

CONFIDENTIAL

C Programming Basic – week 3

For Data Structure and Algorithms

Lecturer :

Do Quoc Huy

**Dept of Computer Science
Hanoi University of Technology**



Today's topics

- Cấu trúc tự tham chiếu trong C
- Cấu trúc dữ liệu “danh sách liên kết đơn”
 - Cài đặt danh sách
 - Thuật toán để dò/quét dữ liệu
 - Thuật toán để chèn, xóa
- Cấu trúc dữ liệu “danh sách liên kết đôi”
 - Cài đặt danh sách
 - Thuật toán để dò/quét dữ liệu
 - Thuật toán để chèn, xóa

Cấu trúc tự tham chiếu

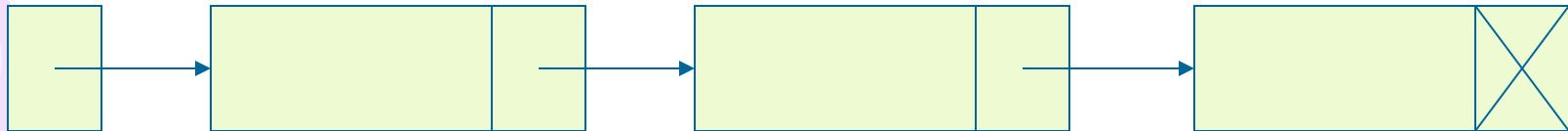
- Một hoặc vài thành phần là con trỏ
trỏ tới bản thân nó.

```
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;
```



Cài đặt danh sách trong C

- “Danh sách - LIST” có nghĩa là cấu trúc dữ liệu chứa thông tin về vị trí của phần tử kế tiếp.
- Các phần tử của “DS liên kết đơn” chỉ có vị trí của phần tử kế tiếp.
- Trong C, con trỏ được sử dụng để trỏ tới vị trí của phần tử tiếp theo.
- **Mảng**: Ta có thể truy cập bất kỳ phần tử nào ngay lập tức.
- **DS liên kết**: Ta có thể thay đổi số dữ liệu trong nó.



Gốc (hay còn gọi là « đầu »)

NULL₄

Khai báo một danh sách liên kết

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

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

Phân phối bộ nhớ cho một phần tử

- Thông qua một con trỏ.

```
struct node * new;
```

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

```
new->element = ...
```

```
new->next = null;
```

- new->addr có nghĩa (*new).addr.

- “biến con trỏ cho cấu trúc bản ghi” -> “tên thành phần”

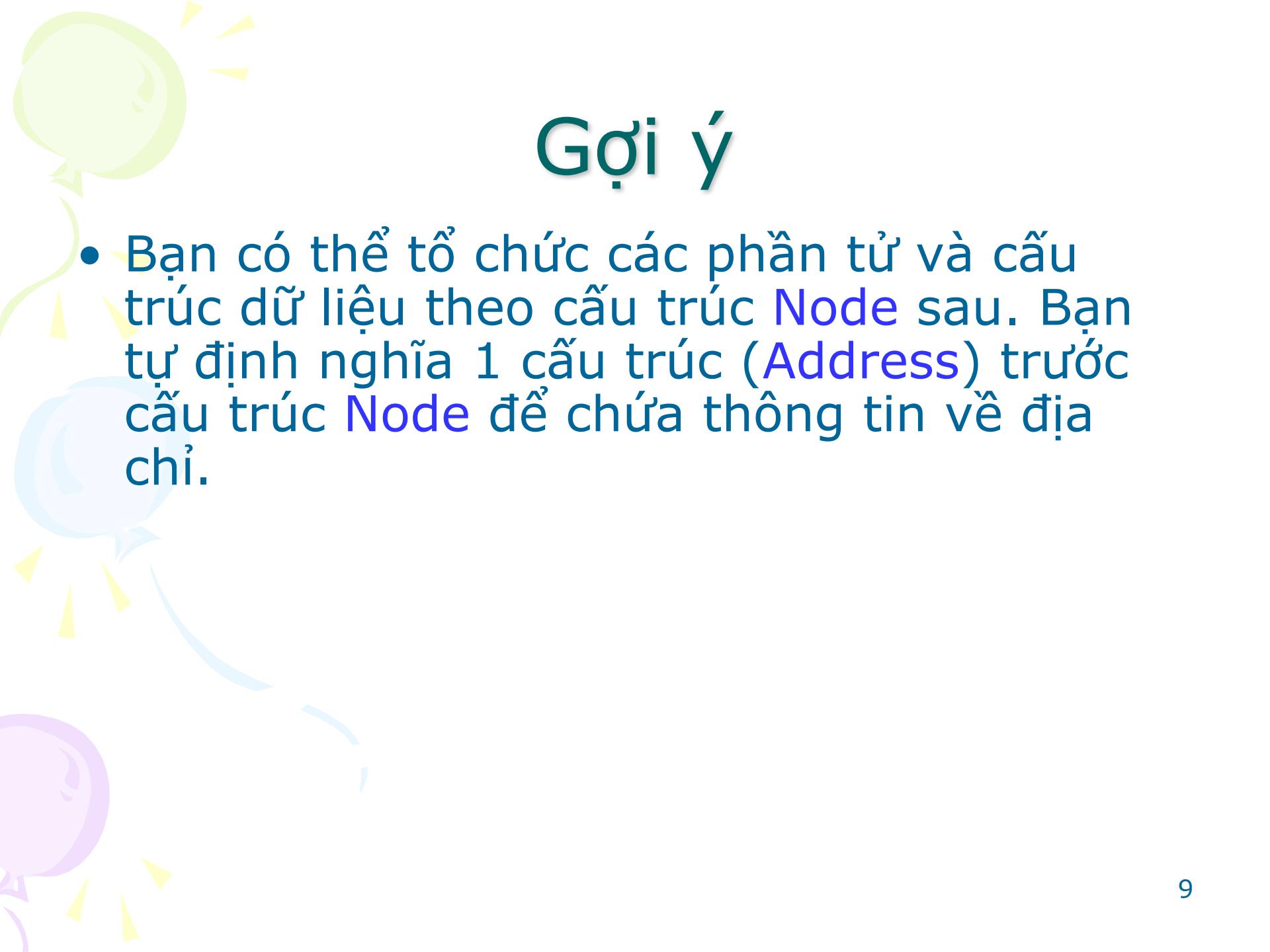
Bài tập 3.1

- Ta thiết kế “address list”(danh sách địa chỉ) cho các số điện thoại di động
- Phải tạo 1 cấu trúc gồm có name, phone number và email address.
- Phải tạo 1 chương trình có thể giải quyết với số lượng dữ liệu tùy ý



Exercise (tiếp)

- Tạo 1 danh sách liên kết đơn chứa danh sách phone address.
- Viết 1 hàm insert 1 phần tử vào list ngay sau phần tử hiện thời, sử dụng nó để thêm node vào list
- Viết 1 hàm duyệt toàn bộ list và in ra thông tin chứa trong nó.
- Viết hàm xoá 1 node khỏi list.



Gợi ý

- Bạn có thể tổ chức các phần tử và cấu trúc dữ liệu theo cấu trúc Node sau. Bạn tự định nghĩa 1 cấu trúc (**Address**) trước cấu trúc **Node** để chứa thông tin về địa chỉ.

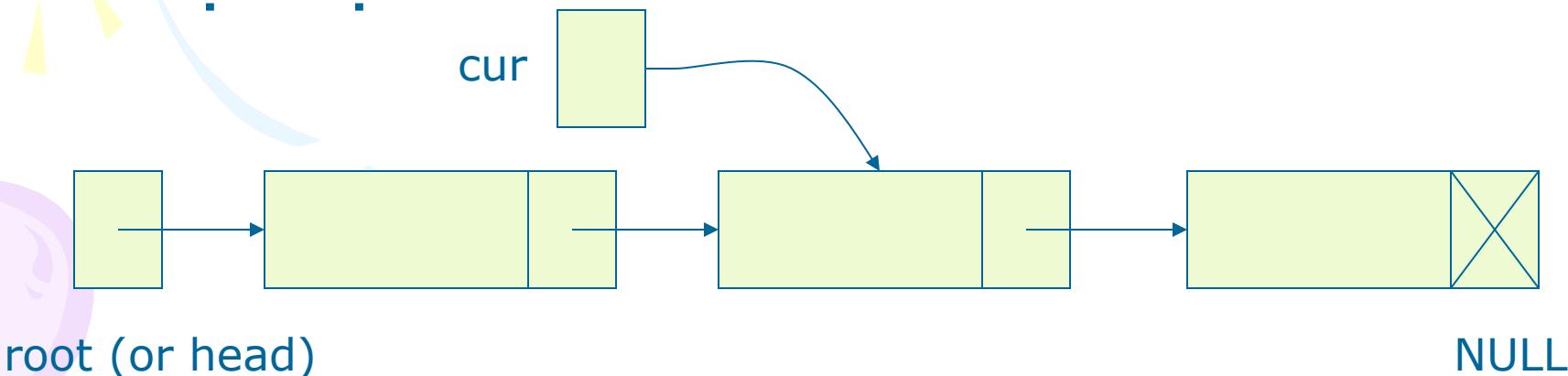
Gợi ý

```
typedef struct Address  
{  
    char name[30];  
    char phone[15];  
    char email[30];  
};
```

```
typedef struct Node  
{  
    Address data;  
    Node *next;  
};
```

3 thửa số quan trọng của 1 list

- Root: đầu của list
- NULL: giá trị của con trỏ. nó báo hiệu kết thúc của list.
- Cur: biến con trỏ lưu giữ phần tử hiện tại.



Link list: insertion(chèn)

