

# HUST

**ĐẠI HỌC BÁCH KHOA HÀ NỘI**

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.



**ĐẠI HỌC**  
**BÁCH KHOA HÀ NỘI**  
HANOI UNIVERSITY  
OF SCIENCE AND TECHNOLOGY

# DANH SÁCH LIÊN KẾT (PHẦN I)

## HB230 C Programming Basic

ONE LOVE. ONE FUTURE.

# Nội dung chính

- **Giới thiệu chung về danh sách liên kết**
- Xây dựng cấu trúc dữ liệu danh sách liên kết đơn
- Ứng dụng danh sách liên kết đơn trong một số bài toán cụ thể



# Cấu trúc dữ liệu sử dụng bộ nhớ động

- Mảng là tập hợp các phần tử **đồng nhất** được lưu trữ ở các vị trí **kế tiếp** trong bộ nhớ.
- Hạn chế của mảng:
  - Là cấu trúc dữ liệu tĩnh.
  - Cần xác định trước kích thước tại thời điểm biên dịch, trong phần lớn các ngôn ngữ lập trình.
  - Không hiệu quả trong thao tác thêm và xóa phần tử.
- Một cấu trúc dữ liệu **động** có thể khắc phục các vấn đề trên.



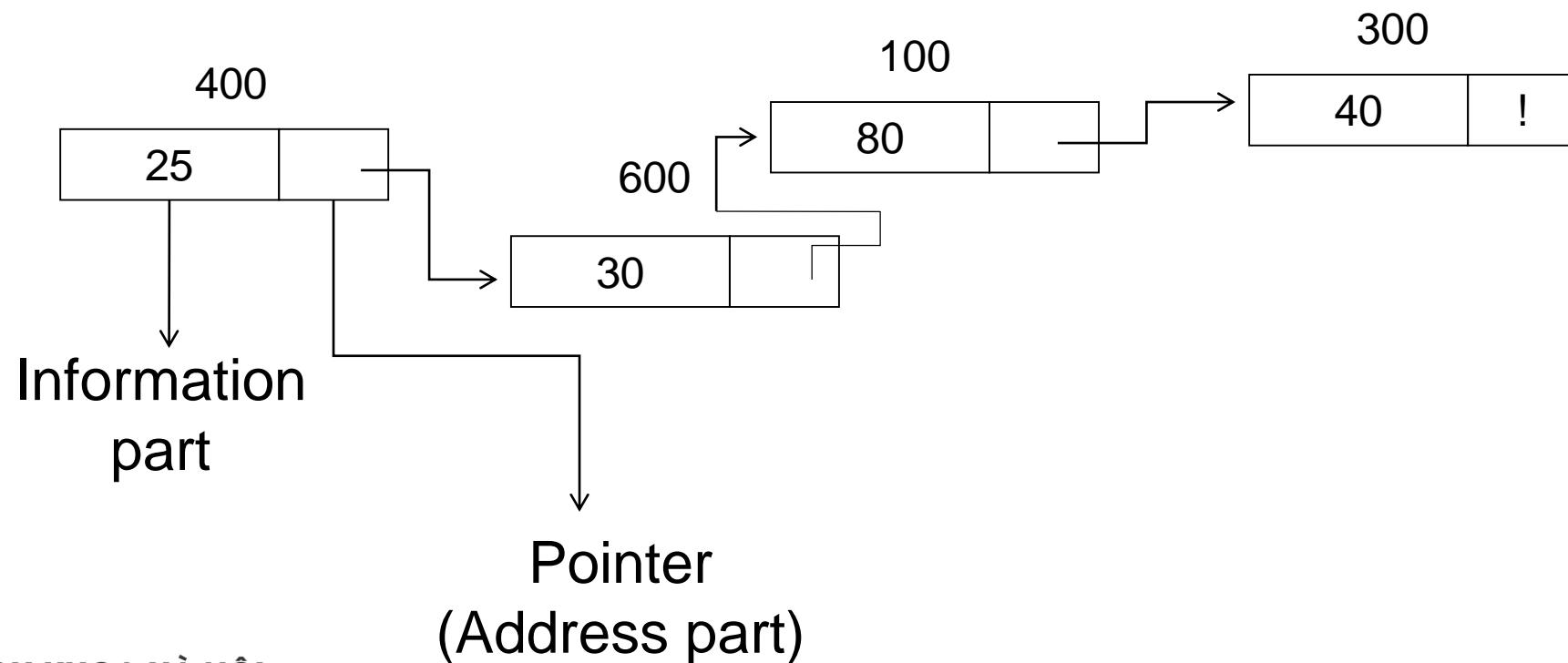
# Thế nào là một cấu trúc dữ liệu động?

- Cấu trúc dữ liệu có thể **thu nhỏ** hoặc **mở rộng** trong quá trình thực thi chương trình
- Kích thước của dữ liệu thuộc cấu trúc dữ liệu động không nhất thiết phải được biết trước **tại thời điểm biên dịch**.
- Thao tác thêm và xóa các phần tử **hiệu quả**.
- Dữ liệu trong cấu trúc dữ liệu động có thể được lưu trữ tại các vị trí **không kế tiếp** (bất kỳ) trong bộ nhớ.
- Danh sách liên kết (**móc nối**) là một ví dụ.



# Danh sách liên kết đơn

- Danh sách liên kết là tập hợp các **phần tử**, mỗi phần tử (nút) chứa một số **thông tin** và một **con trỏ** trỏ tới nút tiếp theo trong danh sách
- Trong ví dụ dưới đây, danh sách gồm bốn nút được lưu trữ ở các vị trí không kế tiếp trong bộ nhớ



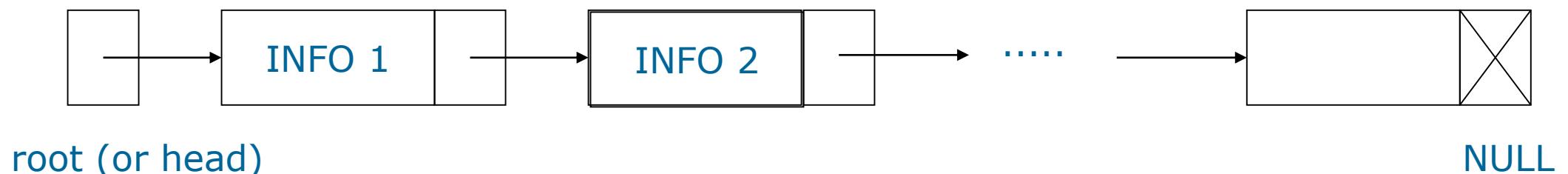
# Nội dung chính

- Giới thiệu chung về danh sách liên kết
- **Xây dựng cấu trúc dữ liệu danh sách liên kết đơn**
- Ứng dụng danh sách liên kết đơn trong một số bài toán cụ thể



# Xây dựng danh sách liên kết đơn trong ngôn ngữ C

- Con trỏ được sử dụng để lưu trữ địa chỉ của nút kế tiếp (NEXT).
  - Giá trị này ở nút cuối là **NULL**.
- Sử dụng dạng cấu trúc tự trỏ
  - Kiểu cấu trúc có trường con trỏ trỏ tới phần tử thuộc chính kiểu cấu trúc đó
  - Có thể định nghĩa kiểu cho trường thông tin dữ liệu (INFO) sử dụng struct và typedef
- Nhân tố quan trọng: con trỏ **root**
  - Luôn **trỏ tới phần tử đầu** danh sách – quản lý truy cập tới danh sách.



# Ví dụ về cấu trúc tự trả

- Một hay nhiều trường là con trả tới chính cấu trúc đó.

```
struct list {  
    char data;  
    struct list *link;  
};
```

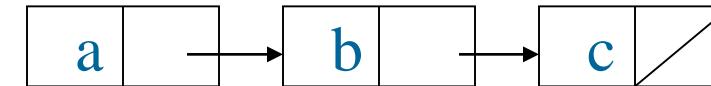
```
list item1, item2, item3;
```

```
item1.data='a';
```

```
item2.data='b';
```

```
item3.data='c';
```

```
item1.link=item2.link=item3.link=NULL;
```



# Các thao tác (hàm) trên danh sách liên kết đơn

- Cấp phát bộ nhớ động – tạo phần tử (nút) mới
- Thêm (chèn) phần tử mới vào danh sách
  - ở vị trí đầu, vào cuối
  - ở giữa (căn cứ theo vị trí phần tử hiện tại hoặc một vị trí tuyệt đối):
    - trước nút hiện hành
    - sau nút hiện hành
- Xóa một phần tử
- Duyệt – Hiển thị nội dung – Tìm kiếm
- Đảo ngược danh sách
- Giải phóng bộ nhớ đã cấp phát cho danh sách
- ...

# Một số bước trong xây dựng và sử dụng một cấu trúc dữ liệu

- Định nghĩa kiểu dữ liệu cho cấu trúc dữ liệu
  - kiểu cho trường INFO (không bắt buộc), kiểu đại diện một phần tử của cấu trúc dữ liệu
- Khai báo biến toàn cục (không bắt buộc)
  - Ví dụ các con trỏ quan trọng
- Cài đặt các hàm thực hiện các thao tác trên cấu trúc dữ liệu
- Sử dụng cấu trúc dữ liệu trong một chương trình – bài toán ứng dụng cụ thể

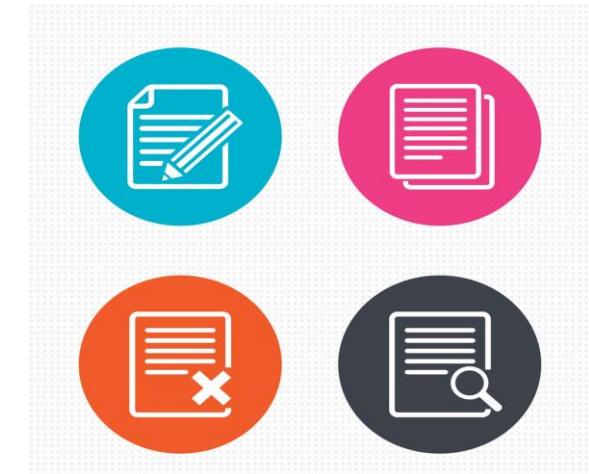


# Bài toán minh họa: Quản lý danh bạ điện thoại

ONE LOVE. ONE FUTURE.

# Description

- Viết chương trình quản lý danh bạ điện thoại di động. Mỗi liên lạc trong danh bạ chứa thông tin về họ tên, số điện thoại và email.
- Chương trình cần sử dụng một danh sách liên kết đơn để lưu trữ và quản lý các liên lạc với một số hàm sau:
  - Thêm một liên lạc (phần tử) mới vào đầu danh sách.
  - Thêm một liên lạc mới vào sau số liên lạc hiện tại trong danh sách
  - Hiển thị nội dung danh bạ (danh sách)
  - Xóa một số liên lạc: ở đầu danh sách; ở vị trí hiện tại,..
  - Đảo ngược danh sách – Giải phóng bộ nhớ cấp phát cho danh sách.



# Khai báo kiểu cho dữ liệu INFO lưu trữ trong mỗi phần tử

- Có thể tự định nghĩa kiểu dữ liệu mới đại diện cho một liên lạc được lưu trữ trong một nút của danh sách.
  - Thường dùng struct và typedef

```
typedef struct contact_t {  
    char name[20];  
    char tel[11];  
    char email[25];  
} contact; // contact is the type for INFO field
```



# Định nghĩa kiểu một phần tử

```
struct list_el {  
    contact el;  
    struct list_el *next;  
};  
typedef struct list_el node;
```

- “next” là biến con trỏ trỏ tới nút tiếp theo..
- “el” là tên trường đại diện cho thông tin contact.

