

# Basic C Programming

## Bài 3

(Lớp học lần 2)


# Nội dung

- Cấu trúc tự trở
- Danh sách liên kết đơn

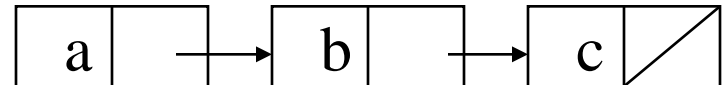
# Cấu trúc tự trỏ

- Một hoặc nhiều thành phần trong cấu trúc của nó là con trỏ, trỏ tới chính nó

```
struct list {  
    char data;  
    struct list *link;  
};  
list item1, item2, item3;
```



```
item1.data='a';  
item2.data='b';  
item3.data='c';  
item1.link=item2;  
item2.link=item3;  
item3.link=NULL;
```



# Danh sách liên kết

- Là một danh sách các phần tử, trong đó mỗi phần tử có lưu trữ địa chỉ của phần tử tiếp theo
- Danh sách liên kết đơn: mỗi phần tử chứa một liên kết (lưu trữ một địa chỉ) của phần tử tiếp theo
- Danh sách liên kết đôi: mỗi phần tử chứa hai liên kết (lưu trữ hai địa chỉ) của phần tử trước và sau nó
- Khác so với mảng?



root (or head)

NULL

# Khai báo danh sách liên kết đơn

```
typedef ... elementtype;  
struct node{  
    elementtype element;  
    struct node* next;  
};  
struct node* root;  
struct node* cur;
```

```
typedef ... elementtype;  
typedef struct node{  
    elementtype element;  
    struct node* next;  
}node;  
node* root;  
node* cur;
```

# Các thao tác trên linked list

- Khởi tạo một phần tử mới
- Chèn phần tử vào danh sách
- Duyệt danh sách
- Xóa một phần tử
- Giải phóng danh sách

# Khởi tạo cho một phần tử mới

- Chúng ta cần cấp phát động bộ nhớ cho mỗi phần tử trong danh sách

```
struct node * new;
```

```
new = (struct node*) malloc(sizeof(struct node));
```

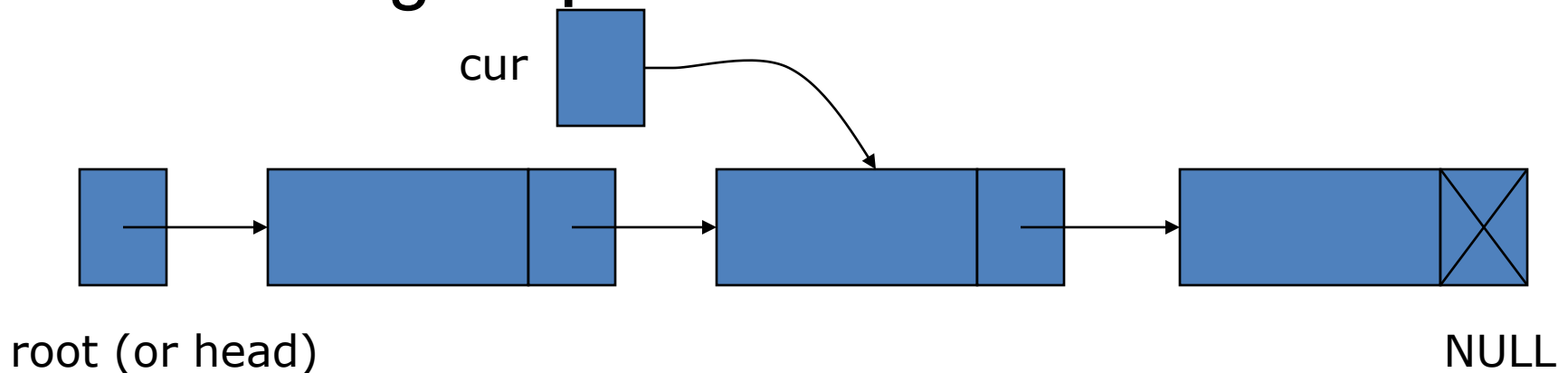
```
new->element = ... //dữ liệu khởi tạo
```

```
new->next = NULL; //chưa trỏ tới phần tử nào
```

- Tại sao lại cấp phát động?
- Nhắc lại: truy cập trường qua con trỏ cấu trúc
  - tên\_con\_trỏ -> tên\_trường
  - (\*tên\_con\_trỏ).tên\_trường

# Chèn một phần tử vào danh sách

- Root: lưu trữ địa chỉ nút đầu tiên trong ds
- Cur: Con trỏ lưu trữ địa chỉ phần tử hiện tại, đang xét
- Lưu ý: nút cuối cùng danh sách, trường next có giá trị NULL

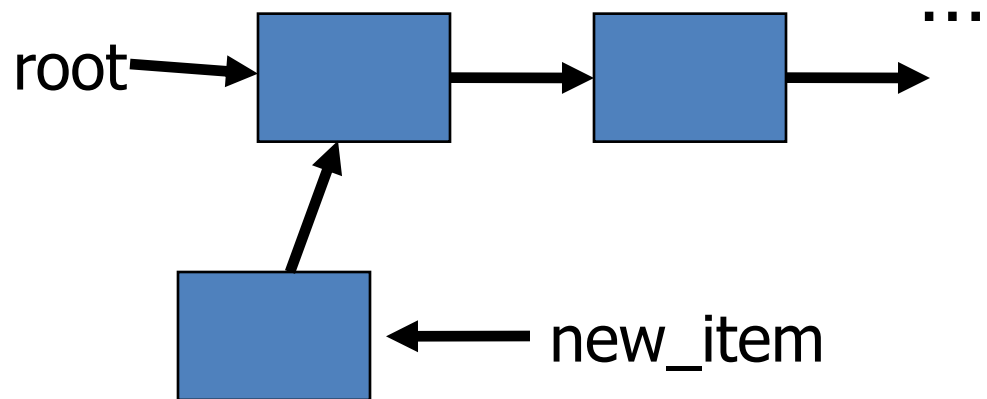




# Chèn một phần tử vào danh sách

- Chèn vào đầu danh sách

```
//tạo phần tử mới new_item  
if(root==NULL) {  
    root=new_item;  
}  
else{  
    new_item->next=root;  
    root=new_item;  
}
```



# Chèn một phần tử vào danh sách

- Chèn sau phần tử trỏ bởi cur

```
//tạo phần tử mới new_item
```

```
if (root==NULL) {
```

```
    root=cur=new_item;
```

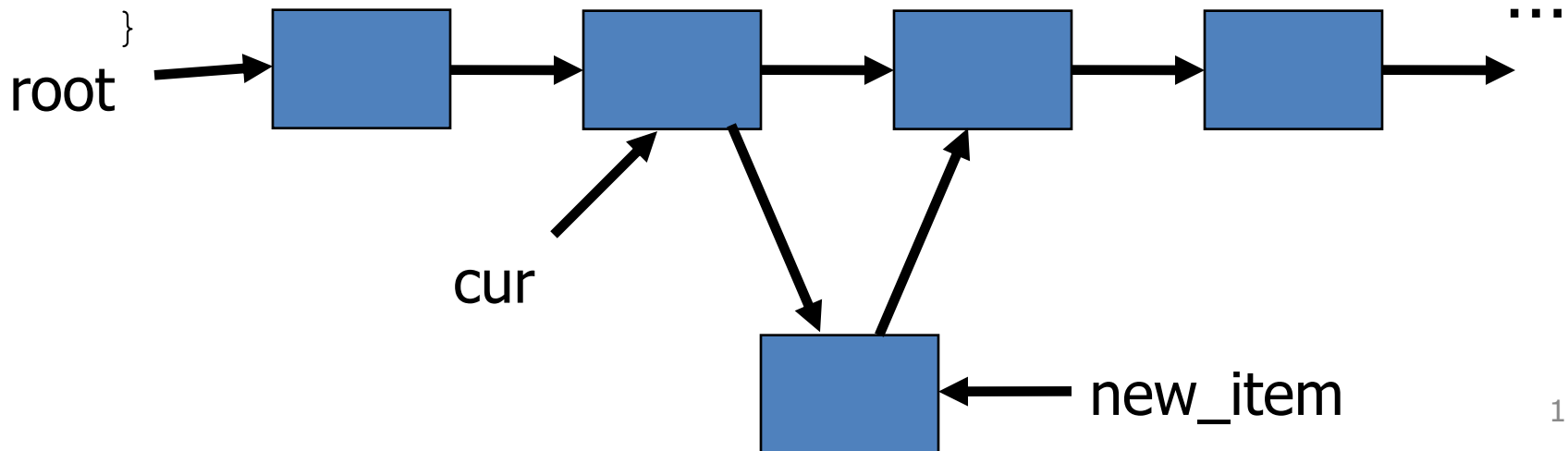
```
}else{
```

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

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

```
    cur= new_item; //hoac cur=cur->next;
```

```
}
```



```
// sai
```

