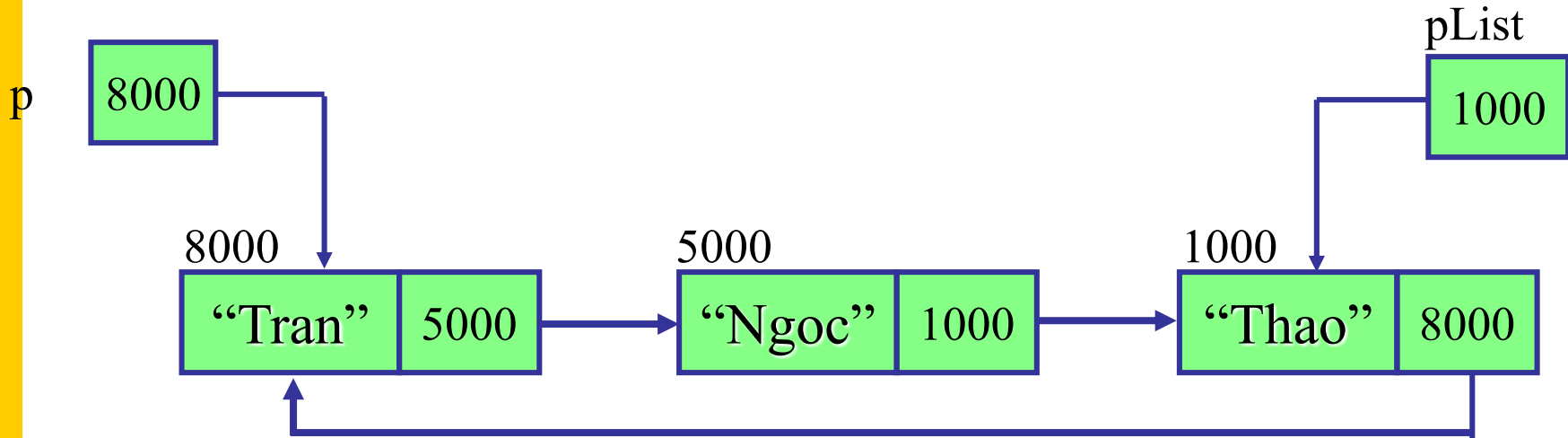
```
p = pList->pNext;  
do {  
    ShowNode(p);  
    p = p->pNext;  
} while (p != pList->pNext);
```

CLL- 5. ShowList



```
p = pList->pNext;  
do {  
    ShowNode(p);  
    p = p->pNext;  
} while (p != pList->pNext);
```

CLL- 5. ShowList



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

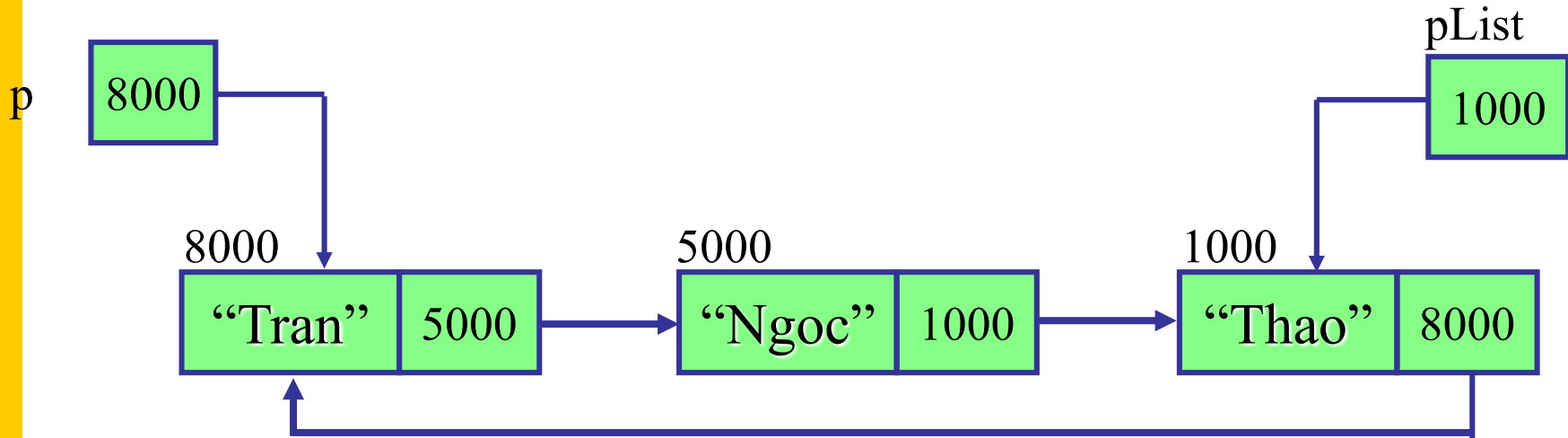
```
do {
```

```
    ShowNode(p);
```

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

```
} while (p != pList->pNext);
```

CLL- 5. ShowList



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

```
do {
```

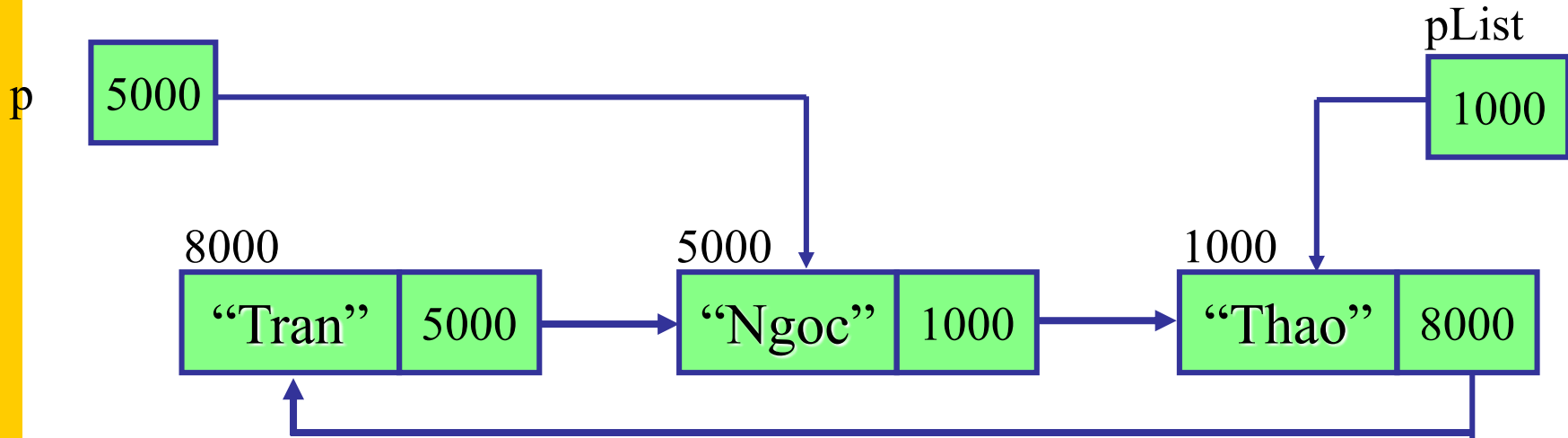
```
    ShowNode(p);
```

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

```
} while (p != pList->pNext);
```