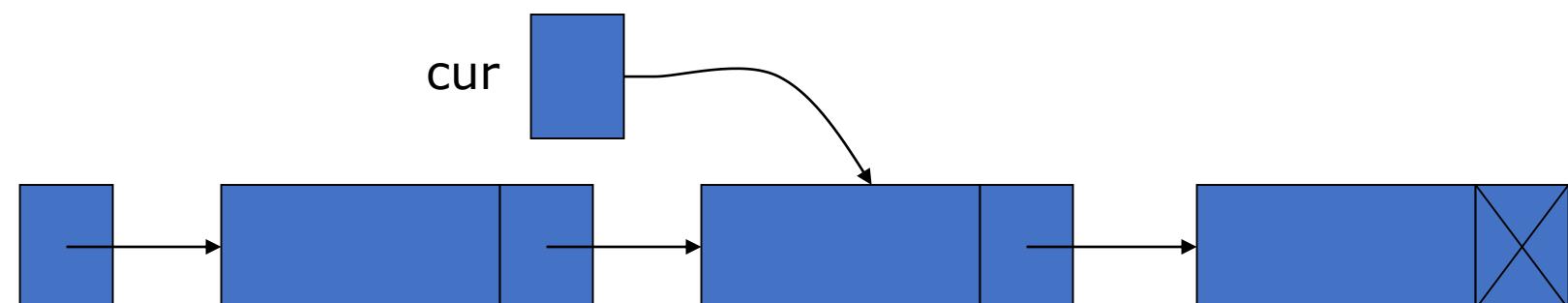


# Khai báo các con trỏ quan trọng – đại diện danh sách liên kết

- Con trỏ **root** (hoặc head) trỏ tới đầu danh sách.
  - Được sử dụng để truy cập tới danh sách, đại diện cho danh sách.
- Con trỏ **cur**: trỏ tới phần tử (nút) hiện hành – chứa dữ liệu vừa được thao tác.
- Con trỏ **prev**: trỏ tới phần tử trước phần tử trỏ bởi cur (không bắt buộc)

`node *root, *cur;`

`node *prev; /* in case you used prev */`



# Xây dựng hàm: Cấp phát bộ nhớ cho một nút (phiên bản draft)

```
node* makeNewNode () {  
    node* new = (node*) malloc(sizeof(node)) ;  
    strcpy( (new->el) .name , "Tran Van Thanh") ;  
    .... // similar statement for other contact fields  
    new->next =NULL;  
    return new;  
}
```

- **Hàm trên cấp phát bộ nhớ và khởi tạo giá trị dữ liệu cho một nút nhưng chưa thêm nó vào danh sách.**
- **Hạn chế:** thiếu tính linh hoạt do khởi tạo trực tiếp trong mã nguồn, khó sử dụng lại.



# Xây dựng hàm: Cấp phát bộ nhớ cho một nút

- Cải tiến hàm makeNewNode
  - nhận dữ liệu sẽ lưu trữ dưới dạng đối số → cấp phát bộ nhớ - lưu dữ liệu và trả về con trỏ trả về nút.
  - tái sử dụng tốt hơn, ví dụ trong các thao tác đọc nhiều dữ liệu từ file và khởi tạo danh sách

```
node* makeNewNode (contact ct) {  
    node* new = (node*) malloc (sizeof (node)) ;  
    new->el= ct;  
    new->next = NULL;  
    return new;  
}
```



# Đọc dữ liệu đầu vào cho một nút

```
contact readNode() {  
    contact tmp;  
    printf("Input the full name:");  
    gets(tmp.name);  
    ...  
    return tmp;  
}
```



# Hiển thị thông tin về một nút

```
void displayNode(node* p){  
    /* display name, tel, email in columns */  
}  
  
void displayInfo(contact ct){  
    /* display name, tel, email in columns */  
  
}
```

- Các hàm (read node, display node) không là thành phần của cấu trúc dữ liệu do thay đổi theo bài toán cụ thể, tuy nhiên cần thiết phải xây dựng.



# Hiển thị thông tin về một nút

```
void displayNode(node* p) {  
    if (p==NULL) {printf("NULL Pointer error.\n"); return; }  
    contact tmp = p->el;  
    printf("%-20s\t%-15s\t%-25s%-p\n", tmp.name, tmp.tel,  
        tmp.email, p->next);  
}  
  
//driver main function  
void main() {  
    contact tmp = readNode();  
    root = makeNewNode(tmp);  
    displayNode(root);  
}
```



# Thêm một nút mới vào đầu danh sách

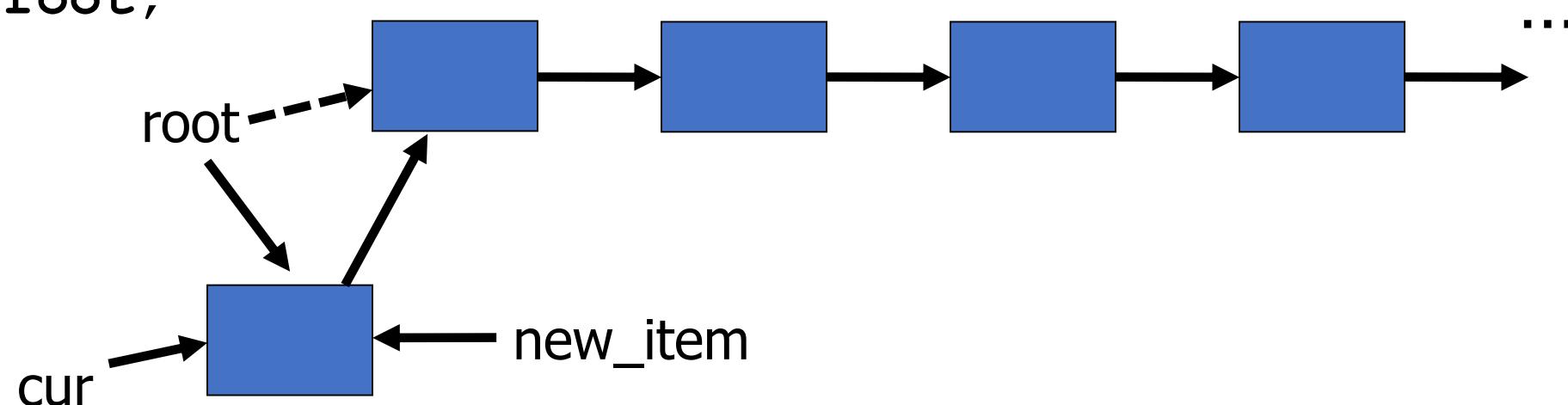
- Gợi ý về thuật toán:

```
create new_item
```

```
new->next = root;
```

```
root = new;
```

```
cur= root;
```



# Thêm một nút mới vào đầu danh sách: mã nguồn tham khảo

```
void insertAtHead(contact ct) {  
    node* new = makeNewNode(ct);  
    new->next = root;  
    root = new;  
    cur = root;  
}  
  
void main() {  
    contact tmp; int i;  
    for(i=0;i<2;i++) {  
        tmp = readNode(); insertAtHead(tmp);  
        displayNode(root);  
    }  
}
```

Name:Cao Dung	Phone number:030035888	Email:caodung@gmail.com	caodung@gmail.com	0000000000000000
Cao Dung	030035888			
Name:Ha Ho	Phone number:0912221122	Email:haho@gmail.com	haho@gmail.com	0000000000026A60
Ha Ho	0912221122			



# Thêm nút mới vào sau nút hiện hành

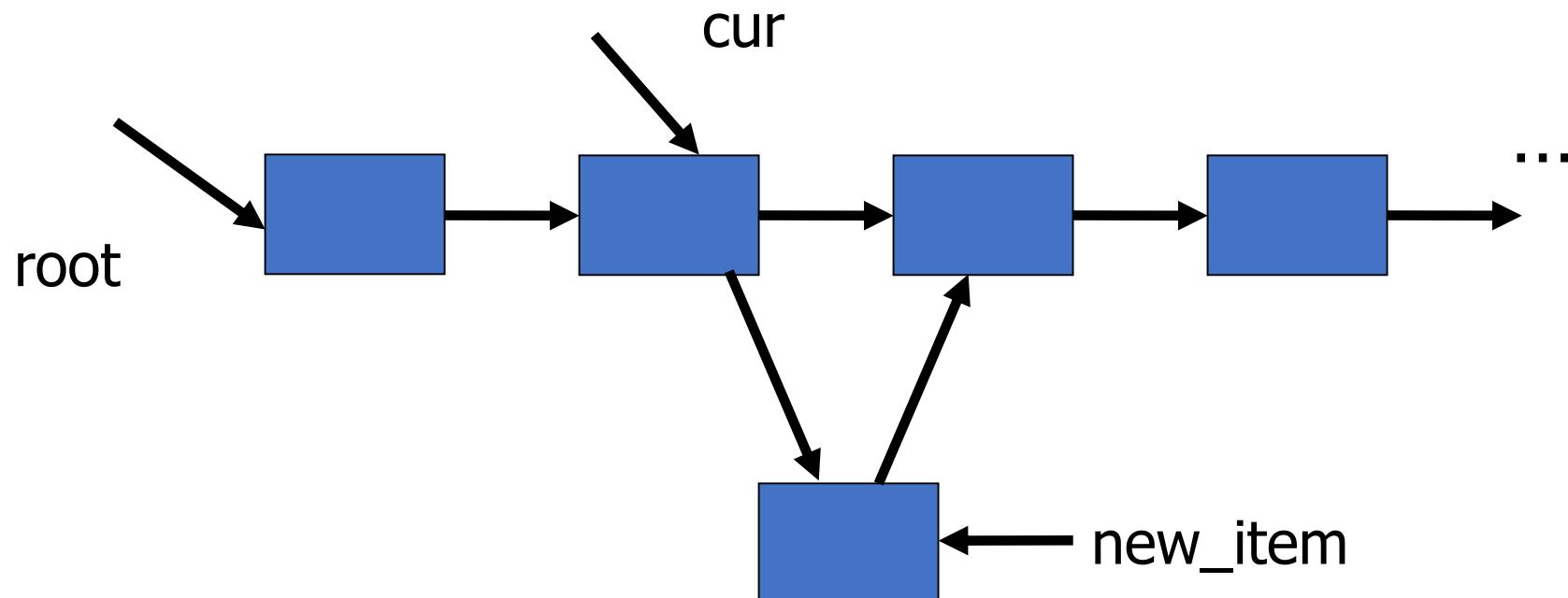
- Logic - Mã giả

```
create new_item
```

```
new->next = cur->next;
```

```
cur->next = new;
```

```
cur= cur->next;
```