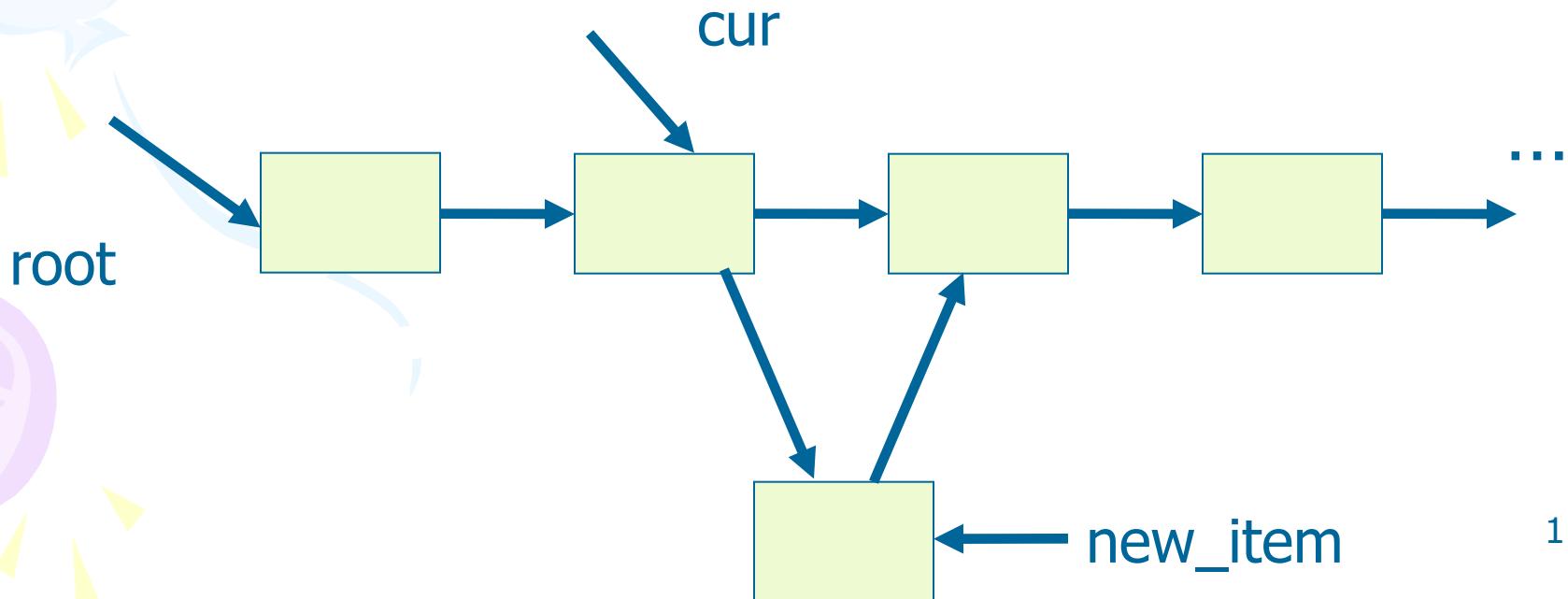
- Chèn ngay sau vị trí hiện tại

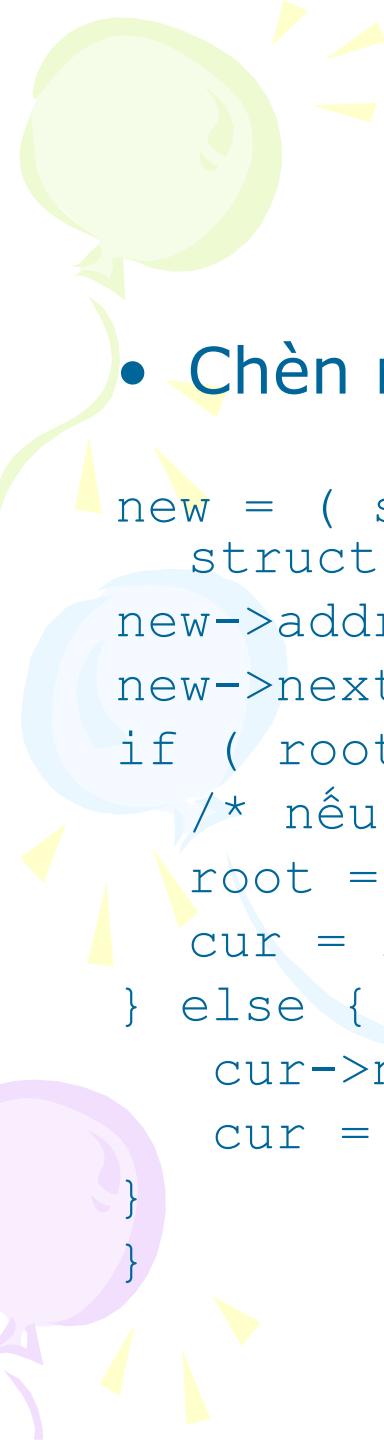
```
create new_item
```

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

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

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





Link list: insertion

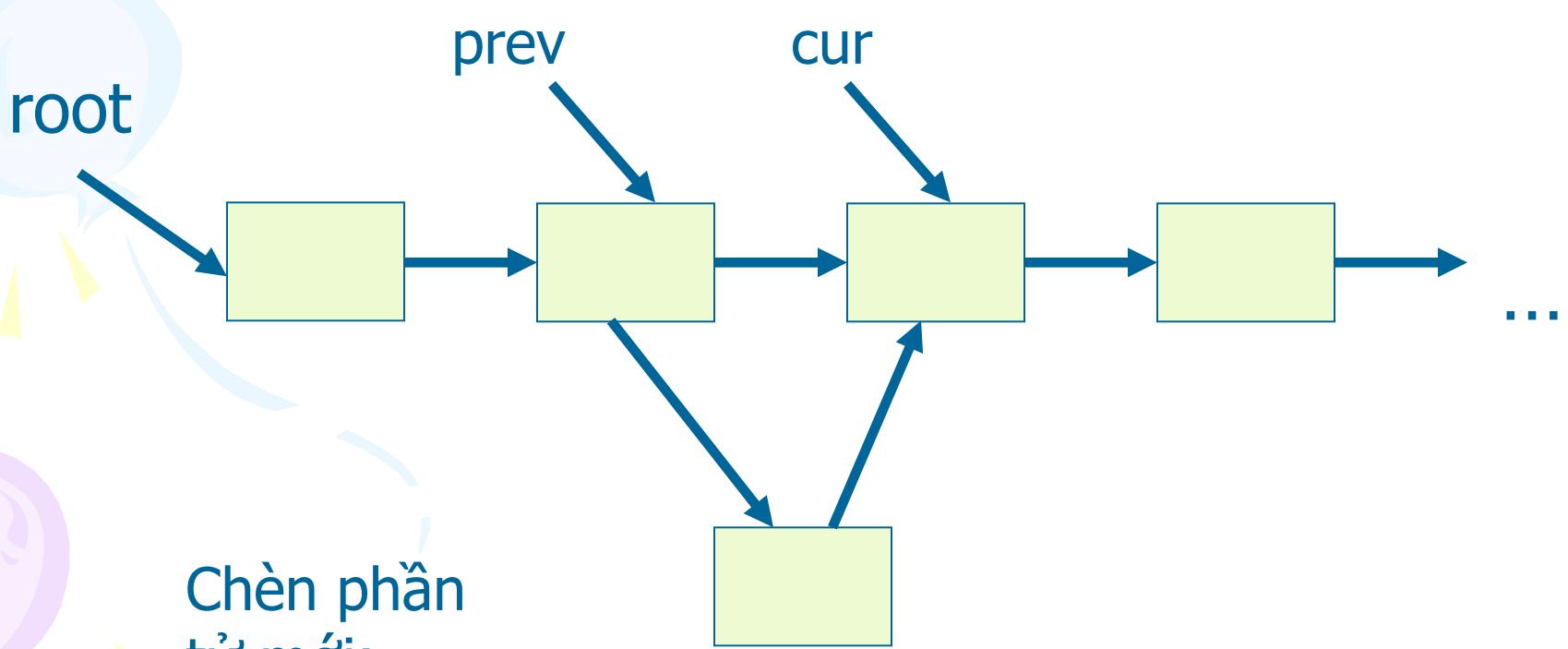
- Chèn ngay sau vị trí hiện tại

```
new = ( struct AddressList * ) malloc( sizeof( struct AddressList ) );
new->addr = addr;
new->next = NULL;
if ( root == NULL ) {
    /* nếu không có phần tử nào*/
    root = new;
    cur = root;
} else {
    cur->next = new;
    cur = cur->next;
}
```

...

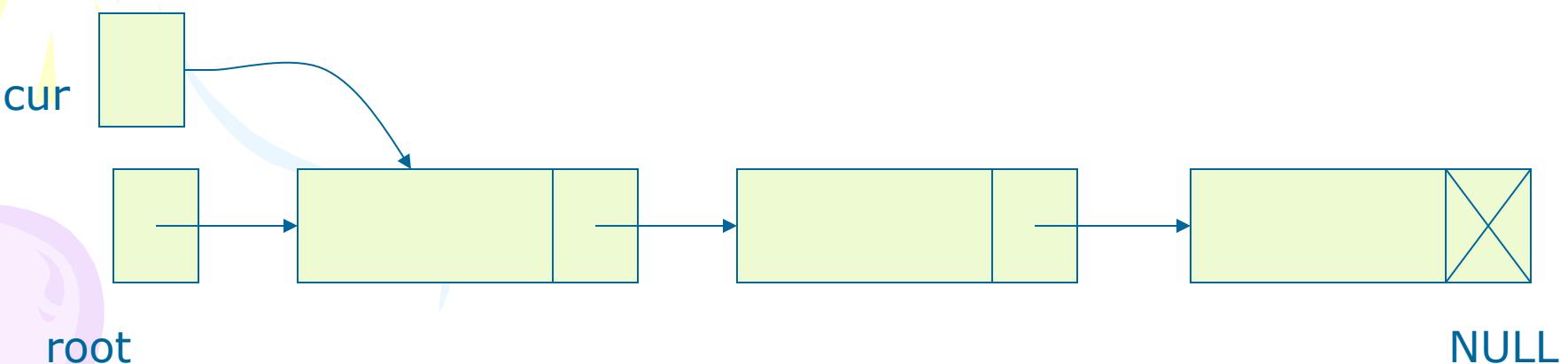
Linked lists – insertion

- Trường hợp khác: Chèn vào trước vị trí hiện tại



Duyệt danh sách

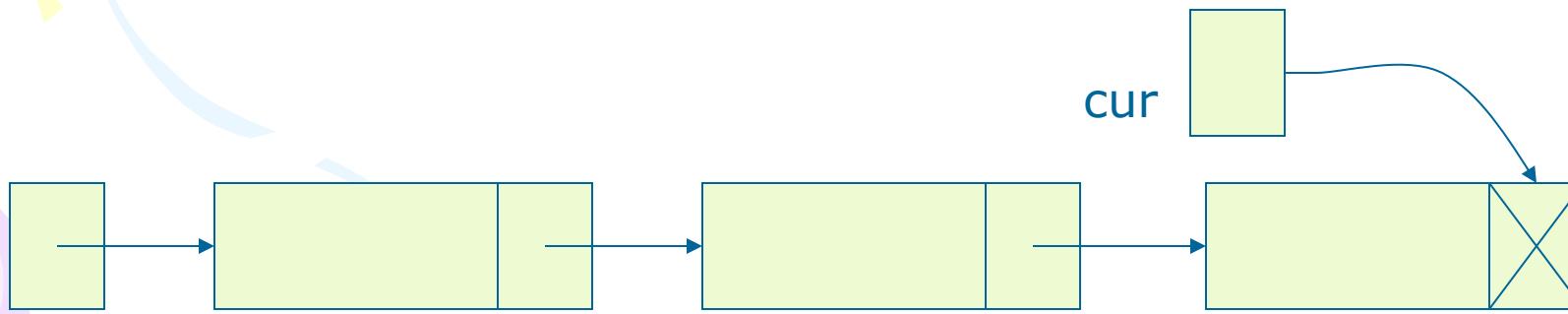
```
for ( cur = root; cur != NULL; cur = cur->next ) {  
    showAddress( cur->addr, stdout );  
}
```



NULL

Duyệt danh sách (tiếp)

- Thay đổi giá trị của biến con trỏ cur trong dãy
- Các biến này được gọi là “biến lặp”
- Hoàn thành duyệt khi giá trị cur = NULL



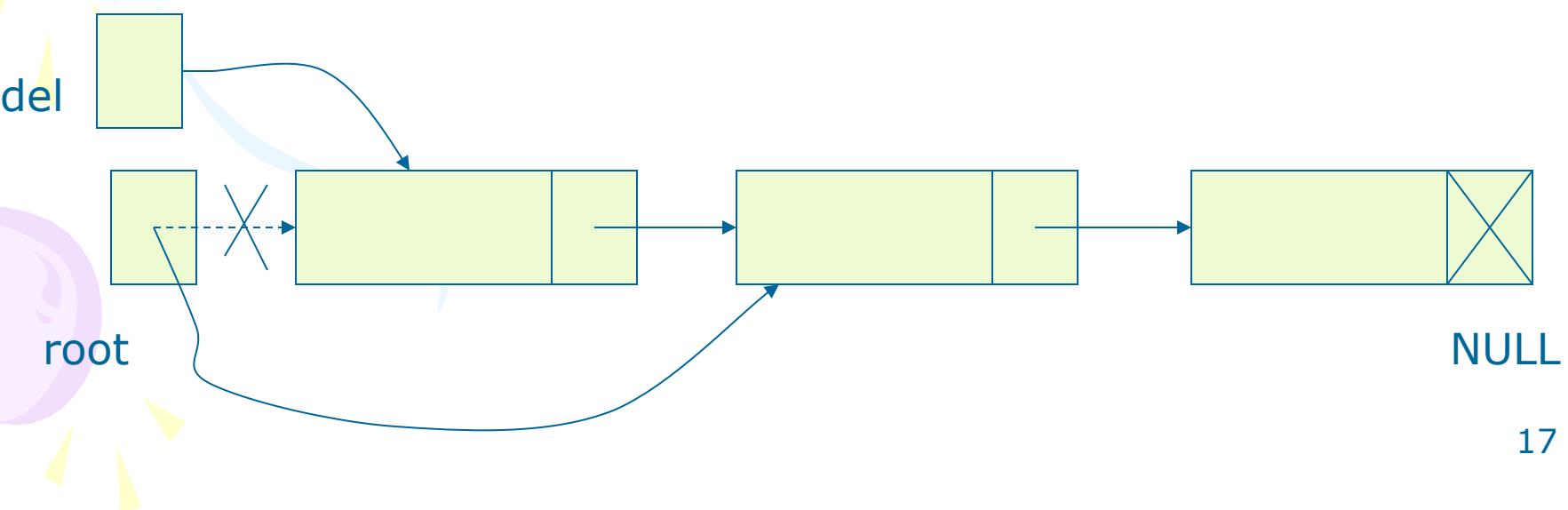
root

cur

NULL

Xóa

- Khi xóa phần tử đầu
 $\text{root} = \text{del} \rightarrow \text{next}; \text{free}(\text{del});$
- Thay đổi giá trị của “root” vào giá trị của “next” được trả bởi “del.”

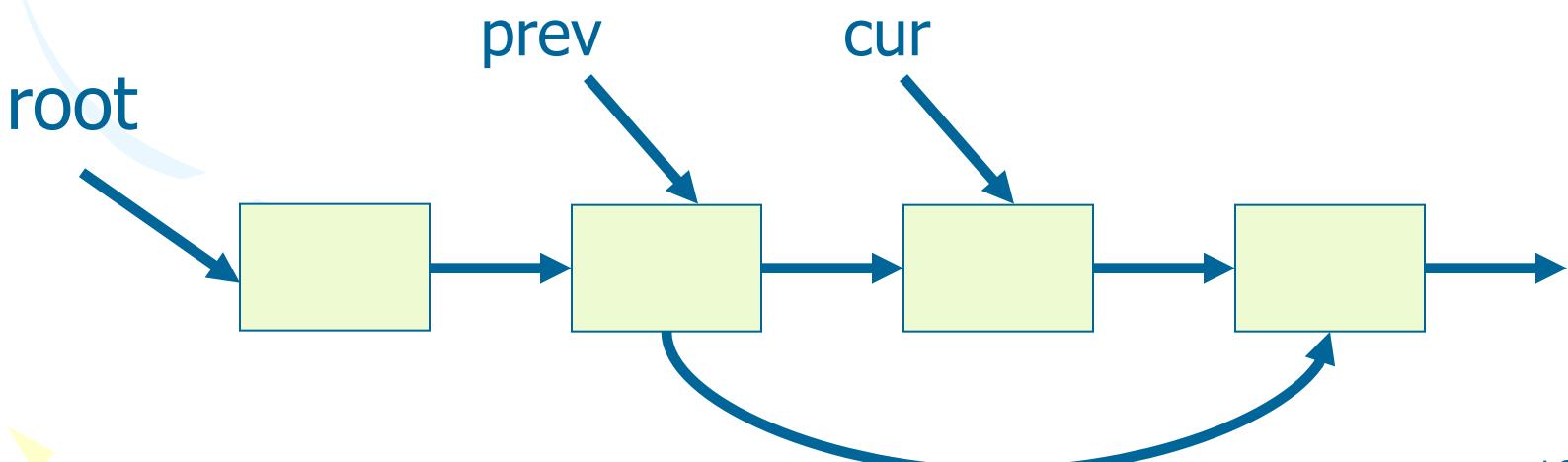


NULL

Xóa từ giữa

- Xóa nốt được trỏ bởi `cur`
- Xác định `prev` trỏ tới node ngay trước node cần xóa

```
prev->next = cur->next;  
free(cur);
```