"Tran"

CLL- 5. ShowList



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

```
do {
```

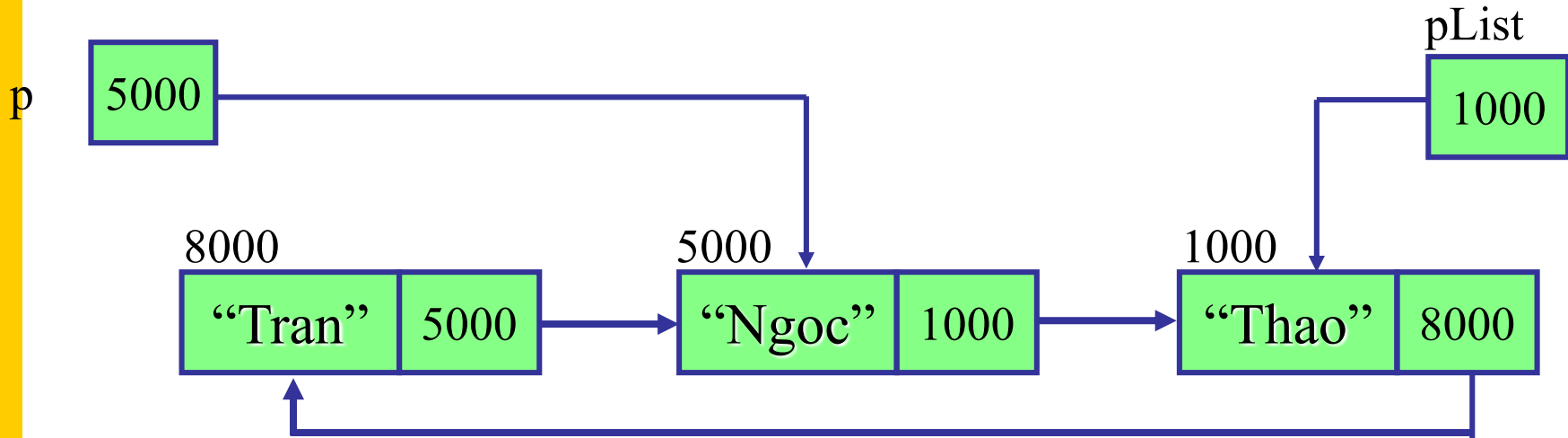
```
    ShowNode(p);
```

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

```
} while (p != pList->pNext);
```

“Tran”

Circular Linked List



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

```
do {
```

```
    ShowNode(p);
```

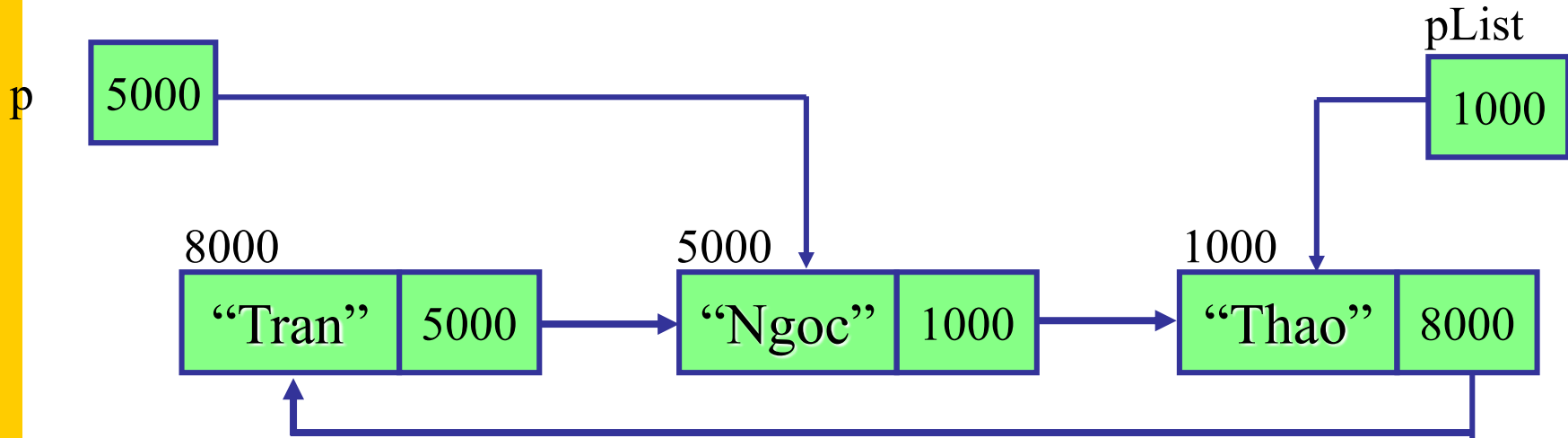
```
    p = p->pNext;
```

```
} while (p != pList->pNext);
```

Tran

Ngoc

CLL- 5. ShowList



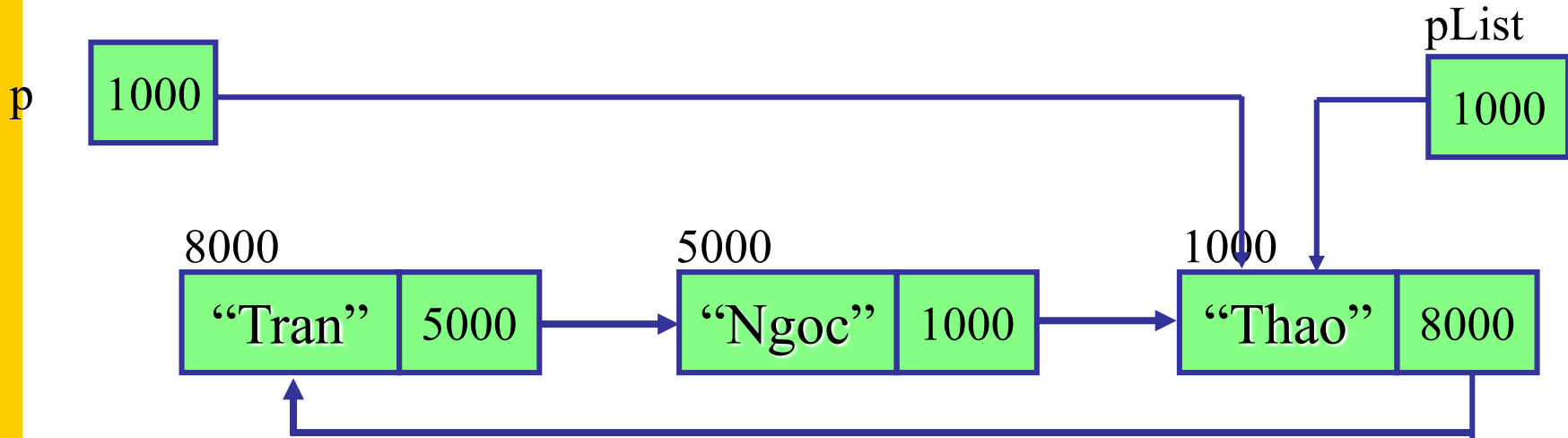
```
p = pList->pNext;  
do {  
    ShowNode(p);  
    p = p->pNext;  
} while (p != pList->pNext);
```