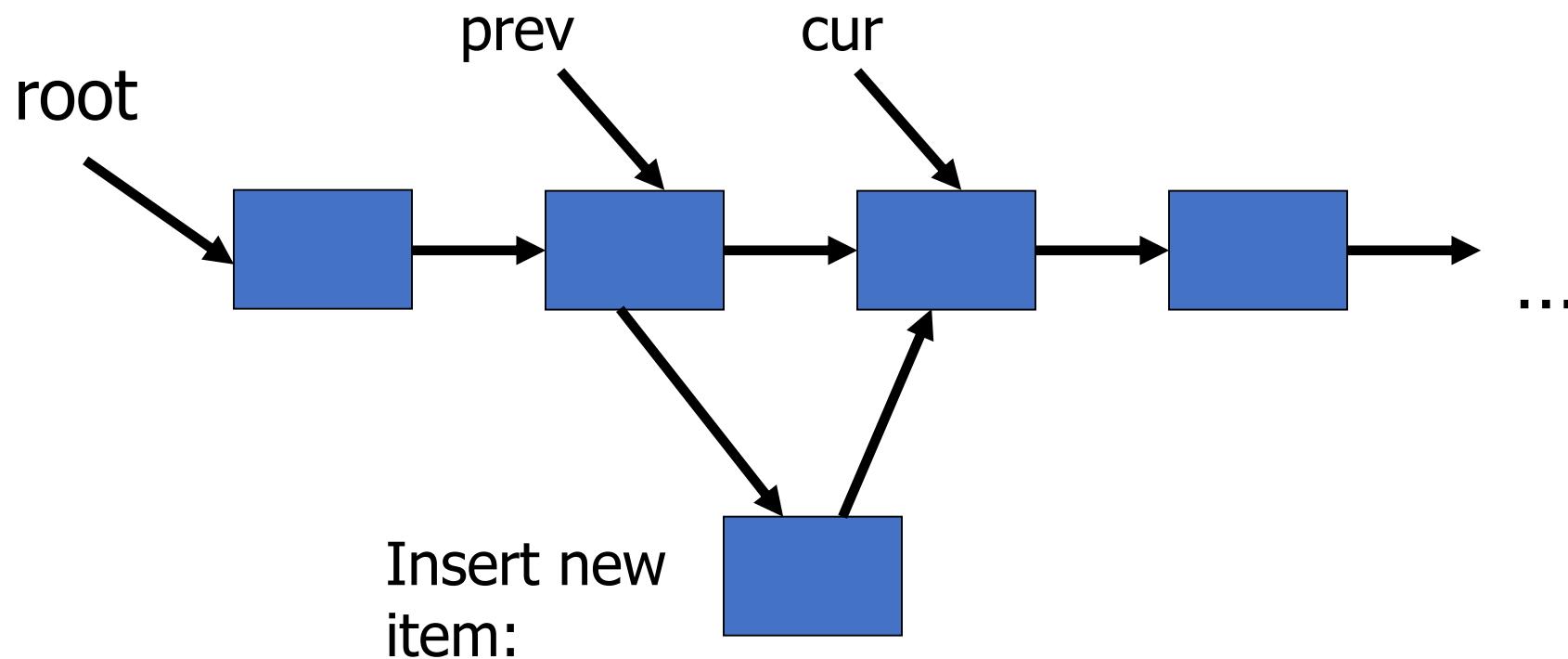


# Thêm nút mới vào sau nút hiện hành (mã nguồn tham khảo)

```
new = makeNewNode(ct); // ct is a contact data
if ( root == NULL ) /* if there is no element */
    root = new;
    cur = root;
}
else if (cur == NULL) return;
else {
    new->next=cur->next;
    cur->next = new;
    /* prev=cur; */
    cur = cur->next;
}
```



# Thêm nút mới vào trước nút hiện hành



# Thêm nút mới vào trước nút hiện hành (mã nguồn tham khảo)

```
void insertBeforeCurrent(contact e) {  
    node_addr * new = makeNewNode(e);  
    if ( root == NULL ) { /* if there is no element */  
        root = new;  
        cur = root;  
        prev = NULL;  
    } else {  
        new->next=cur;  
        if (cur==root) /* if cur pointed to first element */  
            root = new; /* nut moi them vao tro thanh dau danh sach */  
        }  
        else prev->next = new; // assume prev pointer always point to the previous node  
        cur = new;  
    }  
}
```



# Xác định lại prev

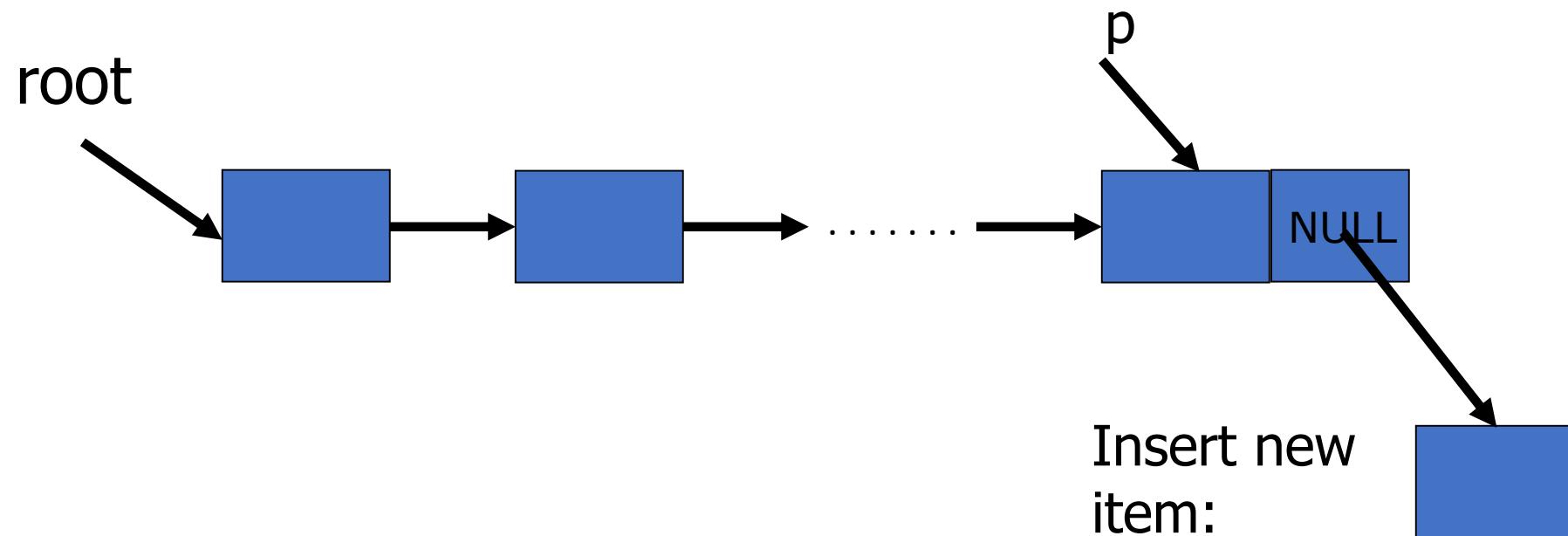
- Nếu chương trình không luôn đồng bộ giá trị của prev với cur thì cần xác định giá trị prev

```
/* determine prev */  
tmp = root;  
while (tmp!=NULL && tmp->next!=cur && cur !=NULL)  
    tmp=tmp->next;  
prev = tmp;
```



# Thêm nút mới vào cuối danh sách

- Cần xác định con trỏ  $p$  trỏ tới nút cuối (trường next là NULL)



# Thêm nút mới vào cuối danh sách : lời giải tham khảo

```
void insertAtTail(contact ct) {  
    node* new = makeNewNode(ct);  
    if (root == NULL) { root = new; cur = new; prev = NULL; return;  
    }  
    node* p = root;  
    while (p->next !=NULL) p=p->next;  
    p->next = new;  
    cur = new; prev = p;  
}  
void main() {  
    contact tmp; int i;  
    for(i=0;i<2;i++){  
        tmp = readNode(); insertAtTail(tmp);  
        displayNode(root);  
    }  
}
```



# Thêm nút mới vào cuối danh sách : phiên bản sử dụng đệ quy

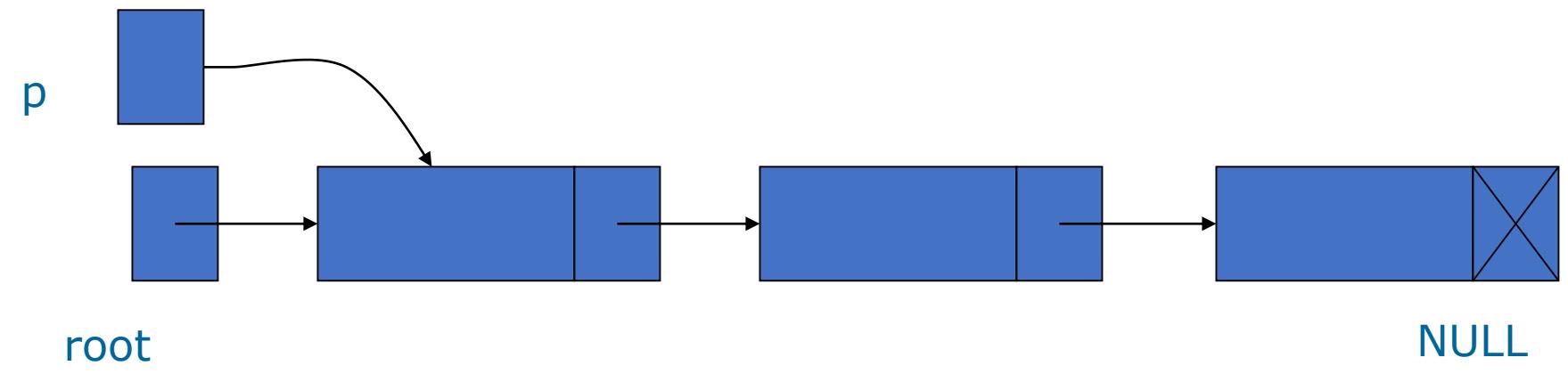
```
node* insertLastRecursive(node* root, contact ct) {
    if(root == NULL) {
        return makeNewNode(ct);
    }
    root->next = insertLastRecursive(root->next, ct);
    return root;
}
void main() {
    contact tmp; int i;
    for(i=0;i<2;i++) {
        tmp = readNode(); root = insertLastRecursive(tmp);
        displayNode(root);
    }
}
```



# Duyệt danh sách liên kết

- Cần thiết trong các tác vụ như hiển thị nội dung hay sao chép nội dung toàn bộ danh sách.
- Quá trình kết thúc khi duyệt xong phần tử cuối.

```
void traversingList(node *root) {  
    node * p;  
    for ( p = root; p != NULL; p = p->next )  
        displayNode(p);  
}
```



# Sử dụng các hàm để tạo và hiển thị danh sách

- Sử dụng vòng lặp để - đọc dữ liệu và thêm vào danh sách. Sau đó hiển thị.

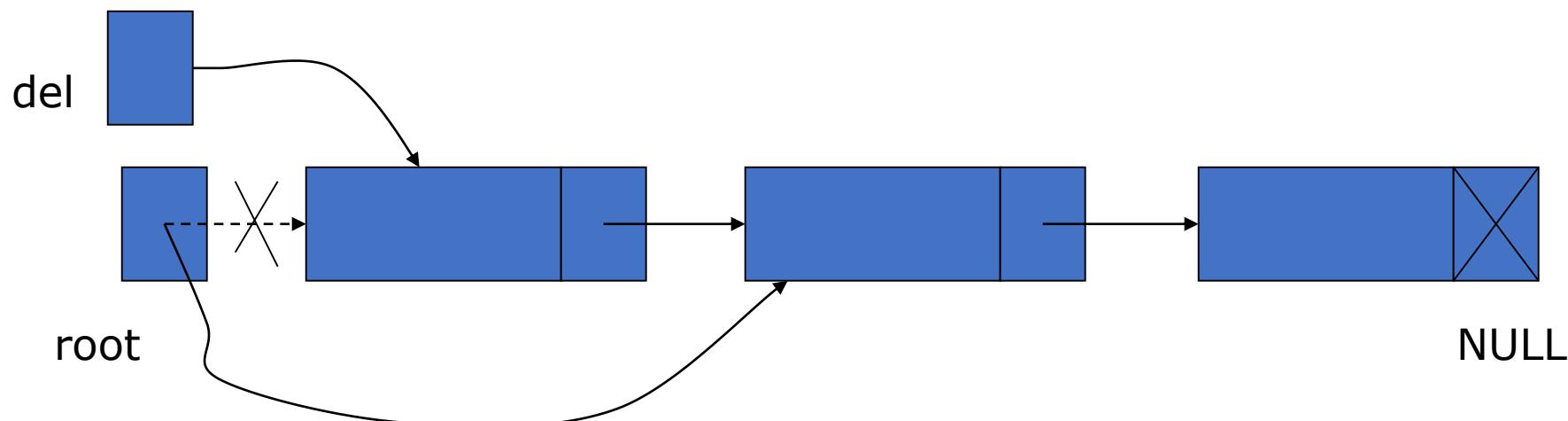
```
void main() {  
    n=5;  
    while (n) {  
        node tmp = readNode();  
        insertAtHead(tmp);  
        // or insertAfter..  
        n--;  
    }  
    traversingList(root);  
}
```