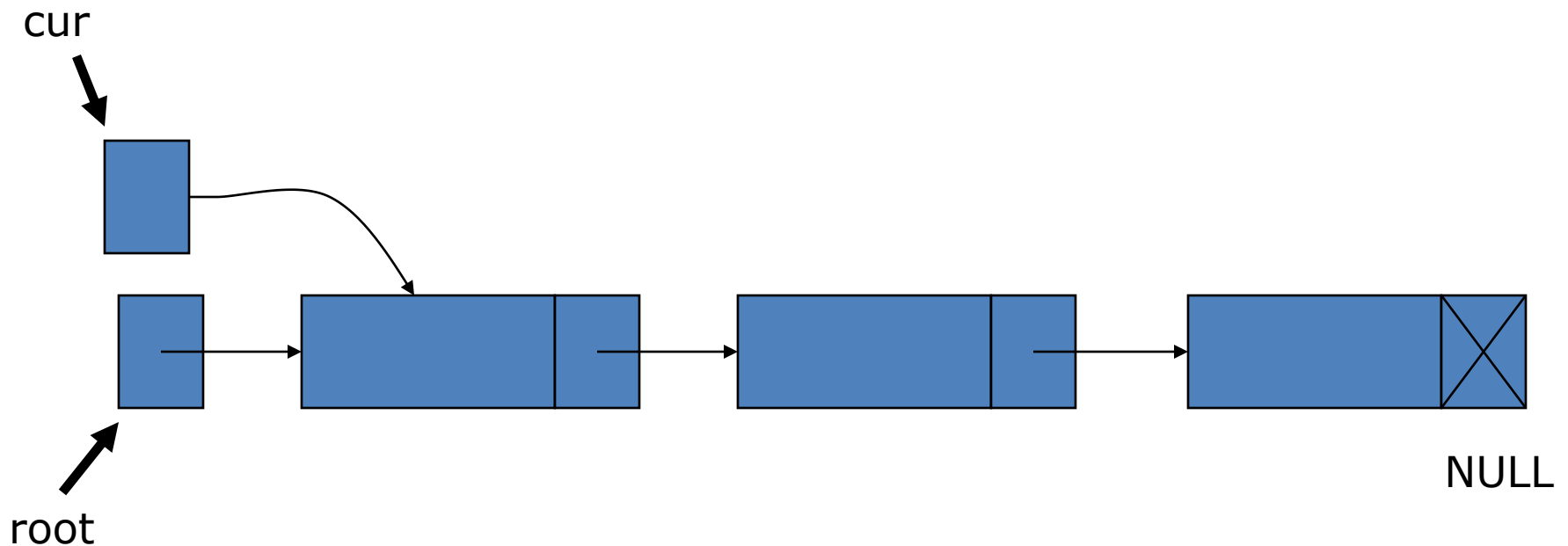
```
cur->next = new_item;
```

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

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

# Duyệt danh sách

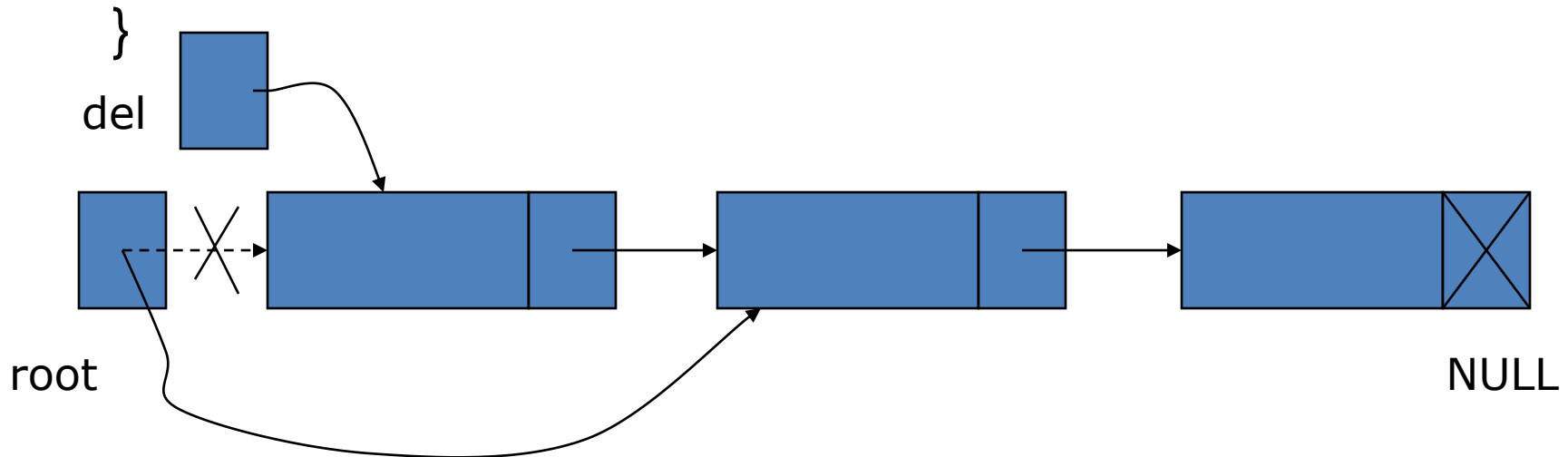
```
for ( cur = root; cur != NULL; cur = cur->next ) {  
    //Hiển thị cur->element ra màn hình  
}
```



# Xóa phần tử trong danh sách

- Xóa phần tử đầu tiên

```
if (root != NULL){  
    del = root;  
    root = del->next;  
    free(del); //không được free(root)  
}
```



# Xóa phần tử trong danh sách

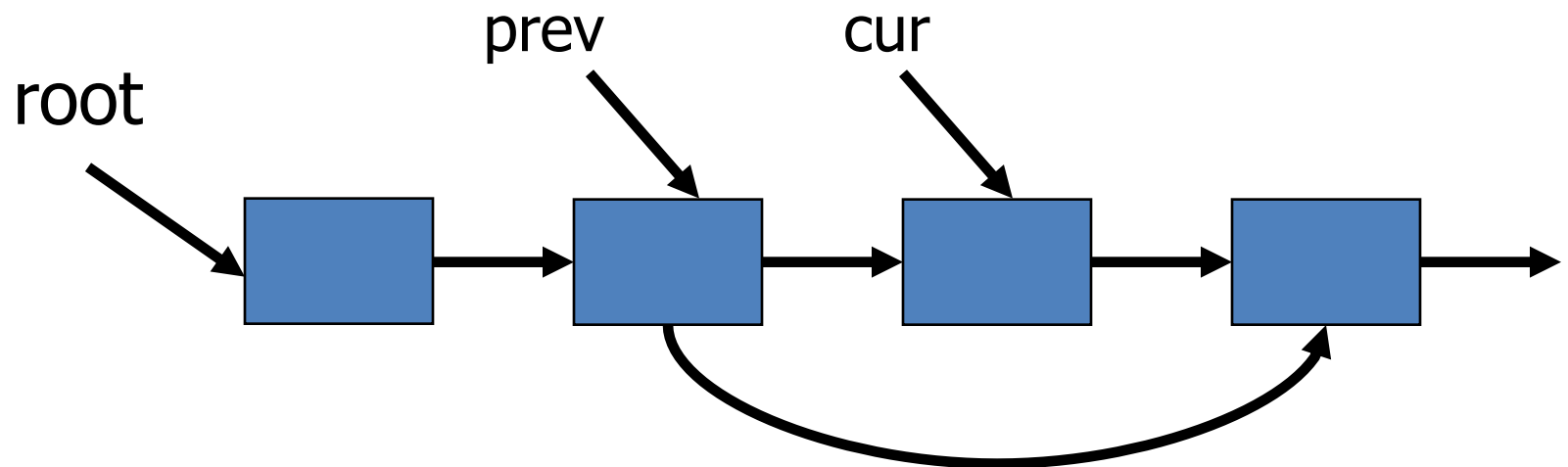
Muốn xóa phần tử trỏ bởi con trỏ cur

– Sử dụng thêm con trỏ prev

Thực hiện:

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

```
free(cur);
```



# Bài tập

- Thông tin danh bạ điện thoại gồm có: tên, số điện thoại, địa chỉ email
- Định nghĩa cấu trúc lưu trữ thông tin trên

# Cấu trúc danh bạ điện thoại

```
typedef struct Address{  
    char name[20];  
    int tel;  
    char email[30];  
}Address;  
typedef struct AddressList{  
    struct AddressList *next;  
    Address addr;  
}AddressList;  
AddressList *root;
```

```
struct Address{  
    char name[20];  
    int tel;  
    char email[30];  
};  
struct AddressList {  
    struct AddressList *next;  
    struct Address addr;  
};  
struct AddressList *root;
```

