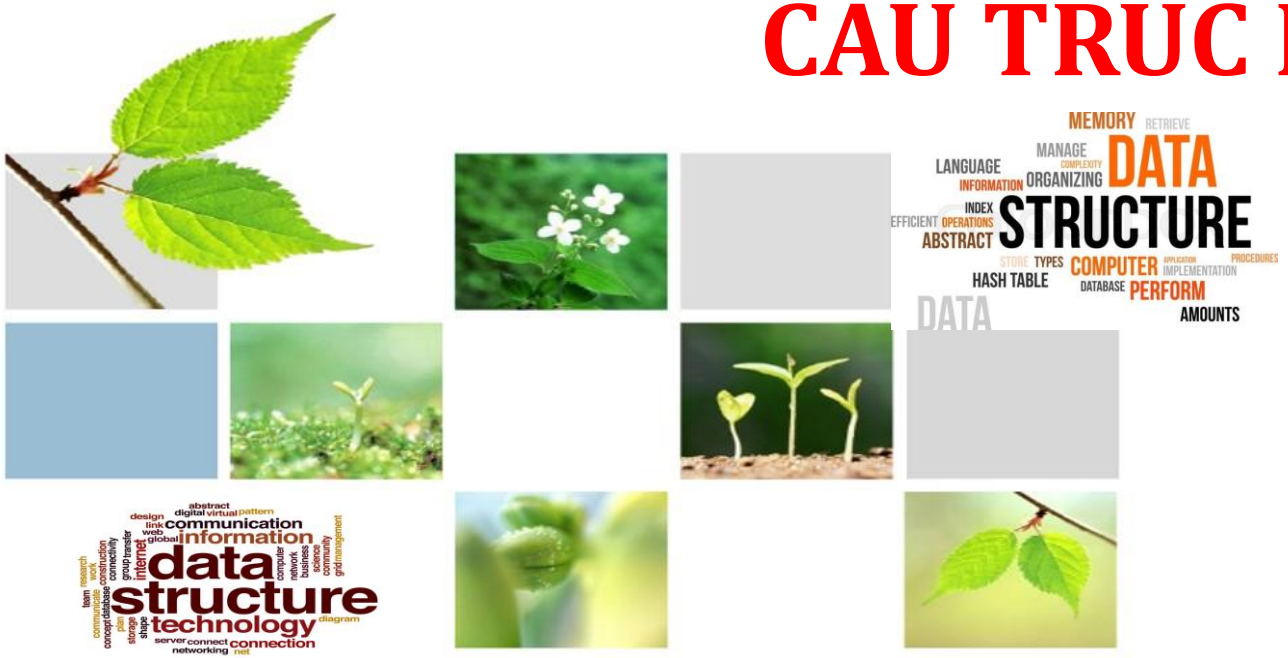
# BUỔI 5: CẤU TRÚC DỮ LIỆU ĐỘNG




## LÝ THUYẾT

# CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

## IT003.N210

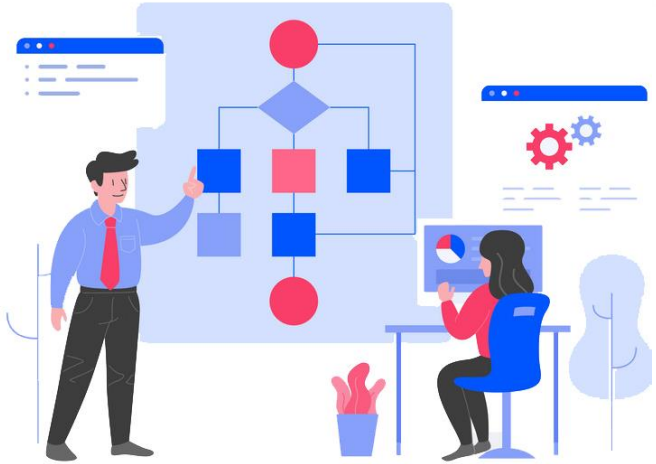
Giảng viên: Ths Đặng Văn Em

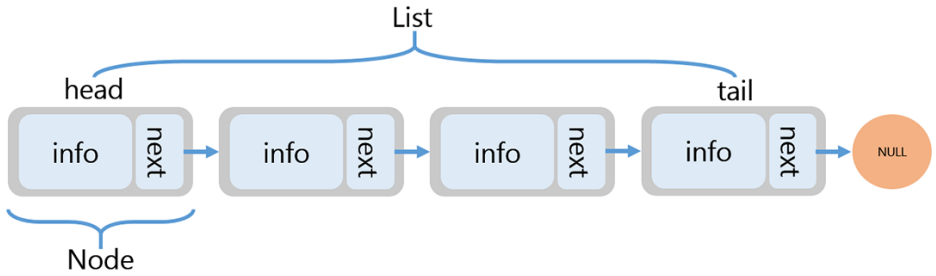
Email: vanem@uit.edu.vn

Điện thoại: 0966661006

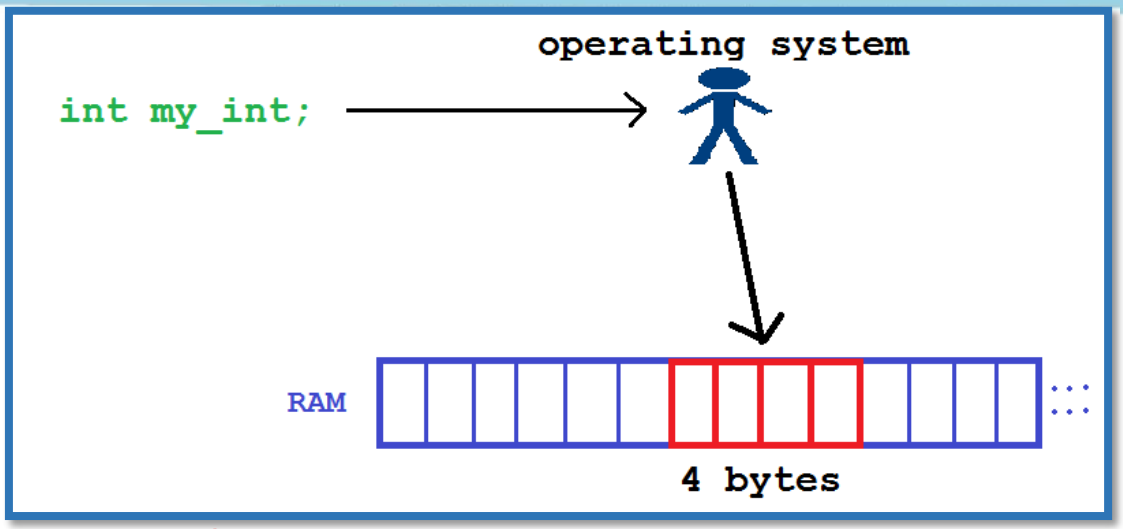
## Cấu trúc dữ liệu động

1. Khái niệm, vai trò cấu trúc dữ liệu động.
2. Kiểu dữ liệu con trỏ.
3. Danh sách liên kết, các hình thức tổ chức danh sách.
4. Danh sách liên kết đơn.
5. Tổ chức danh sách liên kết đơn
6. Các thao tác trên danh sách liên kết đơn.





# Khái niệm, vai trò cấu trúc dữ liệu động




**Biến tĩnh:**

- Được khai báo tường minh, có tên gọi.
- Tồn tại trong phạm vi khai báo.
- Được cấp phát trong stack.
- Kích thước không đổi => **Không tận dụng hiệu quả bộ nhớ**

**Ví dụ** : int x, y; char c; float f[5];

- Khi biết chắc nhu cầu sử dụng đối tượng trước khi thực sự xử lý: dùng biến tĩnh.