Name:Cao Dung	Phone number:035778758	Email:caodung@gmail.com	Cao Dung	035778758	caodung@gmail.com	0000000000000000
Name:Hoang Anh	Phone number:0764676365	Email:hoanganh@vtc.vn	Hoang Anh	0764676365	hoanganh@vtc.vn	0000000000346A60
Testing for the insertion after current position of pointer.	Before insert..		Hoang Anh	0764676365	hoanganh@vtc.vn	0000000000346A60
Hoang Anh	0764676365		Cao Dung	035778758	caodung@gmail.com	0000000000000000
Name:Bui Viet	Phone number:0834787444	Email:buiviet@fpt.vn	Hoang Anh	0764676365	hoanganh@vtc.vn	0000000000346A60
Bui Viet	0834787444		Cao Dung	035778758	caodung@gmail.com	0000000000346B00
			Bui Viet	0834787444	buiviet@fpt.vn	0000000000000000
Name:Cao Dung	Phone number:0931324434	Email:caodung@hust.edu.vn	Cao Dung	0931324434	caodung@hust.edu.vn	0000000000000000
Name:Bui Ha Anh	Phone number:0938734764	Email:buiha@fsoft.vn	Bui Ha Anh	0938734764	buiha@fsoft.vn	0000000000396A60
Bui Ha Anh	0938734764		Testing for the insertion after current position of pointer.	Before insert..		
Bui Ha Anh	0938734764		Bui Ha Anh	0938734764	buiha@fsoft.vn	0000000000396A60
Cao Dung	0931324434		Cao Dung	0931324434	caodung@hust.edu.vn	0000000000000000
Name:Nguyen Linh	Phone number:0123328772	Email:linhalex@hapt.com	Nguyen Linh	0123328772	linhalex@hapt.com	0000000000000000
Bui Ha Anh	0938734764		Testing for the insertion before current position of pointer.	Before insert..		
Cao Dung	0931324434		Name:Vo Hung	0887387843		
Nguyen Linh	0123328772		Phone number:0887387843	Email:vohung@gmail.com		
			Bui Ha Anh	0938734764	buiha@fsoft.vn	0000000000396A60
			Cao Dung	0931324434	caodung@hust.edu.vn	0000000000396B50
			Vo Hung	0887387843	vohung@gmail.com	0000000000396B00
			Nguyen Linh	0123328772	linhalex@hapt.com	0000000000000000



# Xóa phần tử đầu danh sách

- Viết hàm xóa nút đầu tiên trong danh sách
- Logic:  
`del=root; root = del->next; free(del);`
- Cần **thay đổi** để **root** trở tới nút kế tiếp của nút đầu danh sách cũ.



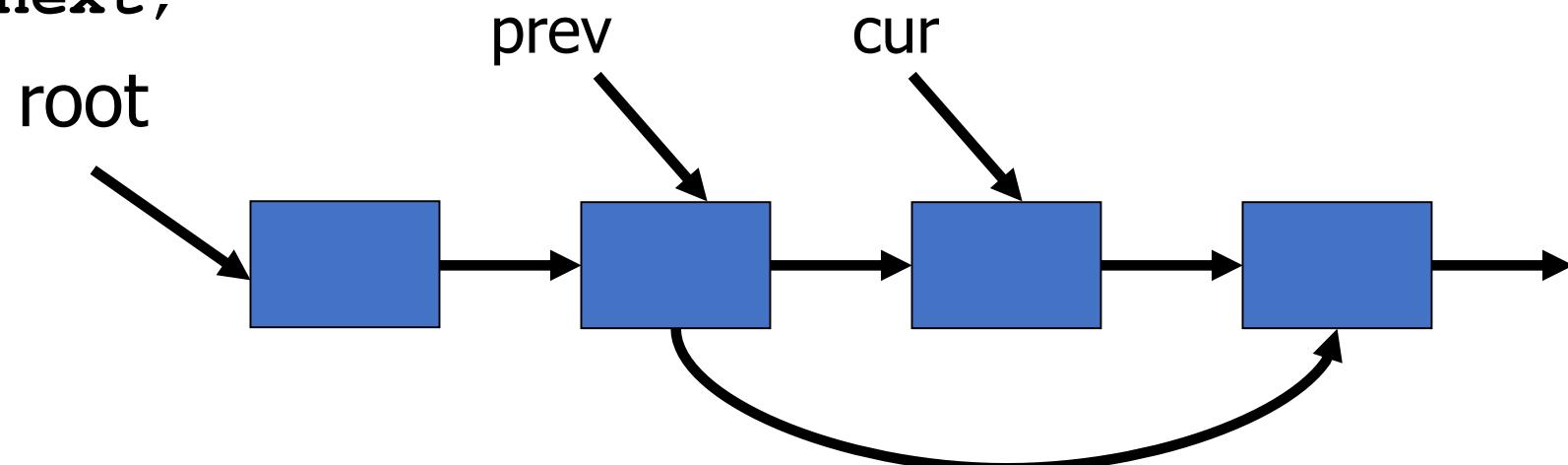
# Xóa phần tử đầu danh sách

```
void deleteFirstElement() {  
    node* del = root;  
    if (del == NULL) return;  
    root = del->next;  
    free(del);  
    cur = root;  
    prev = NULL; //update prev - cur  
}
```



# Xóa phần tử ở giữa danh sách

- Xây dựng hàm xóa nút trỏ bởi con trỏ cur (nút hiện hành):  
***deleteCurrentElement***
- Logic: sử dụng con trỏ prev trỏ tới nút đứng trước nút cần xóa.  
`prev->next = cur->next;`  
`free(cur);`  
`cur = prev->next;`



# Mã nguồn tham khảo: Xóa nút hiện hành

```
void deleteCurrentElement() {  
    if (cur==NULL) return;  
    if (cur==root) deleteFirstElement();  
    else {  
        prev->next = cur->next;  
        free(cur);  
        cur = prev->next; // or cur = root;  
    }  
}
```



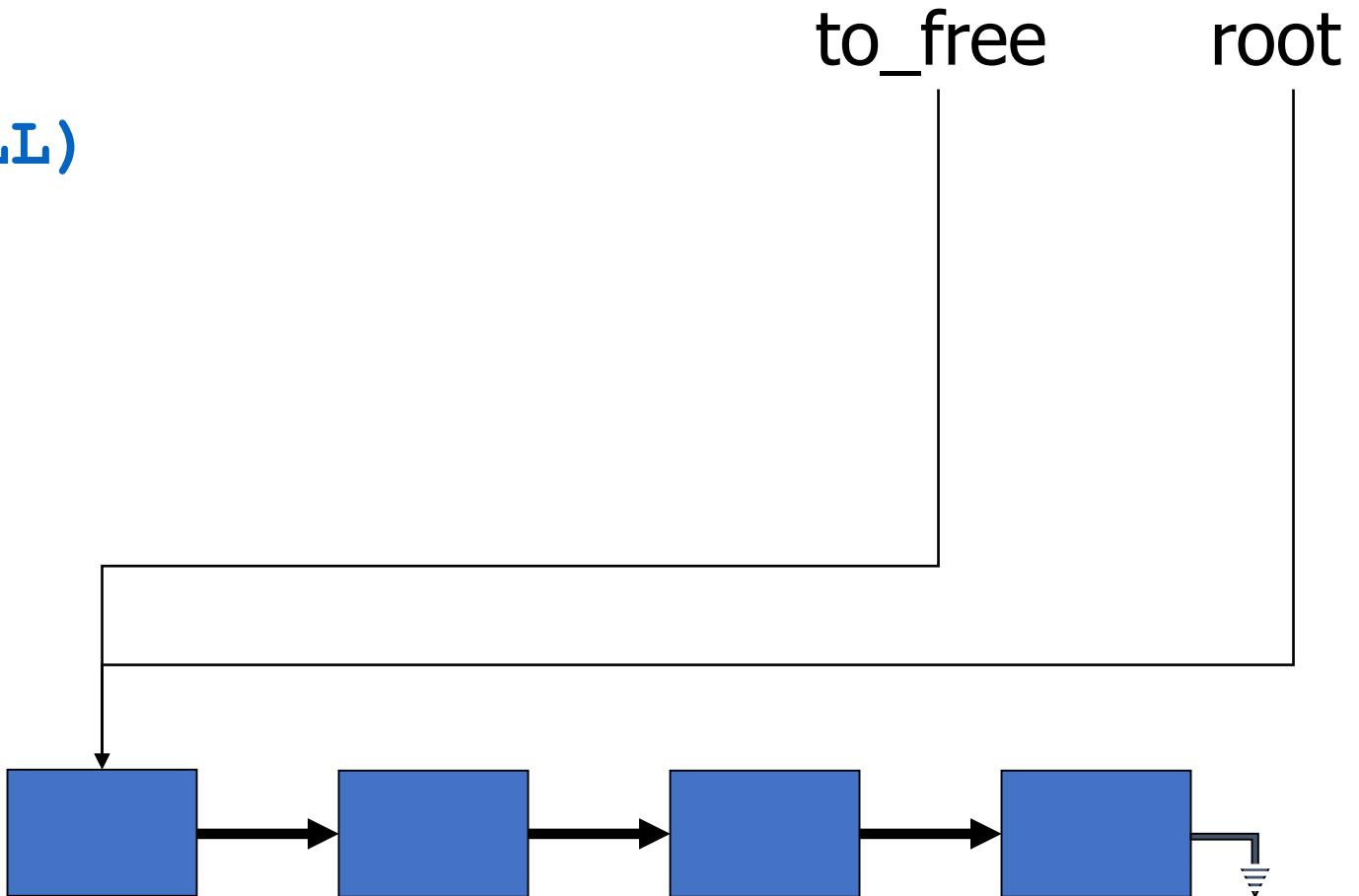
# Xóa phần tử với thông tin liên lạc cụ thể

```
Node* removeNodeRecursive(Node* root, contact e) {  
    if(root == NULL) return NULL;  
    if(root->el == e){  
        Node* tmp = root; root = root->next; free(tmp);  
        return root;  
    }  
    root->next = removeNodeRecursive(root->next, e);  
    return root;  
}
```



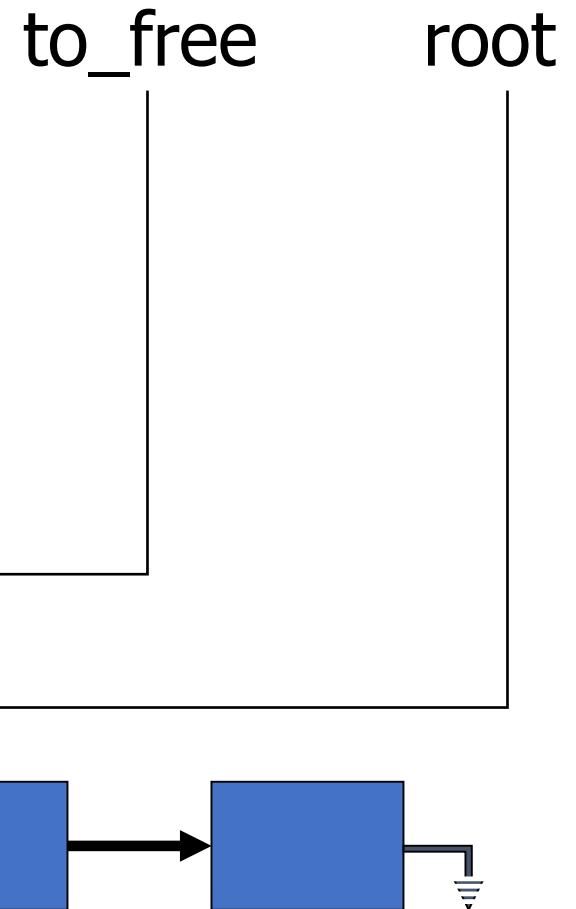
# Giải phóng (bô nhớ) danh sách

```
to_free = root ;  
while (to_free != NULL)  
{  
    root = root->next;  
    free(to_free);  
    to_free = root;  
}
```