Exercise

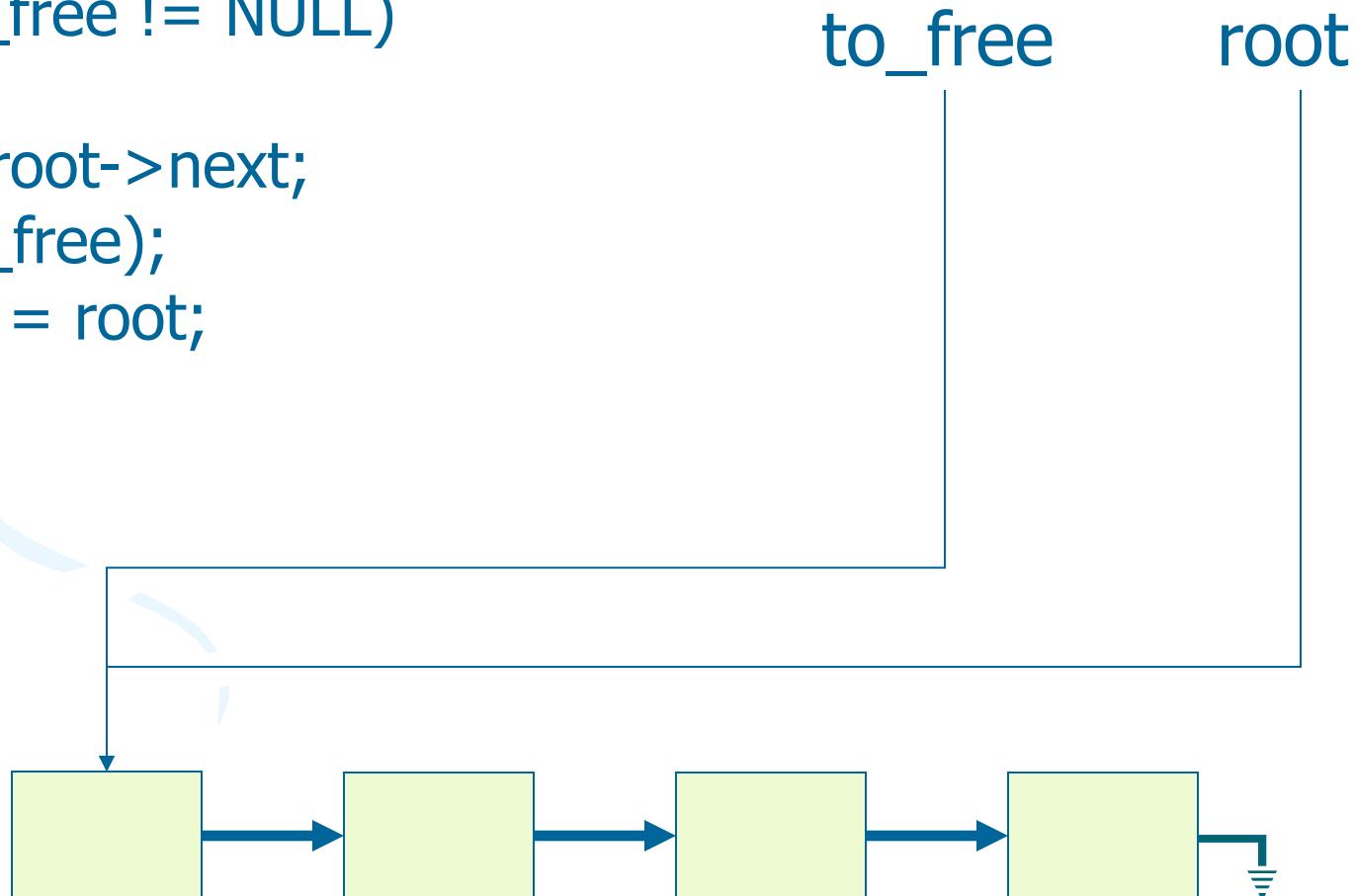
- Tạo hàm insert, delete với tham số n (nguyên) chỉ ra vị trí của node bị tác động đến.
- Vị trí đầu tiên là 0
- n = 1: thêm phần tử vào sau phần tử đầu tiên.
- n = 2: thêm phần tử vào sau phần tử thứ 2.

```
struct AddressList *insert (struct AddressList  
*root, struct Address ad, int n);
```

```
struct AddressList *delete(struct AddressList  
*root, int n);
```