# Khái niệm, vai trò cấu trúc dữ liệu động

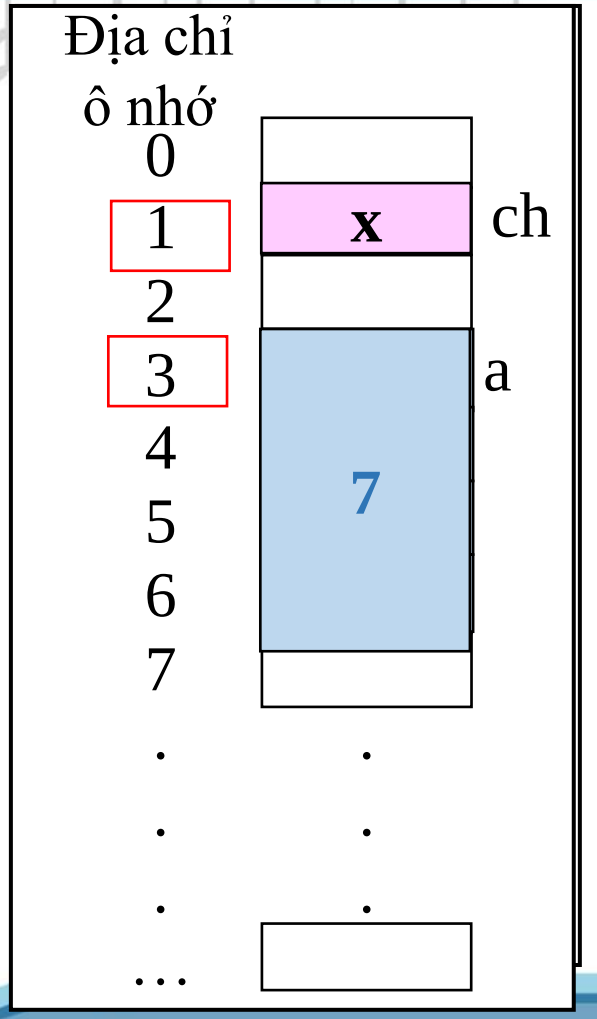
**Hạn chế biến tĩnh:**

- Tổ chức danh sách lớp học
- Dùng mảng tĩnh :

```
typedef  struct Hocvien
{
        char ten[20];
        int maso;
}Hocvien;
Hocvien  danhsach[50];
```

- Số lượng học viên <50 => **lãng phí**
- Số lượng học viên > 50 => **thiếu chỗ !**

```
int main() {
  char ch='x';
  int a = 7;
}
```



# Khái niệm, vai trò cấu trúc dữ liệu động

**Biến động:**

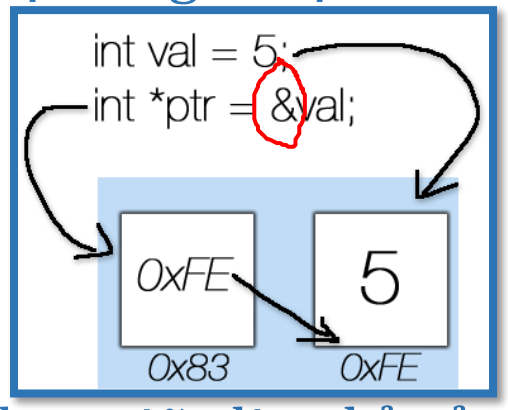
- Không được khai báo tường minh, không có tên gọi
- Xin khi cần, giải phóng khi sử dụng xong.
- Được cấp phát trong heap.
- Linh động về kích thước .
- Vấn đề : biến động không có tên gọi tường minh, làm sao thao tác ?

# Khái niệm, vai trò cấu trúc dữ liệu động

**Kiểu con trỏ:**

- Kiểu con trỏ dùng lưu địa chỉ của một đối tượng dữ liệu khác.
- Biến thuộc kiểu con trỏ Tp là biến mà giá trị của nó là địa chỉ cuả một vùng nhớ ứng với một biến kiểu T, hoặc là giá trị NULL. Bản thân biến con trỏ là không động
- Khai báo trong C :

```
int val=5, *ptr;
*ptr = &val;
```




- Dùng biến con trỏ để lưu giữ địa chỉ của biến động => **Truy xuất biến động thông qua biến con trỏ**

# Khái niệm, vai trò cấu trúc dữ liệu động

- **Cấp phát bộ nhớ tĩnh** (static memory allocation)
  - ✓ Khai báo biến, cấu trúc, mảng, …
  - ✓ Bắt buộc phải biết trước cần bao nhiều bộ nhớ lưu trữ ➔ tốn bộ nhớ, không thay đổi được kích thước, …
- **Cấp phát động** (dynamic memory allocation)
  - ✓ Cần bao nhiêu cấp phát bấy nhiêu.
  - ✓ Có thể giải phóng nếu không cần sử dụng.
  - ✓ Sử dụng vùng nhớ ngoài chương trình (cả bộ nhớ ảo virtual memory).

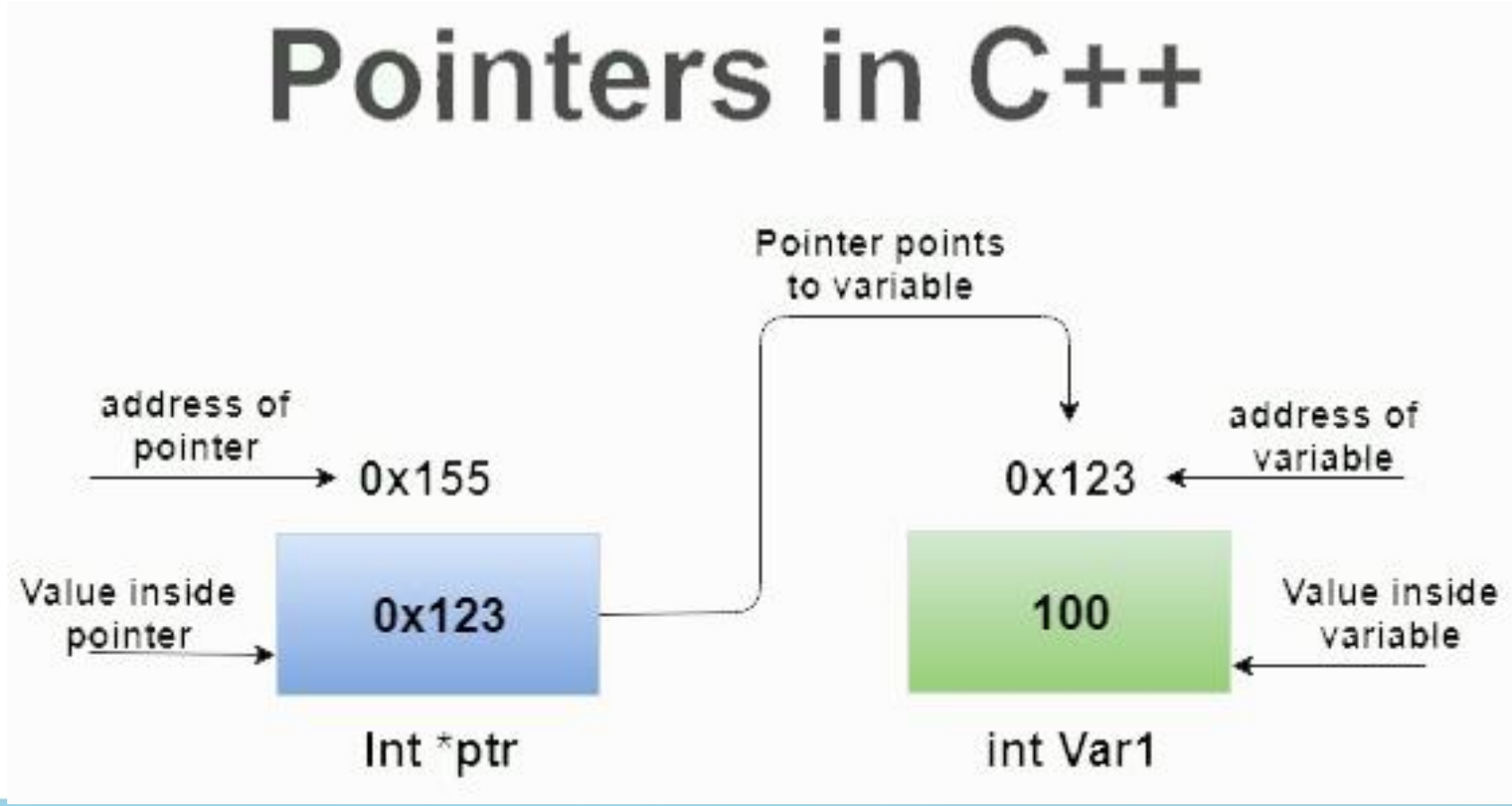
# Khái niệm, vai trò cấu trúc dữ liệu động