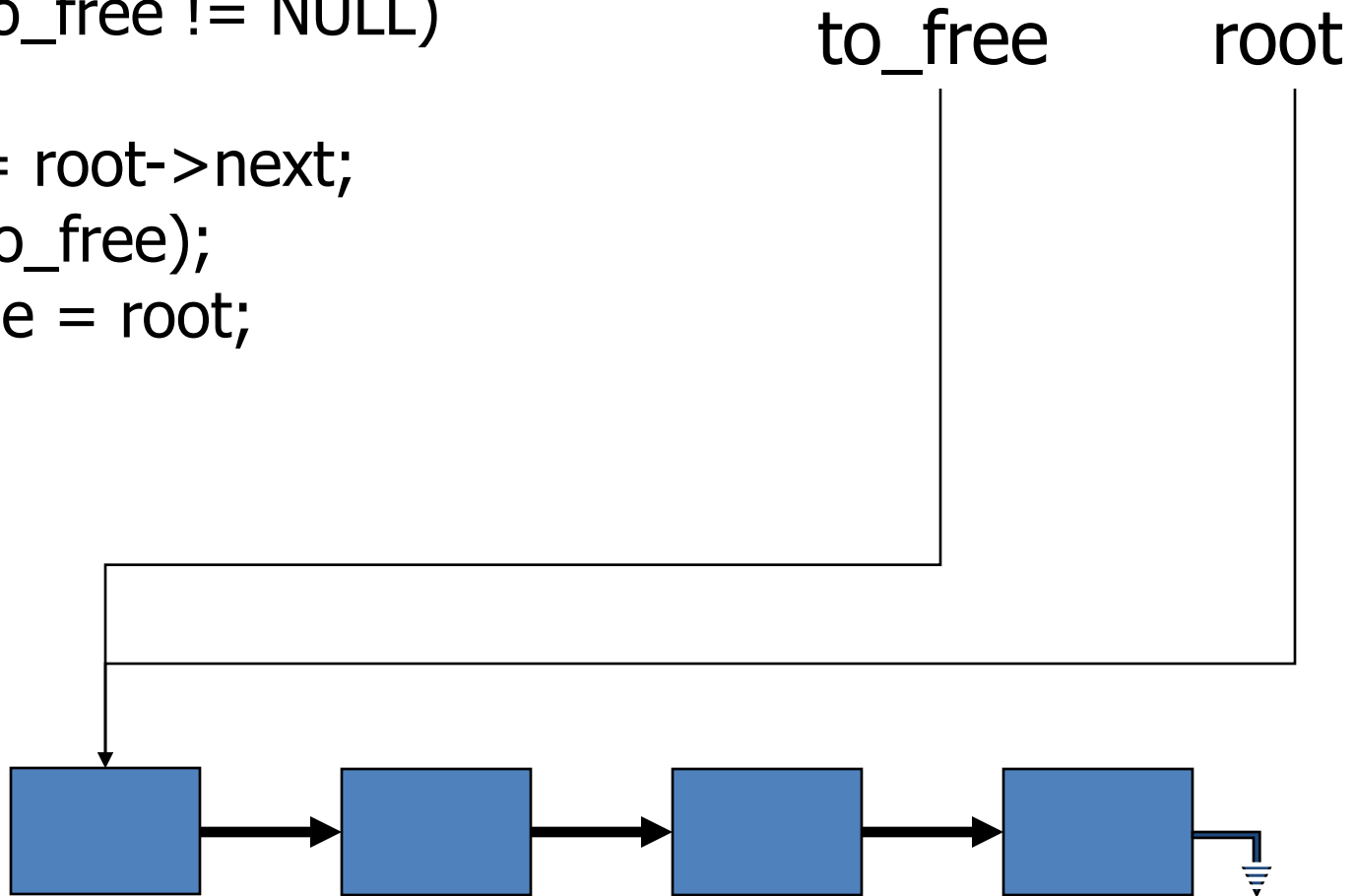
# Bài tập

1. Tạo danh sách liên kết đơn lưu trữ danh bạ điện thoại
2. Viết hàm chèn vào đầu danh sách thông tin của một người mới
3. Viết hàm chèn vào cuối danh sách thông tin của một người mới
4. Viết hàm duyệt danh sách
5. Viết hàm cho phép nhập vào tên của một người, và loại bỏ người này khỏi danh sách
6. Kiểm tra các hàm trên



