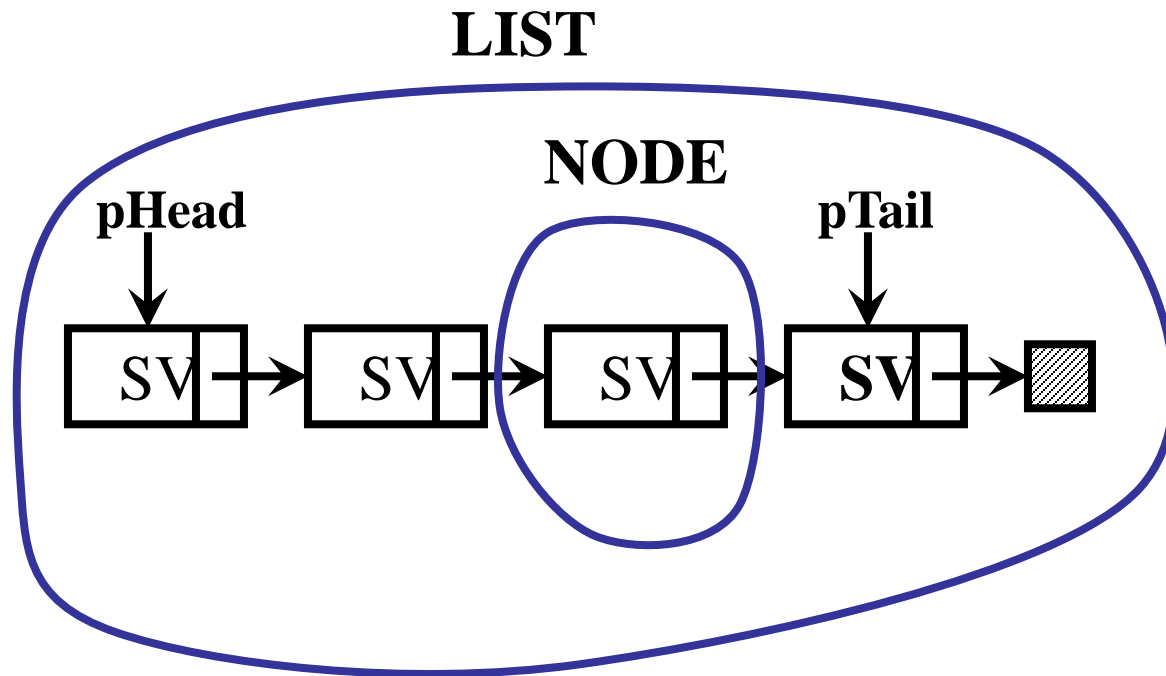


XÂU ĐƠN HAI TRỞ QUẢN LÝ SV

Hình ảnh cấu trúc đơn hai trỏ



```
struct sinhvien
{
    char ho[20];
    char ten[10];
    char mssv [11];
    char lop [9];
    char gioitinh;
    int ntns;
    float toan, ly, tin , DTB ;
};
typedef struct sinhvien sv;
```

Hãy khai báo CTDL cho dslk đơn các quyền sách

1. **struct node**

2. { SV info;

3. struct node*pNext;

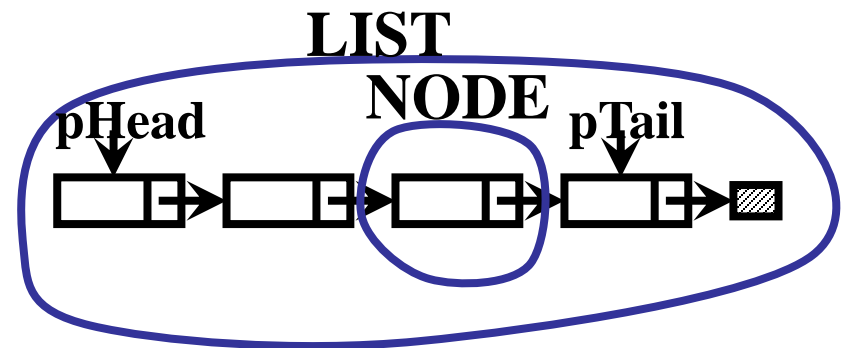
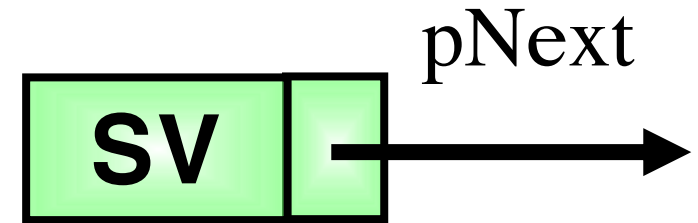
4. }; **typedef struct node NODE;**