**Các thao tác trong kiểu con trỏ:**

- Cấp phát bộ nhớ
  - ✓ Trong C: Hàm **malloc, calloc, realloc** (<stdlib.h> hoặc <alloc.h>)
  - ✓ Trong C++: Toán tử **new**
- Giải phóng bộ nhớ
  - ✓ Trong C: Hàm **free**
  - ✓ Trong C++: Toán tử **delete**

# Khái niệm, vai trò cấu trúc dữ liệu động

**Ví dụ :**

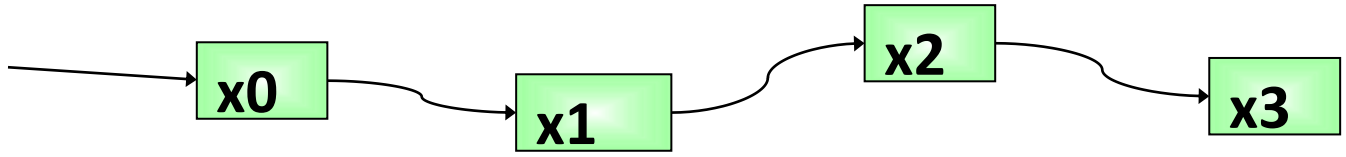


# Khái niệm, vai trò cấu trúc dữ liệu động

**Kiểu danh sách:**

- Danh sách = { các phần tử có cùng kiểu}
- Danh sách là một kiểu dữ liệu tuyến tính :
  - ✓ Mỗi phần tử có nhiều nhất 1 phần tử đứng trước
  - ✓ Mỗi phần tử có nhiều nhất 1 phần tử đứng sau
- Là kiểu dữ liệu quen thuộc trong thực tế :
  - ✓ Danh sách học sinh
  - ✓ Danh mục sách trong thư viện
  - ✓ Danh sách các nhân viên trong công ty.
  - ✓ .....

Cấu trúc dữ liệu cho một phần tử
- ✓ Thông tin bản than
- ✓ Địa chỉ của phần tử kế trong danh sách



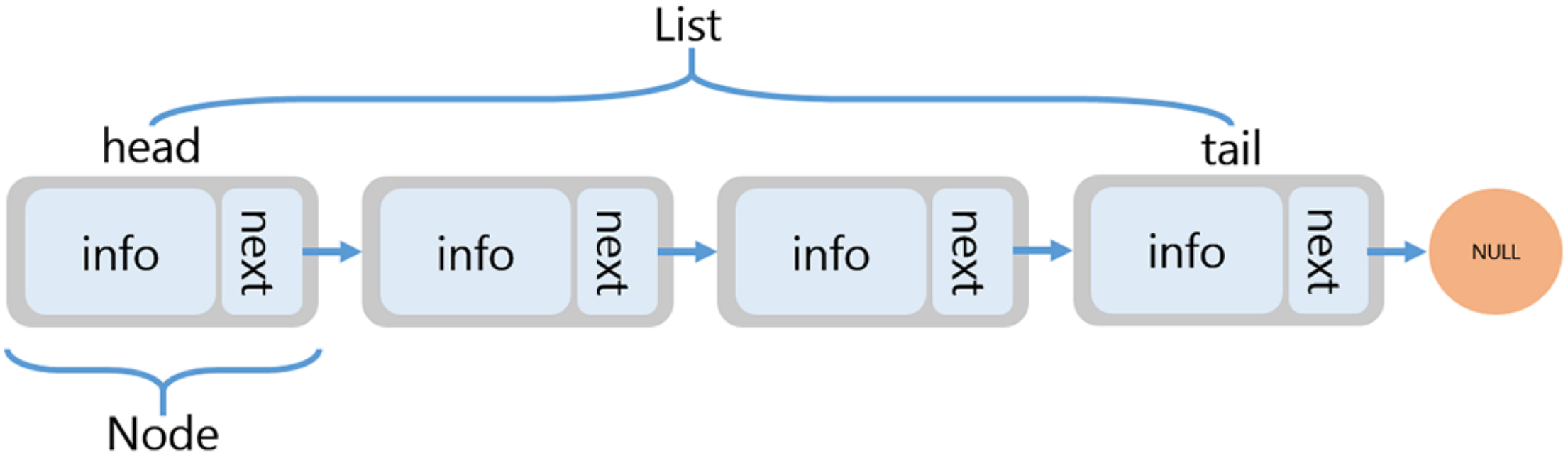
# Danh sách liên kết đơn

**Khái niệm danh sách liên kết đơn:**

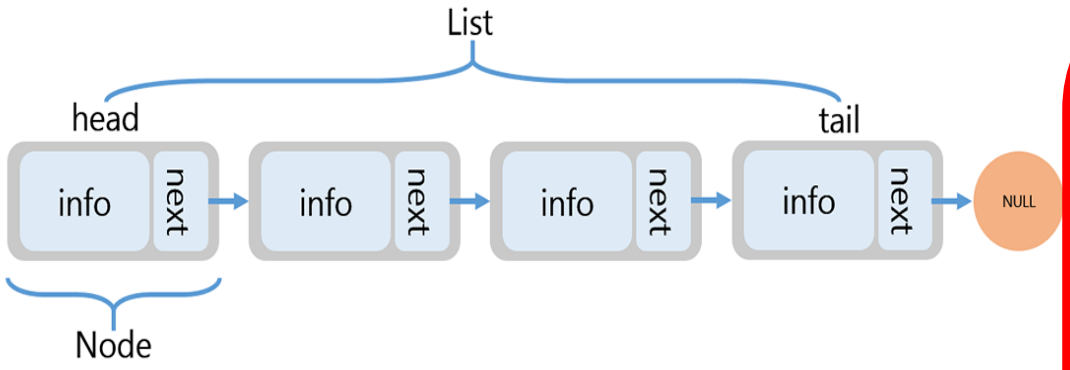
- Danh sách liên kết đơn là một tập hợp các phần tử (node). Mỗi phần tử liên kết với phần tử đứng liền sau trong danh sách.
- Mỗi phần tử trong danh sách liên kết đơn là một cấu trúc có hai thành phần.
  - ✓ **Thành phần dữ liệu**: Lưu trữ thông tin về bản thân phần tử.
  - ✓ **Thành phần liên kết**: Lưu địa chỉ phần tử đứng sau trong danh sách hoặc bằng NULL nếu là phần tử cuối danh sách.

# Danh sách liên kết đơn

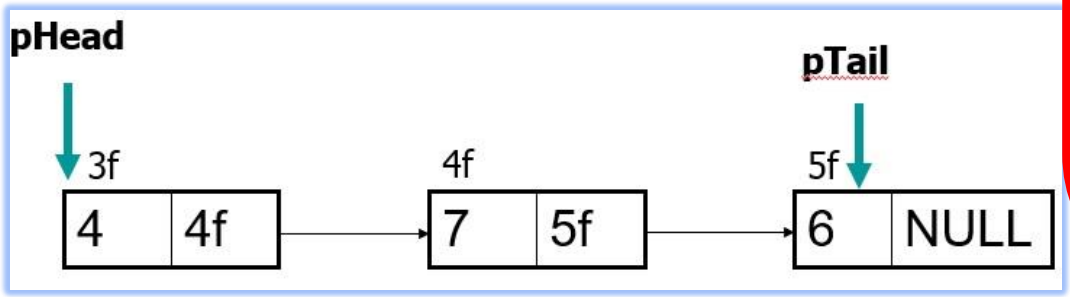
## Khái niệm danh sách liên kết đơn:



# Tổ chức Danh sách liên kết đơn




**Ví dụ:** trên thành phần dữ liệu một số nguyên




❖ **Cấu trúc dữ liệu của 1 nút trong DSLK đơn**

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

❖ **Cấu trúc dữ liệu của DSLK đơn**

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

# Các thao tác Danh sách liên kết đơn

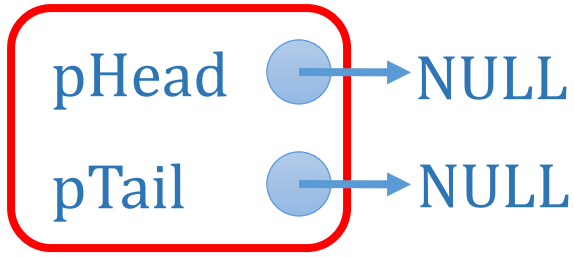
- Tạo 1 danh sách liên kết đơn rỗng.
- Tạo 1 nút có trường Infor bằng X.
- Thêm một phần tử vào danh sách.
- Hủy một phần tử X trong danh sách.
- Tìm một phần tử có Info bằng X.
- Duyệt danh sách liên kết đơn.
- Sắp xếp danh sách liên kết đơn.