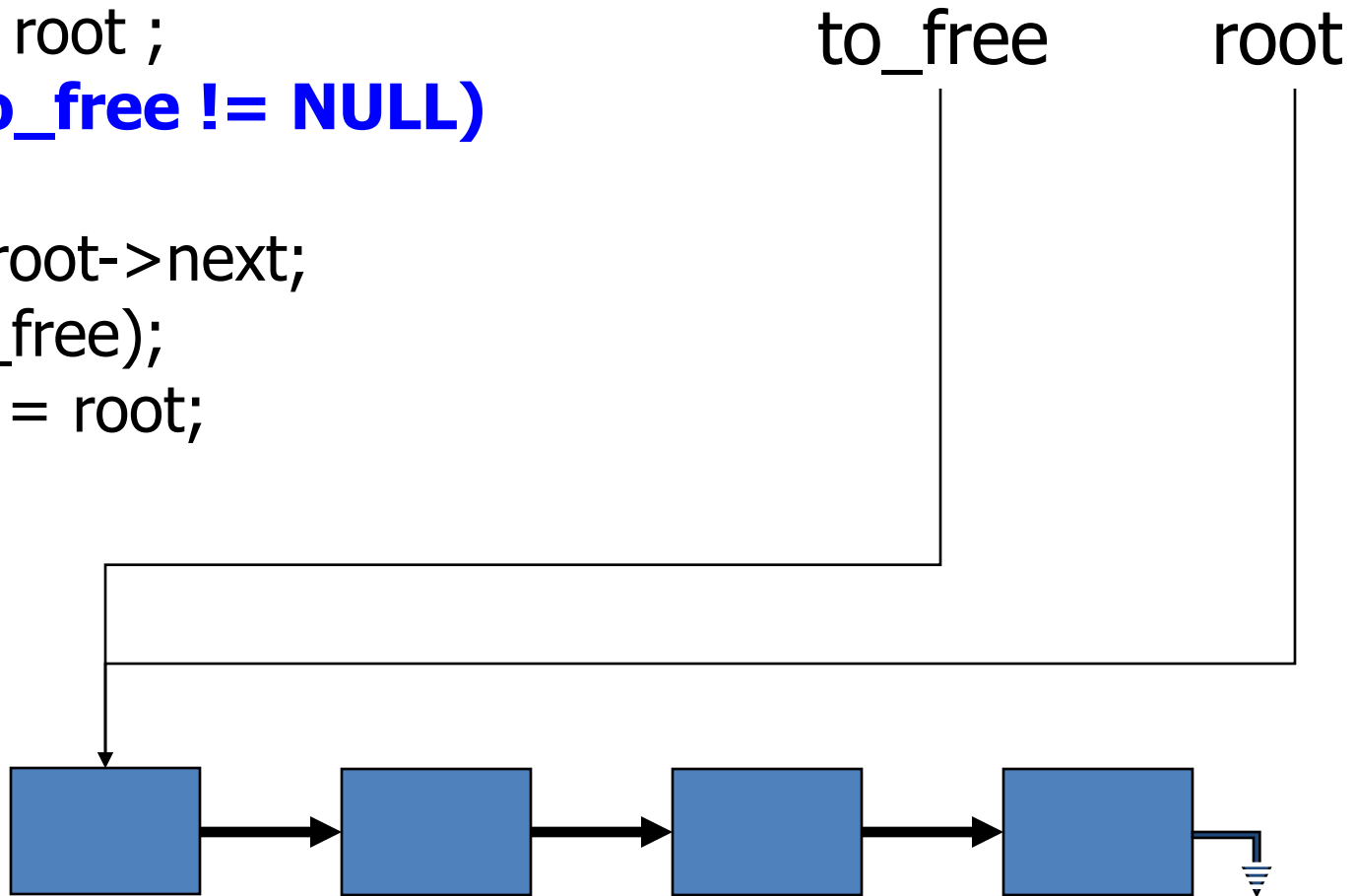
# Giải phóng danh sách

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



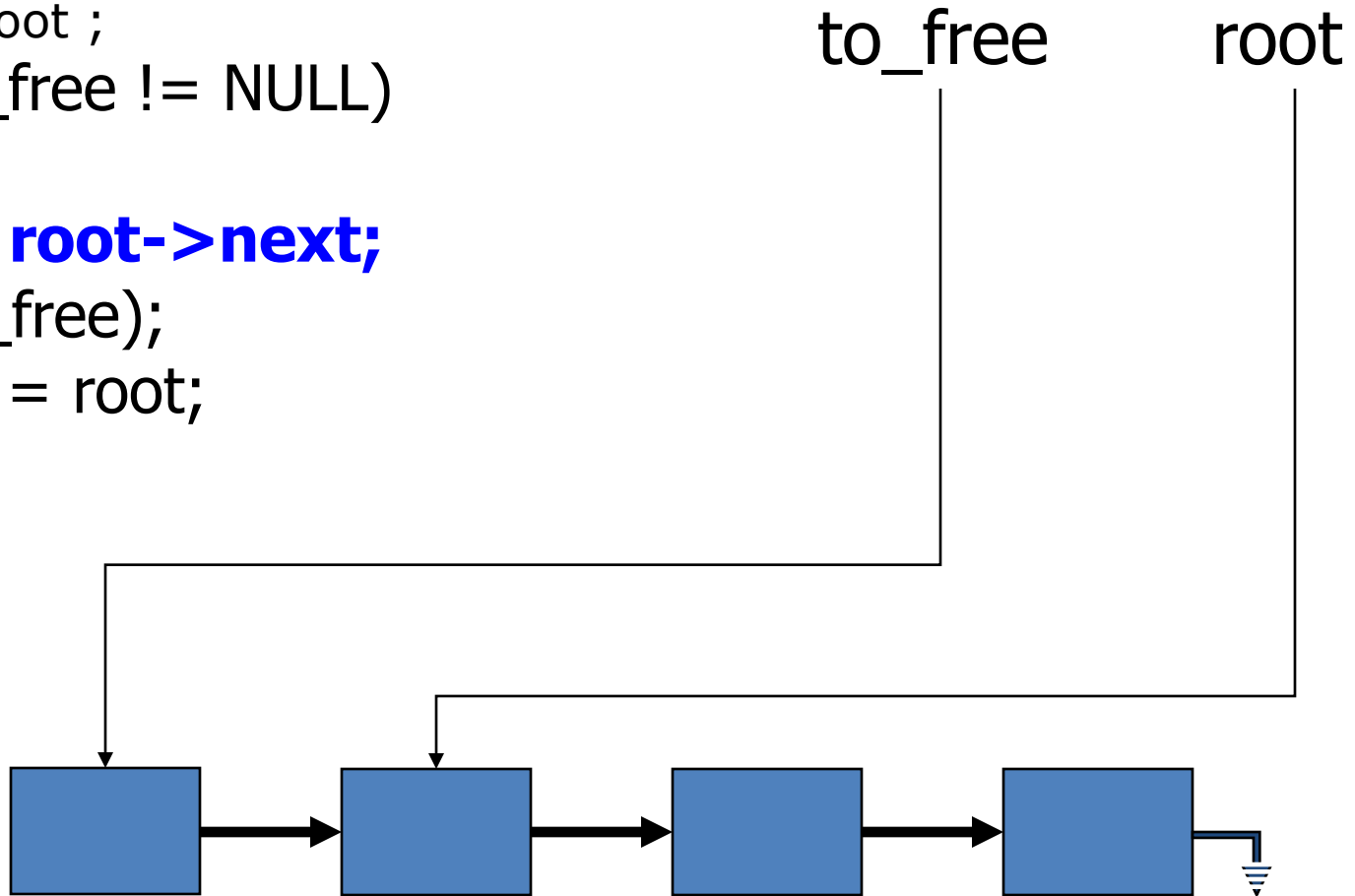
# Giải phóng danh sách

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



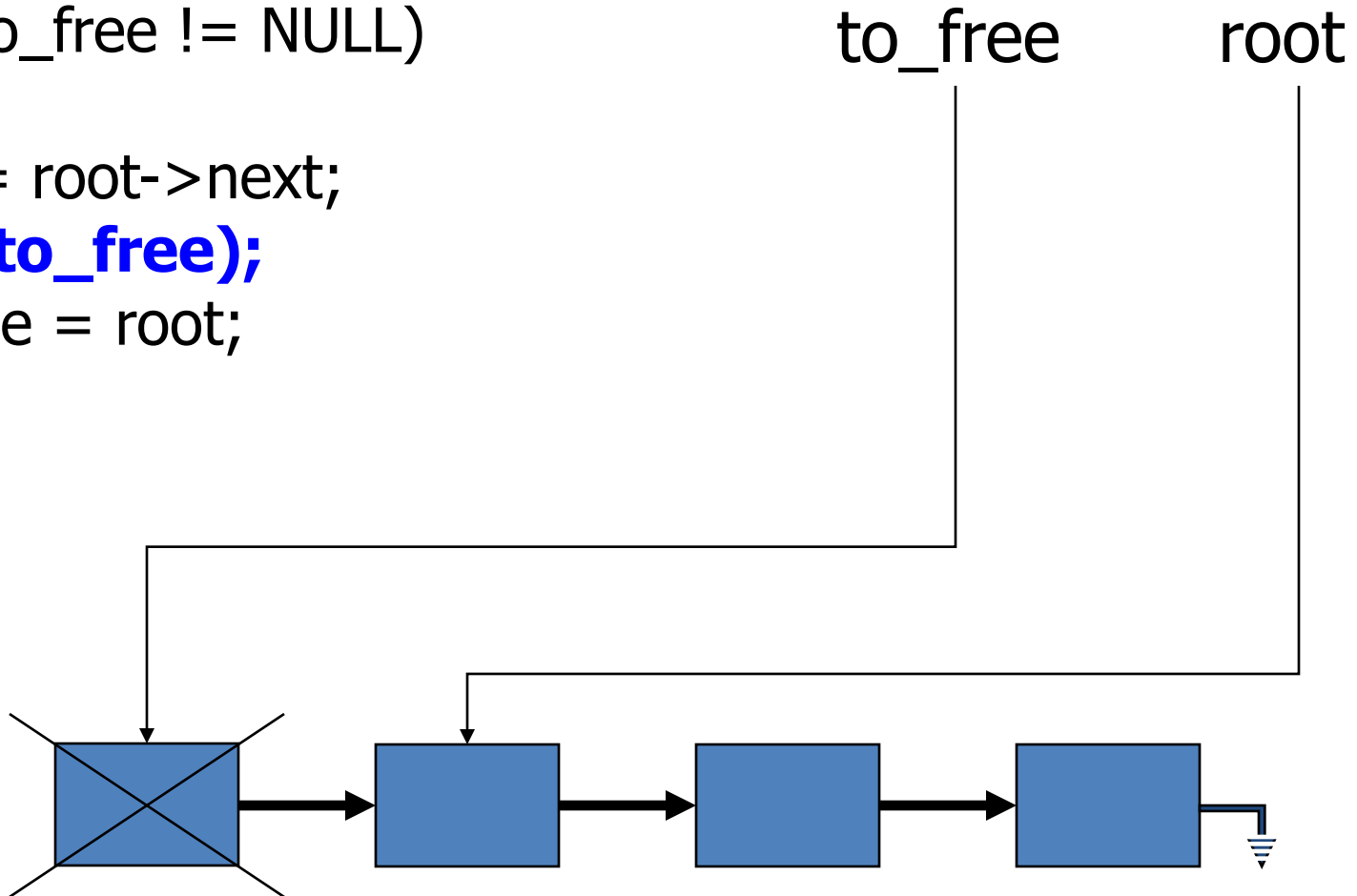
# Giải phóng danh sách

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



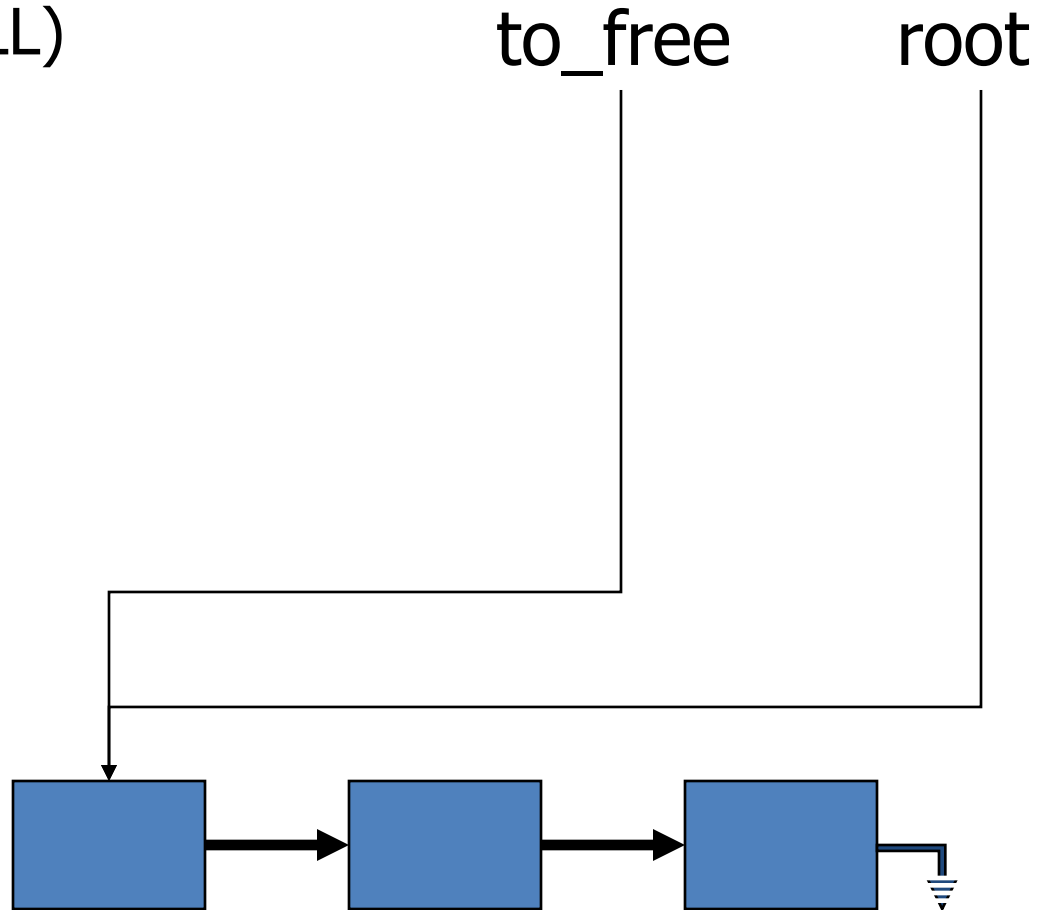
# Giải phóng danh sách

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