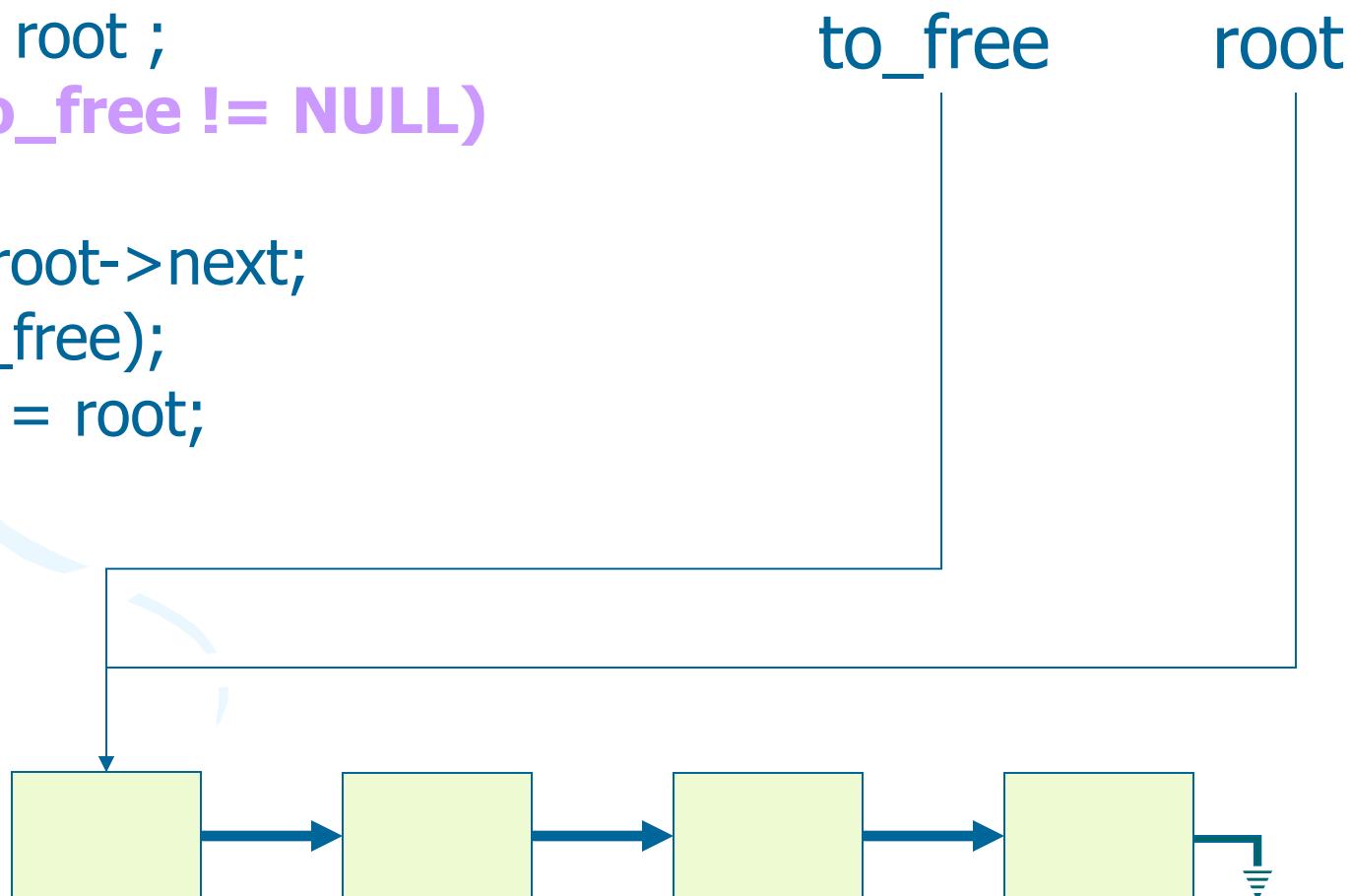
Giải phóng list

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



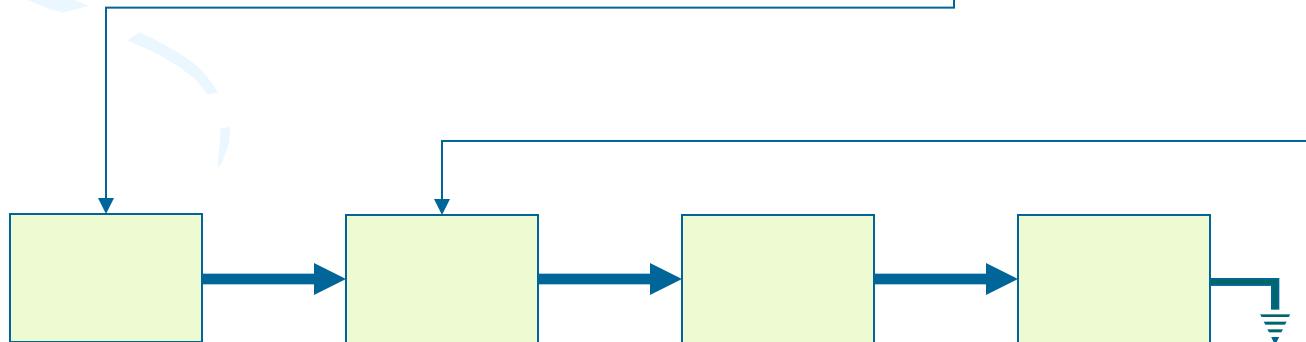
Giải phóng list

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



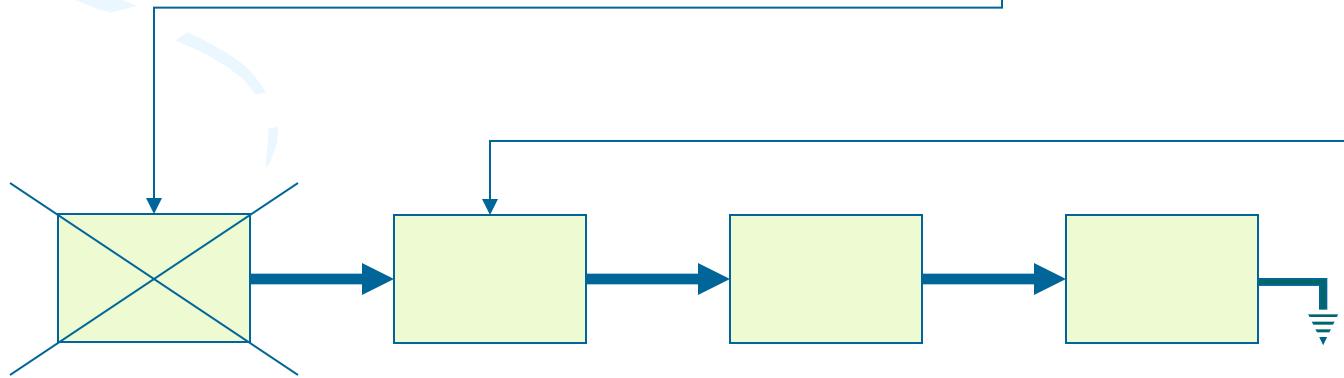
Giải phóng list

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



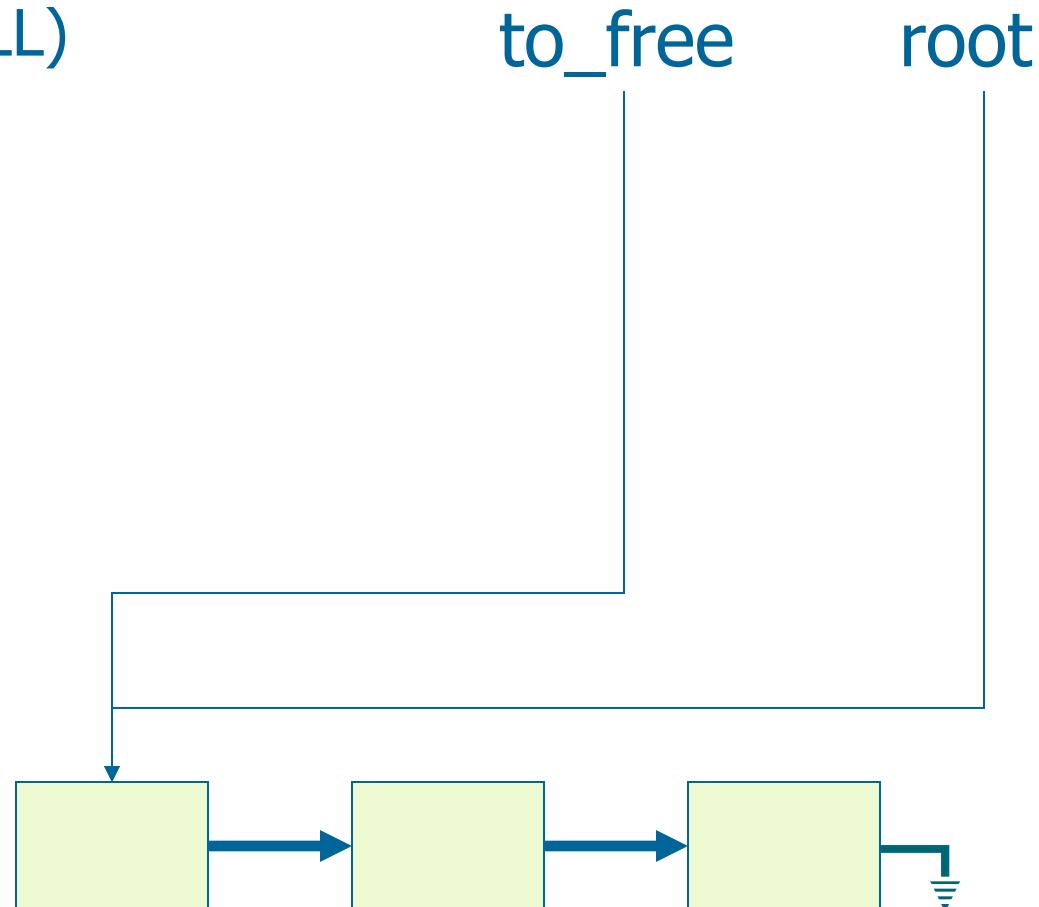
Giải phóng list

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



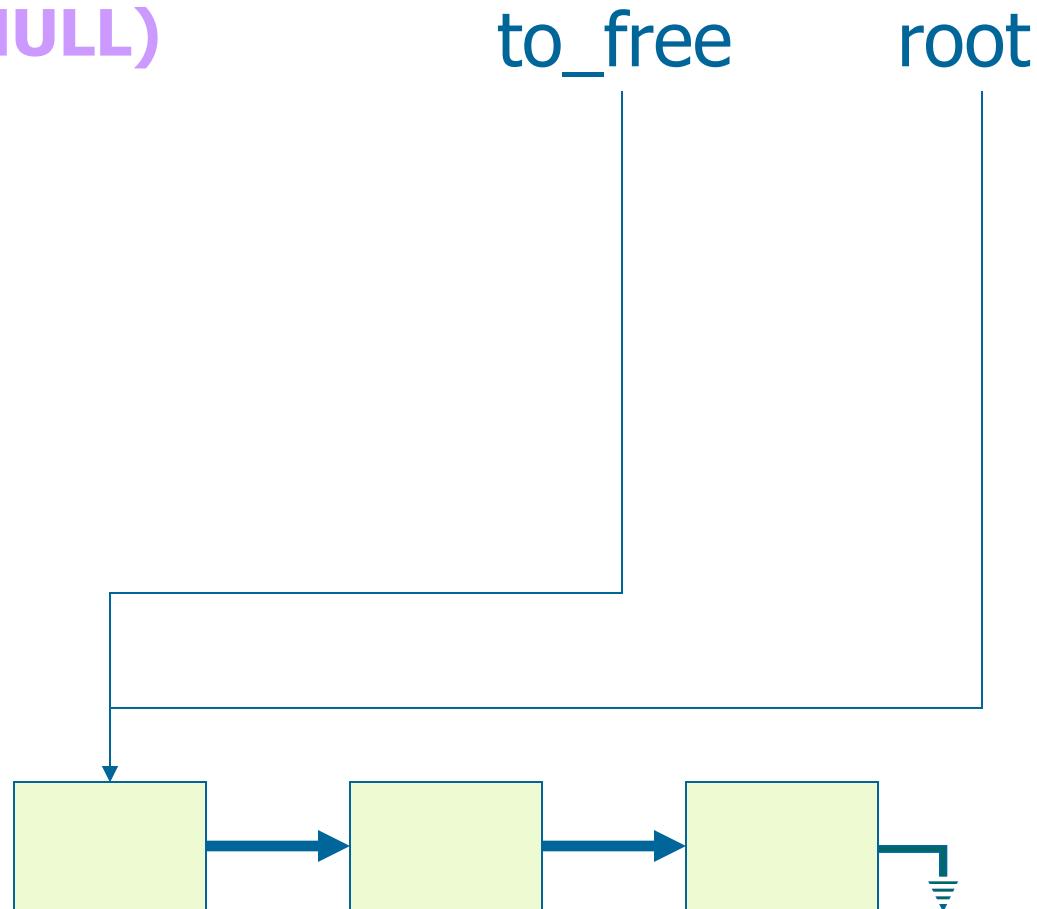
Giải phóng list

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



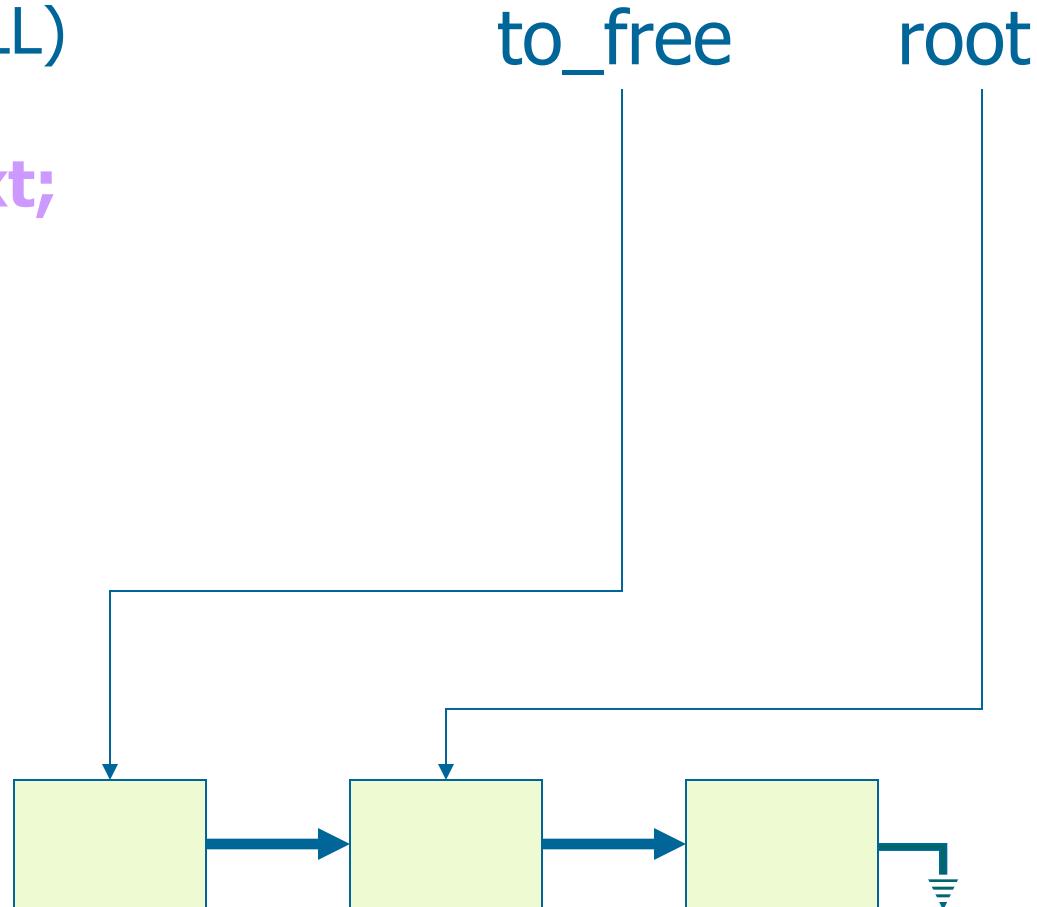
Giải phóng list

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



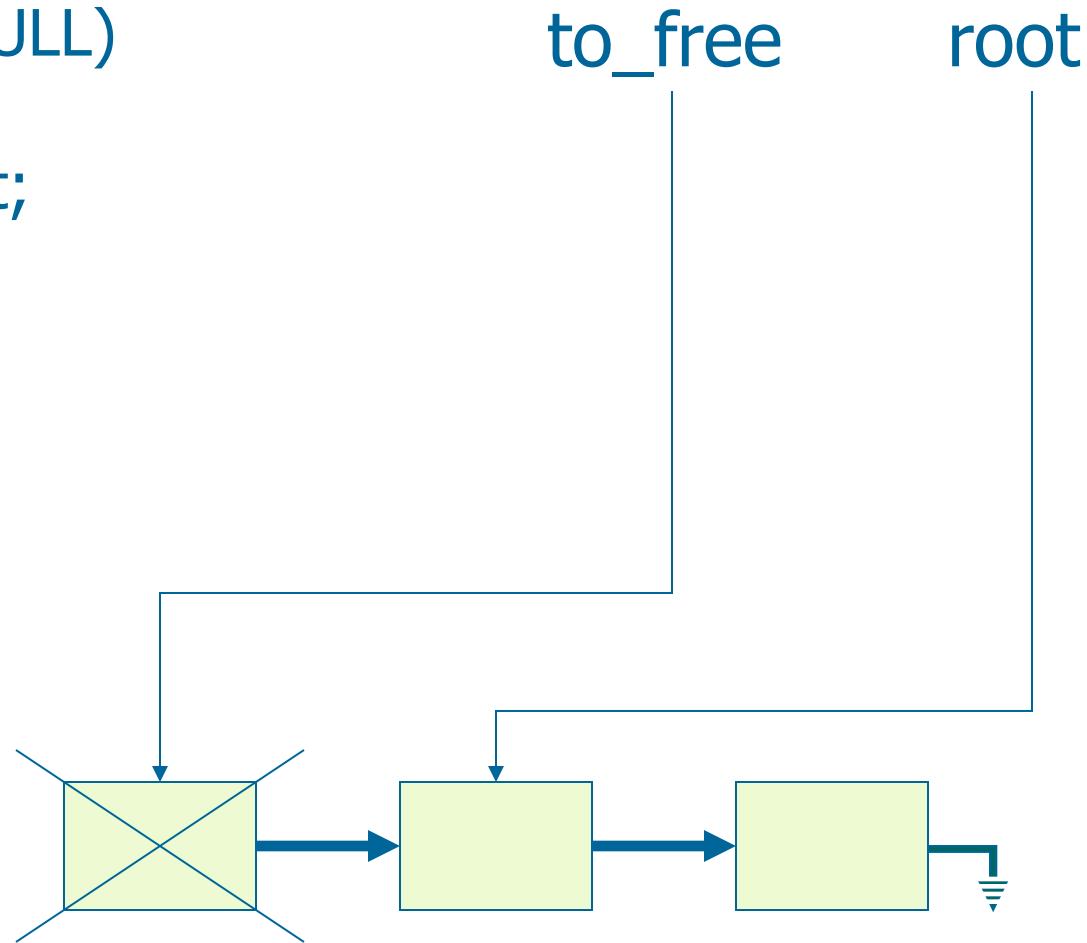
Giải phóng list

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



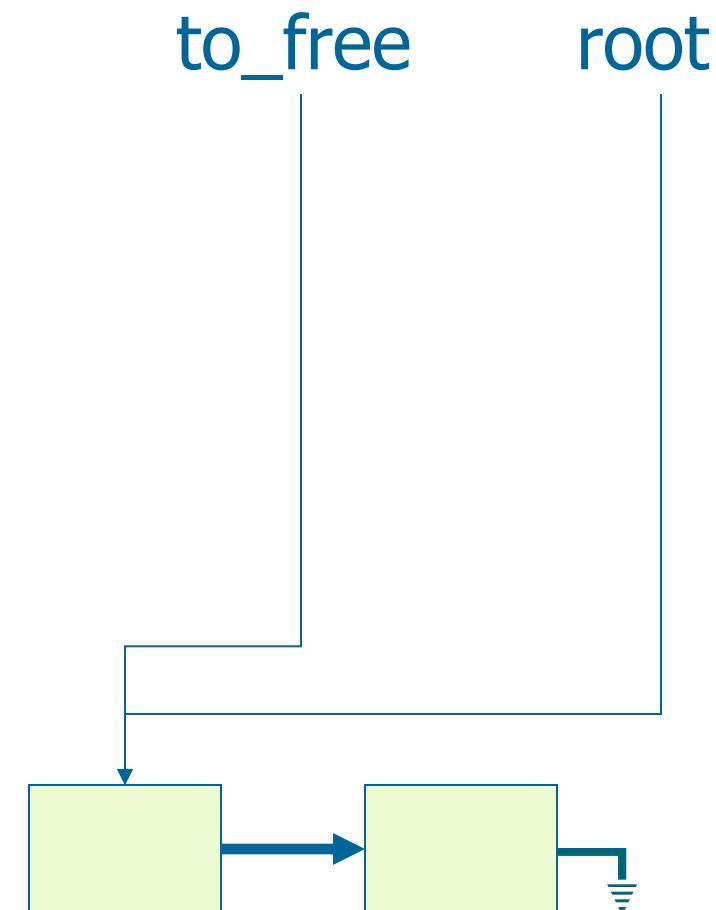
Giải phóng list

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



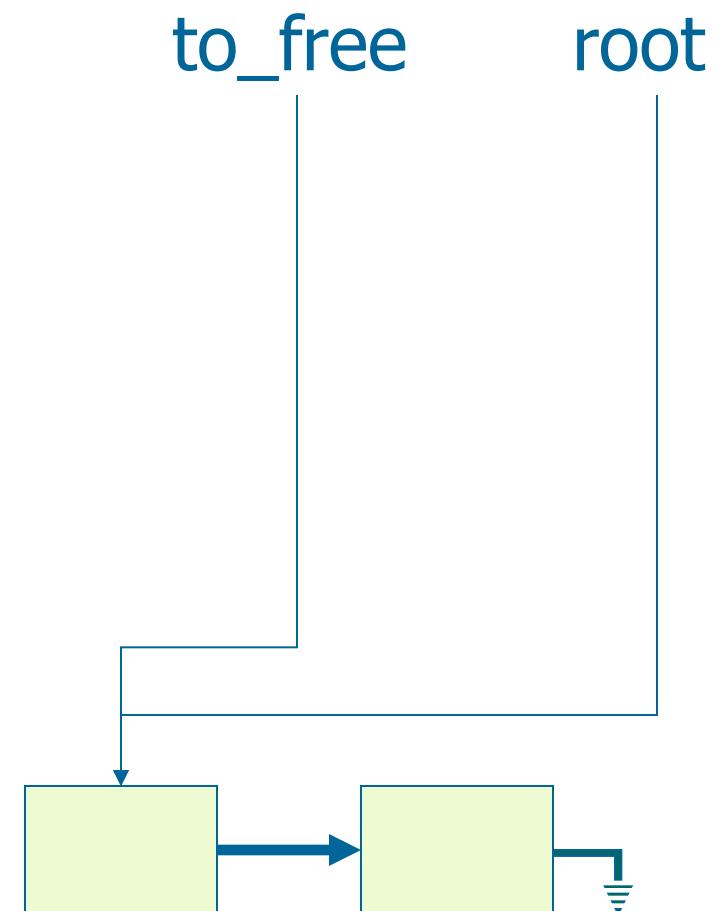
Giải phóng list

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



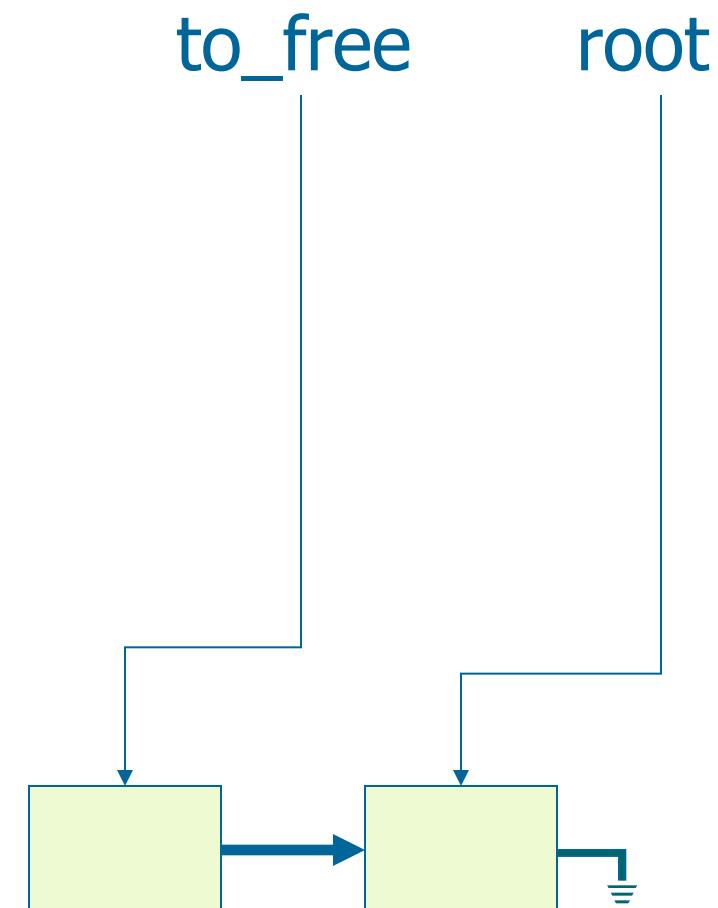
Giải phóng list

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



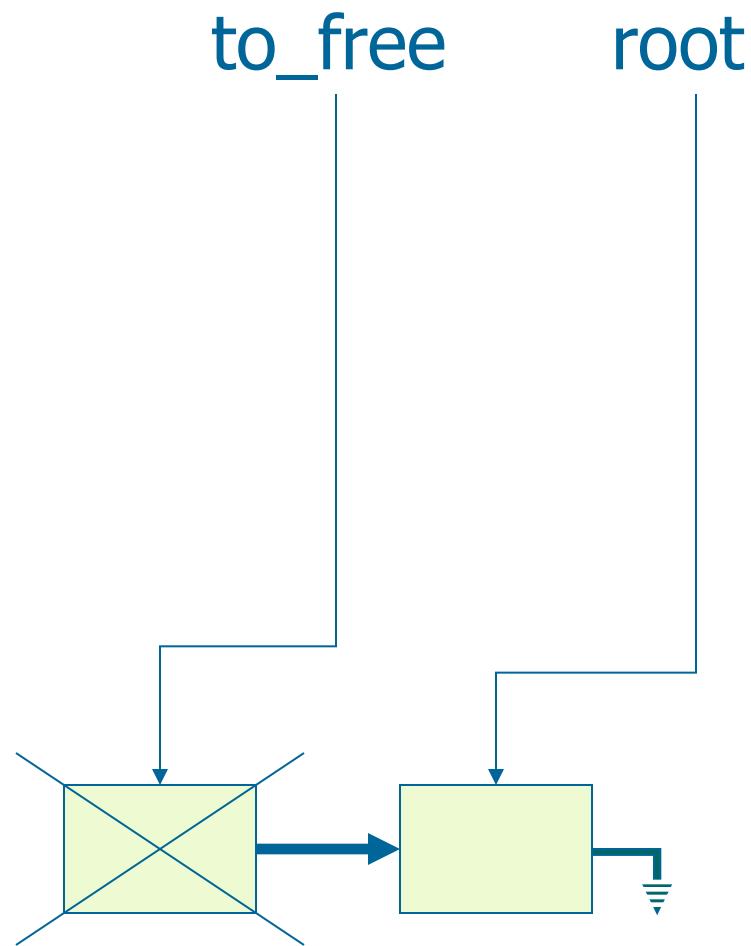
Giải phóng list

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



Giải phóng list