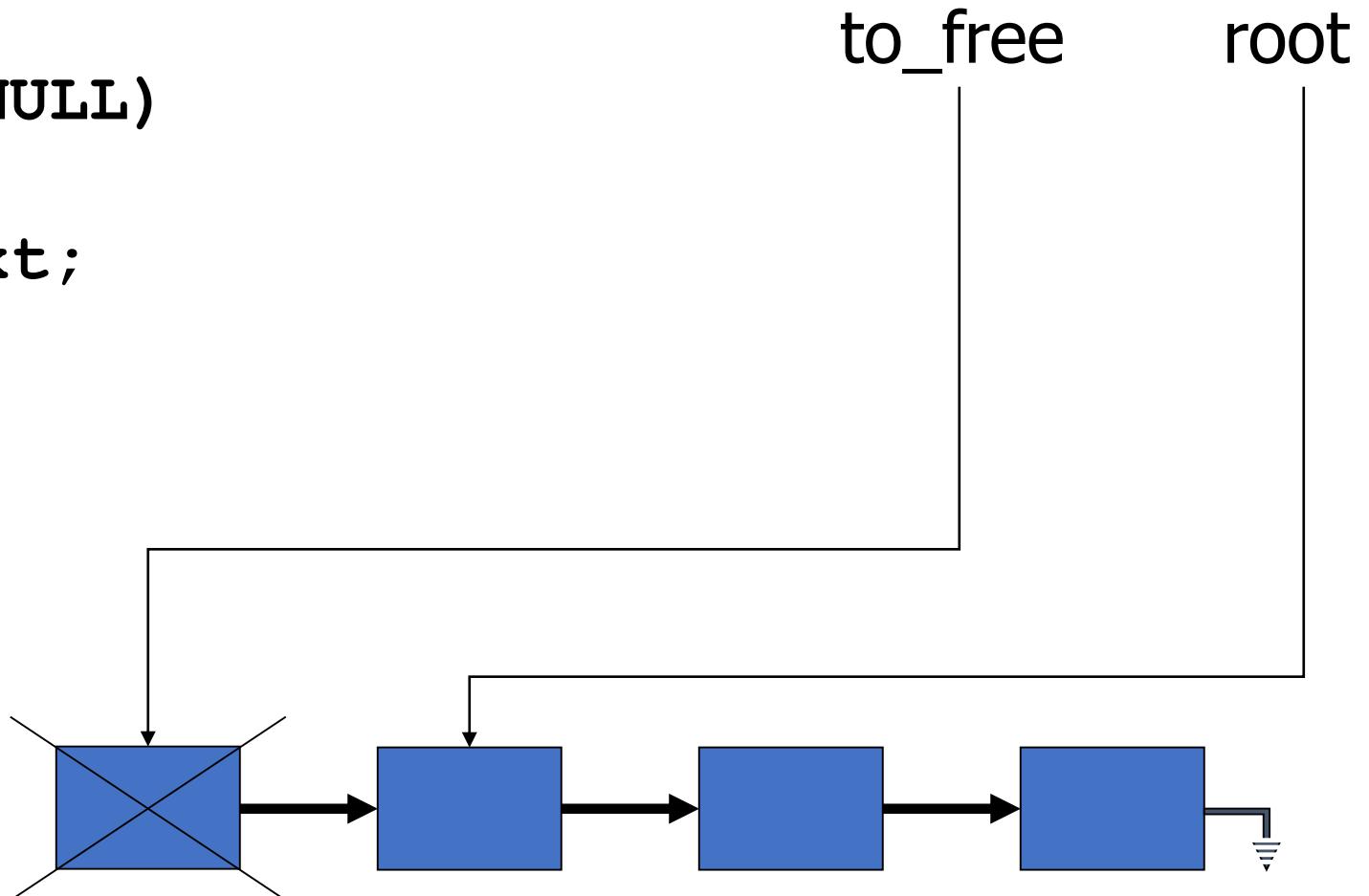
# Giải phóng (bộ nhớ) danh sách

```
to_free = root ;  
while (to_free != NULL)  
{  
    root = root->next;  
    free(to_free);  
    to_free = root;  
}
```



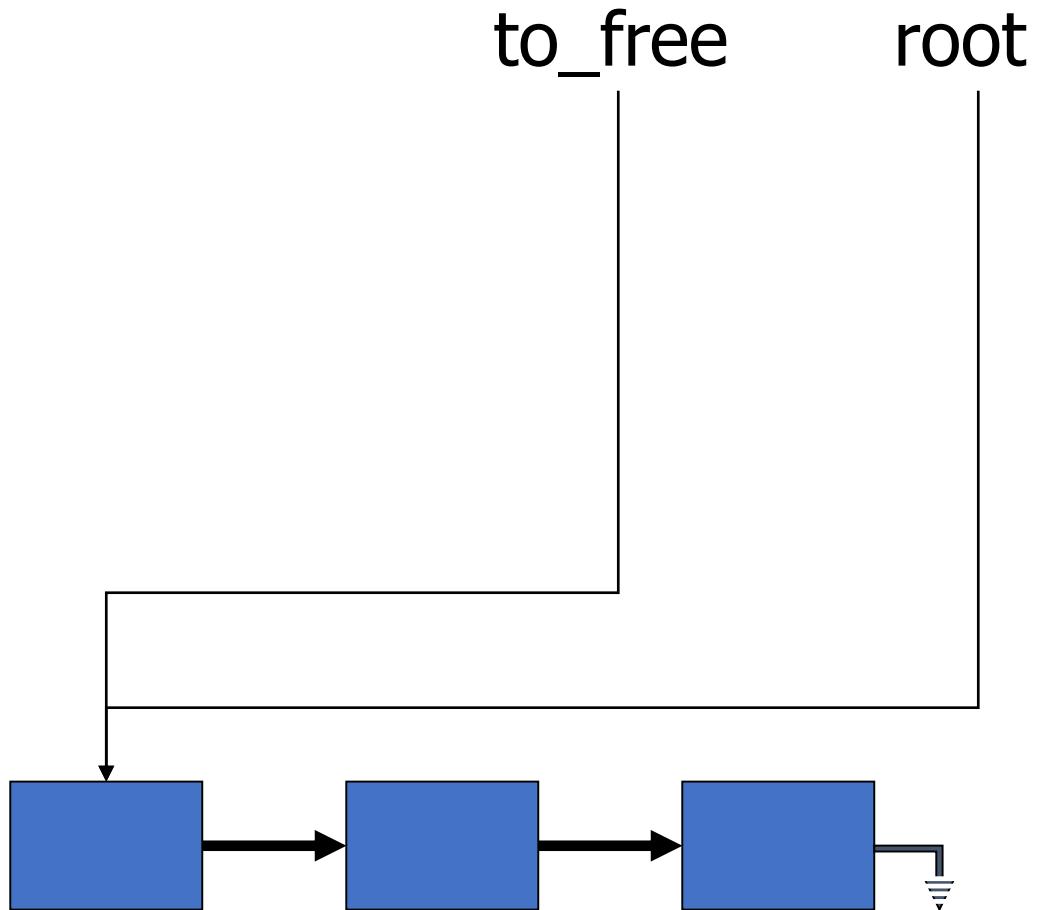
# Giải phóng (bô nhớ) danh sách

```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



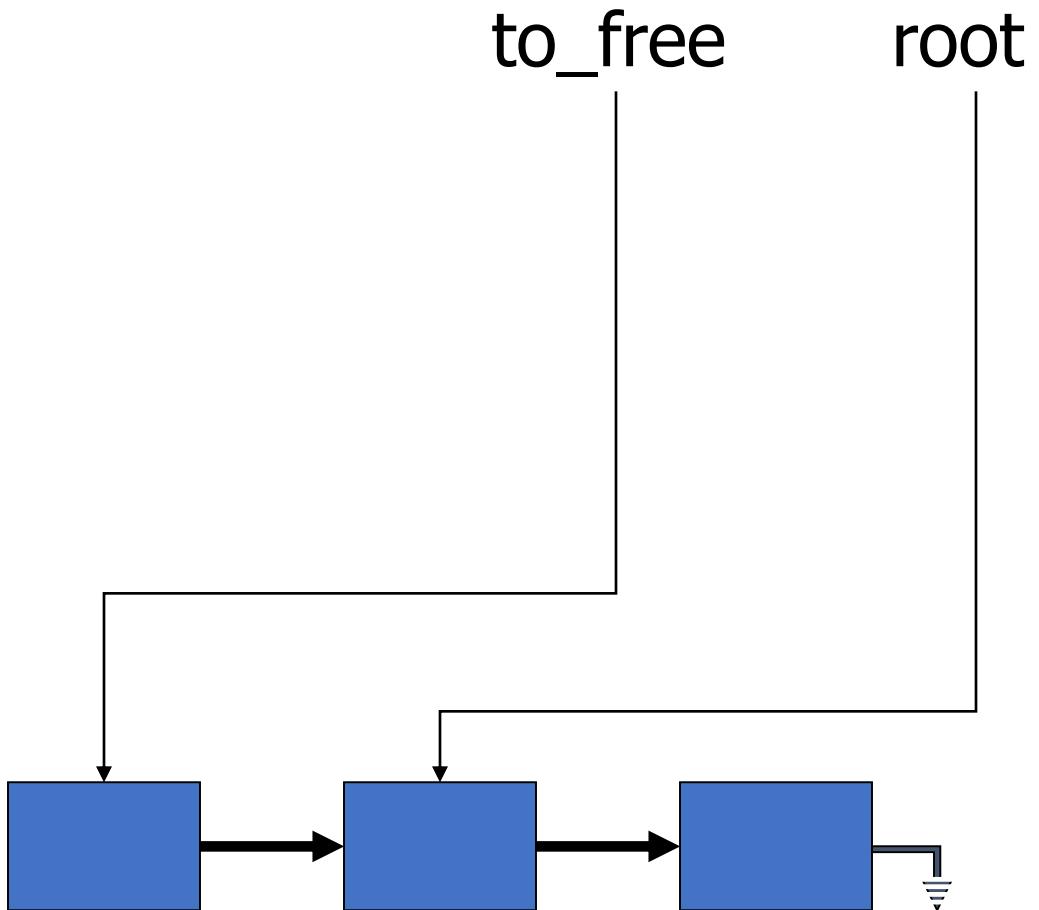
# Giải phóng (bộ nhớ) danh sách