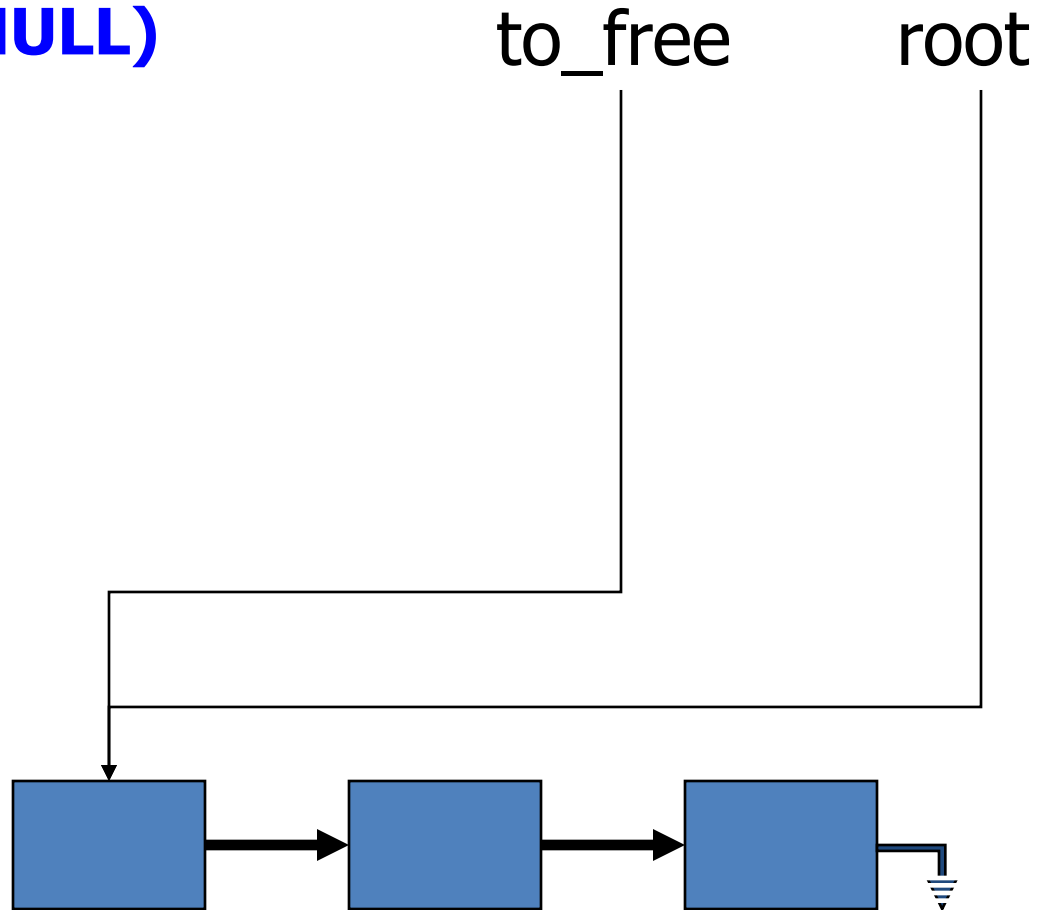
# Giải phóng danh sách

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



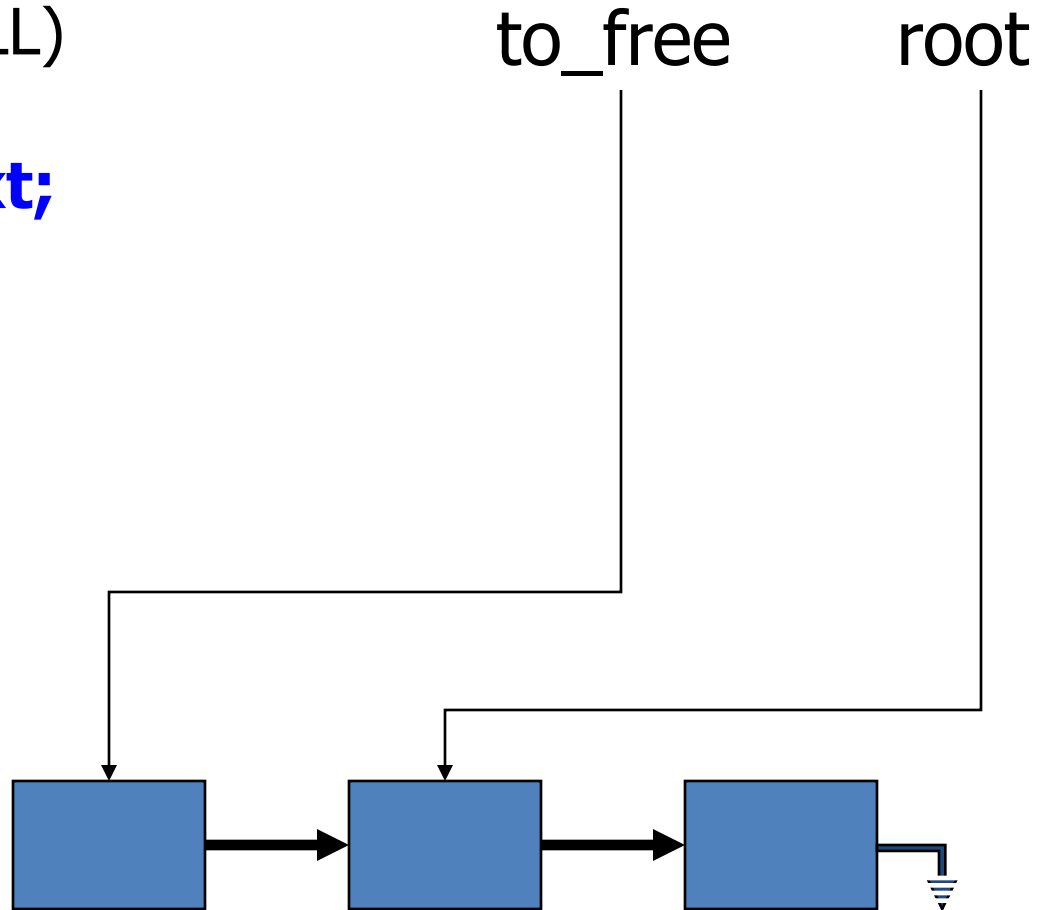
# Giải phóng danh sách

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



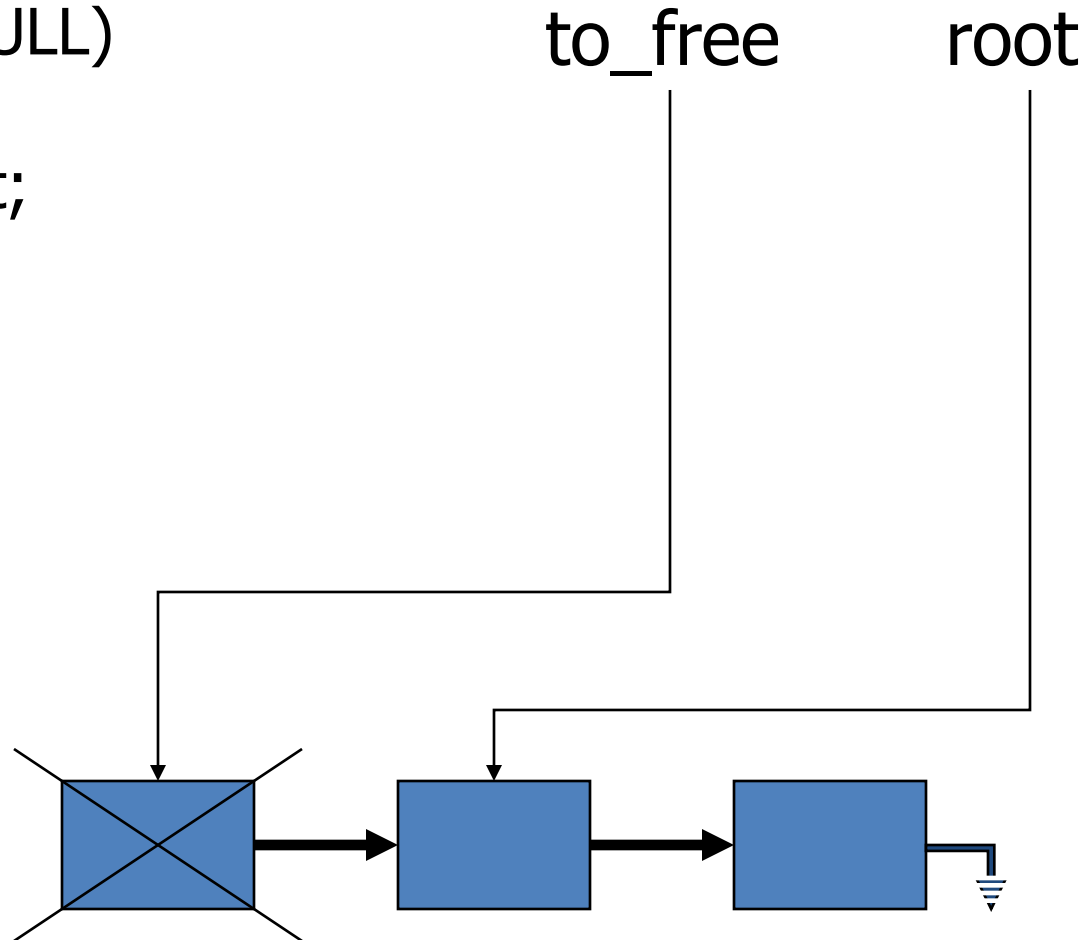
# Giải phóng danh sách

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



# Giải phóng danh sách

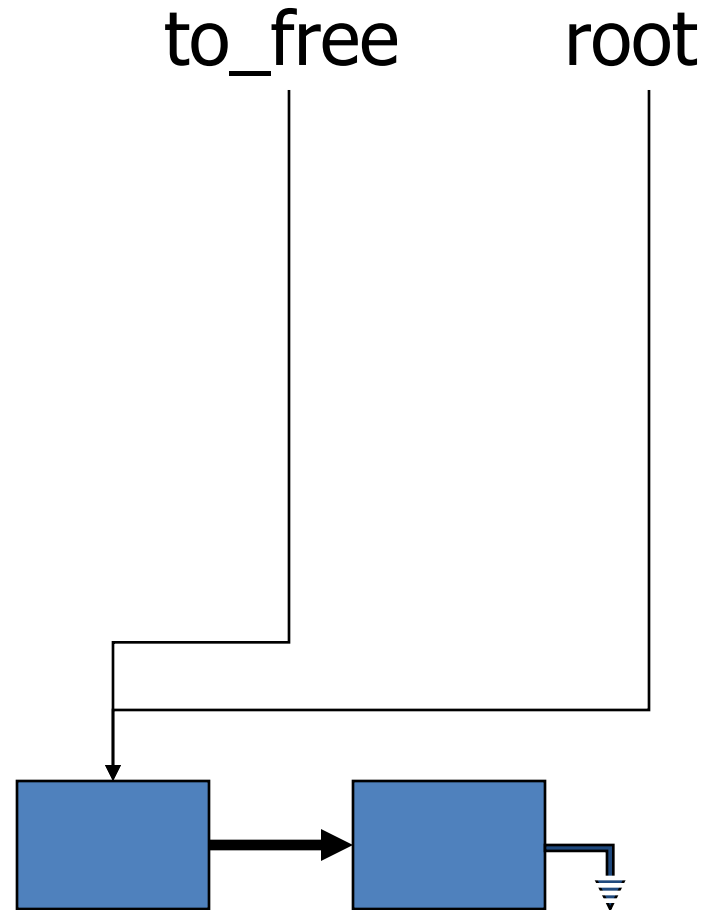
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```