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



Giải phóng list

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

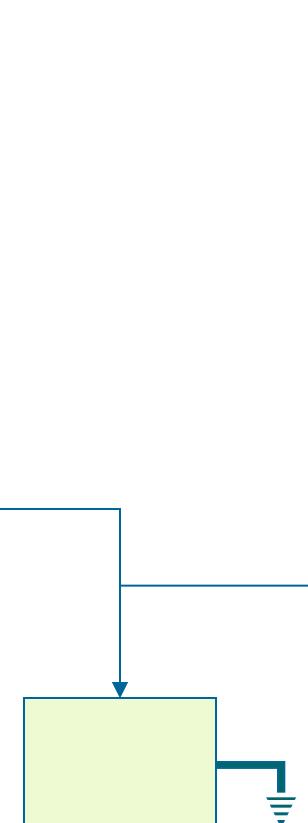
to_free root



Giải phóng list

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

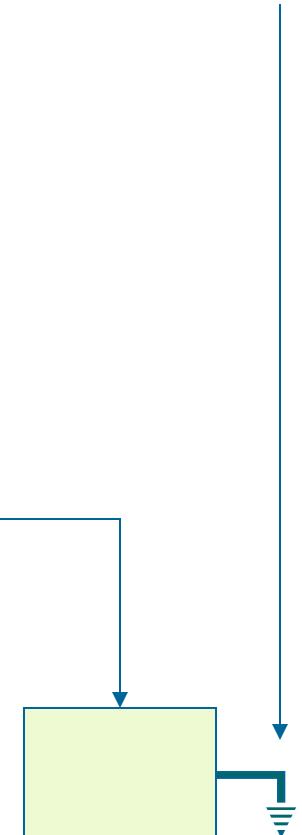
to_free root



Giải phóng list

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

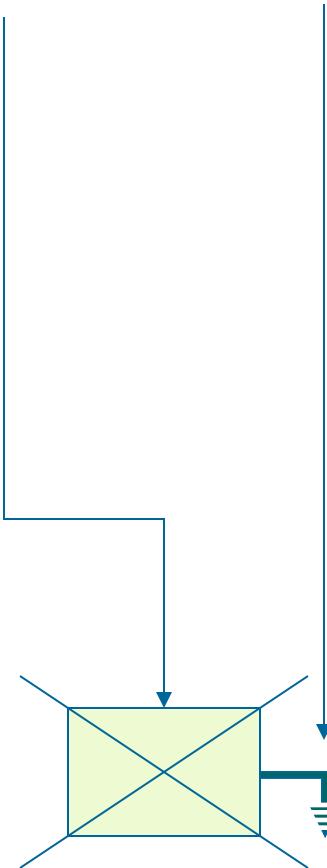
to_free root



Giải phóng list

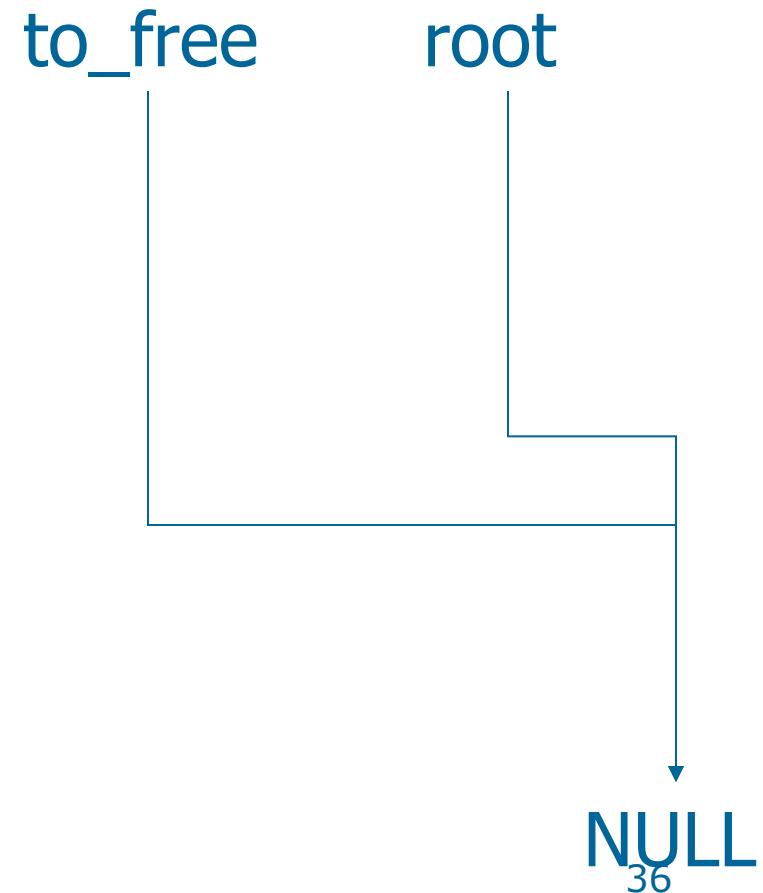
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```

to_free root



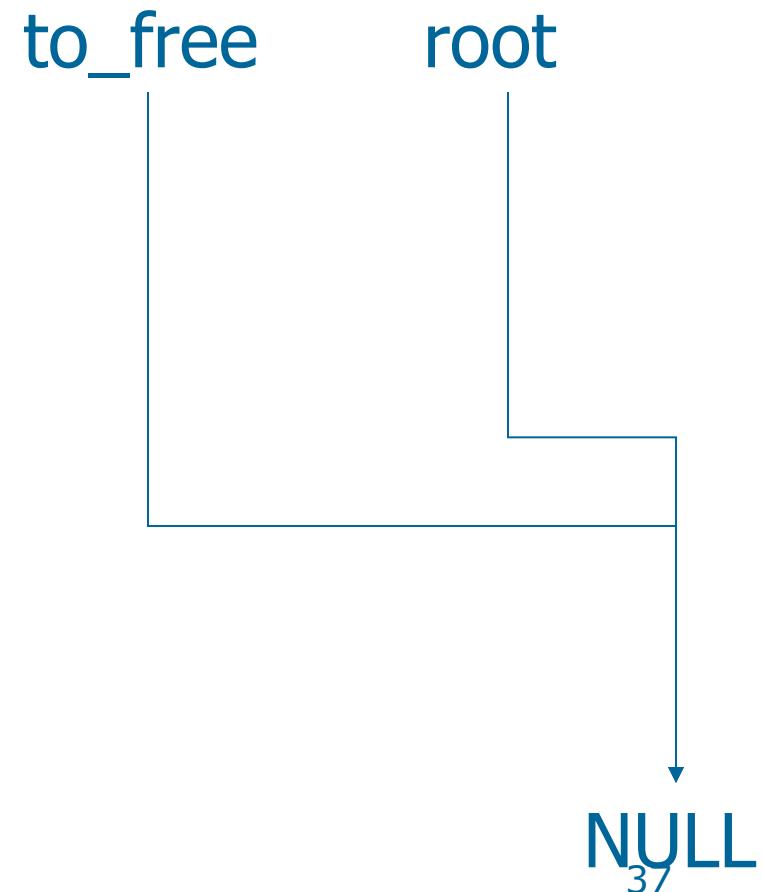
Giải phóng list

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



Giải phóng list

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



Exercise: Đảo một danh sách

- Cho danh sách được định nghĩa như sau:

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

- Viết hàm đảo ngược danh sách này.

Exercise 3-3

- Xây dựng 1 chương trình quản lý sinh viên đơn giản sử dụng linked list gồm node có cấu trúc như sau:

```
typedef struct Student_t {  
    char    id[ID_LENGTH];  
    char    name[NAME_LENGTH];  
    int     grade;  
  
    struct Student_t *next;  
} Student;
```

Exercise 3-3

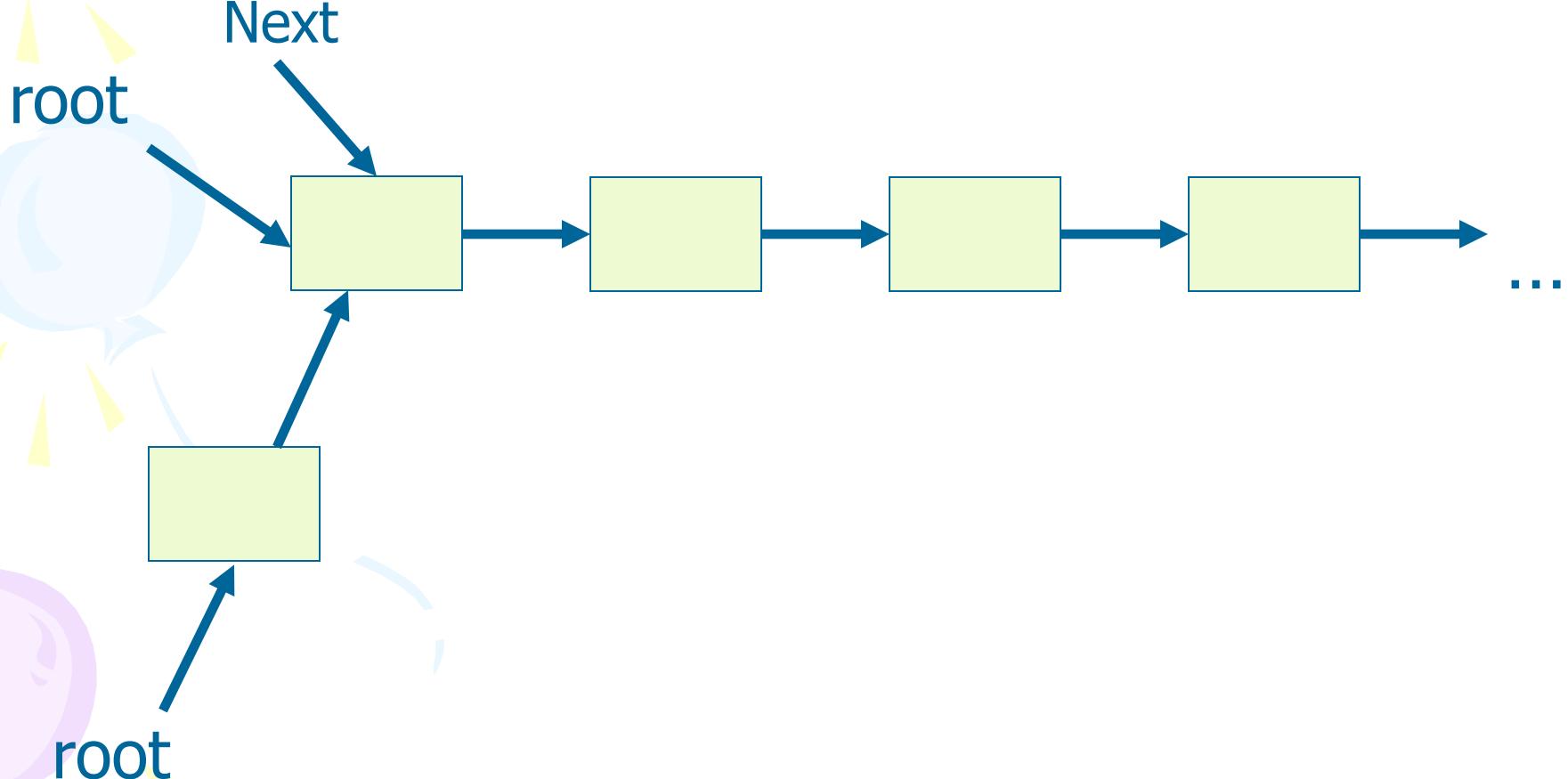
Yêu cầu:

List được sắp xếp theo thứ tự giảm dần của điểm sinh viên.