“Tran”

true

ThS.Nguyễn Thúy Loan

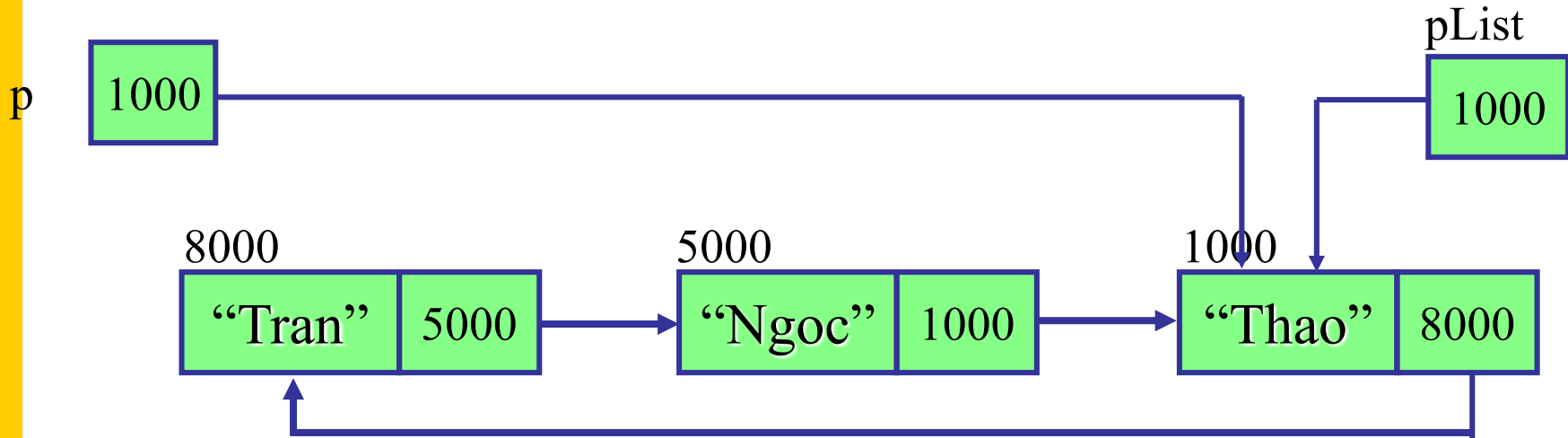
CLL- 5. ShowList



```
p = pList->pNext;  
do {  
    ShowNode(p);  
    p = p->pNext;  
} while (p != pList->pNext);
```

Tran
Ngoc

CLL- 5. ShowList



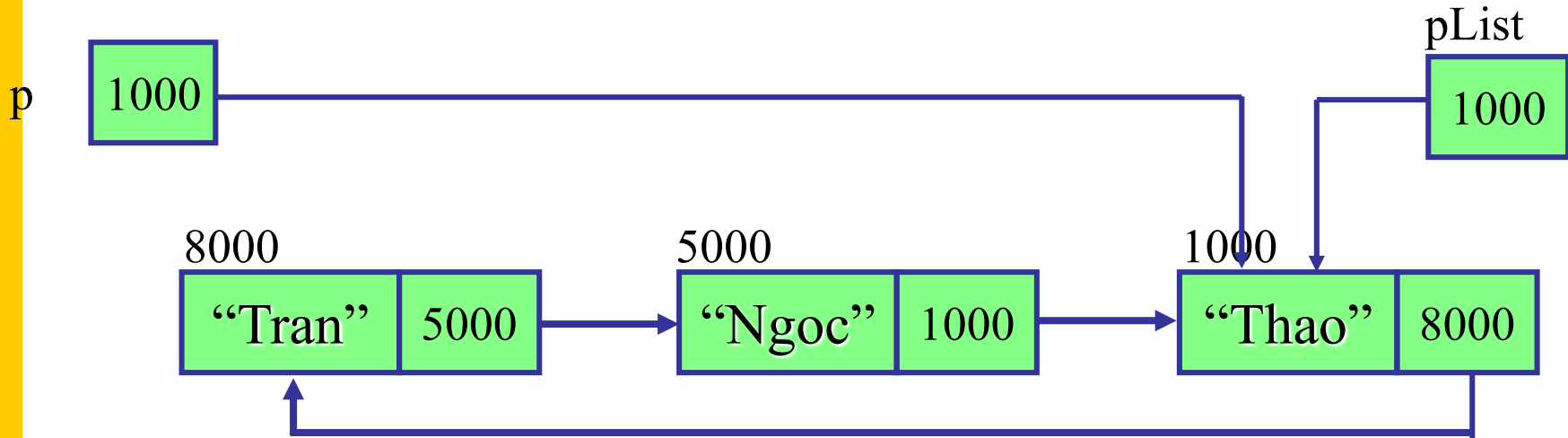
```
p = pList->pNext;  
do {  
    ShowNode(p);  
    p = p->pNext;  
} while (p != pList->pNext);
```