```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



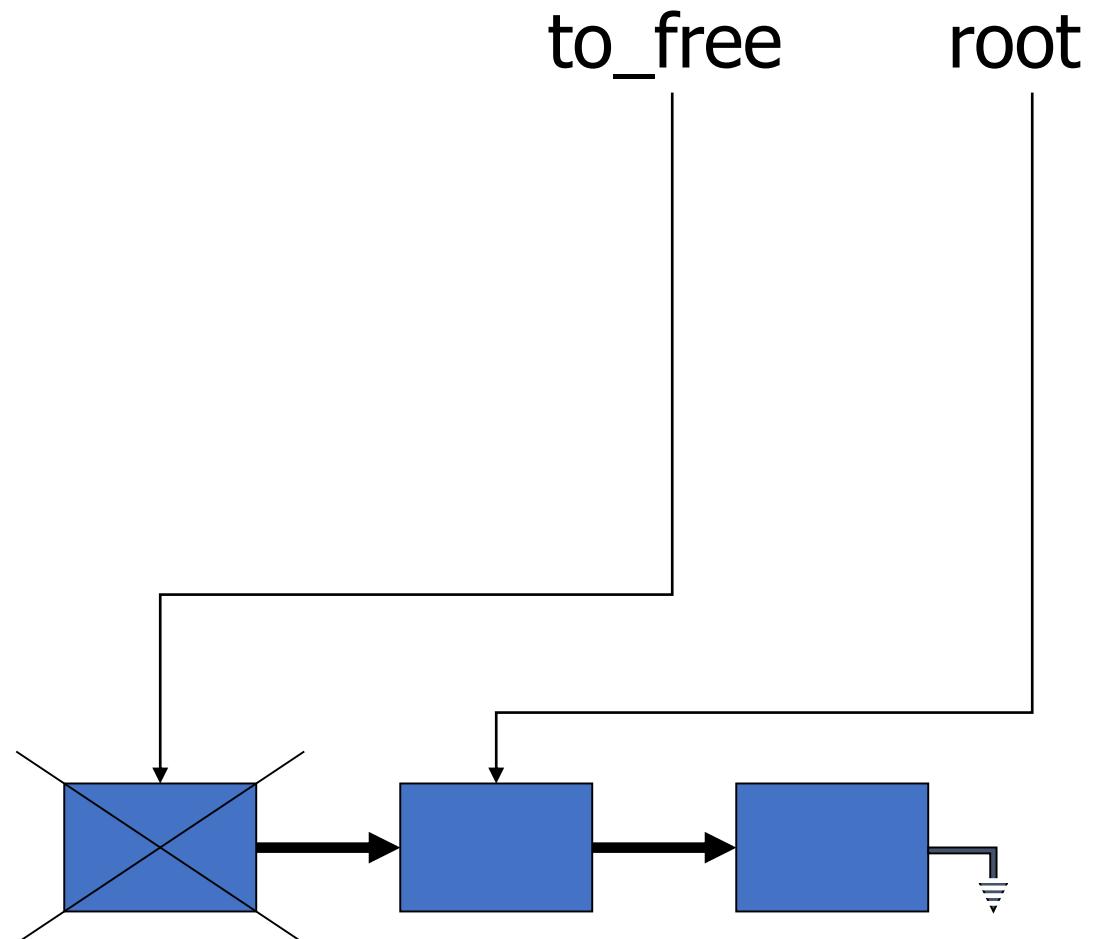
# Giải phóng (bộ nhớ) danh sách

```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



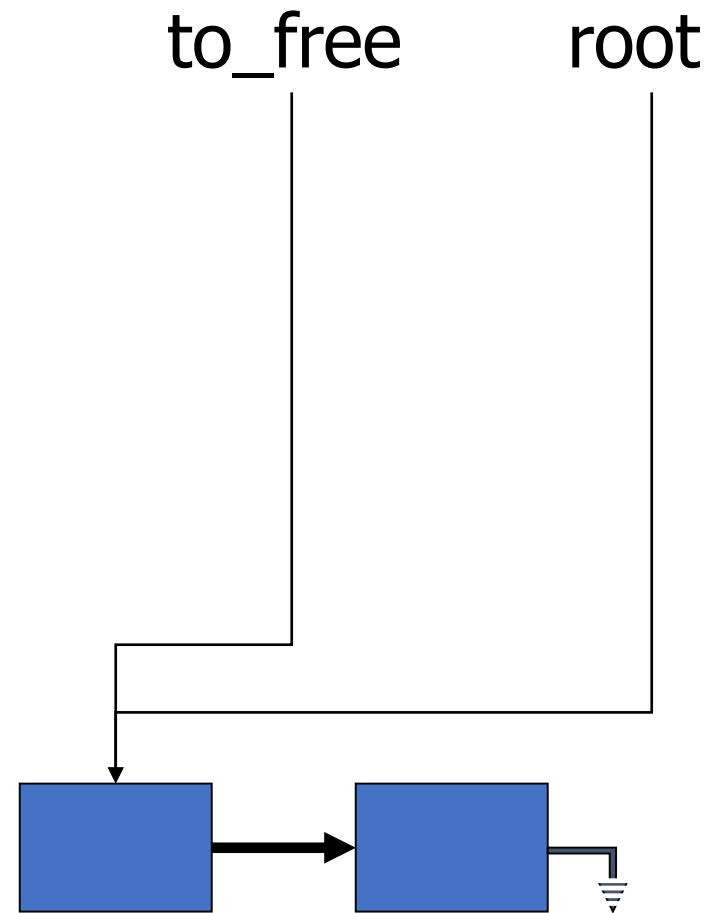
# Giải phóng (bô nhớ) danh sách

```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



# Giải phóng (bộ nhớ) danh sách

```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```

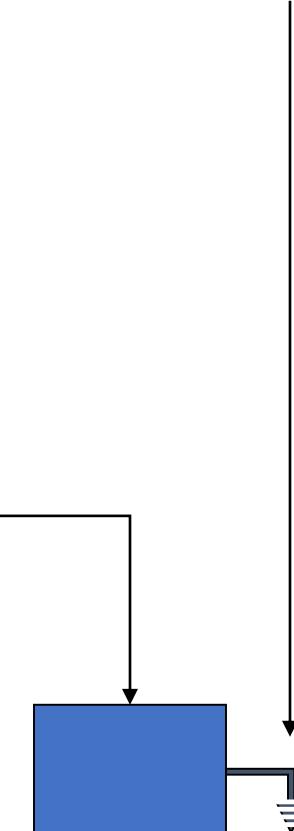


# Giải phóng (bộ nhớ) danh sách

- After some iteration ...

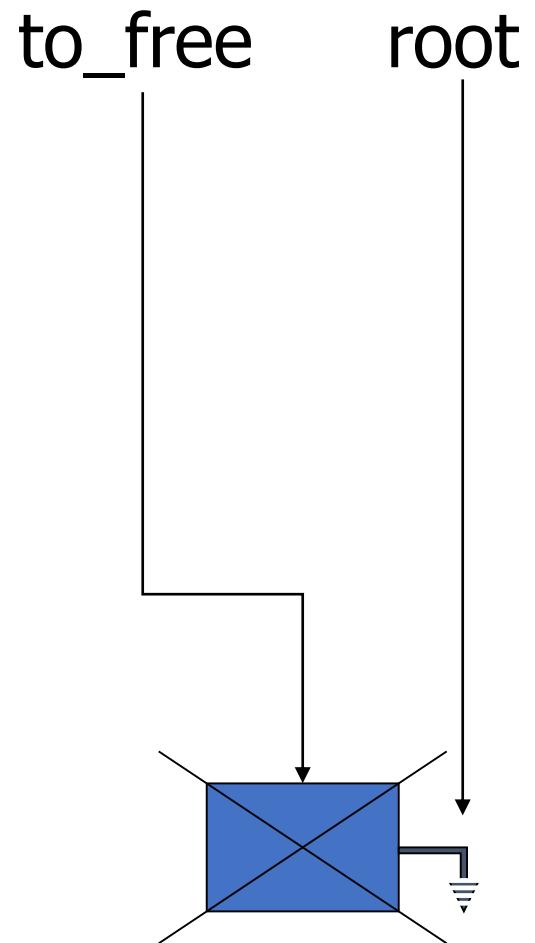
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```

to\_free      root



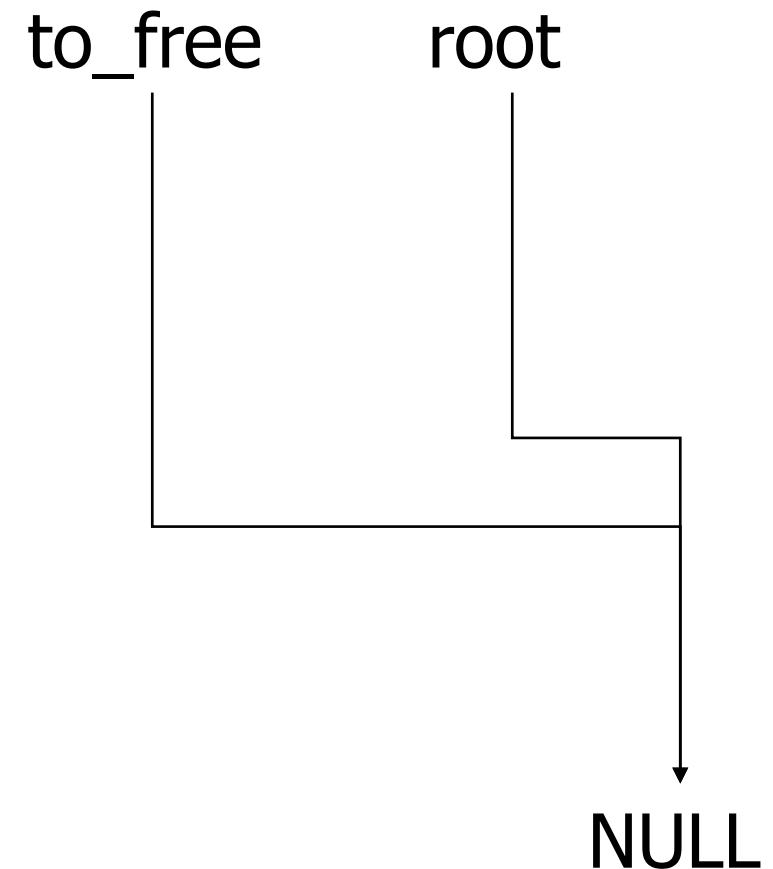
# Giải phóng (bô nhớ) danh sách

```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



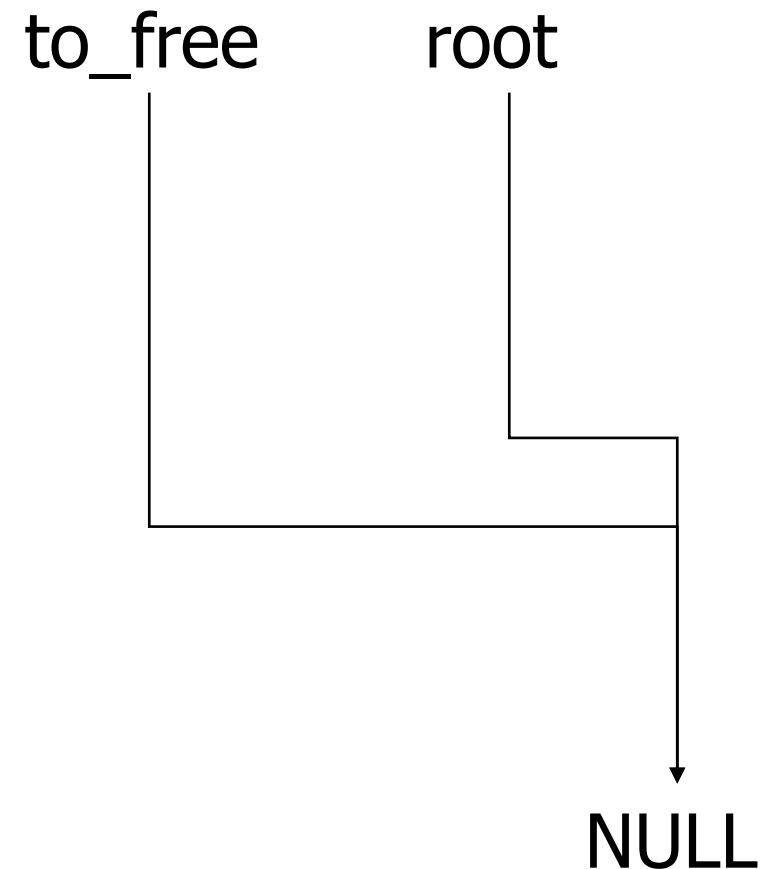
# Giải phóng (bô nhớ) danh sách

```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



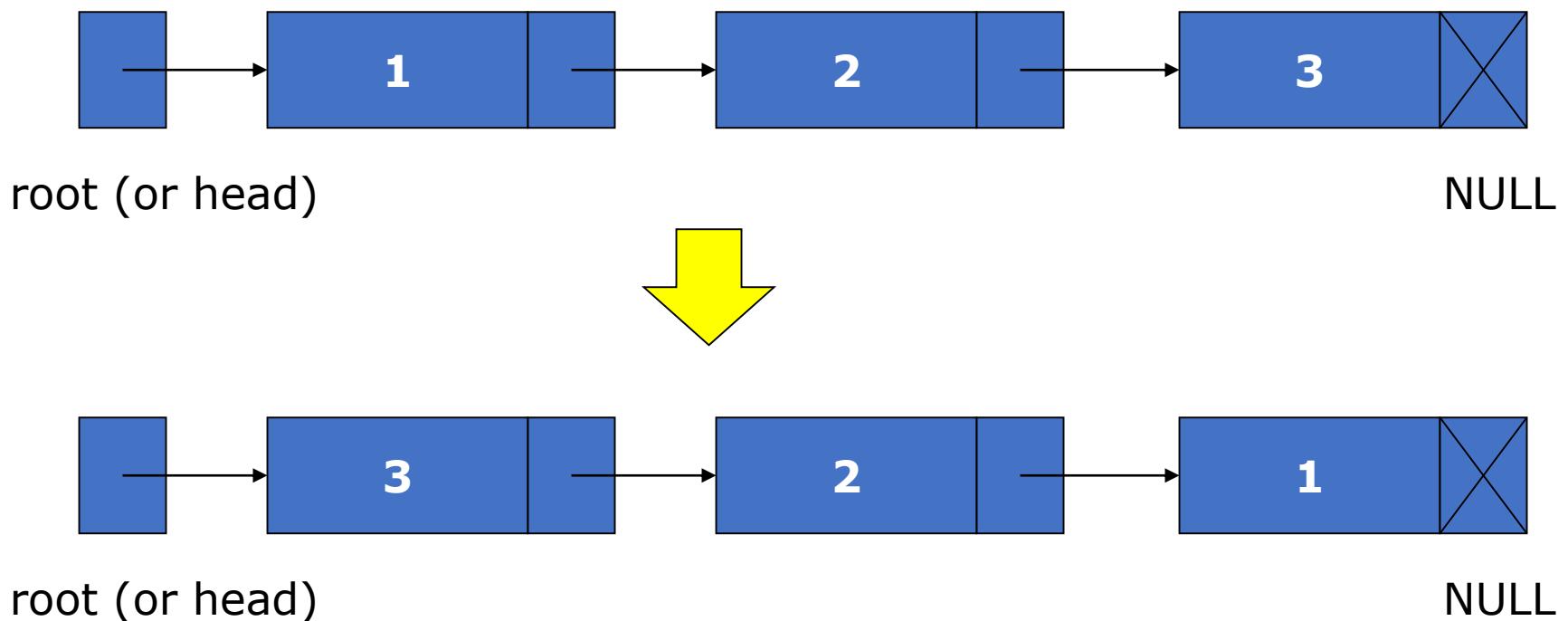
# Giải phóng (bộ nhớ) danh sách