# Các thao tác Danh sách liên kết đơn

## Tạo 1 danh sách liên kết đơn rỗng:

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

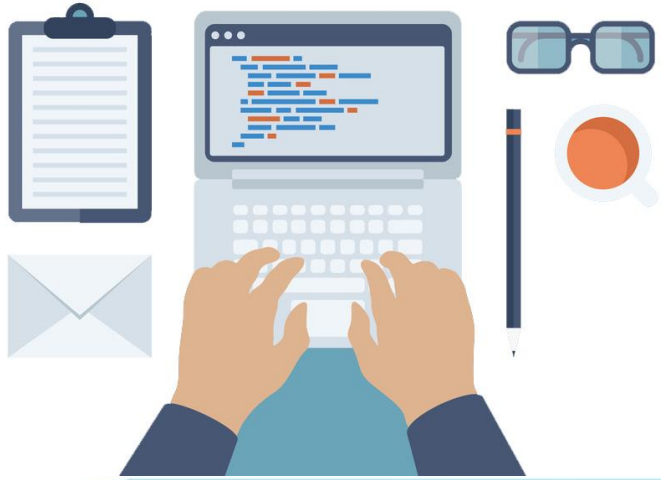
```
void CreateList(List &l)
{
        l.pHead=NULL;
        l.pTail=NULL;
}
```



# Các thao tác Danh sách liên kết đơn

**Tạo 1 nút có trường Infor bằng X:**

```
Node*   CreateNode(Data x) // Hàm trả về địa chỉ phần tử mới tạo:
{   Node *p;
    p = new Node;  //Cấp phát vùng nhớ cho phần tử
    if ( p==NULL)  exit(1);
    p ->Info = x;    //gán dữa liệu cho nút
    p->pNext = NULL;
    return p;
}
```

# Các thao tác Danh sách liên kết đơn

## Thêm một phần tử có Info bằng X vào danh sách:

- **Nguyên tắc thêm:** Khi thêm 1 phần tử vào danh sách thì có làm cho pHead, pTail thay đổi?
- **Các vị trí cần thêm 1 phần tử vào danh sách:**
  - ✓ Thêm phần tử vào đầu danh sách.
  - ✓ Thêm phần tử vào cuối danh sách.
  - ✓ Thêm phần tử vào sau 1 phần tử q trong danh sách.

# Các thao tác Danh sách liên kết đơn

**Thuật toán thêm một phần tử p vào ĐẦU danh sách:**

**Bắt đầu**:

Nếu List rỗng thì

- pHead = p;
- pTail = pHead;

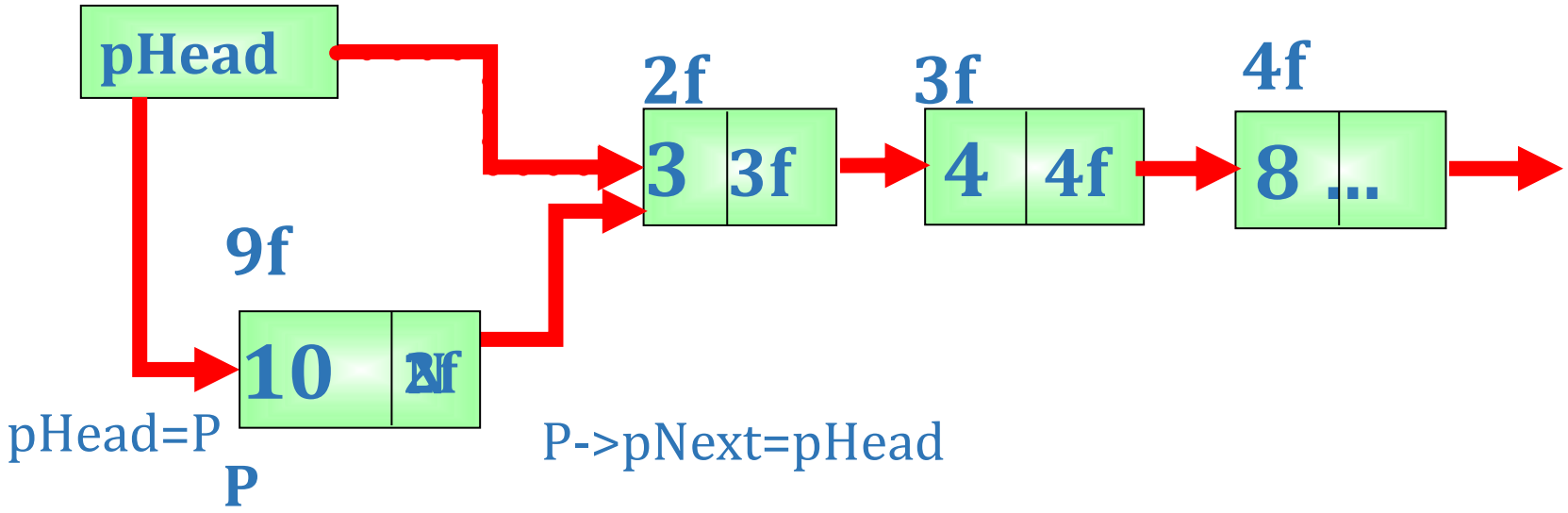
Ngược lại

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

```
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;
        }
}
```

# Các thao tác Danh sách liên kết đơn

**Minh họa Thuật toán thêm một phần tử p vào ĐẦU danh sách:**



# Các thao tác Danh sách liên kết đơn

**Thuật toán thêm một phần tử vào CUỐI danh sách:**

**Bắt đầu**:

Nếu List rỗng thì

- pHead = p;
- pTail = pHead;