Tran
Ngoc

true

ThS.Nguyễn Thúy Loan

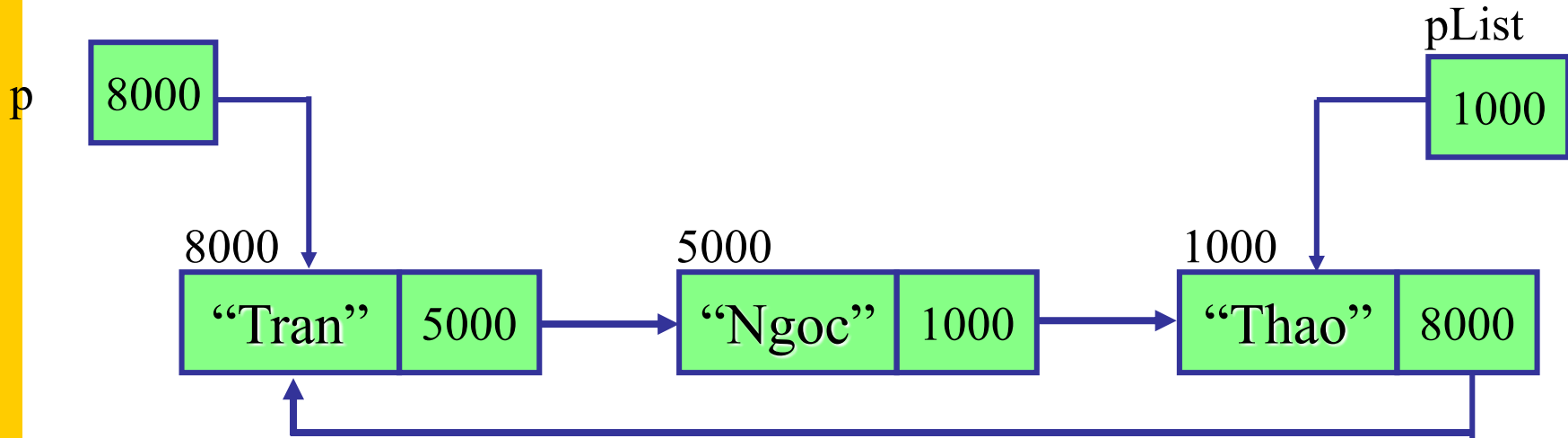
CLL- 5. ShowList



```
p = pList->pNext;  
do {  
    ShowNode(p);  
    p = p->pNext;  
} while (p != pList->pNext);
```

Tran
Ngoc
Thao

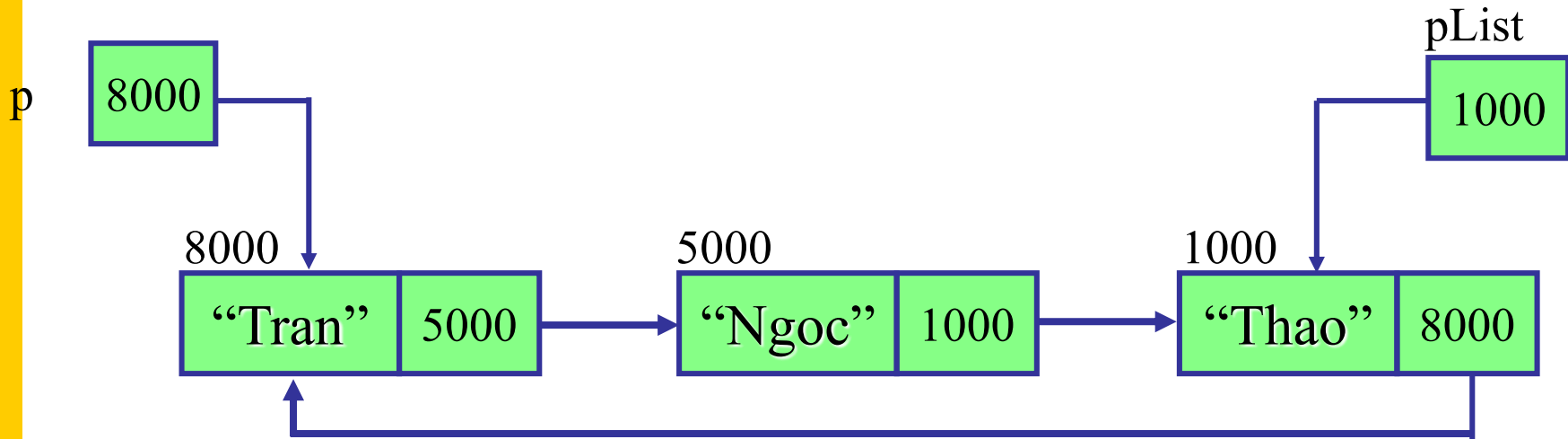
CLL- 5. ShowList



```
p = pList->pNext;  
do {  
    ShowNode(p);  
    p = p->pNext;  
} while (p != pList->pNext);
```

Tran
Ngoc
Thao

CLL- 5. ShowList



```
p = pList->pNext;  
do {  
    ShowNode(p);  
    p = p->pNext;  
} while (p != pList->pNext);
```

Tran
Ngoc
Thao

false

ThS.Nguyễn Thúy Loan

CLL- 6. Search

➤ Search:

- ↪ Xuất phát từ đầu danh sách
- ↪ Nếu tìm thấy trả về địa chỉ nút đó
- ↪ Ngược lại qua phần tử tiếp theo
- ↪ Điều kiện dừng khi quay lại phần tử đầu tiên
- ↪ Không tìm thấy trả về NULL

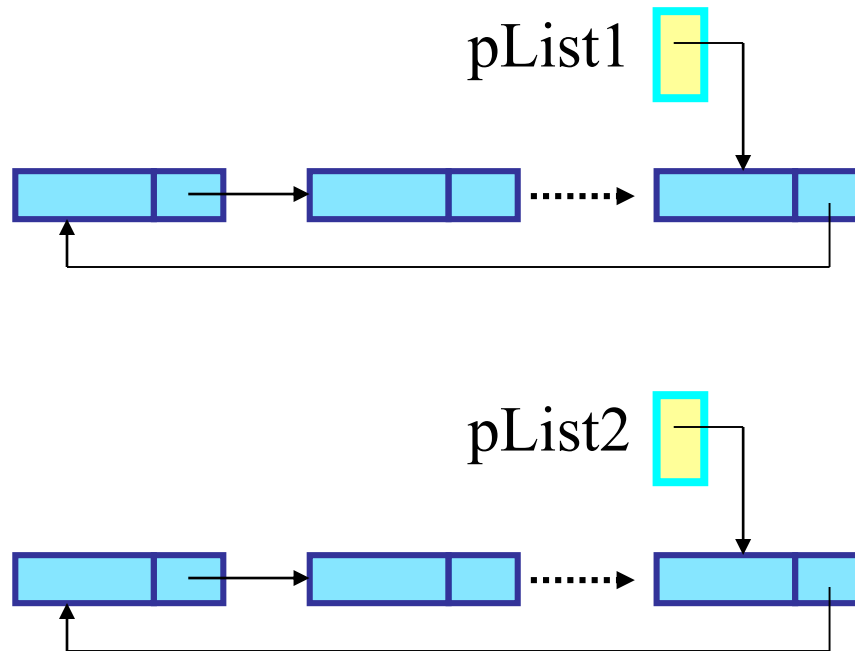
CLL- 6. Search

```
Node* Search( Clist pList, int x)
{
    NODE * p;
    if (pList == NULL) return NULL;

    p = pList->pNext;    //Lấy nút đầu DS
    while (p->info != x && p != pList->pNext)
        p = p->pNext;
    if ( p->info == x)    return p;
    return NULL;
}
```