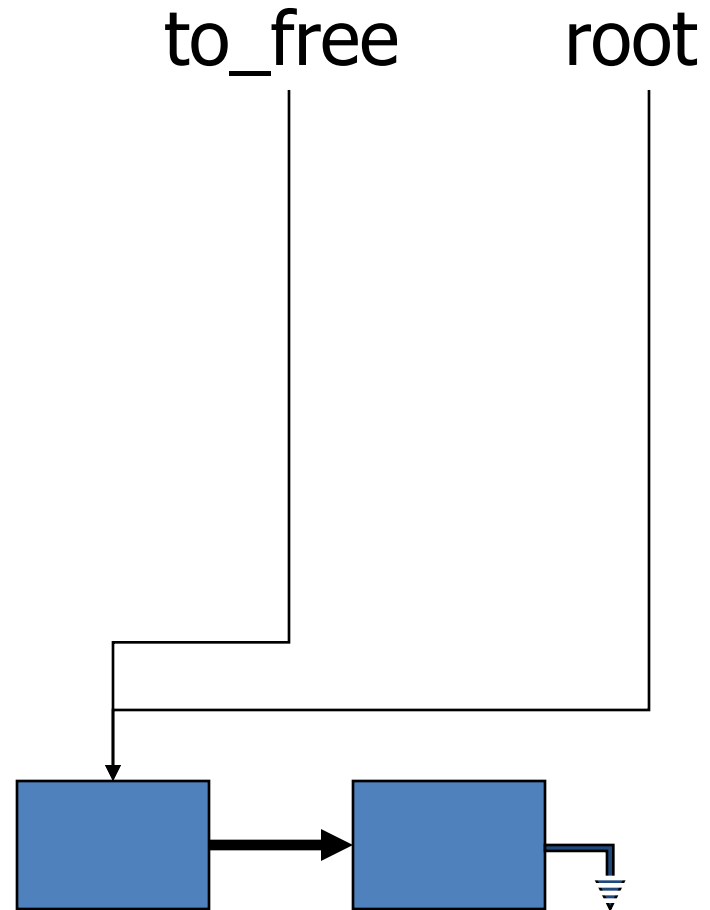
# Giải phóng danh sách

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



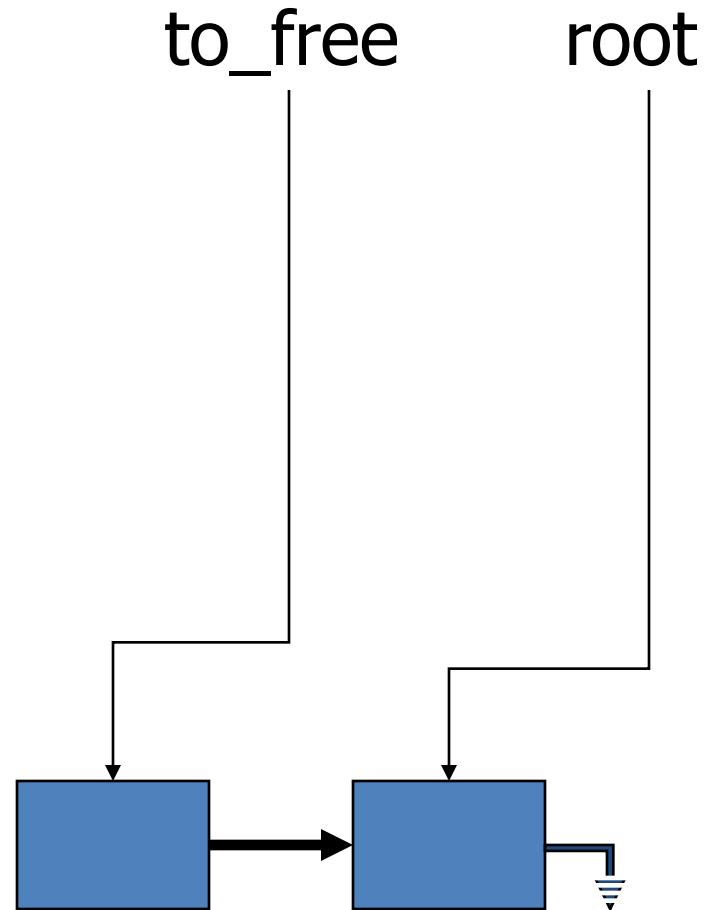
# Giải phóng danh sách

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



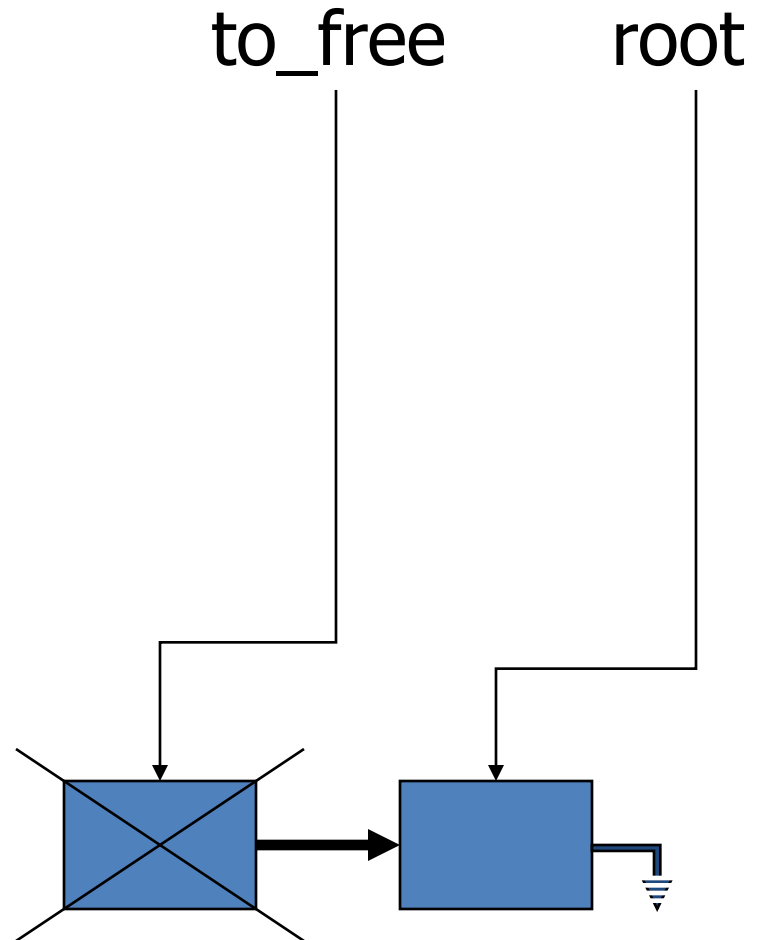
# Giải phóng danh sách

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



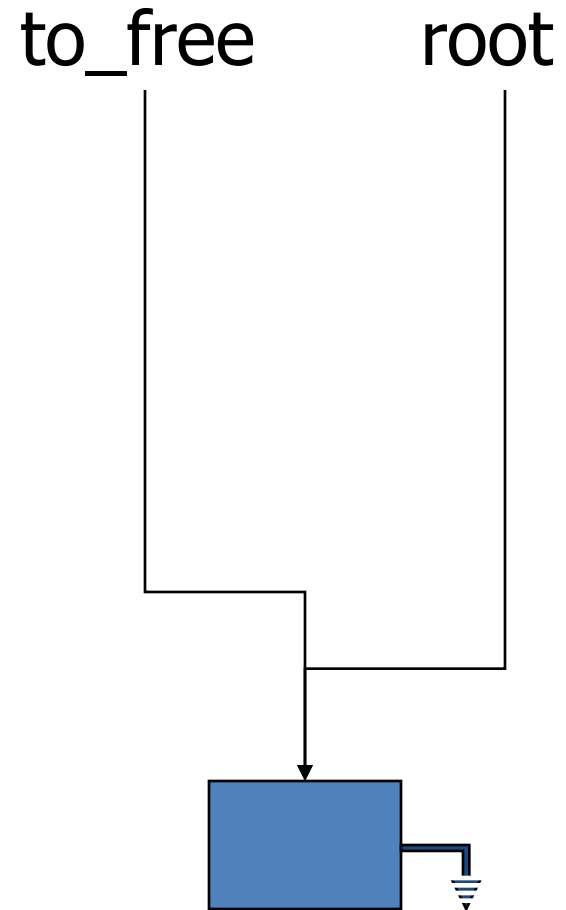
# Giải phóng danh sách

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



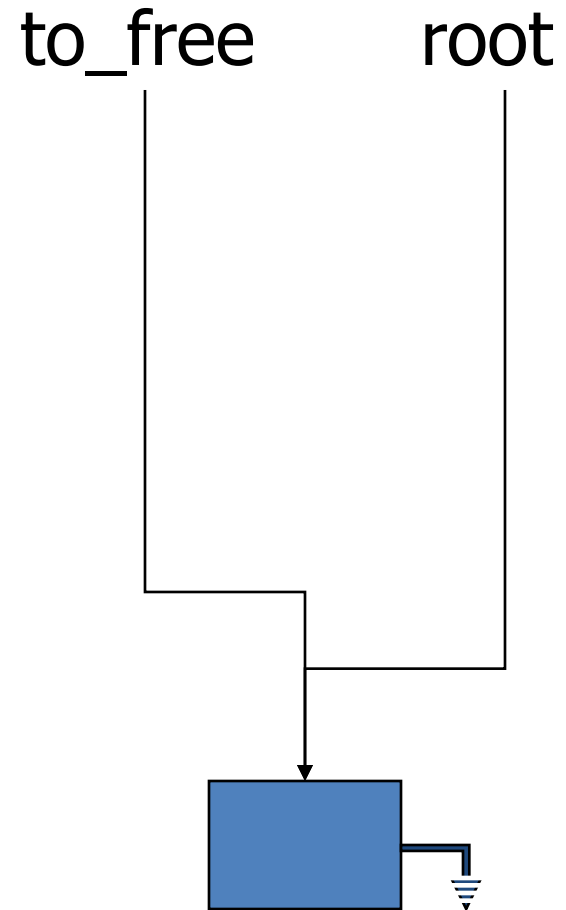
# Giải phóng danh sách

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



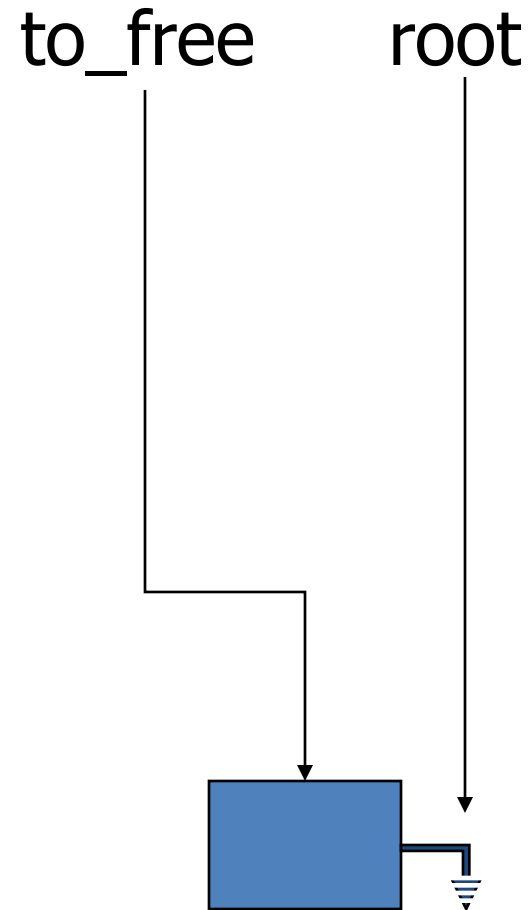
# Giải phóng danh sách

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



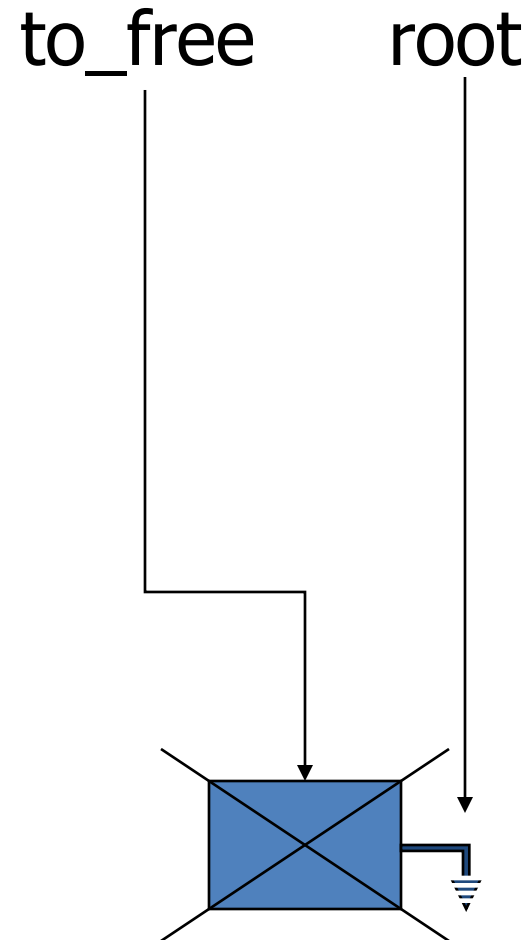
# Giải phóng danh sách

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



# Giải phóng danh sách

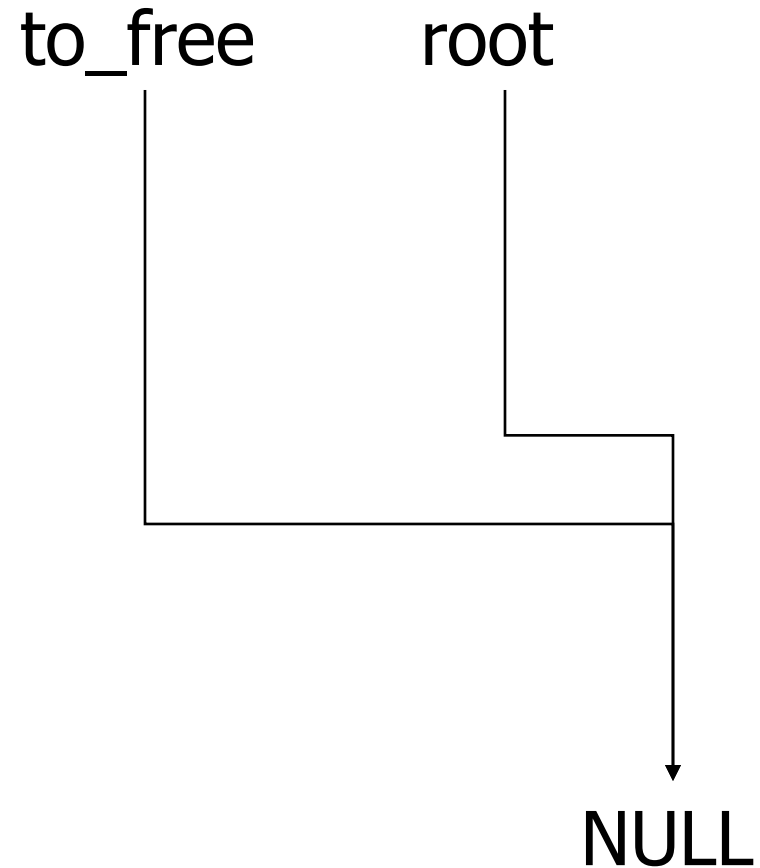
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```





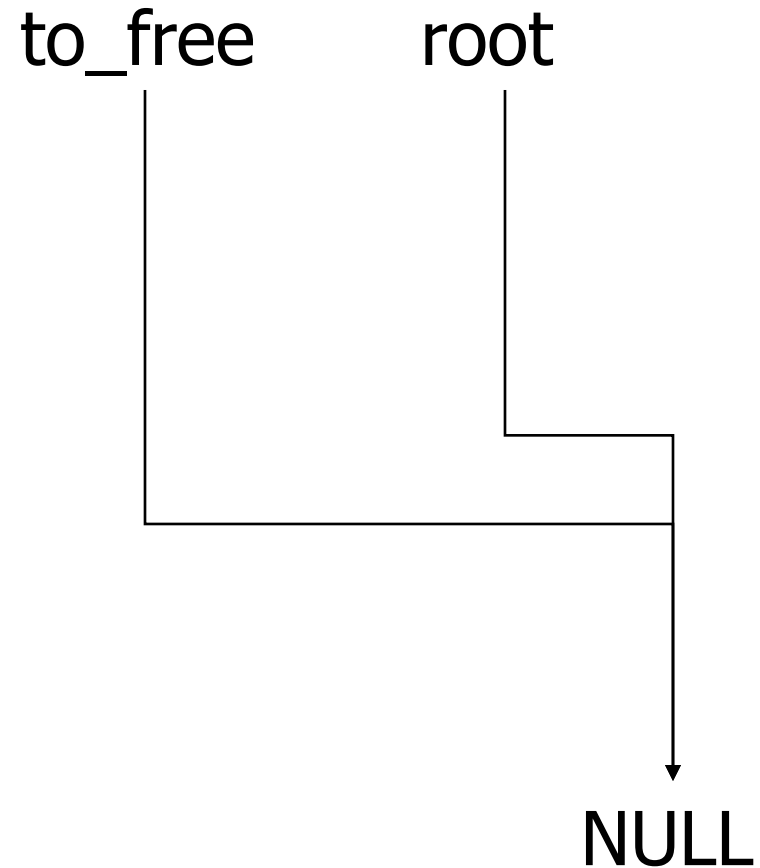