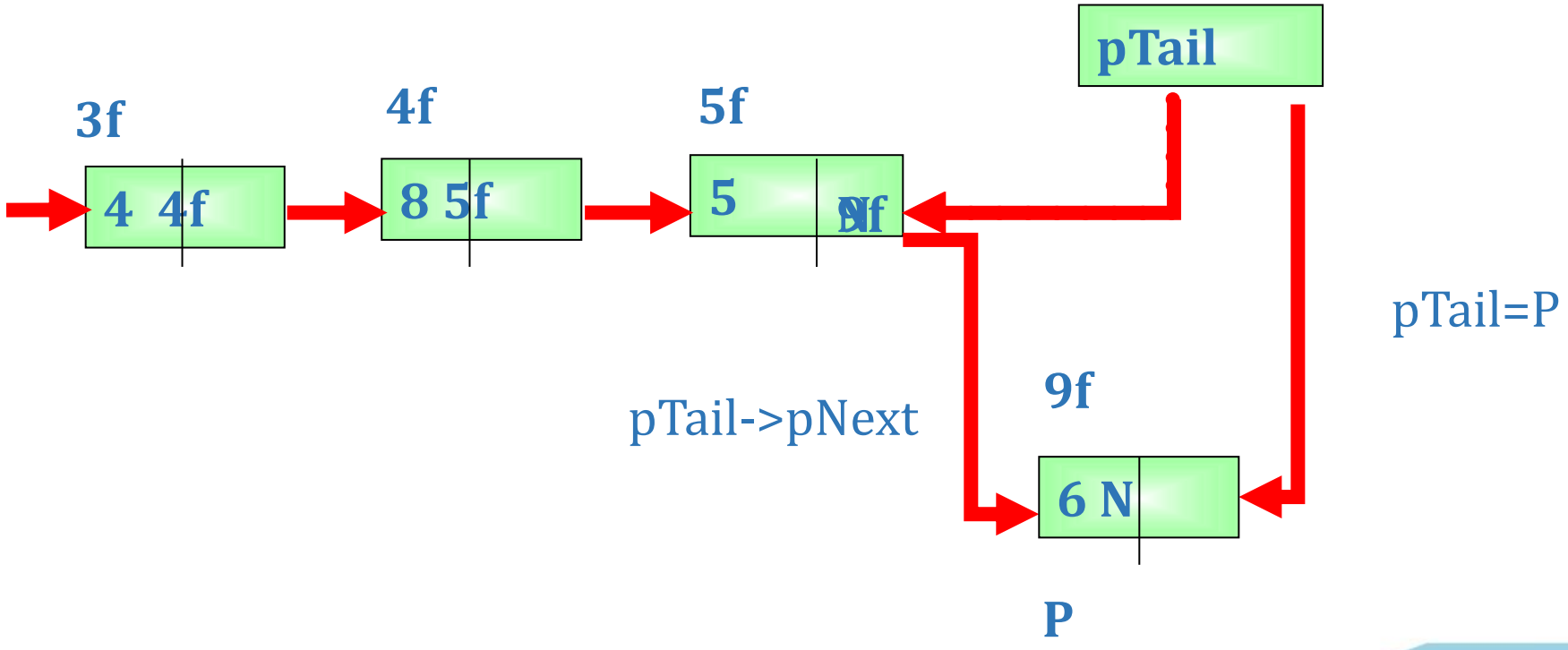
Ngược lại

- pTail->pNext=p;
- pTail=p;

```
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;
        }
}
```

# Các thao tác Danh sách liên kết đơn

**Minh họa thuật toán thêm một phần tử p vào CUỐI danh sách:**



# Các thao tác Danh sách liên kết đơn

**Thuật toán thêm một phần tử p vào sau phần tử q trong danh sách:**

**Bắt đầu**:

Nếu (q!=NULL) thì

B1: p->pNext = q->pNext

B2:

q->pNext = p

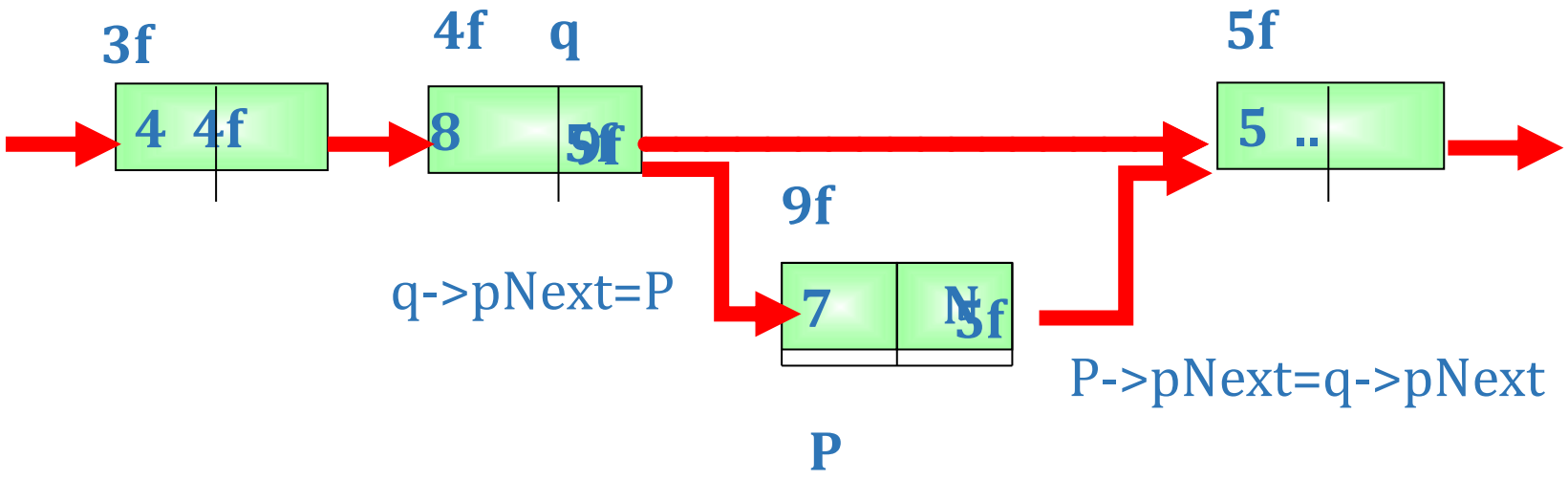
nếu q = pTail

thì pTail=p

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

# Các thao tác Danh sách liên kết đơn

**Minh họa thuật toán thêm một phần tử p vào sau phần tử q trong danh sách:**



# Các thao tác Danh sách liên kết đơn

## Thuật toán hủy một phần tử p trong danh sách:

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

# Các thao tác Danh sách liên kết đơn

## Thuật toán hủy một phần tử p ĐẦU danh sách:

**Bắt đầu:**

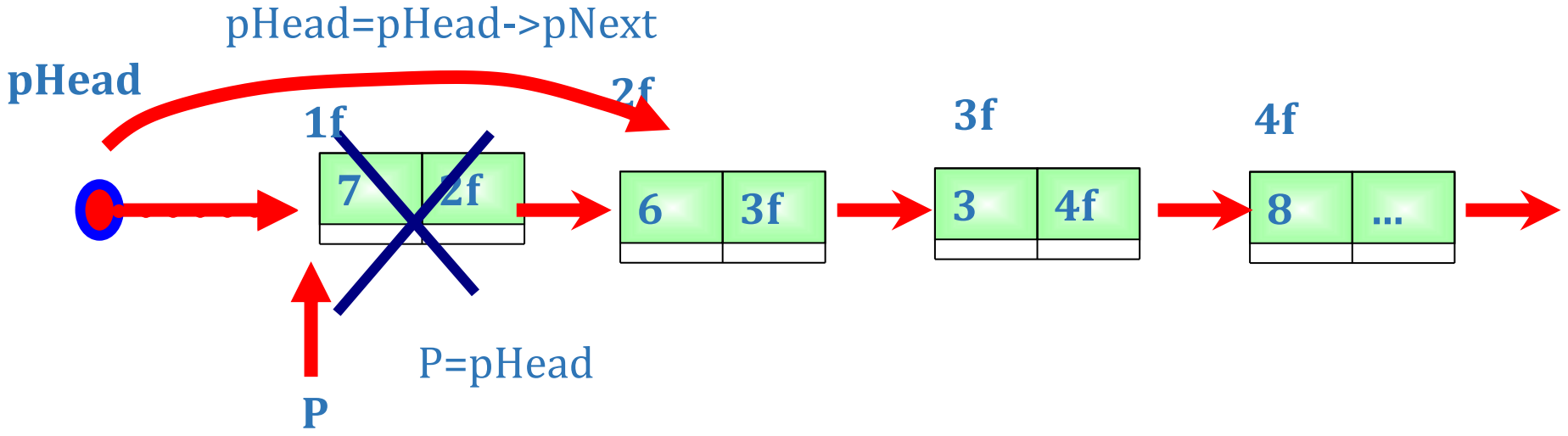
- Nếu (pHead!=NULL) thì
- B1: p=pHead
- B2:
  - ✓ pHead = pHead->pNext
  - ✓ delete (p)
- B3:

  Nếu pHead==NULL thì pTail=NULL

```
//Hủy được hàm trả về 1, ngược lại hàm trả về 0
void  RemoveHead(List &l)
{       Node *p;
        if(l.pHead!=NULL)
                {
                        p=l.pHead;
                        l.pHead=p->pNext;
                                delete p;
                        if(l.pHead==NULL)
                                l.pTail=NULL;
                }
        }
```

# Các thao tác Danh sách liên kết đơn

## Minh họa Thuật toán hủy một phần tử p ĐẦU danh sách:



# Các thao tác Danh sách liên kết đơn

**Thuật toán hủy một phần tử p sau phần tử q trong danh sách:**

Bắt đầu:

Nếu (q!=NULL) thì //q tồn tại trong List

B1: p=q->pNext;// p là phần tử cần hủy

B2: Nếu (p!=NULL) thì // q không phải là phần tử cuối

q->pNext=p->pNext;// tách p ra khỏi xâu

nếu (p== pTail) // nút cần hủy là nút cuối

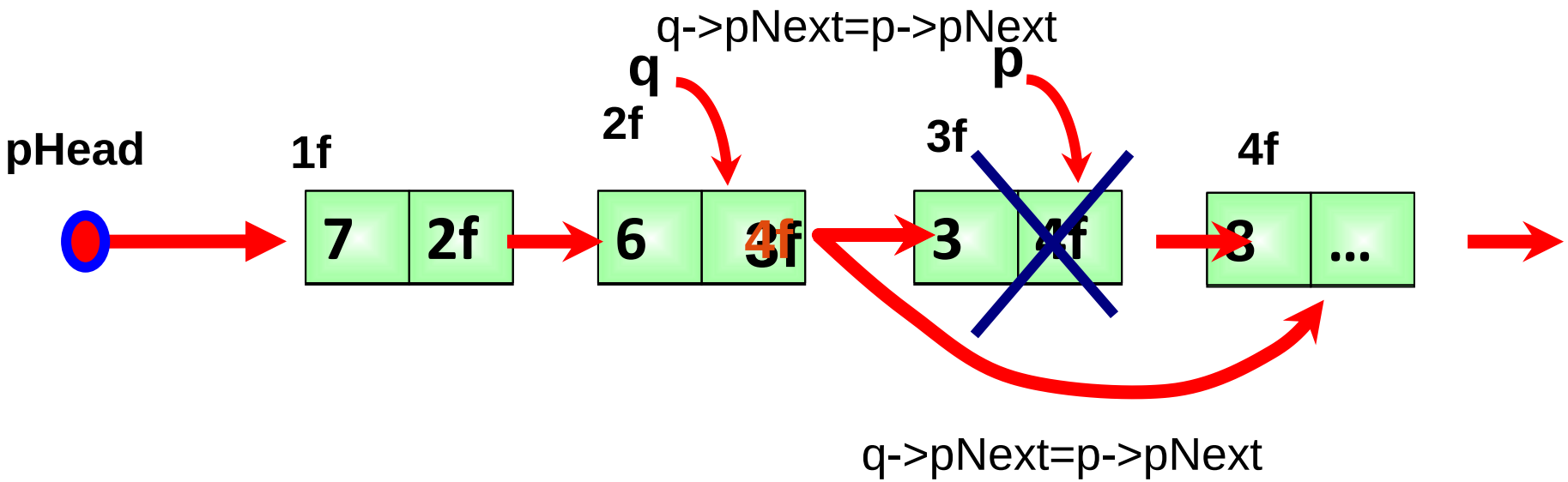
pTail=q;

delete p;// hủy p

```
int RemoveAfterQ(List &l, Node *q, int &x)
{   Node *p;
    if(q!=NULL)
    {       p=q->pNext; //p là nút cần xoá
            if(p!=NULL) // q không phải là nút cuối
            {       if(p==l.pTail) //nút cần xoá là nút cuối cùng
                            l.pTail=q;// cập nhật la pTail
                    q->pNext=p->pNext; x=p->Info;
                    delete p;
            }
            return 1;
    }
    else
        return 0;   }
```

# Các thao tác Danh sách liên kết đơn

**Minh họa Thuật toán hủy một phần tử p sau phần tử q trong danh sách:**



# Các thao tác Danh sách liên kết đơn

**Thuật toán hủy một phần tử khóa x trong danh sách:**

**Bước 1:**

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

**Bước 2:**

Nếu (p!=NULL) thì //tìm thấy phần tử có khoá bằng x

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

Ngược lại

Báo không tìm thấy phần tử có khoá

```
void RemoveX(List &l, char *x)
{   Node *p,*q = NULL; p=l.Head;
    while((p!=NULL)&&(strcmp(p->Info.MSSV,x)!=0))
    {   q=p;
        p=p->Next;
    }
    if(p==NULL) //không tìm thấy phần tử có khoá bằng x
        printf("Tim khong tahy");
    else //tim thay
        if(q==NULL)//tìm thấy phần tử có khoá bằng x
                RemoveHead(l);
        else //co q, tuc q khac NULL
        {
                q->pNext=p->pNext;
                if(p==l.pTail)
                        l.pTail=q;
                        delete p;
        }
}
```

# Các thao tác Danh sách liên kết đơn

**Thuật toán tìm một phần tử có info bằng X trong danh sách:**

❖ Tìm tuần tự  các phần tử các nút có Info bằng x trong danh sách.

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

**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

# Các thao tác Danh sách liên kết đơn

## Thuật toán tìm một phần tử có info bằng X trong danh sách:

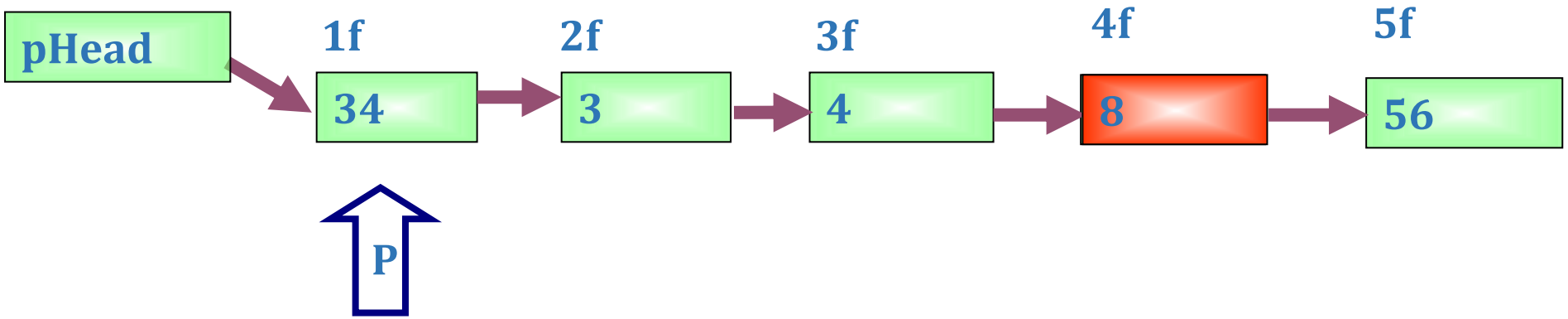
❖ Hàm tìm phần tử có Info = x, hàm trả về địa chỉ của x, ngược lại hàm trả về NULL.

```
Node *Search(LIST l, int  x)
{
        Node      *p;
        p = l.pHead;
        while((p!= NULL)&&(p->Info != x))
                p = p->pNext;
        return p;
}
```


**Bài tập**: Đếm số sinh viên có tên là X trong lớp học, nếu không có thì thông báo không có

# Các thao tác Danh sách liên kết đơn

**Minh hoạt Thuật toán tìm một phần tử có info bằng X trong danh sách:**




X = 8

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

# Các thao tác Danh sách liên kết đơn

**Thuật toán tìm một phần tử có info bằng X trong danh sách:**

- 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 tất cả các phần tử trong danh sách thảo điều kiện
  - ✓ Hủy toàn bộ danh sách

# Các thao tác Danh sách liên kết đơn

**Thuật toán duyện các phần tử trong danh sách:**

**Bước 1**:

p = pHead;

**Bước 2:**

Trong khi (danh sách chưa hết) thực hiện

- xử lý phần tử p
- p=p->pNext;// *qua phần tử kế*

```
void PrintList(List l)
{
        Node *p;
        p=l.pHead;
        while(p!=NULL)
        {       printf("%d    ", p->Info);
                p=p->pNext;
        }
}
```