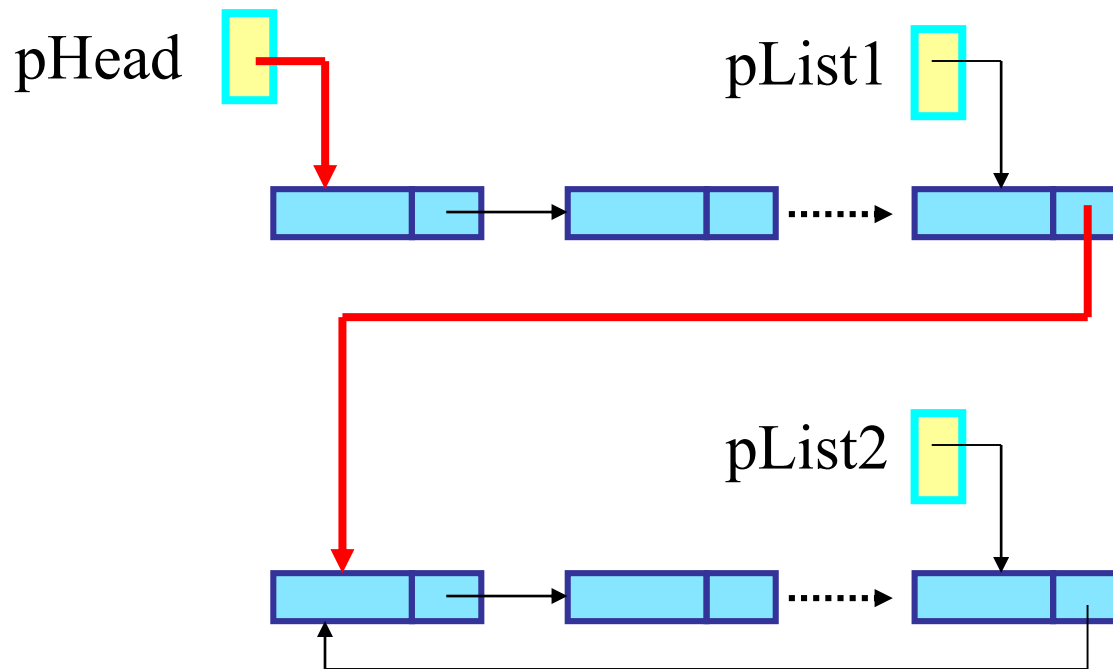
CLL- 7. AddList

- Nối danh sách vòng pList2 vào pList 1



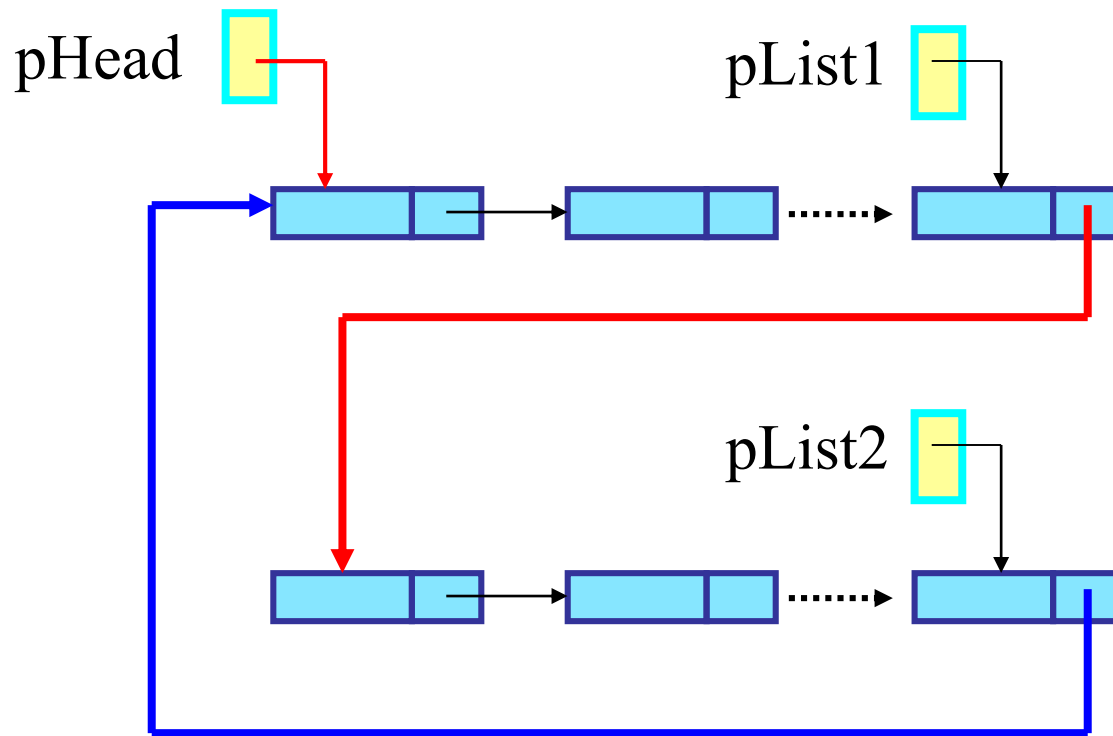
CLL- 7. AddList

- Nối danh sách vòng pList2 vào pList 1



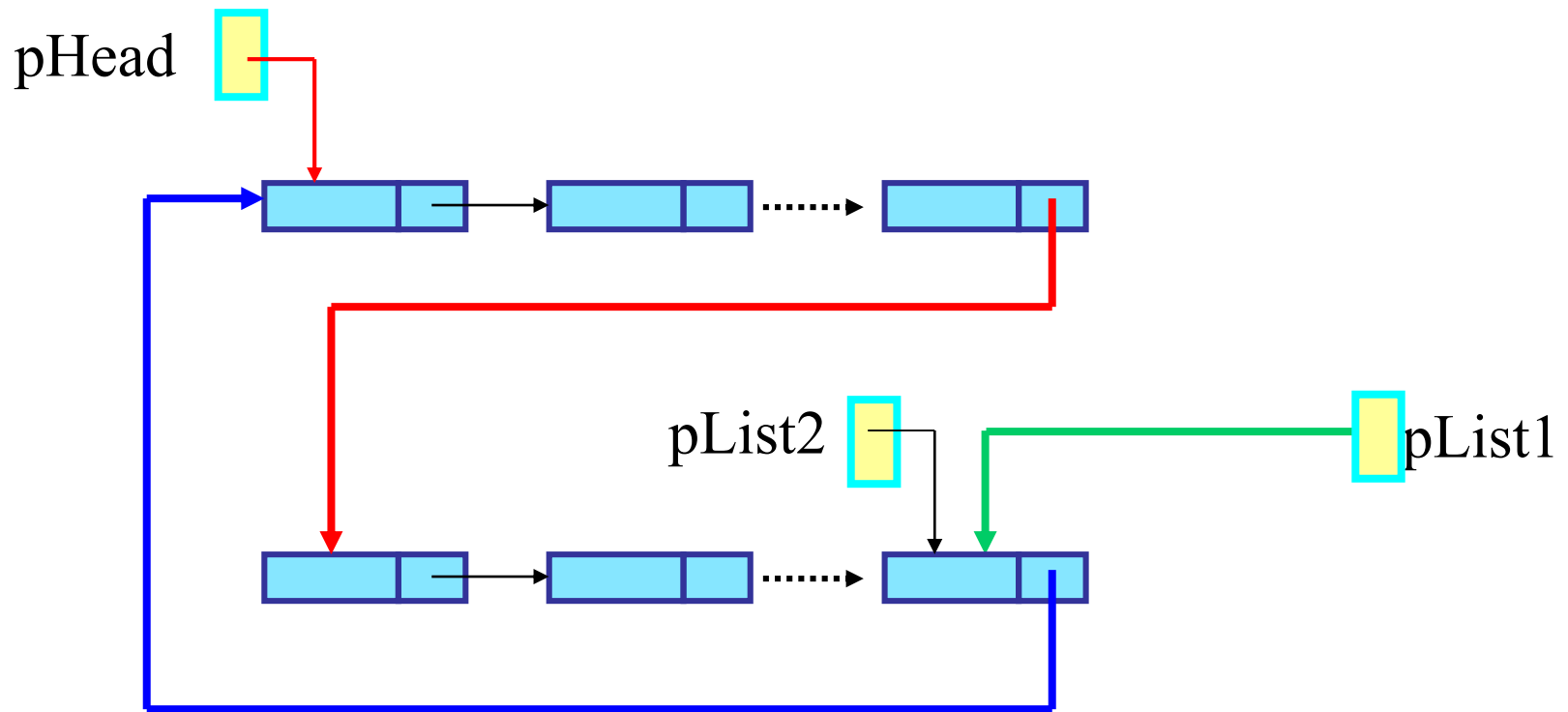
CLL- 7. AddList

- Nối danh sách vòng pList2 vào pList 1



CLL- 7. AddList

- Nối danh sách vòng pList2 vào pList 1



CLL- 7. AddList

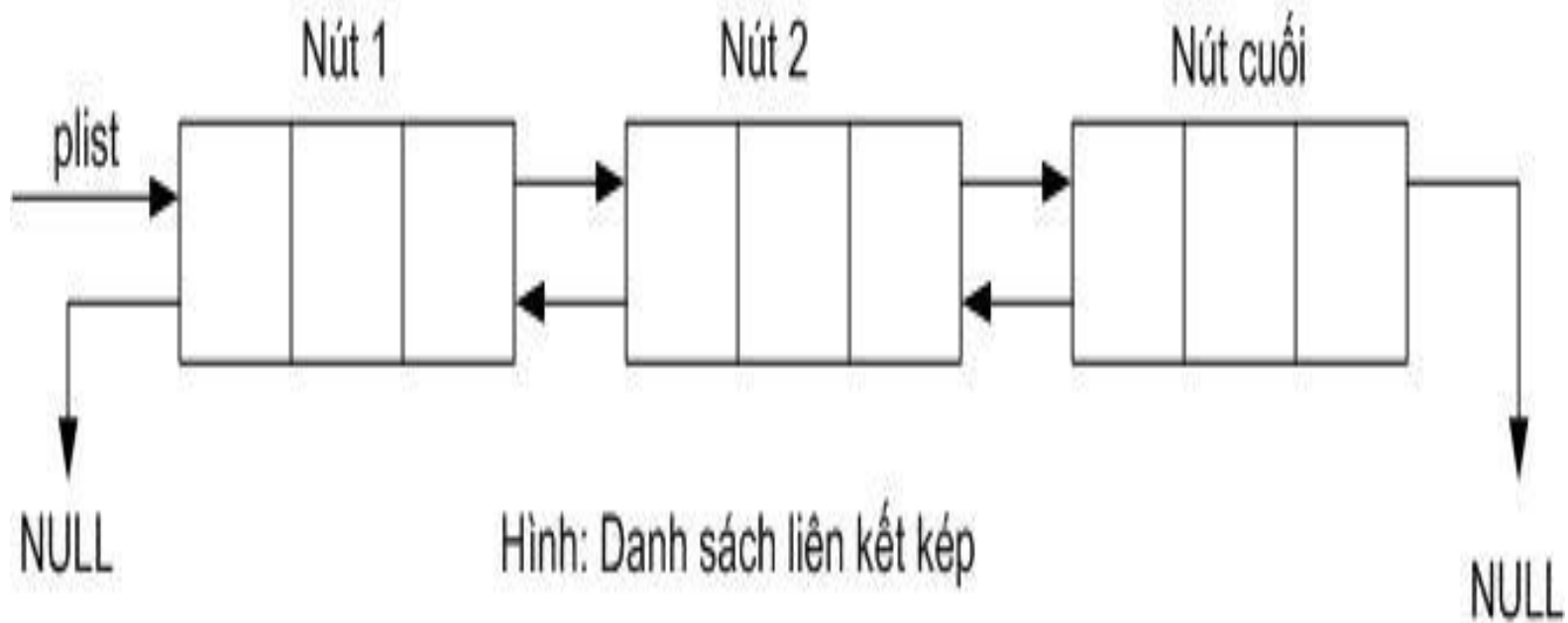
```
void AddList(NodePtr &pList1, NodePtr &pList2)
```

```
{
```

```
}
```

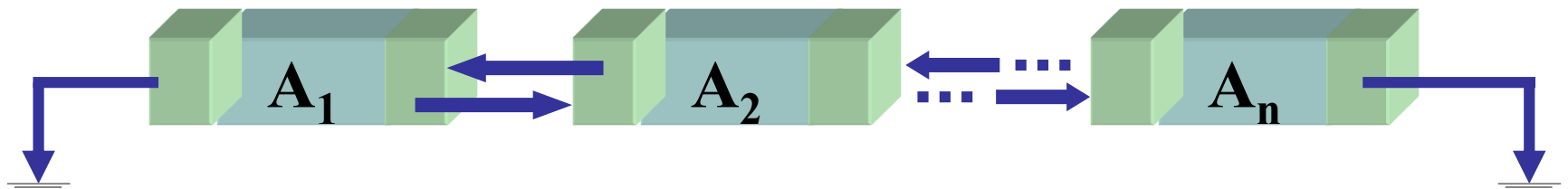
Doubly Linked List

DANH SÁCH LIÊN KẾT KÉP



Doubly Linked List

- Cho phép di chuyển 2 chiều đến nút trước và sau.
 - ↪ Liên kết nút trước là: prev
 - ↪ Liên kết nút sau là: pNext
- Nút đầu có prev là NULL
- Nút cuối có pNext là NULL



Doubly Linked List

➤ Khai báo

```
typedef struct node  
{
```

```
    DataType    info;
```

```
    struct node * prev;
```

```
    struct node * next;
```

```
}NODE ;
```

```
Typedef  NODE * DHead;
```



trở đến nút trước



trở đến nút sau



DHead quản lý ds kép

Doubly Linked List

➤ Các thao tác cơ bản

↪ CreateNode, Init, IsEmpty...