# Giải phóng danh sách

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



# Giải phóng danh sách

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



# Bài 2

- Định nghĩa cấu trúc

```
struct list_int {  
    int val;  
    struct list_int *next;  
};  
struct list_int *head=NULL;
```

- Viết hàm đảo ngược lại danh sách

- Gợi ý:

- Sử dụng 2 con trỏ prev, cur
- Duyệt từ đầu -> cuối danh sách, đảo ngược lại liên kết

# Bài 3

- Định nghĩa cấu trúc lưu trữ thông tin sinh viên như sau

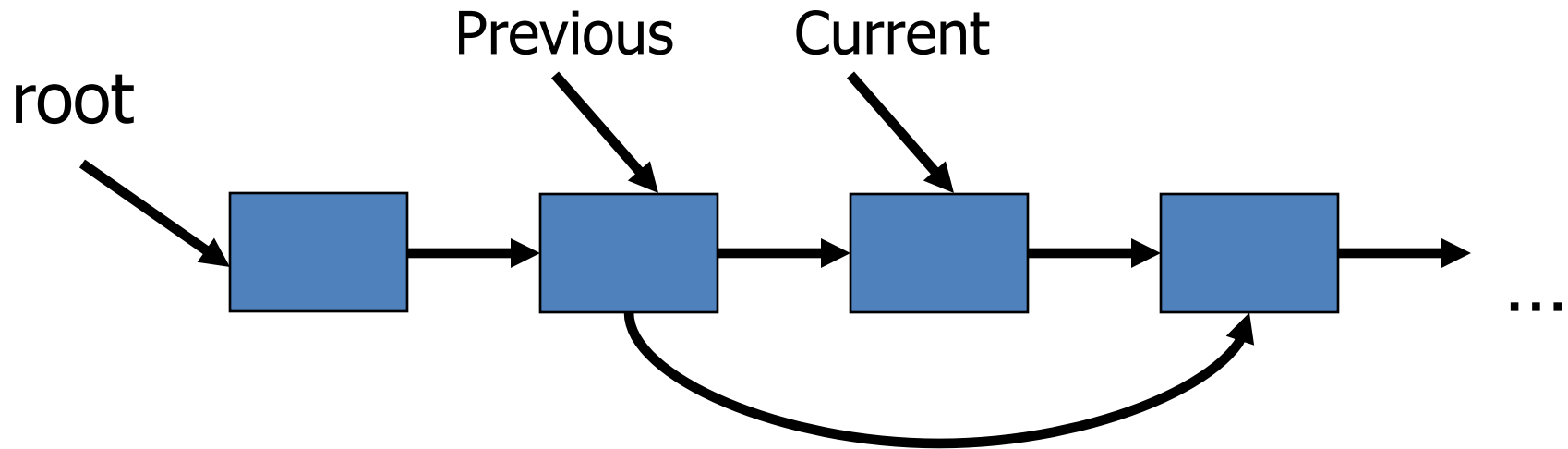
```
typedef struct student_t {  
    char  name[30];  
    int   grade;  
    struct student_t *next;  
}student;
```

# Bài 3

Viết chương trình có các chức năng

- Nhập thông tin sinh viên từ bàn phím và chèn vào danh sách sao cho điểm theo thứ tự giảm dần
- Xóa tất cả những sinh viên có tên được nhập vào từ bàn phím

# Lưu ý về xóa nút



# Xóa nút gốc

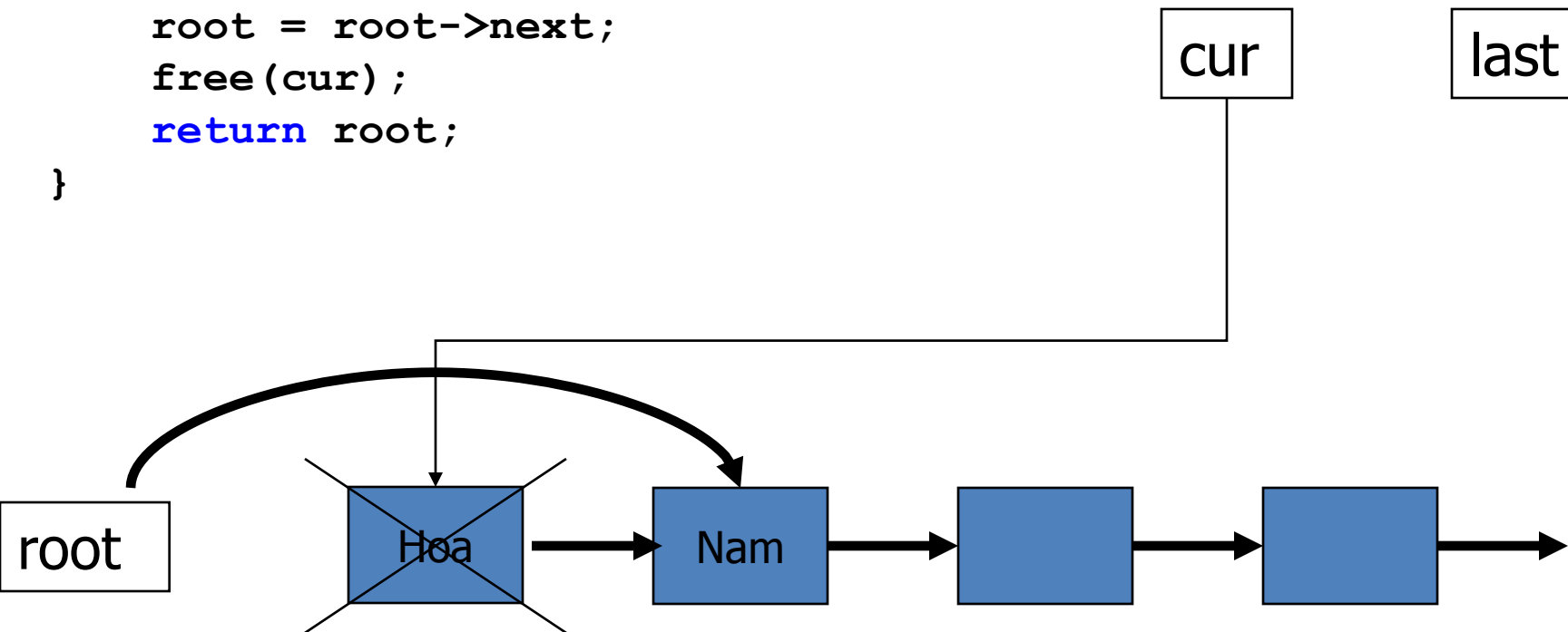
```
if (root == NULL)
    return root;
cur = root;
if (strcmp(cur->name, s) == 0)
{
    root = root->next;
    free(cur);
    return root;
}
```