Chương trình cung cấp các chức năng sau:

- Chèn new student (khi chèn thì trước hết phải tìm ra vị trí đúng của student đó đã rồi mới chèn).
- Tìm 1 student qua ID: trả về con trỏ.
- Xoá 1 student (xác định qua ID) khỏi list

Thêm một sinh viên- phần đầu danh sách



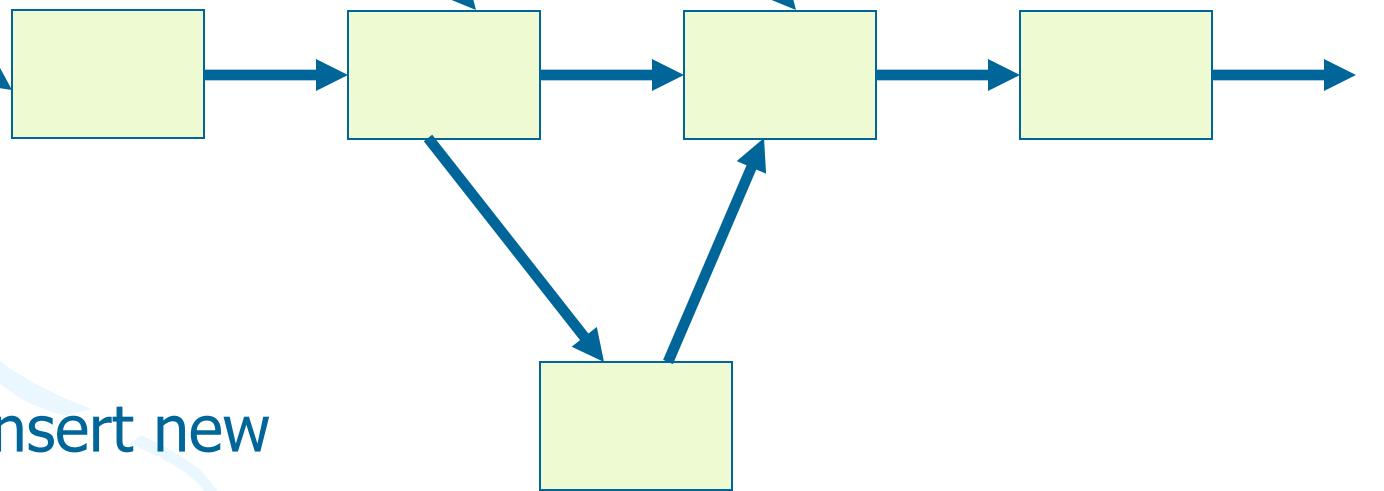
Thêm một sinh viên- mid/end

root

Insert new
item:

Previous

Next



```
Student *add_student(Student *root, Student *to_add)
{
    Student *curr_std, *prev_std = NULL;

    if (root == NULL)
        return to_add; ← Xử lý danh sách rỗng

    if (to_add->grade > root->grade)
    {
        to_add->next = root;
        return to_add; ← Xử lý phần đầu
    }

    curr_std = root;
    while (curr_std != NULL && to_add->grade < curr_std->grade)
    {
        prev_std = curr_std;
        curr_std = curr_std->next;
    }

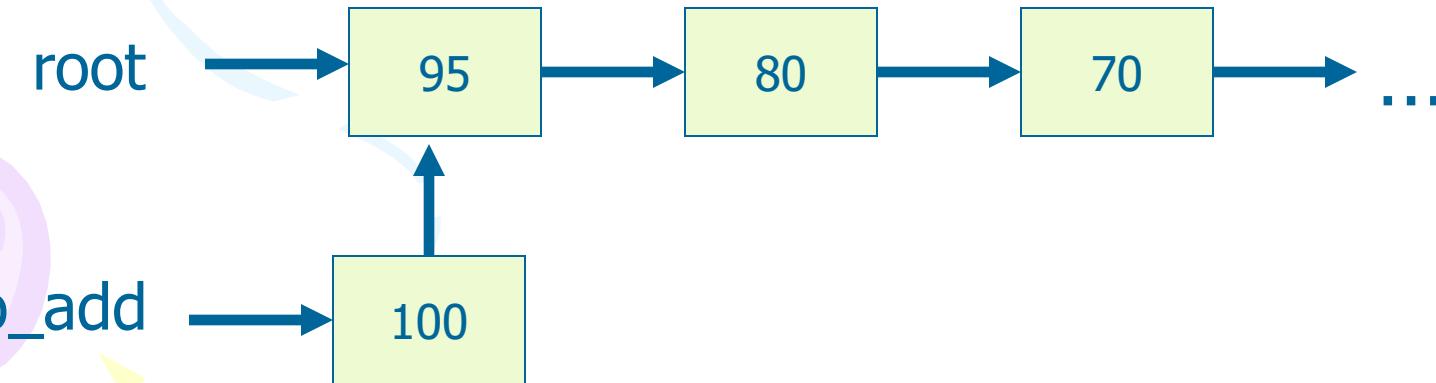
    prev_std->next = to_add;
    to_add->next = curr_std;

    return root;
}
```

the rest

Adding a student - beginning

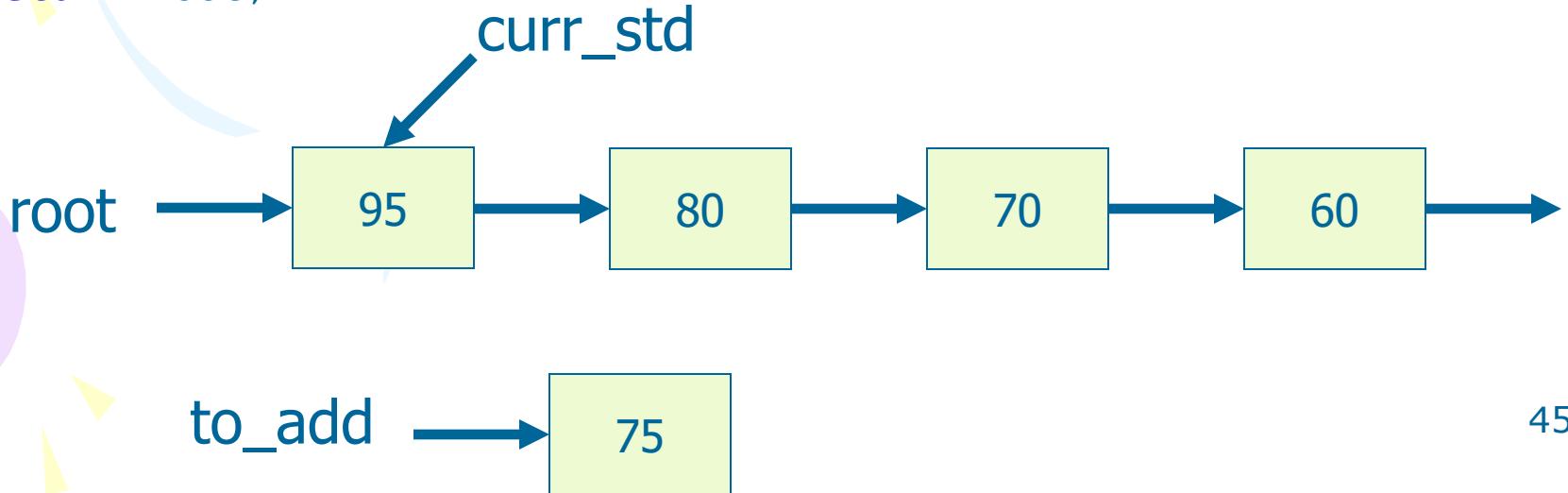
```
if (root == NULL)  
    return to_add;  
  
if (to_add->grade > root->grade)  
{  
    to_add->next = root;  
    return to_add;  
}  
}
```



Adding a student - mid / end

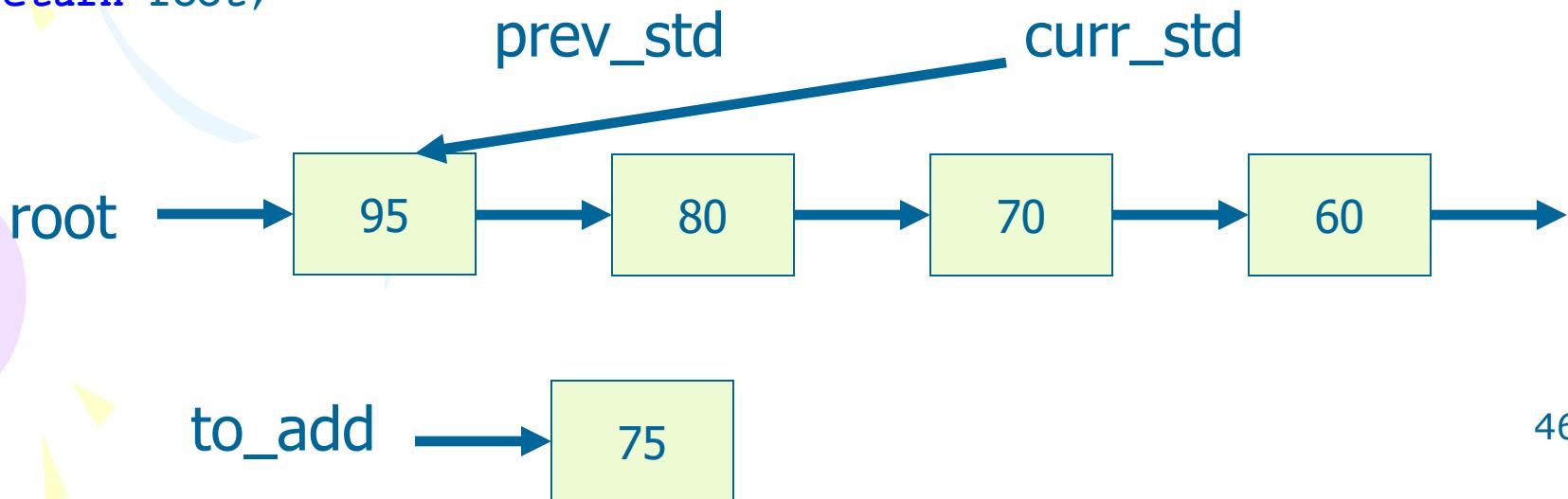
```
curr_std = root;
while (curr_std != NULL && to_add->grade < curr_std->grade)
{
    prev_std = curr_std;
    curr_std = curr_std->next;
}

prev_std->next = to_add;
to_add->next = curr_std;
return root;
```



Adding a student - mid / end

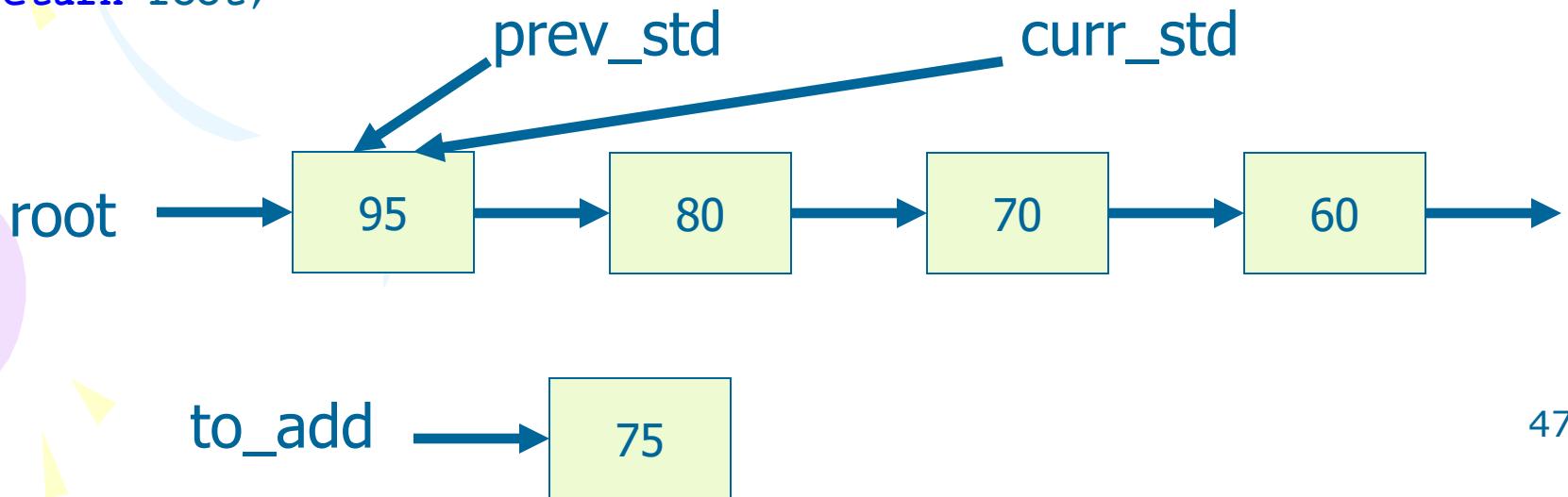
```
curr_std = root;  
while (curr_std != NULL && to_add->grade < curr_std->grade)  
{  
    prev_std = curr_std;  
    curr_std = curr_std->next;  
}  
  
prev_std->next = to_add;  
to_add->next = curr_std;  
return root;
```



Adding a student - mid / end

```
curr_std = root;
while (curr_std != NULL && to_add->grade < curr_std->grade)
{
    prev_std = curr_std;
    curr_std = curr_std->next;
}

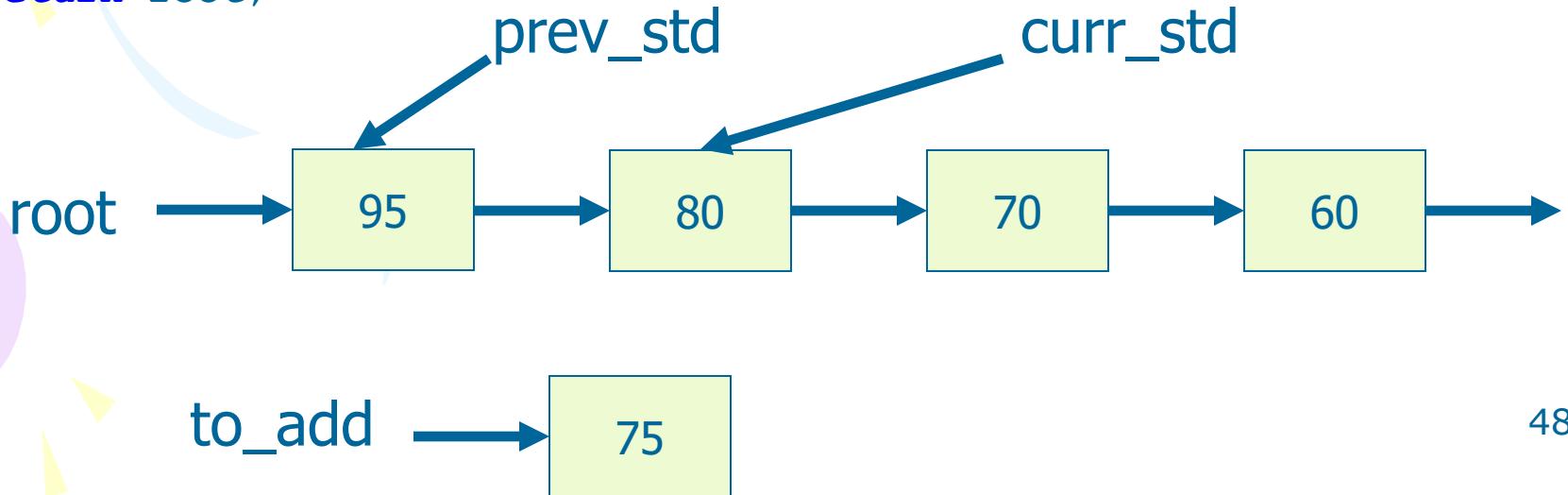
prev_std->next = to_add;
to_add->next = curr_std;
return root;
```