# Các thao tác Danh sách liên kết đơn

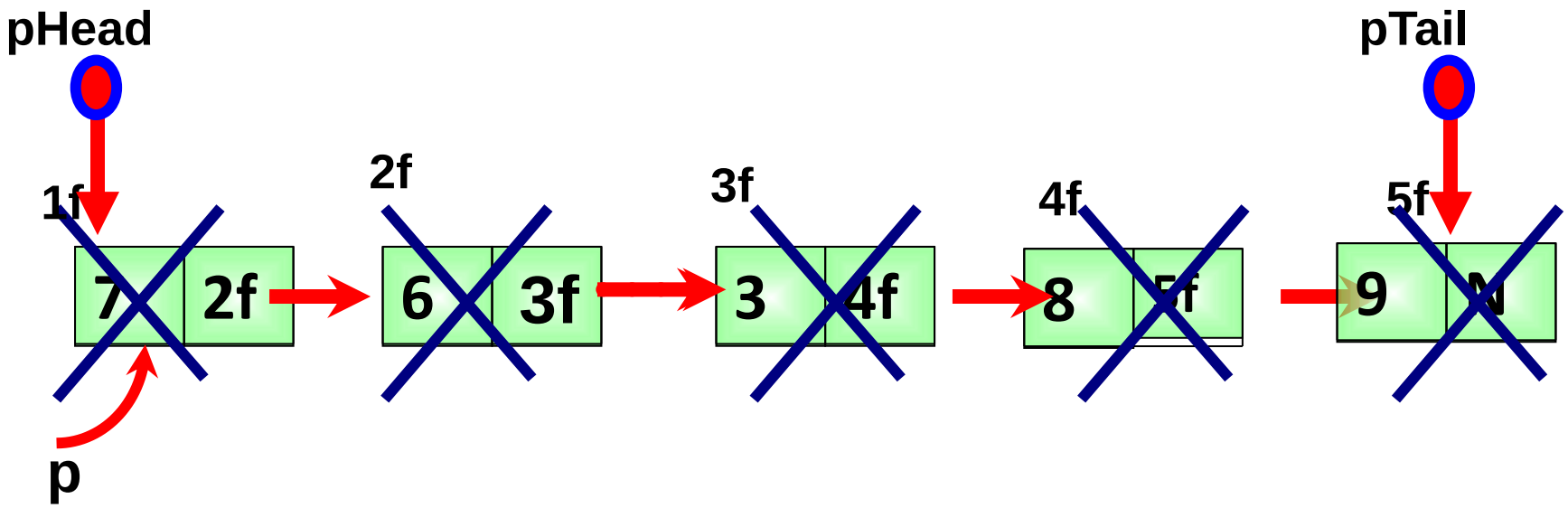
**Thuật toán xóa phần tử trong danh sách:**

- **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 xâu rỗng*

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

# Các thao tác Danh sách liên kết đơn

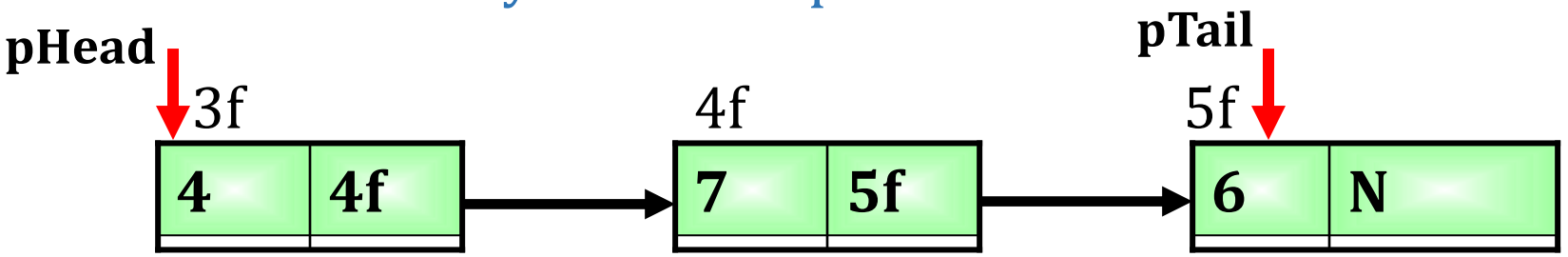
## Minh họa Thuật toán xóa phần tử trong danh sách:

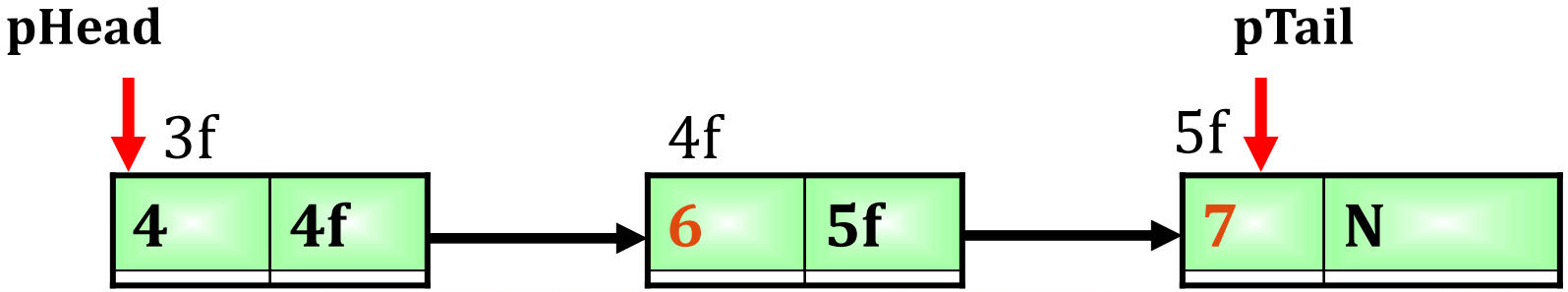


# Các thao tác Danh sách liên kết đơn

**Thuật toán sắp xếp các phần tử trong danh sách:**

- Có 02 cách tiếp cận.
- **Cách 01:** là thay đổi thành phần info.