s

Hoa

cur

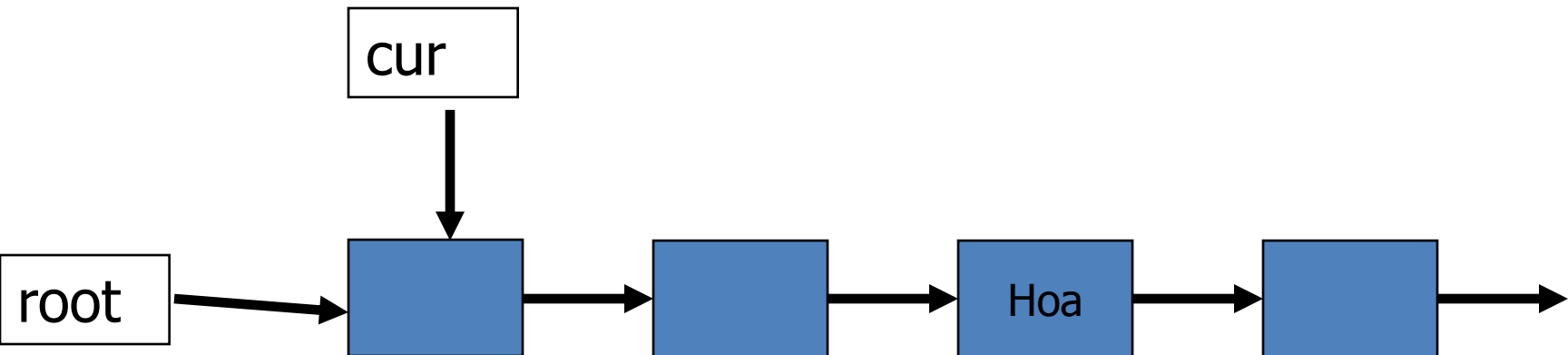
last



# Xóa nút giữa

```
while (cur != NULL && strcmp(cur->name,s) != 0)
{
    prev = cur;
    cur = cur->next;
}
if (cur != NULL)
{
    prev->next = cur->next;
    free(cur);
}
return root;
```

Hoa





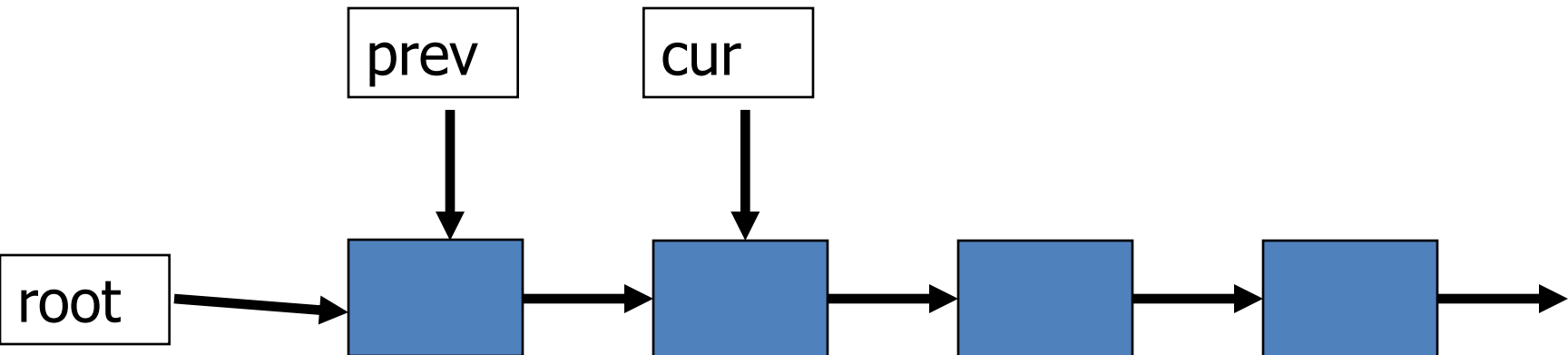
# Xóa nút giữa

```
while (cur != NULL && strcmp(cur->name, s) != 0)
{
    prev = cur;
    cur = cur->next;
}

if (cur != NULL)
{
    prev->next = cur->next;
    free(cur);
}

return root;
```

S  
Hoa



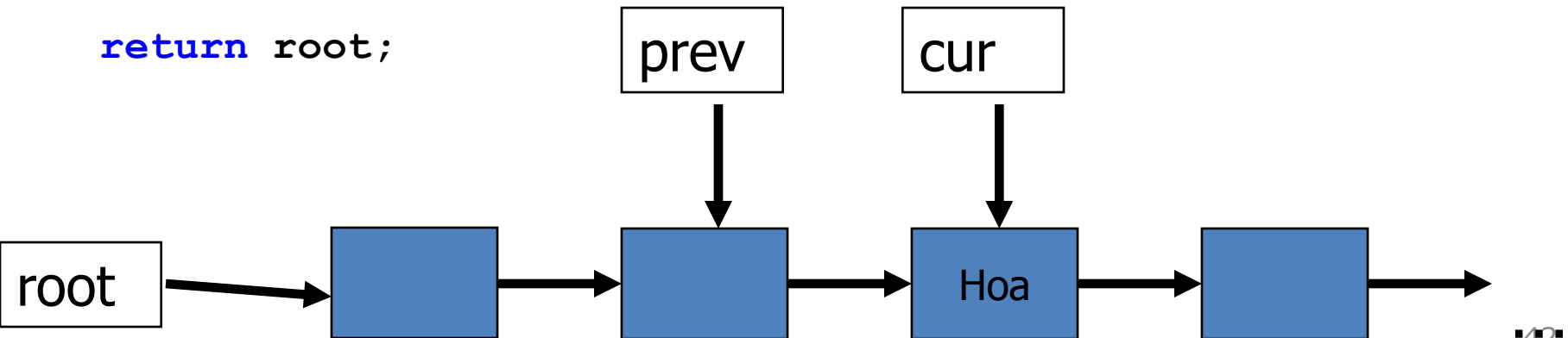
# Xóa nút giữa

```
while (cur != NULL && strcmp(cur->id, id) != 0)
{
    prev = cur;
    cur = cur->next;
}

if (cur != NULL)
{
    prev->next = cur->next;
    free(cur);
}

return root;
```

S  
Hoa



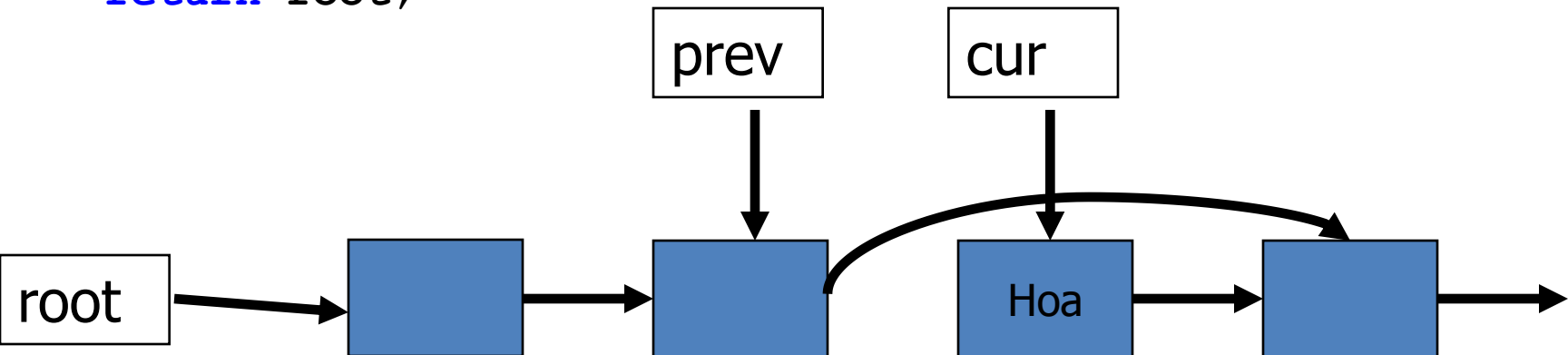
# Xóa nút giữa

```
while (cur != NULL && strcmp(cur->id, id) != 0)
{
    prev = cur;
    cur = cur->next;
}

if (cur != NULL)
{
    prev->next = cur->next;
    free(cur);
}

return root;
```

S  
Hoa



# Xóa nút giữa

```
while (cur != NULL && strcmp(cur->id, id) != 0)
{
    prev = cur;
    cur = cur->next;
}

if (cur != NULL)
{
    prev->next = cur->next;
    free(cur);
}

return root;
```

S  
Hoa