Adding a student - mid / end

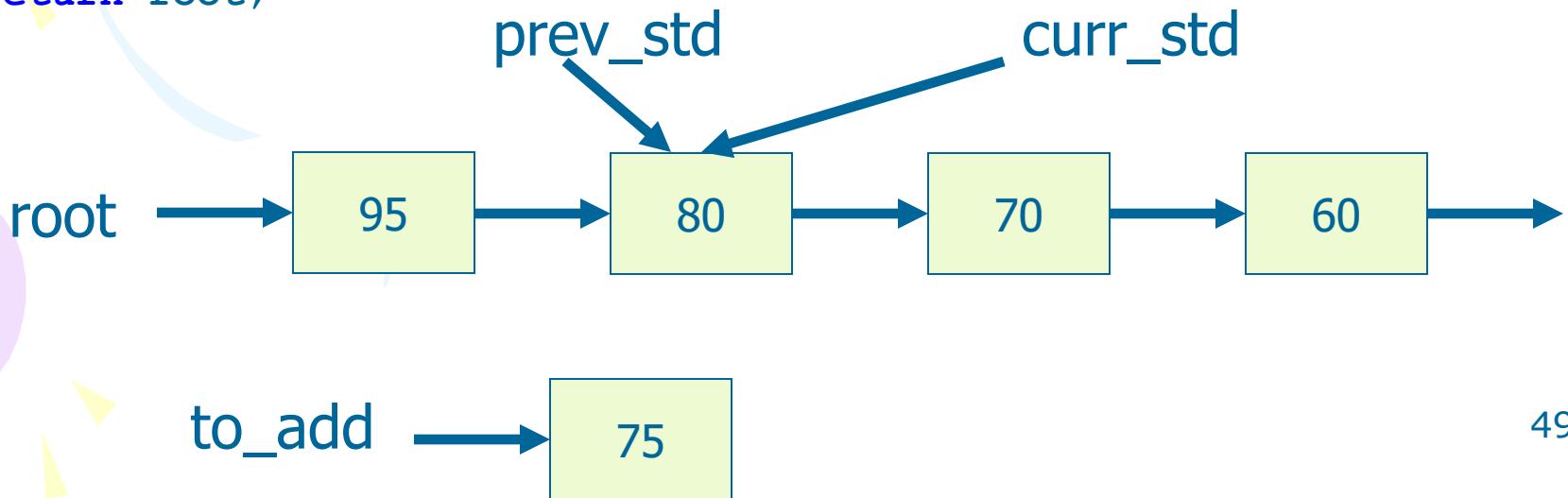
```
curr_std = root;
while (curr_std != NULL && to_add->grade < curr_std->grade)
{
    prev_std = curr_std;
    curr_std = curr_std->next;
}

prev_std->next = to_add;
to_add->next = curr_std;
return root;
```



Adding a student - mid / end

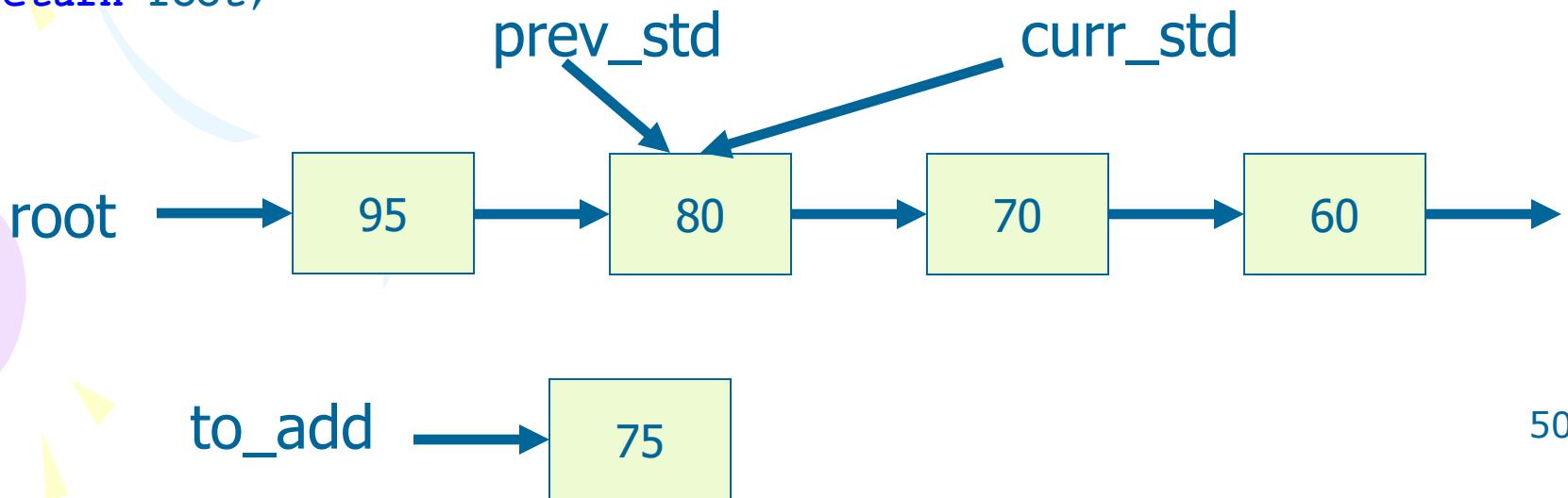
```
curr_std = root;  
while (curr_std != NULL && to_add->grade < curr_std->grade)  
{  
    prev_std = curr_std;  
    curr_std = curr_std->next;  
}  
  
prev_std->next = to_add;  
to_add->next = curr_std;  
return root;
```



Adding a student - mid / end

```
curr_std = root;
while (curr_std != NULL && to_add->grade < curr_std->grade)
{
    prev_std = curr_std;
    curr_std = curr_std->next;
}

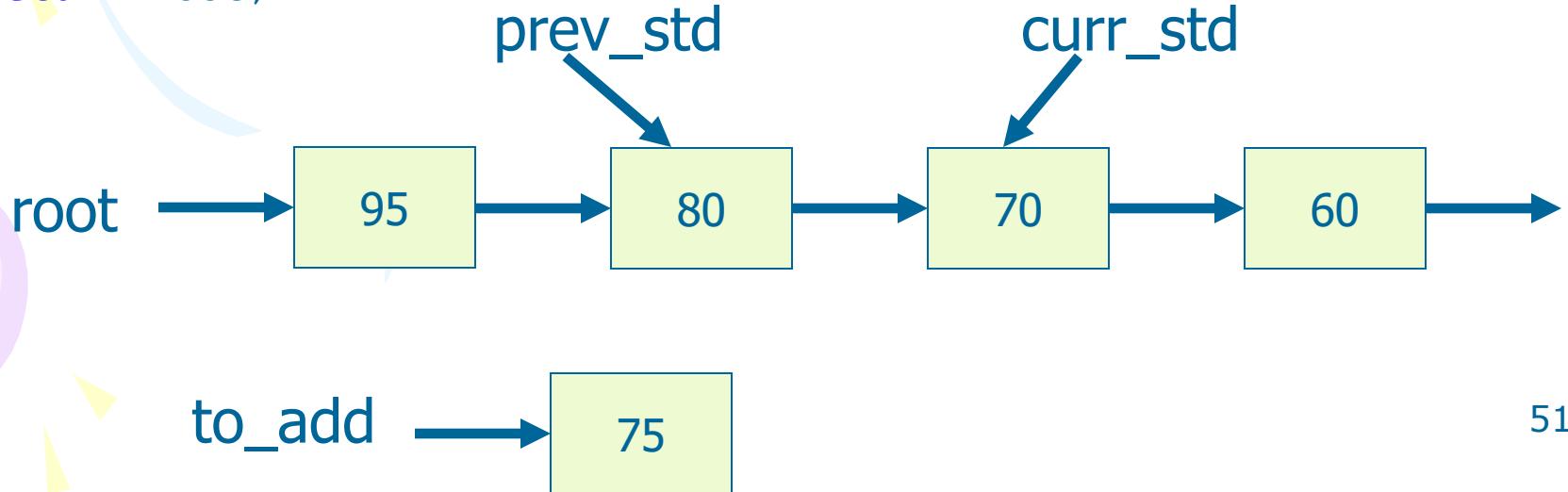
prev_std->next = to_add;
to_add->next = curr_std;
return root;
```



Adding a student - mid / end

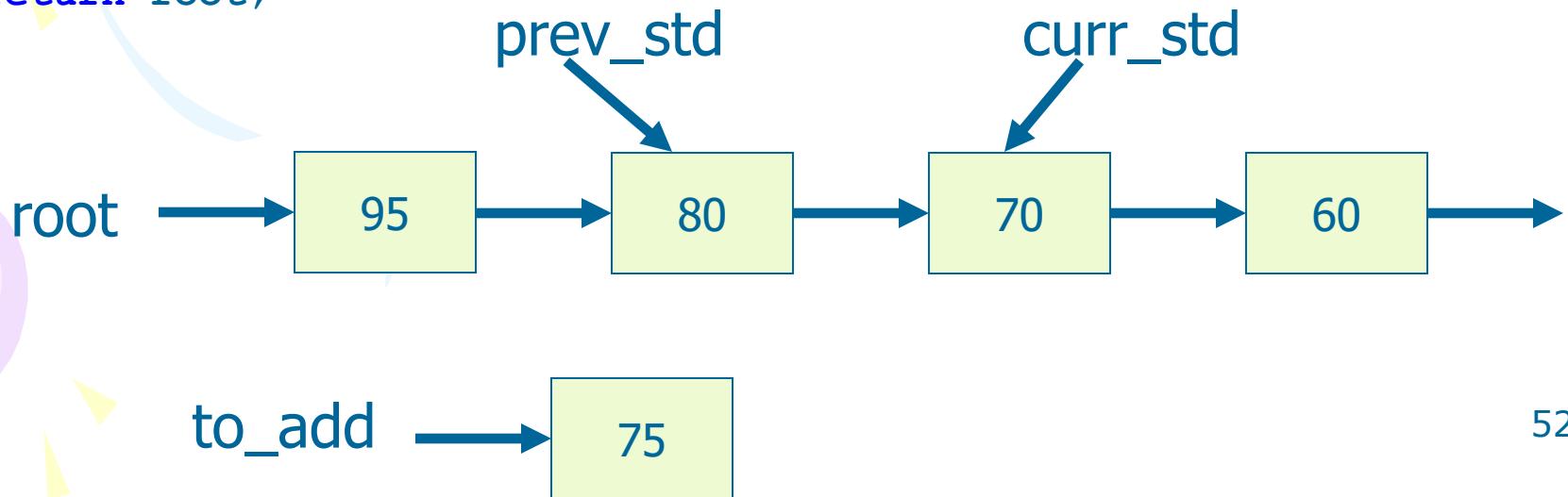
```
curr_std = root;
while (curr_std != NULL && to_add->grade < curr_std->grade)
{
    prev_std = curr_std;
    curr_std = curr_std->next;
}

prev_std->next = to_add;
to_add->next = curr_std;
return root;
```



Adding a student - mid / end

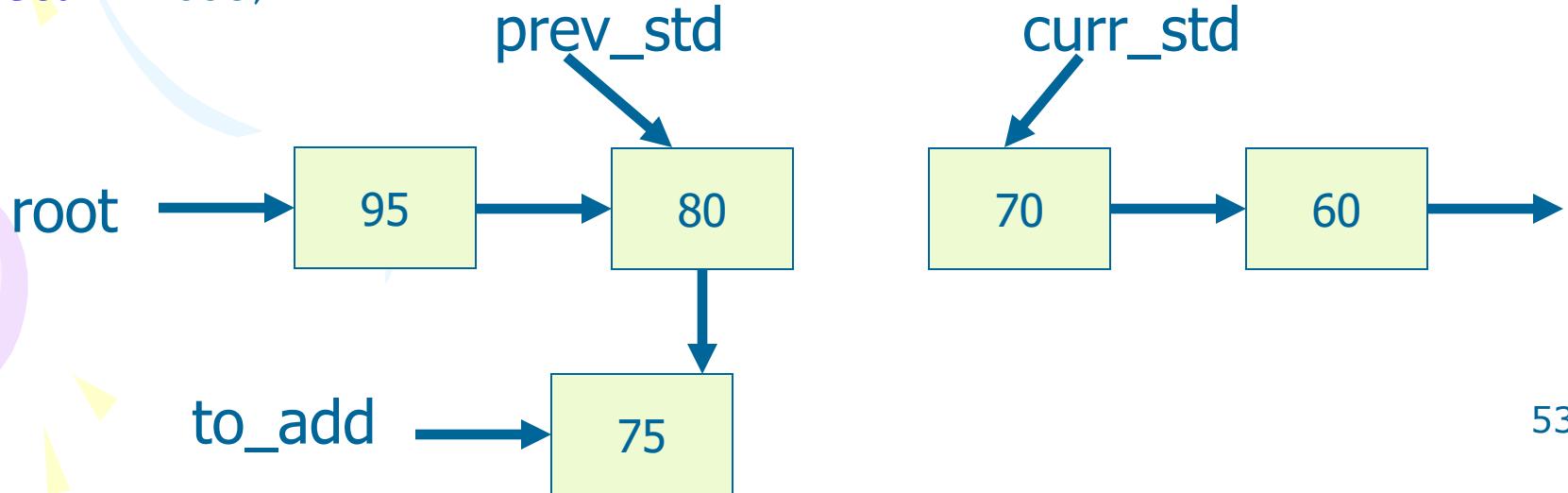
```
curr_std = root;  
while (curr_std != NULL && to_add->grade < curr_std->grade)  
{  
    prev_std = curr_std;  
    curr_std = curr_std->next;  
}  
  
prev_std->next = to_add;  
to_add->next = curr_std;  
return root;
```



Adding a student - mid / end

```
curr_std = root;
while (curr_std != NULL && to_add->grade < curr_std->grade)
{
    prev_std = curr_std;
    curr_std = curr_std->next;
}

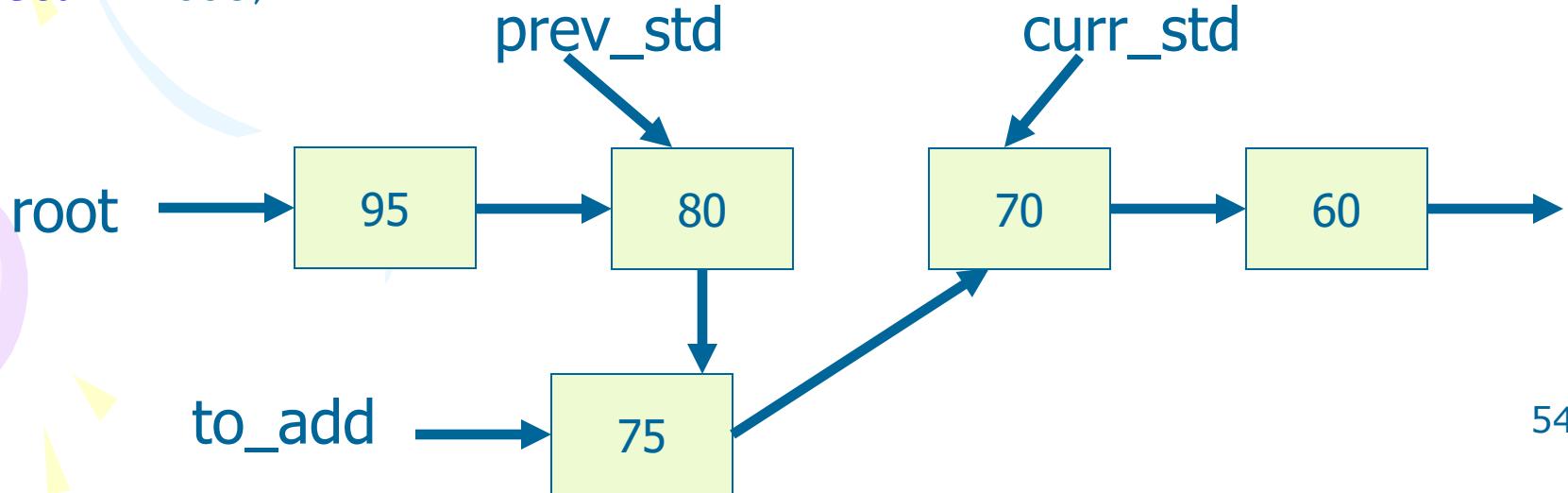
prev_std->next = to_add;
to_add->next = curr_std;
return root;
```