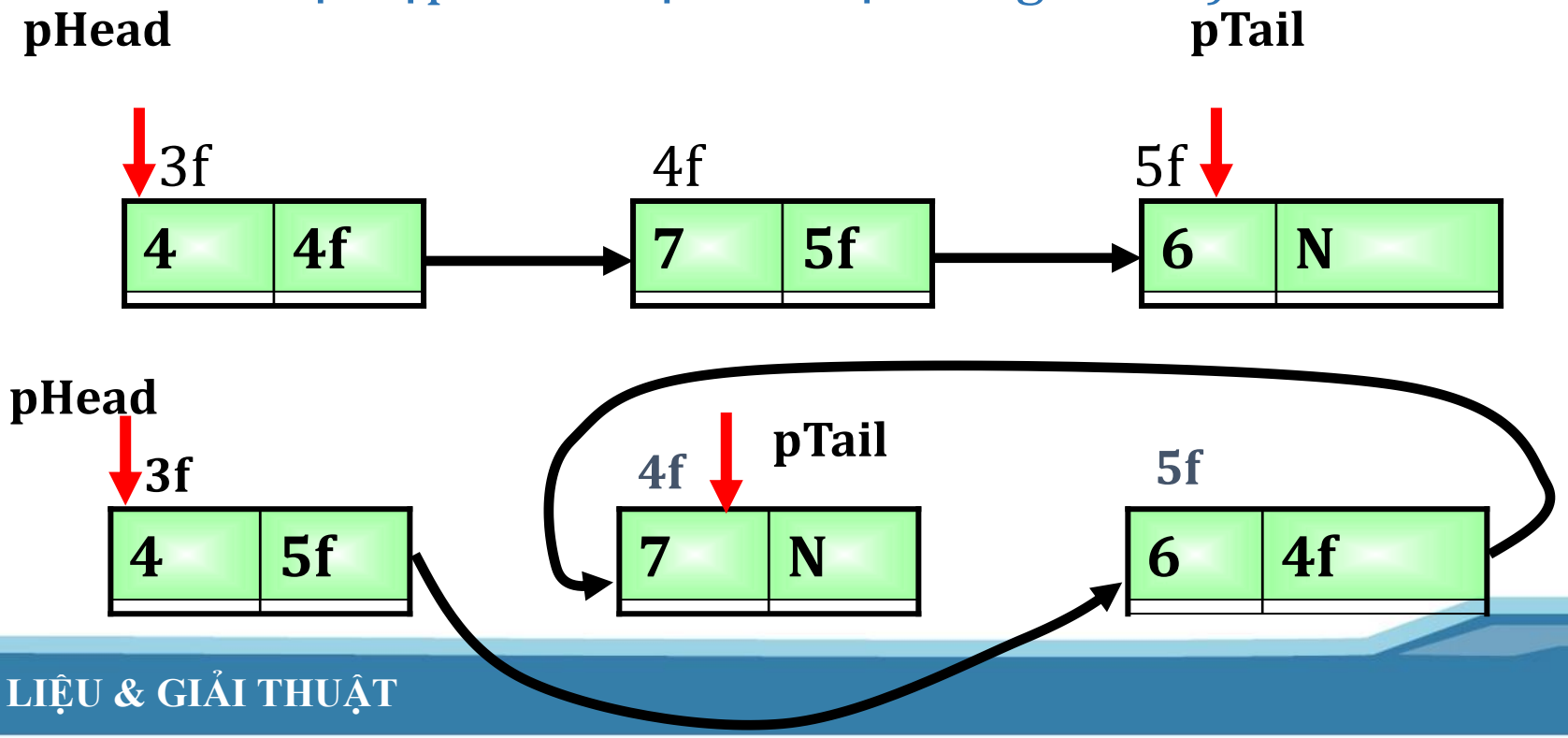


# Các thao tác Danh sách liên kết đơn

**Thuật toán sắp xếp các phần tử trong danh sách:**

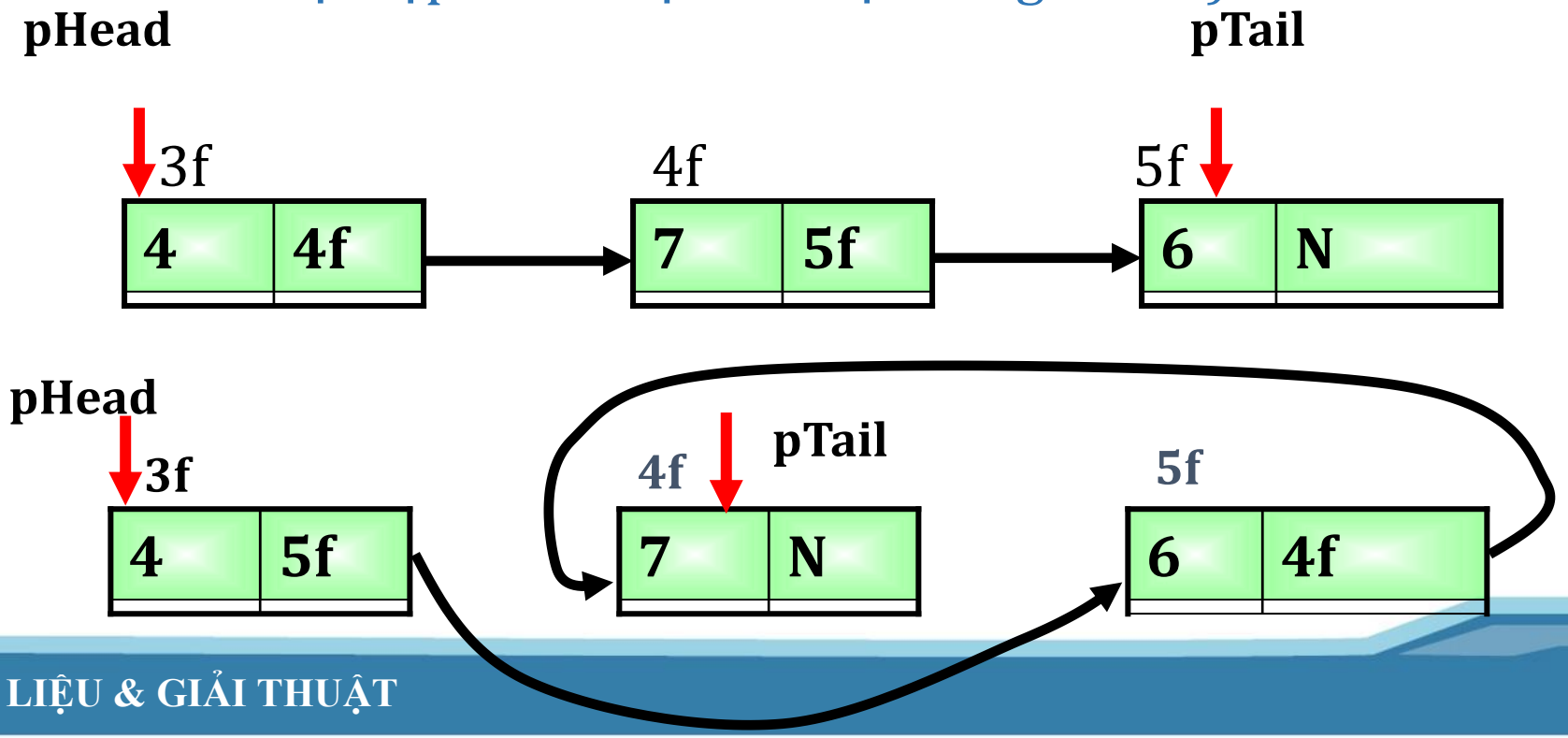
❖ **Cách 02:** Thay đổi thành phần pNext (thay đổi trình tự móc nối của các phần tử sao cho tạo lập nên được thứ tự mong muốn).



# Các thao tác Danh sách liên kết đơn

**Thuật toán sắp xếp các phần tử trong danh sách:**

❖ **Cách 02:** 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).



# Các thao tác Danh sách liên kết đơn

**Minh họa Thuật toán sắp xếp các phần tử trong danh sách:**

```
void SelectionSort(LIST &l)
{
   Node *p,*q,*min;
   p=l.pHead;
   while(p!=l.pTail)
   {
      min=p;
      q=p->pNext;
   while(q!=NULL)
   {
      if(q->Info<p->Info)
      min=q;
      q=q->pNext;
   }
   HV(min->Info,p->Info);
   p=p->pNext;
   }
}
```

# Bài tập Danh sách liên kết đơn

**Bài tập 1:** 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.

# Bài tập Danh sách liên kết đơn

**Bài tập 2:** 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

# Bài tập Danh sách liên kết đơn

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>=50 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

...

# Bài tập Danh sách liên kết đơn

❖ **Cấu trúc dữ liệu của một sinh viên**

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

❖ **Cấu trúc dữ liệu của 1 nút trong danh sách liên kết**

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

# Bài tập Danh sách liên kết đơn

**Bài tập 3:** Thông tin của một nhân viên gồm: Mã số nhân viên (**MSNV**), Họ tên nhân viên (**HOTEN**), hệ số lương (**HeSoLuong**) và Lương (**Luong**).

1. Hãy khai báo cấu trúc dữ liệu dạng DSLK đơn để lưu danh sách nhân viên.
2. Nhập danh sách các nhân viên, mỗi nhân viên có hệ số lương 2.34<=HeSoLuong<=8.0). Lương được tính (Luong=hesoluong*luongcoban)
3. Sắp xếp danh sách lương nhân viên tăng/giảm. Tìm một nhân viên có lương cao nhất trong danh sách.
4. Tìm nhân viên có MSVV bằng X (X nhập từ bàn phím). *(Tìm nhân viên theo tên nhập)*
5. Xoá một sinh viên có MSSV bằng X (X nhập từ bàn phím).
6. Liệt kê danh sách nhân viên có lương dưới 6.000.000vnđ

# Bài tập Danh sách liên kết đơn

❖ **Cấu trúc dữ liệu của một nhân viên**

```
typedef struct Nhanvien
{
        int manv;
        string hoten;
        float hesoluong;
        float luong;
}Nhanvien;
```

❖ **Cấu trúc dữ liệu của 1 nút trong DSLK**

```
typedef struct Node
{
        Nhanvien data;
        struct Node *pNext;
}Node;

typedef struct List
{
        Node *pHead, *pTail;
}Node;
```

# Câu hỏi Bài tập Danh sách liên kết đơn

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.