↪ InsertFirst: chèn vào đầu

↪ InsertPrev: chèn trước nút p

↪ InsertNext: chèn sau nút p

↪ DeleteFirst: xóa nút đầu

↪ DeleteNode: xóa nút p

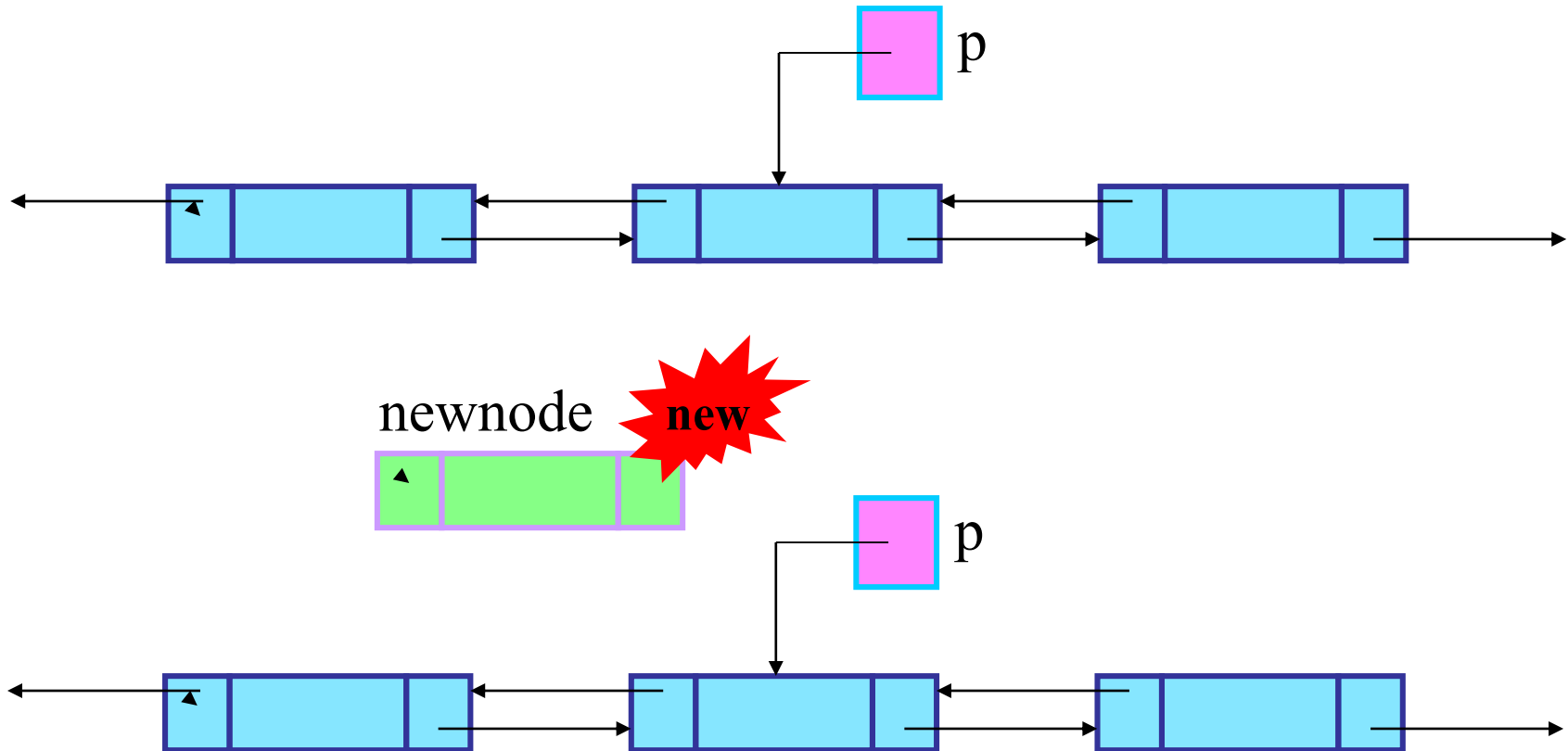
↪ ShowList: duyệt ds

↪ ShowReverse: duyệt từ cuối danh sách

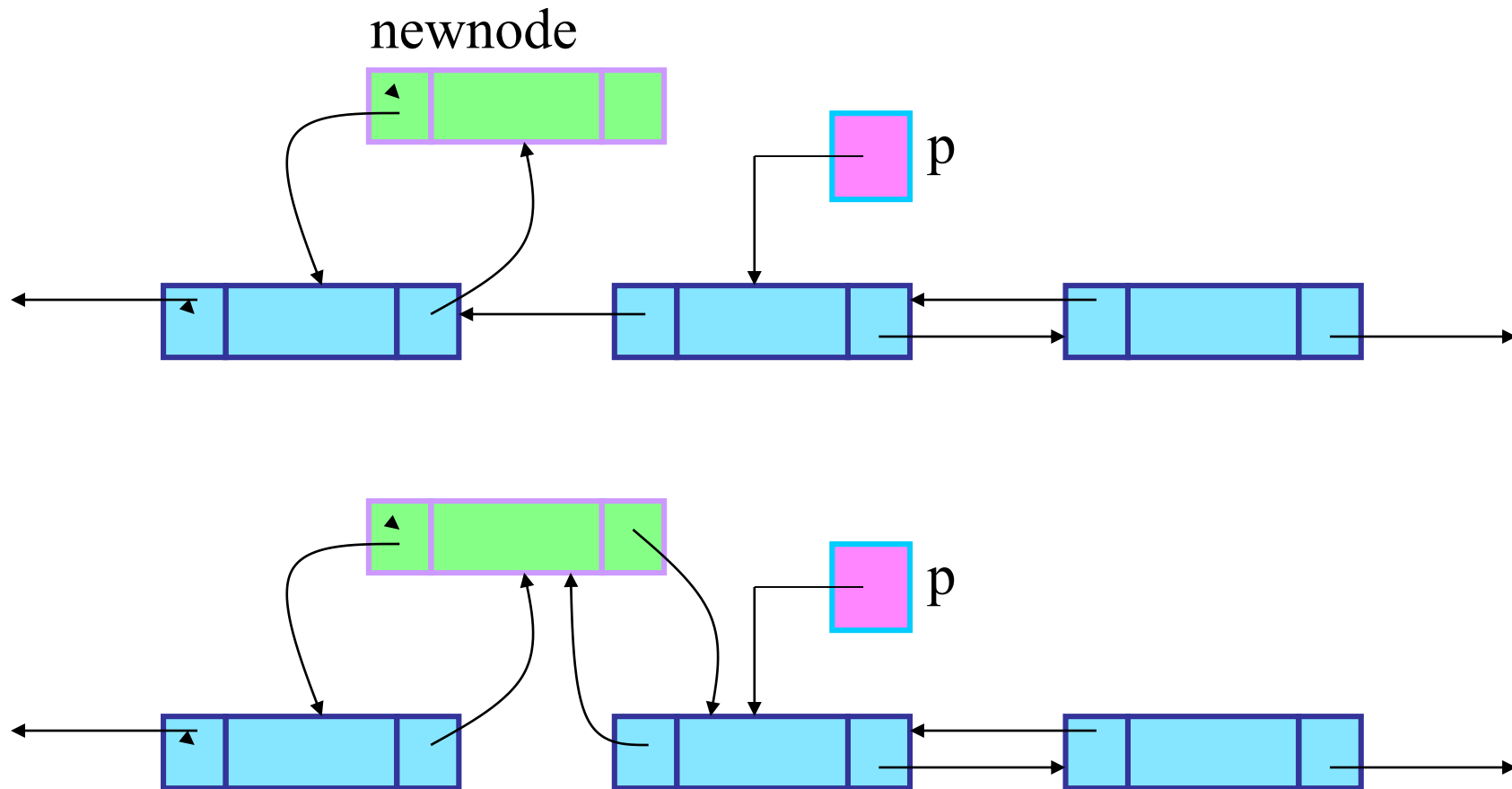
↪ ClearList: xóa toàn bộ ds

Doubly Linked List

➤ InsertPrev: chèn vào trước nút p



Doubly Linked List



Doubly Linked List

```
void InsertPrev(LIST &l, Node* p, Node *q)
{
    if (p == l.pHead)    addhead(l,q);
    else {
        Node *k = p->prev;           //lấy nút trước nút p
        q->prev = k; //gắn nút mới vào nút left
        k->next = q;

        q->next = p; //gắn nút mới vào nút p
        p->prev = q;
    }
}
```

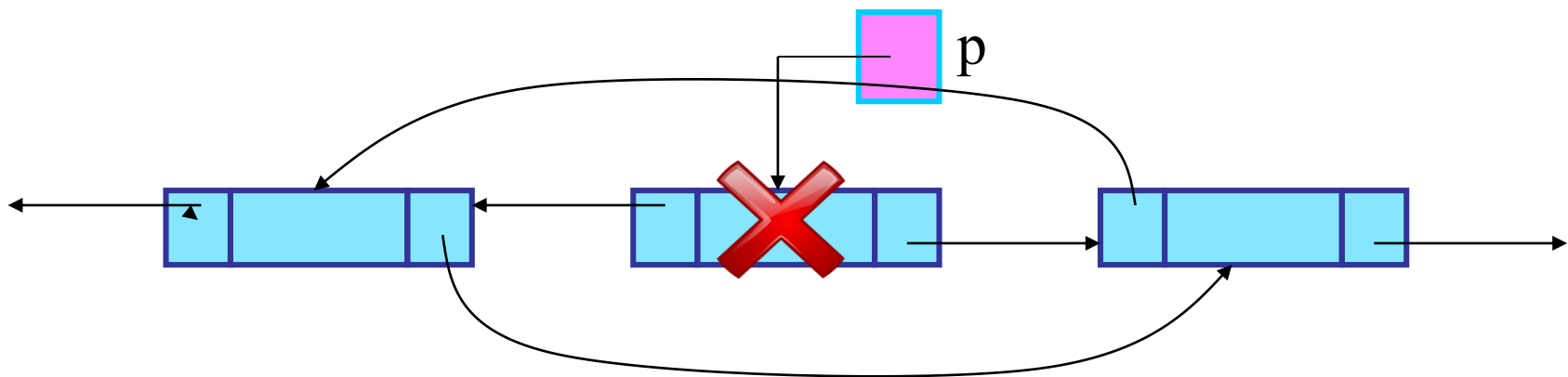
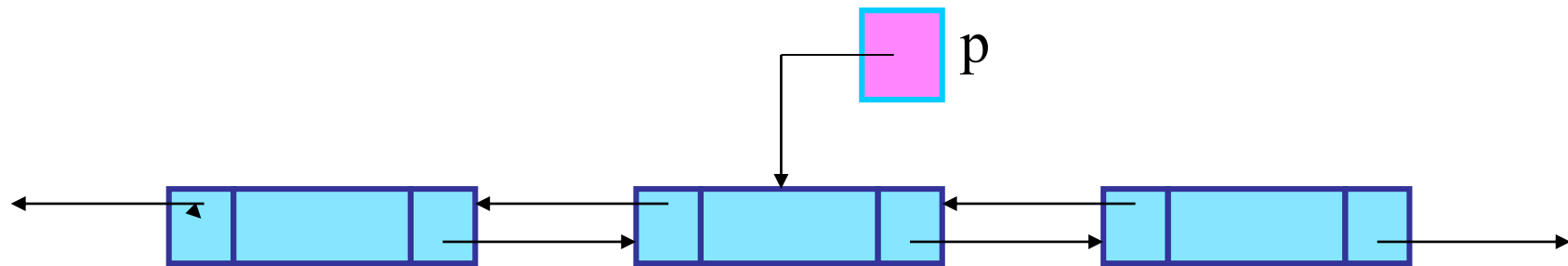
Doubly Linked List

```
void InsertPrev(Node*&pHead, Node* &p, int x){
    Node* newnode, * q;
    if (p == NULL)        return;
    if (p == pHead)       InsertFirst(pHead,x);
    else {
        newnode = CreateNode(x); //tạo nút mới chứa dl x
        q = p->prev;          //lấy nút trước nút p
        newnode->prev = q; //gắn nút mới vào nút left
        q->next = newnode;

        newnode->next = p; //gắn nút mới vào nút p
        p->prev = newnode;
    }
}
```

Doubly Linked List

➤ DeleteNode: xoá nút p



ThS. Nguyễn Thúy Loan

Doubly Linked List

```
void DeleteNode(Node* &pHead, Node* &p) {  
    Node* left, *right;  
    if (p == NULL) return;  
    if (p==pHead) DeleteFirst(pHead);  
    else {  
        left = p ->prev;  
        right = p->next;  
        left->next = right;  
        if (right != NULL)  
            right->prev = left;  
        delete p;  
    }  
}
```

Doubly Linked List

➤ Các thao tác còn lại SV tự làm!

Bài tập nâng cao

- Xây dựng cấu trúc danh sách liên kết đôi vòng
 - ↳ Mỗi nút trên danh sách có hai trường liên kết
 - Prev: trỏ đến nút trước
 - Next: trỏ đến nút sau
 - ↳ Nút cuối cùng trong danh sách có trường next là nút đầu tiên
 - ↳ Nút đầu tiên có trường prev là nút cuối cùng.
 - ↳ Các thao tác trên danh sách :