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 (SV x)
```

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

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

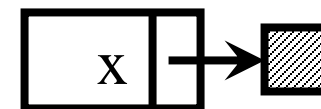
```
        return NULL;
```

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

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

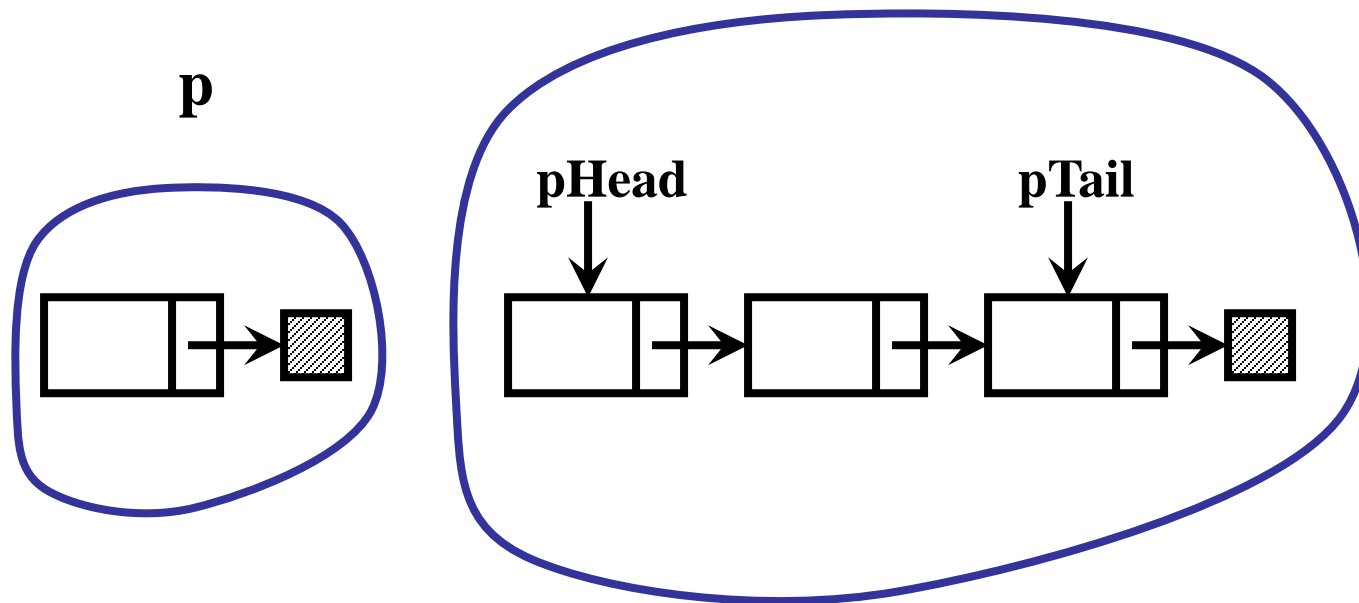
```
3.     return p;
```