```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



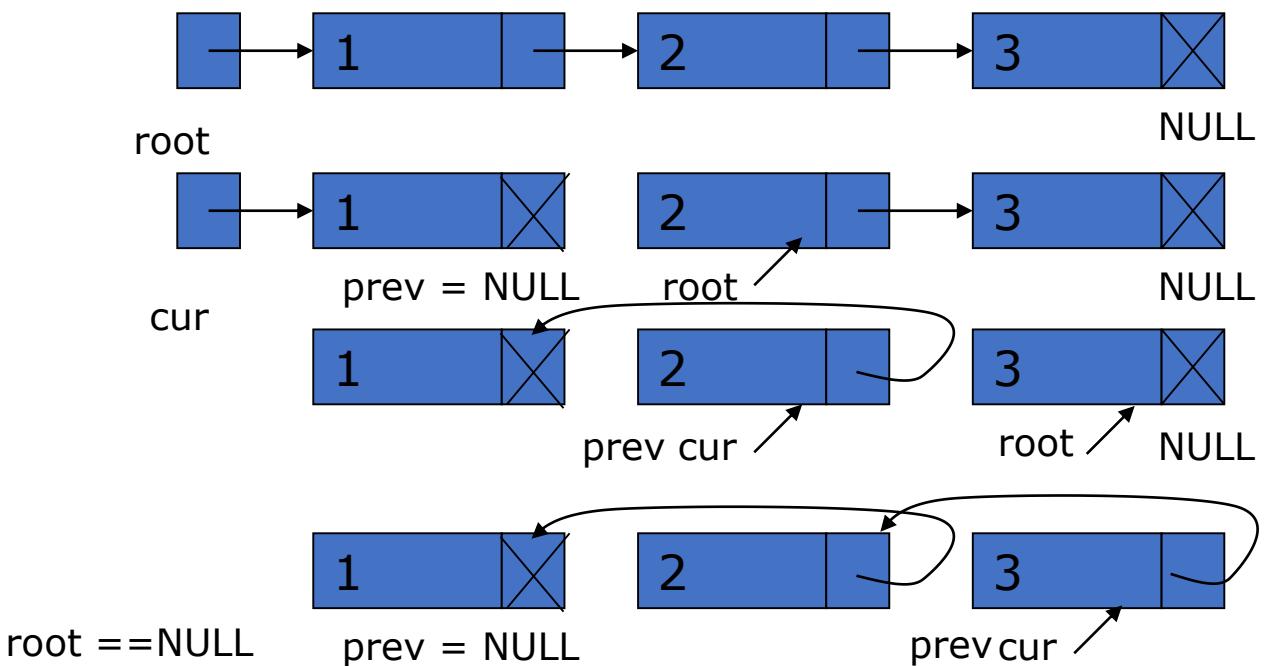
# Đảo ngược danh sách

- Viết hàm đảo ngược danh sách (không tạo ra danh sách mới).



# Đảo ngược danh sách: Lời giải tham khảo

```
node* list_reverse (node* root)
{
    node *cur, *prev;
    cur = prev = NULL;
    while (root != NULL) {
        cur = root;
        root = root->next;
        cur->next = prev;
        prev = cur;
    }
    return prev;
}
```



# Đầu ra của chương trình

```
Phone number:0912211313
Email:haanh@gmail.com
Ha Anh      0912211313    haanh@gmail.com    0000000000000000
Name:Luu Vu
Phone number:0932323223
Email:luuvu@fpt.vn
Luu Vu      0932323223    luuvu@fpt.vn    0000000000566A90
Testing for the insertion after current position of pointer.Before insert..
Luu Vu      0932323223    luuvu@fpt.vn    0000000000566A90
Ha Anh      0912211313    haanh@gmail.com    0000000000000000
Name:Nguyen Quang Anh
Phone number:0921211221
Email:qa@vnpt.com
Luu Vu      0932323223    luuvu@fpt.vn    0000000000566A90
Ha Anh      0912211313    haanh@gmail.com    0000000000566B30
Nguyen Quang Anh 0921211221    qa@vnpt.com    0000000000000000
Testing for the insertion before current position of pointer.Before insert..
Name:Bui Long
Phone number:0112121122
Email:bui long@yahoo.com
Luu Vu      0932323223    luuvu@fpt.vn    0000000000566A90
Ha Anh      0912211313    haanh@gmail.com    0000000000566B80
Bui Long    0112121122    builong@yahoo.com  0000000000566B30
Nguyen Quang Anh 0921211221    qa@vnpt.com    0000000000000000
Testing for the deletion of the first element..
Ha Anh      0912211313    haanh@gmail.com    0000000000566B80
Bui Long    0112121122    builong@yahoo.com  0000000000566B30
Nguyen Quang Anh 0921211221    qa@vnpt.com    0000000000000000
Testing for the deletion of the middle element..
Bui Long    0112121122    builong@yahoo.com  0000000000566B30
Nguyen Quang Anh 0921211221    qa@vnpt.com    0000000000000000
Testing for the reverse list operation..
Nguyen Quang Anh 0921211221    qa@vnpt.com    0000000000566B80
Bui Long    0112121122    builong@yahoo.com  0000000000000000
```



# Nội dung chính

- Giới thiệu chung về danh sách liên kết
- Xây dựng cấu trúc dữ liệu danh sách liên kết đơn
- **Ứng dụng danh sách liên kết đơn trong một số bài toán cụ thể**



# Lab 1: Thao tác trên danh sách liên kết đơn

- Viết chương trình thực hiện công việc sau
  - Xây dựng danh sách liên kết với các khóa được cung cấp ban đầu là dãy  $a_1, a_2, \dots, a_n$ .
  - Thực hiện các thao tác trên danh sách:
    - Thêm 1 phần tử vào đầu, vào cuối danh sách
    - Thêm vào trước hay sau một phần tử đã xác định (bởi giá trị dữ liệu) trong danh sách
    - hoặc xóa một phần tử khỏi danh sách.
- Nộp và chạy chương trình trên hệ thống chấm chương trình tự động.



# Định dạng dữ liệu nhập xuất

- **Input**

- Dòng 1: nhập một số nguyên dương n ( $1 \leq n \leq 1000$ )
- Dòng 2: nhập dãy n số nguyên dương  $a_1, a_2, \dots, a_n$ .
- Các dòng tiếp theo lần lượt là các lệnh để thao tác (kết thúc bởi ký hiệu #) :
  - **addlast k**: thêm phần tử có key bằng k vào cuối danh sách (nếu k chưa tồn tại)
  - **addfirst k**: thêm phần tử có key bằng k vào đầu danh sách (nếu k chưa tồn tại)
  - **addafter u v**: thêm phần tử có key bằng u vào sau phần tử có key bằng v trên danh sách (nếu v đã tồn tại trên danh sách và u chưa tồn tại)
  - **addbefore u v**: thêm phần tử có key bằng u vào trước phần tử có key bằng v trên danh sách (nếu v đã tồn tại trên danh sách và u chưa tồn tại)
  - **remove k**: loại bỏ phần tử có key bằng k khỏi danh sách
  - **reverse**: đảo ngược thứ tự các phần tử của danh sách (không được cấp phát mới các phần tử)
- **Output**: ghi ra dãy khóa của danh sách thu được sau 1 chuỗi các lệnh thao tác đã cho



# Ví dụ về dữ liệu nhập xuất

## Input

5

5 4 3 2 1

addlast 3

addlast 10

addfirst 1

addafter 10 4

remove 1

#

## Output

5 4 3 2 10



# Lời giải tham khảo

```
#include <stdio.h>
#include <string.h>
typedef int elementtype;
struct node_t{
    elementtype element;
    struct Node* next;// point to the next element of the current
element
};
typedef struct node_t Node;
Node* root = NULL;
Node* cur = NULL;
Node* prev = NULL;
```



# Lời giải tham khảo

```
Node* makeNewNode(elementtype e) {  
    Node* new = (Node*) malloc(sizeof(Node)) ;  
    new->element=e;  
    new->next =NULL;  
    return new;  
}  
  
Node* find(Node* root, elementtype e) {  
    Node* p;  
    for(p = root; p != NULL; p = p->next) {  
        if(p->element == e) return p;  
    }  
    return NULL;  
}
```



# Lời giải tham khảo

```
void insertAtTail(elementtype e) {  
    Node* new = makeNewNode(e);  
    if (root == NULL) { root = new; cur = new; prev = NULL;  
    return;  
}  
    Node* p = root;  
    while (p->next !=NULL) p=p->next;  
    p->next = new;  
    cur = new; prev = p;  
}
```



# Lời giải tham khảo

```
Node* insertLastRecursive(Node* root, elementtype e) {  
    if(root == NULL) {  
        return makeNewNode(e);  
    }  
    root->next = insertLastRecursive(root->next, e);  
    return root;  
}  
  
void insertAtHead(elementtype e) {  
    Node* new = makeNewNode(e);  
    new->next = root;  
    root = new;  
    cur = root;  
}
```



# Lời giải tham khảo

```
Node* removeNodeRecursive(Node* root, elementtype e) {
    if(root == NULL) return NULL;
    if(root->element == e) {
        Node* tmp = root; root = root->next; free(tmp); return root;
    }
    root->next = removeNodeRecursive(root->next, e);
    return root;
}

void freeList() {
    Node* to_free=root;
    while (to_free != NULL) {
        root = root->next; free(to_free);      to_free = root;
    }
}
```