```
4. }
```

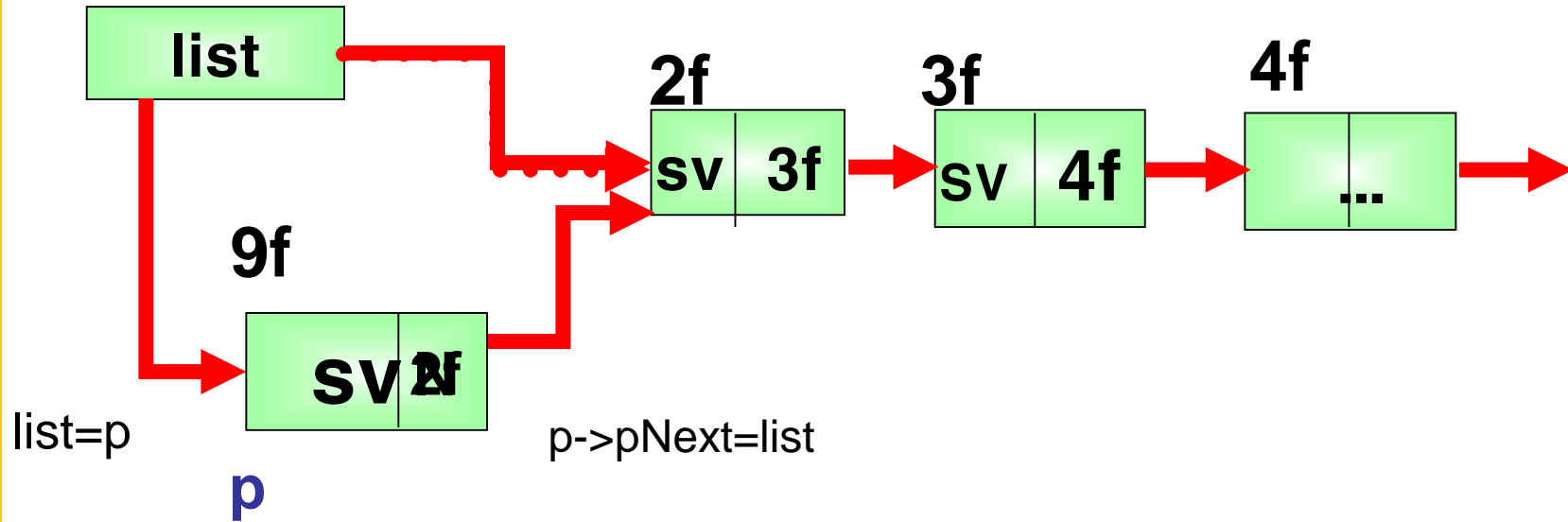


6. THÊM MỘT NODE VÀO ĐẦU DS LIÊN KẾT ĐƠN

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



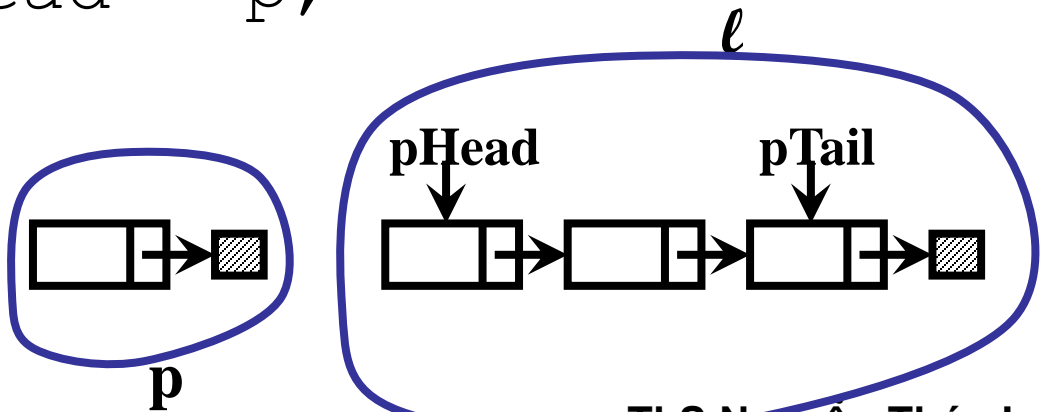
Minh họa thuật toán thêm vào đầu



6. THÊM MỘT NODE VÀO ĐẦU DS LIÊN KẾT ĐƠN

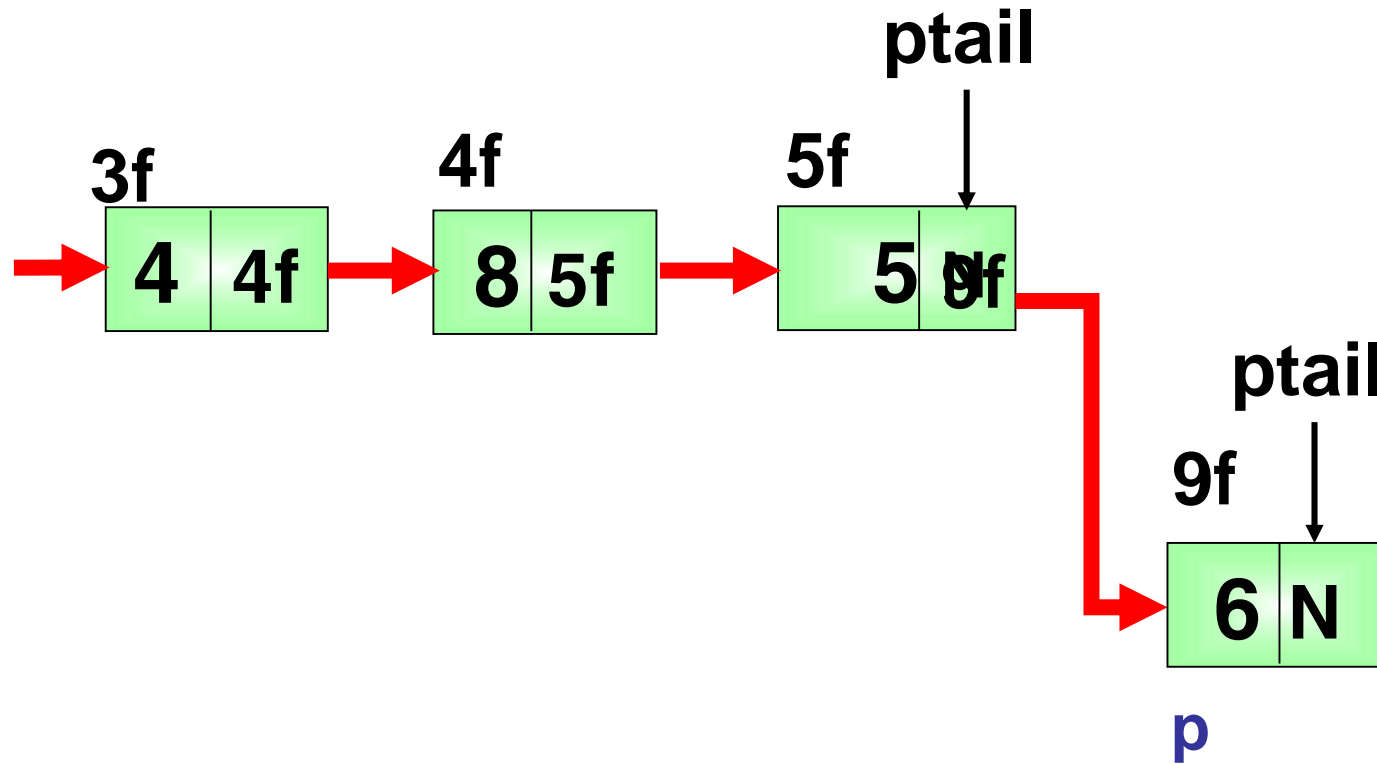
➤ Định nghĩa hàm:

```
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

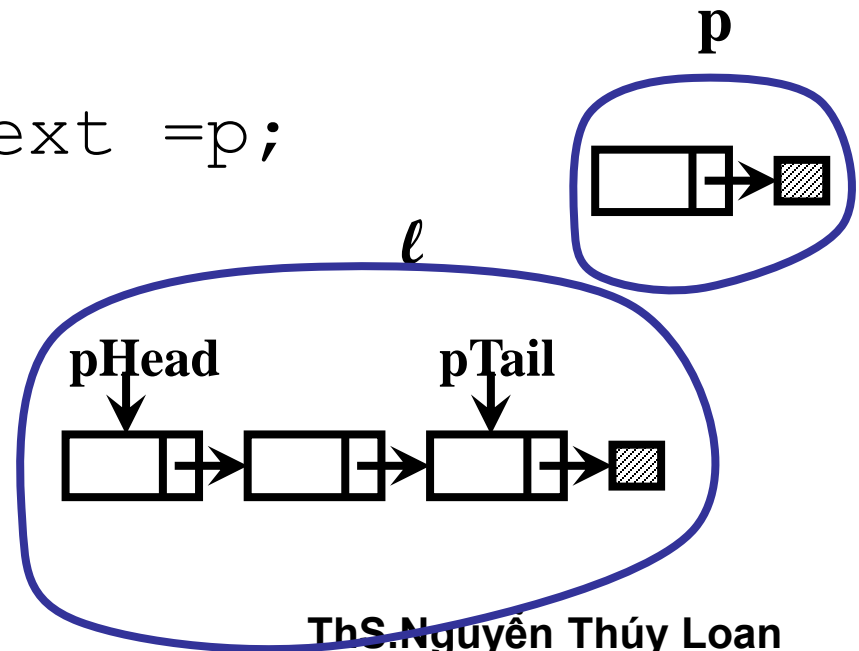
Minh họa thuật toán thêm vào cuối



6. THÊM MỘT NODE VÀO CUỐI DS LIÊN KẾT ĐƠN

➤ Định nghĩa hàm:

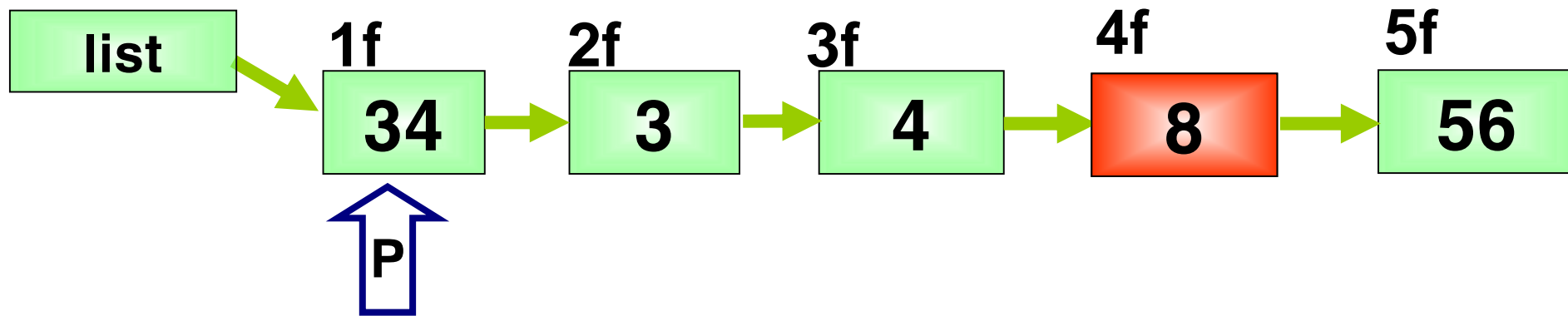
```
1. void Addtail (LIST&l, NODE*p)
2. {
3.     if (l.pHead==NULL)
4.         l.pHead = l.pTail = p;
5.     else
6.     {
7.         l.ptail->pNext = p;
8.         l.ptail = p;
9.     }
10. }
```



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

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

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;

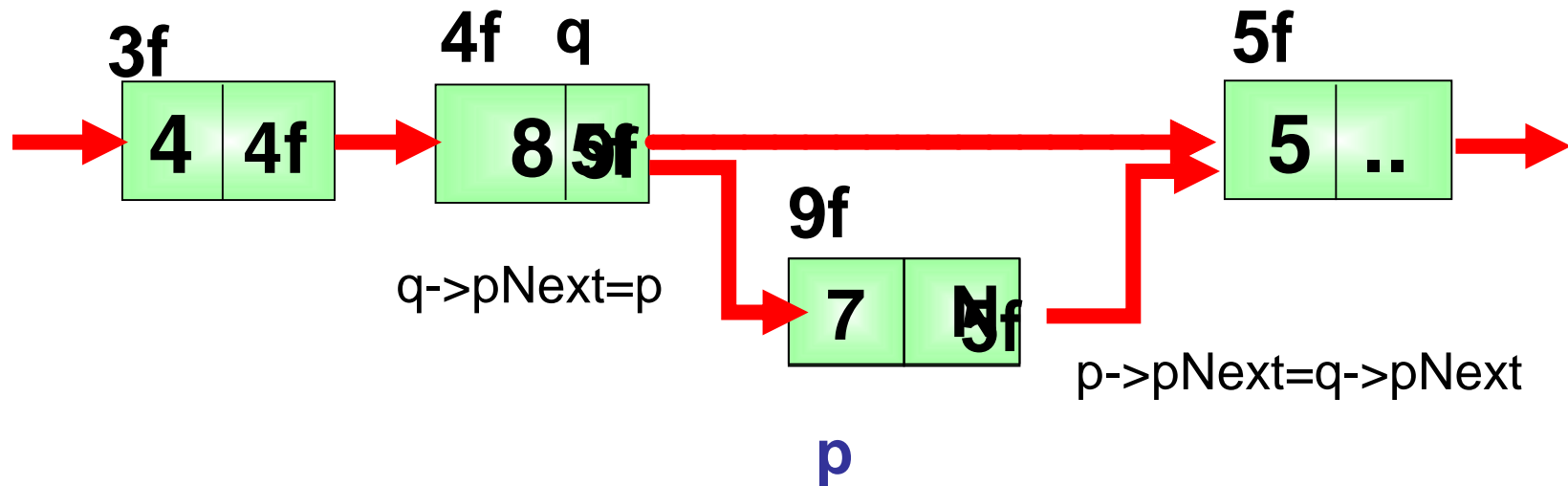
 while(p!=NULL && p->info!=x)

 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

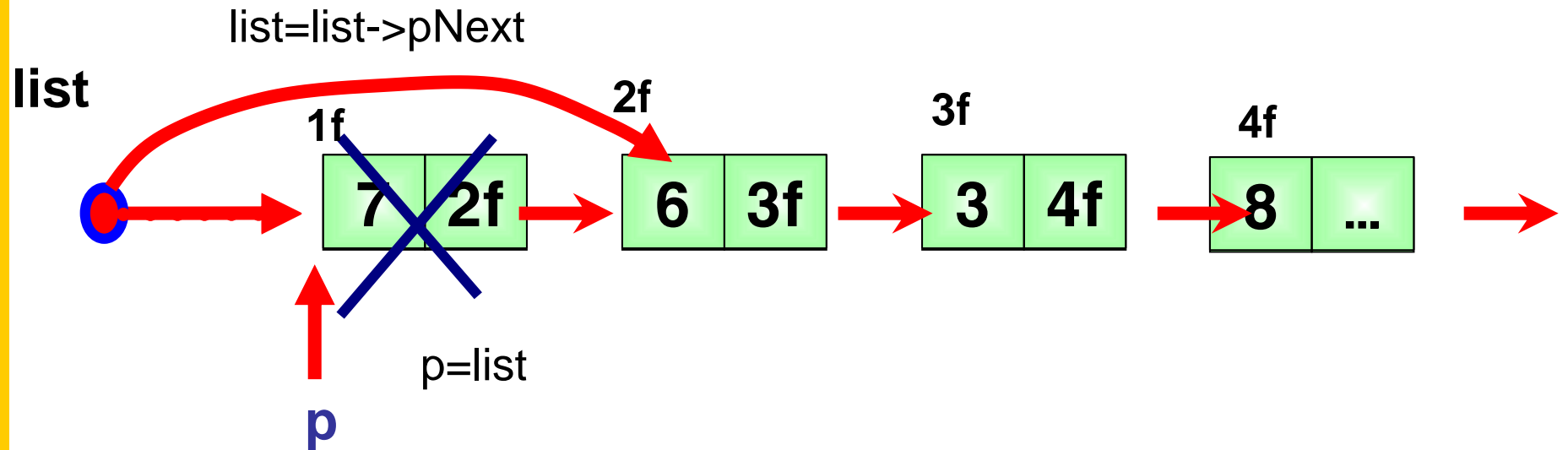
Cài đặt thuật toán

```
➤ 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:

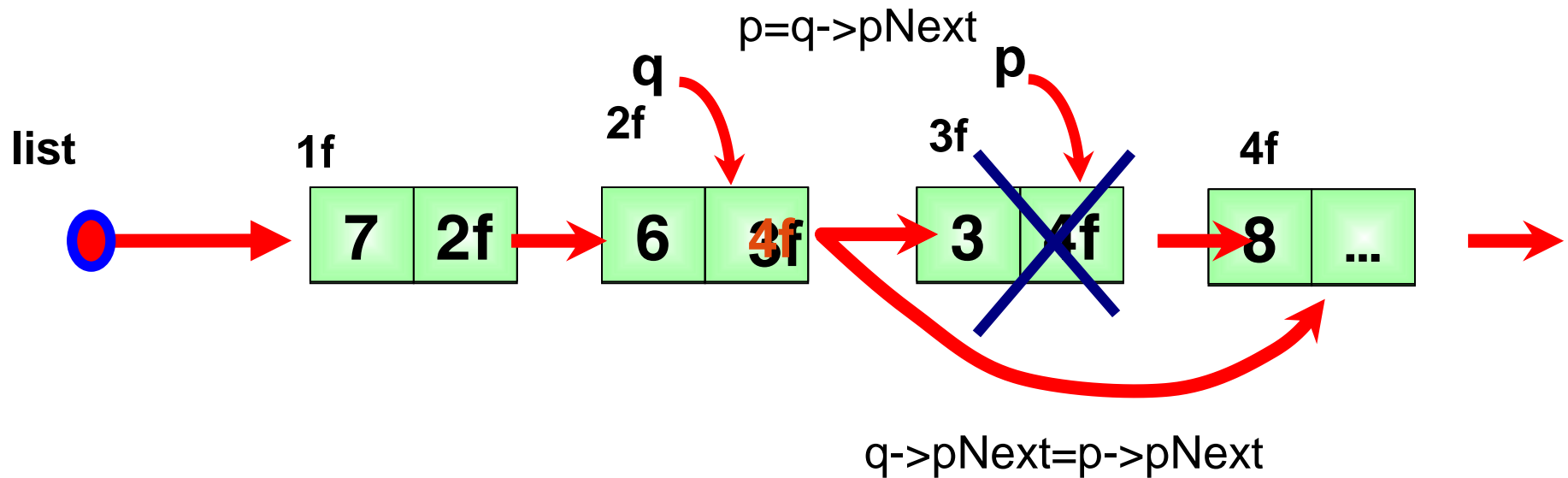
↪ Nếu ($l \neq \text{NULL}$) thì

- 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

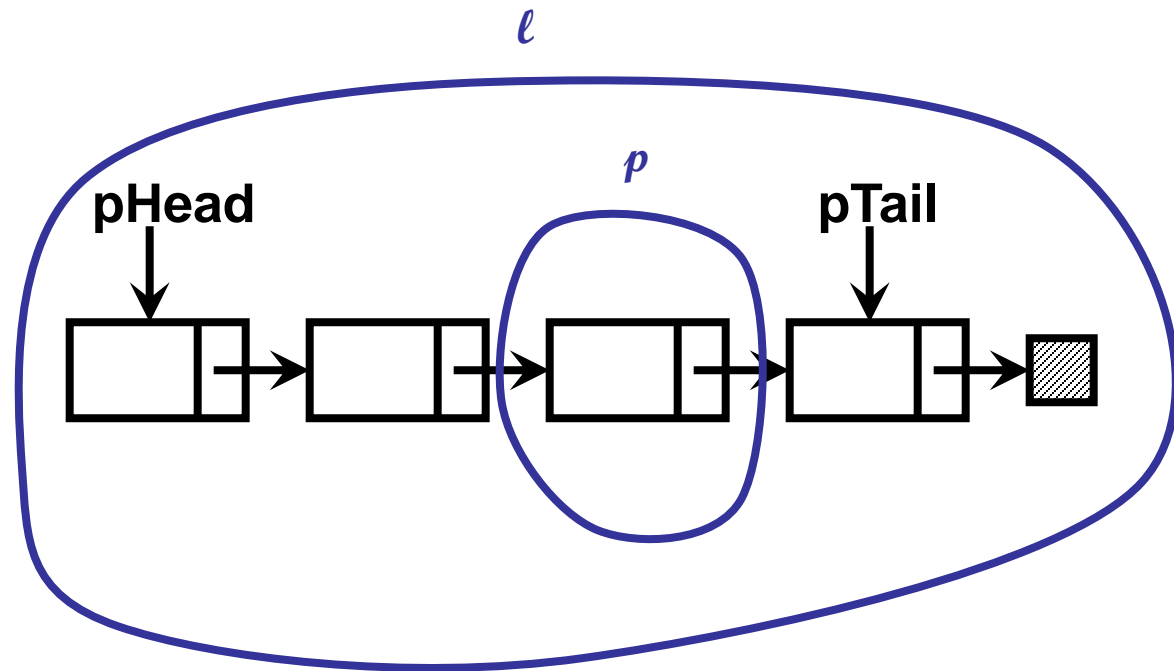
Hủy p ra khỏi danh sách bằng cách
hủy phần tử đứng sau q

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.

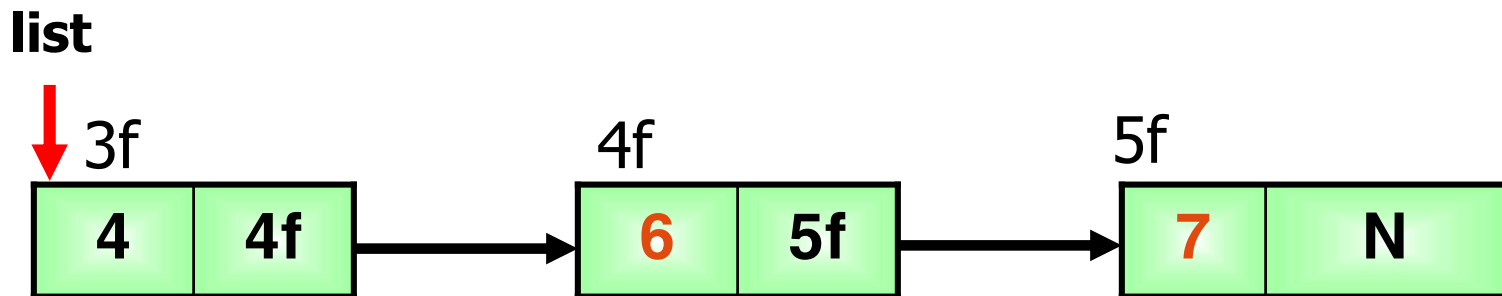
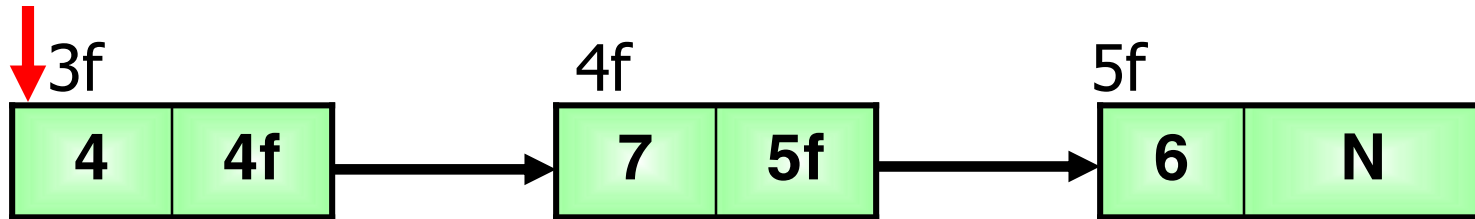
```
void remove_anynode(list &l)
{
    node *p,*q;
    int msv;
    printf("\tNhap mssv can xoa: ");
    scanf("%d",&msv);
    p=l.phead;
    if(p->info.mssv == msv)
    {
        l.phead = l.phead->pnext;