# Lời giải tham khảo

```
Node* addBefore(Node* root, elementtype u, elementtype v) {  
    if(root == NULL) return NULL;  
    if(find(root,u) != NULL) return root;// do nothing  
    if(root->element == v) {  
        Node* q = makeNewNode(u) ;  
        q->next = root; return q;  
    }  
    root->next = addBefore(root->next,u,v) ;  
    return root;  
}
```



# Lời giải tham khảo

```
Node* addAfter(Node* root, elementtype u, elementtype v) {  
    if(root == NULL) return NULL;  
    if(root->element == v) {  
        Node* q = makeNewNode(u);  
        q->next = root->next;  
        root->next = q; return root;  
    }  
    root->next = addAfter(root->next,u,v);  
    return root;  
}
```



# Lời giải tham khảo

```
Node* reverse(Node *root) {  
    Node* p = root;  
    Node* pp = NULL;  
    Node* np = NULL;  
    while(p != NULL) {  
        np = p->next;  
        p->next = pp;  
        pp = p;  
        p = np;  
    }  
    return pp;  
}
```



# Lời giải tham khảo

```
void traverseList(Node* root) {  
    Node* p = root;  
    while(p != NULL) {  
        printf("%d ",p->element);  
        p = p->next;  
    }  
    printf("\n");  
}  
  
int main(){  
    solve(); traverseList(root); freeList();  
}
```



# Lời giải tham khảo

```
void solve() {  
    int n, i;  
    scanf("%d", &n);  
    for(i = 1; i <= n; i++) {  
        int k;  
        scanf("%d", &k);  
        h = insertLastRecursive(h, k);  
    }  
    while(1) {  
        char cmd[256];  
        scanf("%s", cmd);  
        if(strcmp(cmd, "#") == 0) break;  
    }  
}
```



# Lời giải tham khảo

```
if(strcmp(cmd,"addlast") == 0) {  
    int k; scanf("%d",&k);  
    if(find(root,k) == NULL) //h = insertLastRecursive(h,k);  
        insertAtTail(k);  
} else if(strcmp(cmd,"addfirst") == 0) {  
    int k; scanf("%d",&k);  
    if(find(root,k) == NULL) insertAtHead(k);  
} else if(strcmp(cmd,"addafter") == 0) {  
    int u, v; scanf("%d%d",&u,&v);  
    if(find(root,u) == NULL)  
        root = addAfter(root,u,v);  
}
```



# Lời giải tham khảo

```
else if(strcmp(cmd,"addbefore") == 0) {  
    int u, v; scanf("%d%d", &u, &v);  
    if(find(root,u) == NULL) root = addBefore(root,u,v);  
} else if(strcmp(cmd,"remove") == 0) {  
    int k; scanf("%d", &k);  
    root = removeNodeRecursive(root,k);  
} else if(strcmp(cmd,"reverse") == 0) {  
    root = reverse(root);  
}  
}
```



# Thao tác trên đa thức

- Một đa thức  $p(x)$  là một biểu thức với biến  $x$  với công thức dạng  $a_nx^n + a_{n-1}x^{n-1} + \dots + a_1x + a_0$ ,
  - với các hệ số  $a_i$  là các số thực
  - $n$  là số nguyên không âm, còn được gọi là bậc của đa thức.
- Một số thao tác cơ bản trên đa thức
  - Tạo và biểu diễn đa thức
  - Cộng (trừ) các đa thức
  - Nhân các đa thức
  - Chuẩn hóa đa thức
  - ..



# Biểu diễn đa thức sử dụng mảng

- Lưu trữ các hệ số vào các phần tử mảng với chỉ số mảng tương ứng số mũ

$$P(x) = [a_0 \quad a_1 \quad a_2 \quad \dots \quad a_n]$$

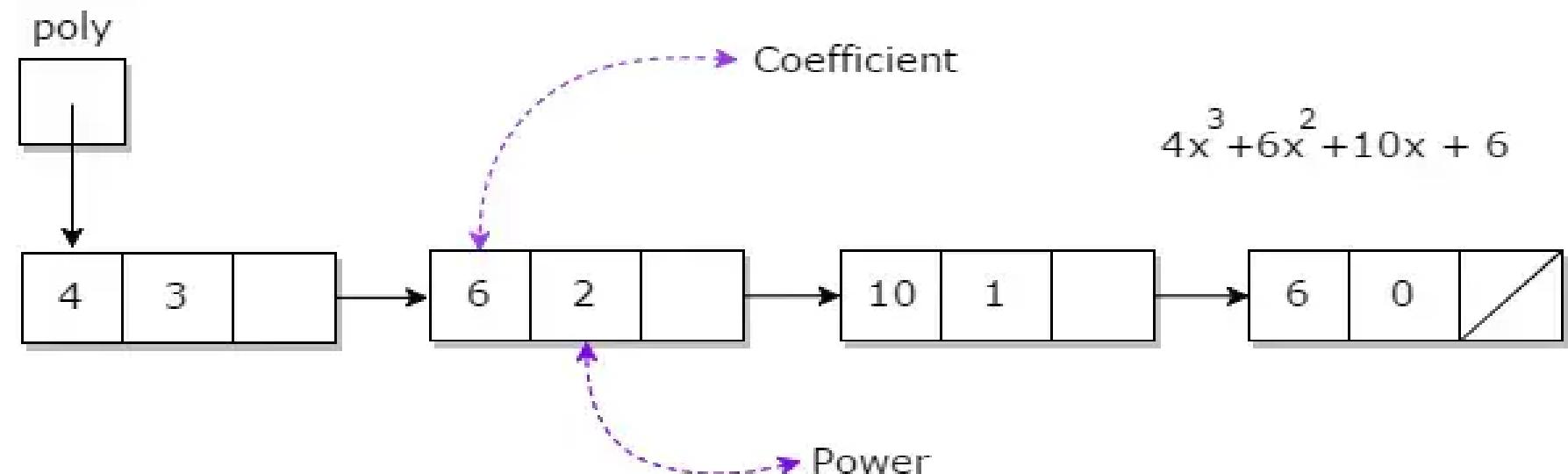
$$8x^3 + 3x^2 + 2x + 6 \leftarrow [6 \quad 2 \quad 3 \quad 8]$$

- Hạn chế: lãng phí không gian bộ nhớ với các đa thức "thưa" bậc cao.

$$8x^{100} + 3x^2 + 2x + 6 \leftarrow [6 \quad 2 \quad 3 \quad 0 \quad \dots \quad 0 \quad 8]$$

# Biểu diễn đa thức sử dụng danh sách liên kết

- Mỗi số hạng của đa thức được lưu trong một nút của danh sách với 2 trường: **hệ số** và **số mũ**.
- Các nút được sắp xếp theo chiều giảm dần hoặc tăng dần của số mũ.
- Không tồn tại hai nút có cùng giá trị số mũ
- **Tiết kiệm** không gian bộ nhớ lưu trữ.



# Lab2: Thao tác với đa thức

- Viết chương trình cung cấp các lệnh thao tác sau trên đa thức, biết rằng mỗi đa thức có một mã số định danh là một số nguyên dương từ 1 tới 10000:
- **Create <poly\_id>**: tạo một đa thức có mã định danh <pol\_id> nếu đa thức này không tồn tại, nếu không thì không làm gì.
- **AddTerm <poly\_id> <coef> <exp>**: Thêm một số hạng có hệ số <coef> và số mũ <exp> vào đa thức có định danh <poly\_id> (tạo đa thức mới nếu nó không tồn tại)
- **EvaluatePoly <poly\_id> <variable\_value>**: Tính giá trị của đa thức có định danh <poly\_id> và <variable\_value> là giá trị của biến (in 0 nếu đa thức không tồn tại)
- **AddPoly <poly\_id1> <poly\_id2> <result\_poly\_id>**: Thực hiện phép cộng trên hai đa thức <pol\_id1> và <poly\_id2>. Đa thức kết quả sẽ có mã định danh <result\_poly\_id> (nếu đa thức <result\_poly\_id> tồn tại thì sẽ ghi đè đa thức hiện có)
- **PrintPoly <poly\_id>**: in đa thức <poly\_id> (nếu có) ra dòng ra chuẩn dưới dạng <c\_1> <e\_1> <c\_2> <e\_2> ... (chuỗi các cặp (hệ số, số mũ) số hạng của đa thức theo thứ tự giảm dần của số mũ)
- **Destroy <poly\_id>**: Xóa và giải phóng danh sách có mã số <poly\_id>



# Định dạng đầu vào

- **Input:** Each line contains a command described above (terminated by a line containing \*)
- Example:

AddTerm 1 3 2

AddTerm 1 4 0

AddTerm 1 6 2

AddTerm 2 3 2

AddTerm 2 7 5

PrintPoly 1

PrintPoly 2

AddPoly 2 1 3

PrintPoly 3

EvaluatePoly 2 1



# Định ra đầu ra

- **Output:** Each line contains the information printed out by the PrintPoly and EvaluatePoly above
- Example:

9 2 4 0

7 5 3 2

7 5 12 2 4 0

10



# Lời giải tham khảo

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#define N 10001
typedef struct TNode{
    int coef;
    int exp;
    struct TNode* next;
} Node;
Node* P[N];// P[i] is the pointer to the first element of linked
list representing the polynomial id = i
```



# Lời giải tham khảo

```
Node* makeNode(int c, int e){  
    Node* p = (Node*)malloc(sizeof(Node));  
    p->coef = c; p->exp = e; p->next = NULL;  
    return p;  
}  
  
void printPoly(Node* p){  
    for(Node* q = p; q!=NULL; q = q->next){  
        printf("%d %d ", q->coef, q->exp);  
    }  
    printf("\n");  
}
```



# Lời giải tham khảo

```
Node* addTerm(int c, int e, Node* p){  
    if(p == NULL) return makeNode(c,e);  
    if(e > p->exp) {  
        Node* q = makeNode(c,e);  
        q->next = p; return q;  
    }  
    if(e == p->exp) { p->coef += c; return p;  
    }  
    p->next = addTerm(c,e,p->next);  
    return p;  
}  
  
void processAddTerm(int id, int c, int e){ P[id] = addTerm(c,e,P[id]);  
}
```



# Lời giải tham khảo

```
Node* addPoly(Node* p1, Node* p2) {
    Node* prs = NULL;
    Node* last = prs;
    // copy poly p1 to prs
    for(Node* q = p1; q != NULL; q = q->next) {
        Node* newNode = makeNode(q->coef, q->exp);
        if(prs == NULL) prs = newNode;
        else last->next = newNode;
        last = newNode;
    }
    for(Node* q = p2; q != NULL; q = q->next) {
        prs = addTerm(q->coef, q->exp, prs);
    }
    return prs;
}
```



# Lời giải tham khảo

```
void processAddPoly(int id1, int id2, int idrs){  
    P[idrs] = addPoly(P[id1], P[id2]);  
}  
  
void processEvaluatePoly(int id, int x){  
    long long rs = 0;  
    for(Node* q = P[id]; q != NULL; q = q->next) {  
        rs = rs + q->coef* pow(x,q->exp);  
    }  
    printf("%lld",rs);  
}  
int main(){  
    char cmd[50];  
    for(int id = 1; id <= N-1; id++) P[id] = NULL;
```



# Lời giải tham khảo

```
while(1) {
    scanf ("%s", cmd);
    if(strcmp(cmd, "*") == 0) break;
    else if(strcmp(cmd, "AddTerm") == 0) {
        int id, c, e; scanf ("%d%d%d", &id, &c, &e);
        processAddTerm(id, c, e);
    }else if(strcmp(cmd, "PrintPoly") == 0) {
        int id; scanf ("%d", &id);
        printPoly(P[id]);
    }else if(strcmp(cmd, "AddPoly") == 0) {
        int id1, id2, idrs; scanf ("%d%d%d", &id1, &id2, &idrs);
        processAddPoly(id1, id2, idrs);
    }else if(strcmp(cmd, "EvaluatePoly") == 0) {
        int id, x; scanf ("%d%d", &id, &x);
        processEvaluatePoly(id, x);
    }
}
return 0;
```





**HUST**

**XIN CẢM ƠN !**