        delete p;
    }
    else
```

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



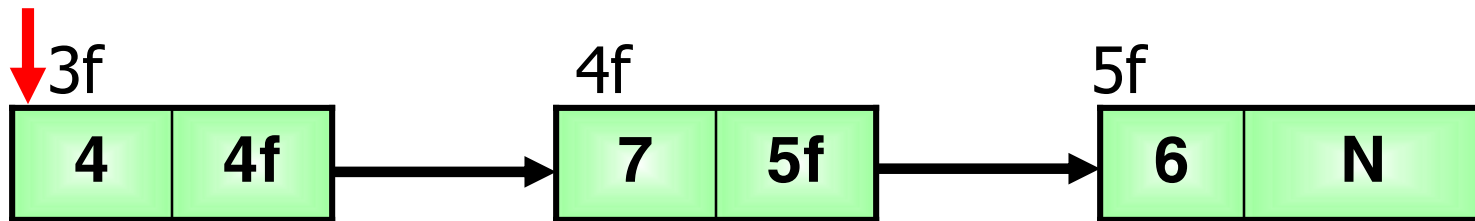
Tìm 1 node trong danh sách liên kết đơn.

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

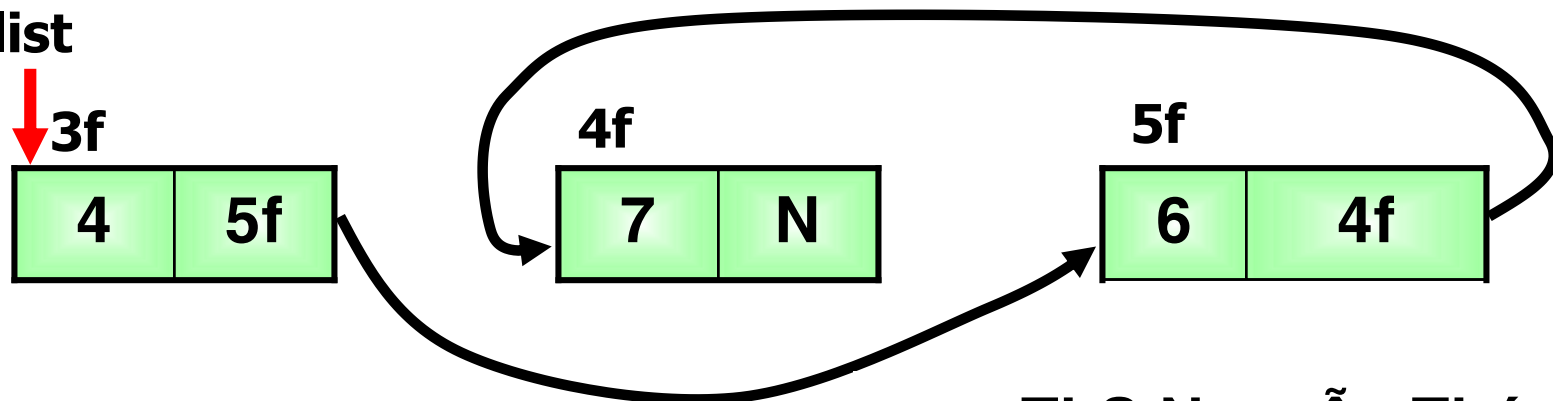
Sắp xếp danh sách

- **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

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 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 số thực.

```
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.         float x;
9.         printf("Nhap so thuc: ");
10.        scanf("%f", &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 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. }
```

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. }
```

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.}
```

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

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

```
// 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

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


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

```
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 n: ");
13.     scanf("%d", &n);
14.     Init(ℓ);
15.     for(int i=1; i<=n; i++)
16.     {   int x;
17.         printf("Nhap so nguyen:");
18.         scanf("%d", &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.}
```

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.