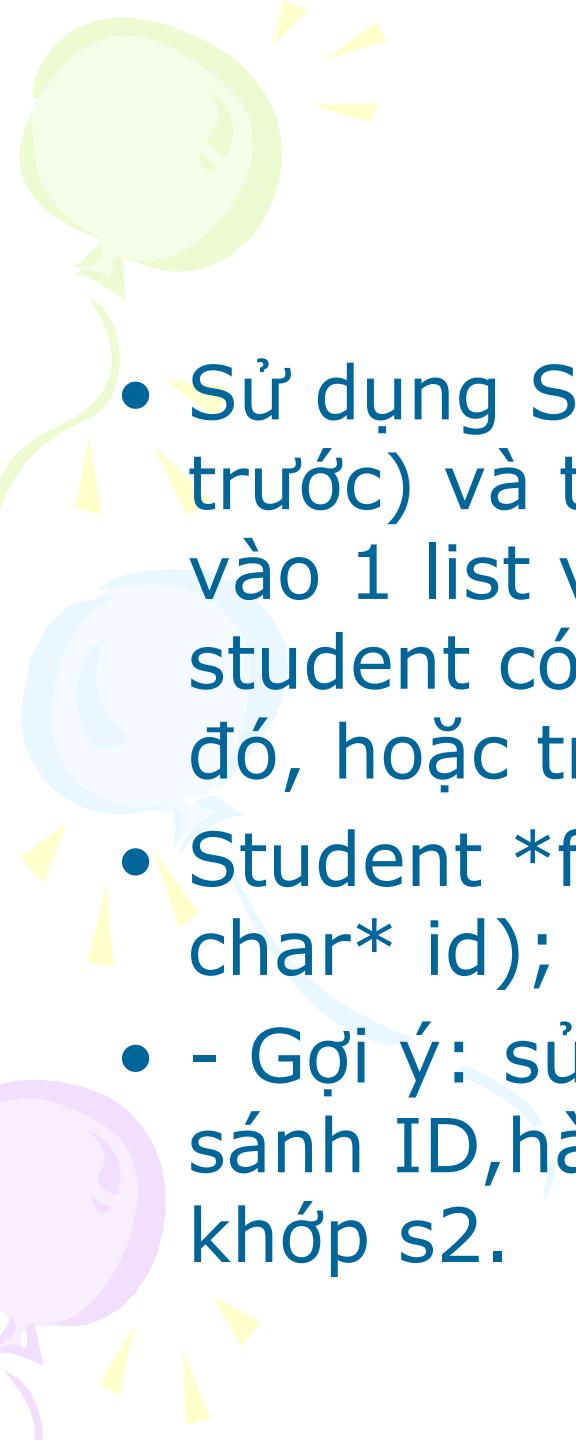


Adding a student - mid / end

```
curr_std = root;
while (curr_std != NULL && to_add->grade < curr_std->grade)
{
    prev_std = curr_std;
    curr_std = curr_std->next;
}

prev_std->next = to_add;
to_add->next = curr_std;
return root;
```





Exercise 3.5

- Sử dụng Student_Package2.c (các bt trước) và tạo hàm find_student() nhận vào 1 list và 1 số ID, trả về con trỏ tới student có số ID trùng khớp ở trong list đó, hoặc trả về 0 nếu không tìm thấy.
- Student *find_student(Student *root, char* id);
- - Gợi ý: sử dụng hàm strcmp(s1,s2) để so sánh ID,hàm này trả về 0 nếu s1 trùng khớp s2.



loại bỏ một student

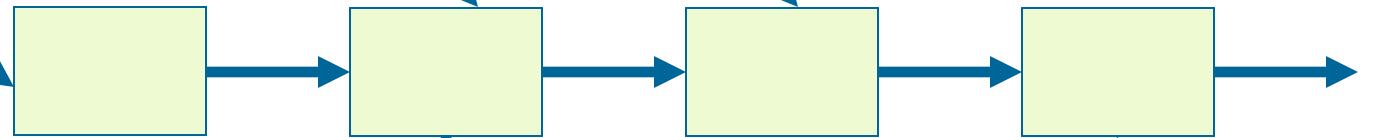
- Ta muốn có thể loại bỏ student qua ID.
- Hàm thực hiện: remove_student()

Removing a student - reminder

root

Previous

Current



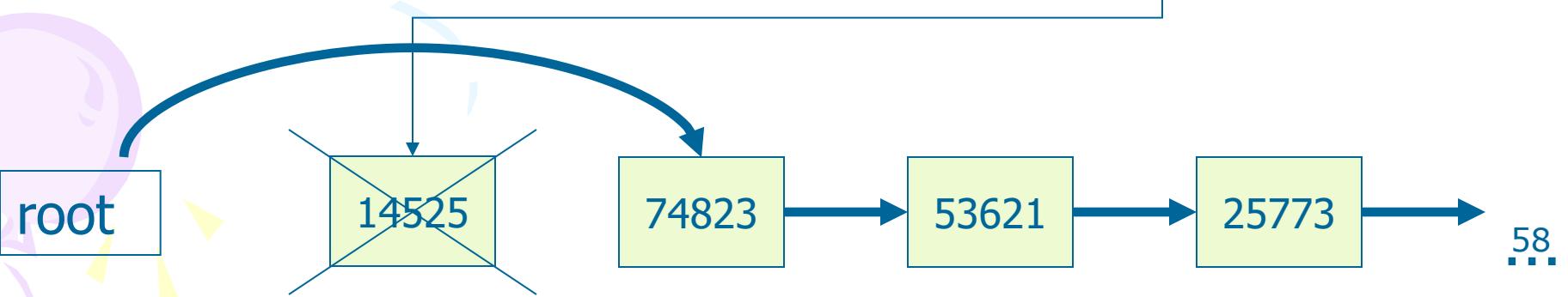
Removing a student – beginning

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

ID
14525

cur

last



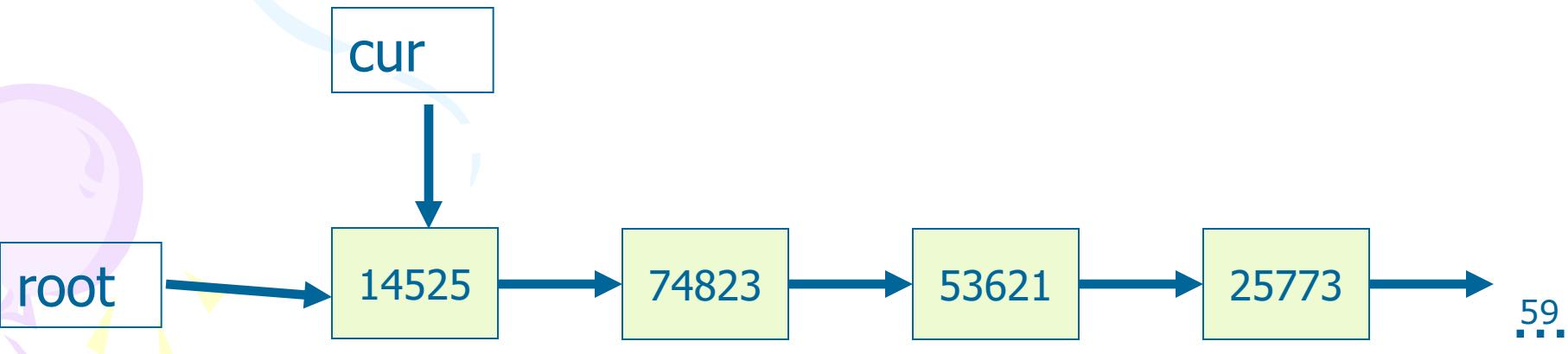
Removing a student - mid list

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

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

return root;
```

ID
53621



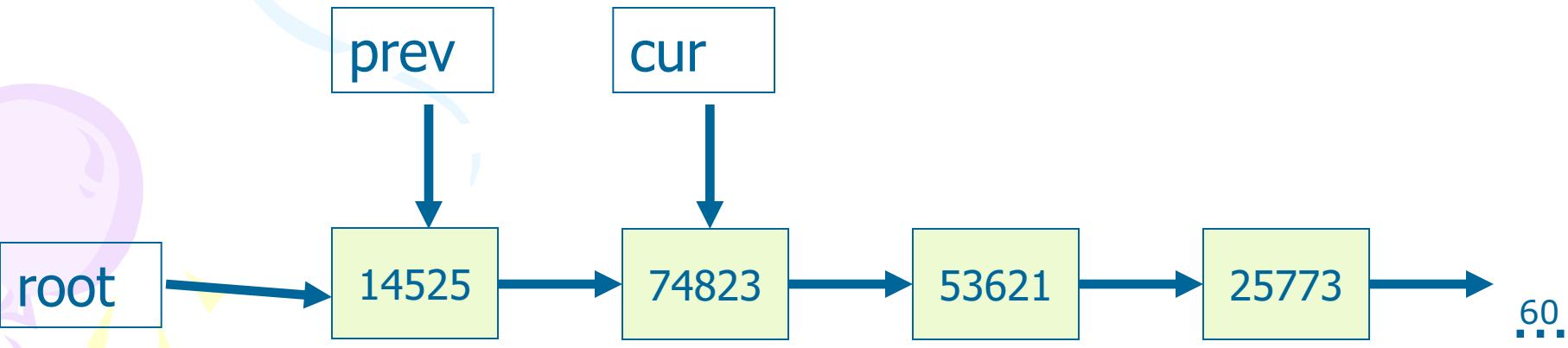
Removing a student - mid list

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

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

return root;
```

ID
53621



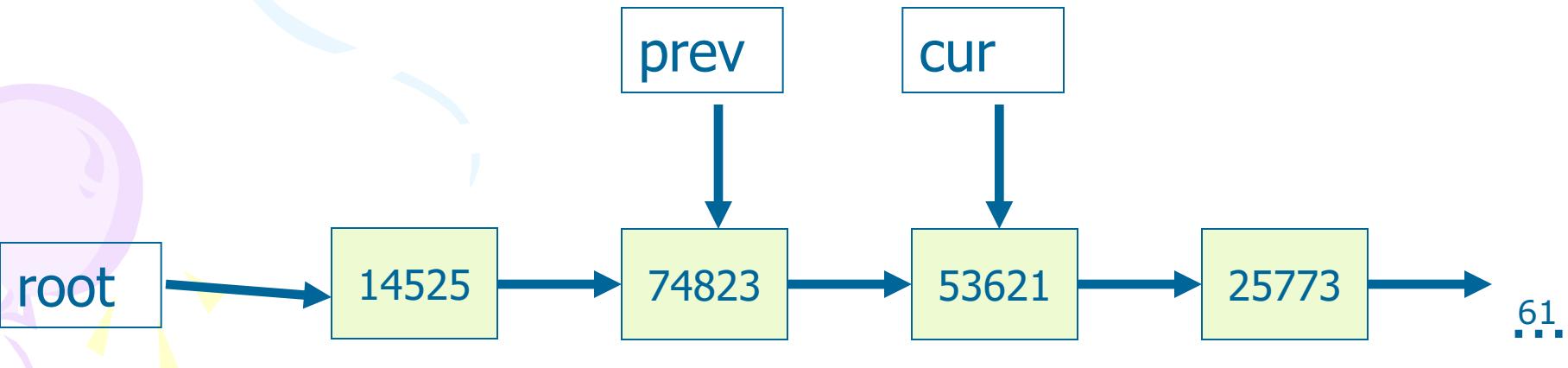
Removing a student - mid list

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

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

return root;
```

ID
53621



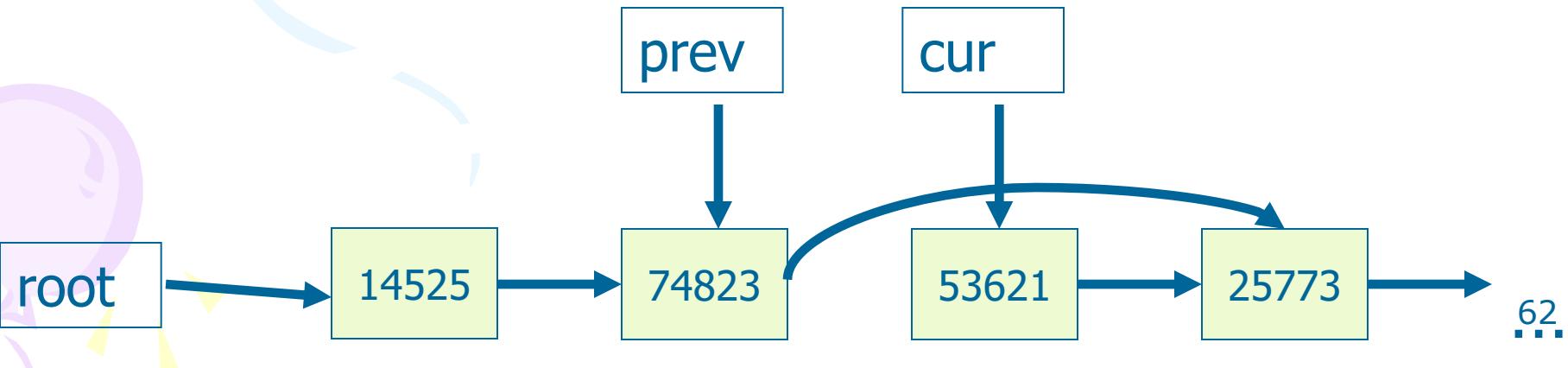
Removing a student - mid list

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

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

return root;
```

ID
53621



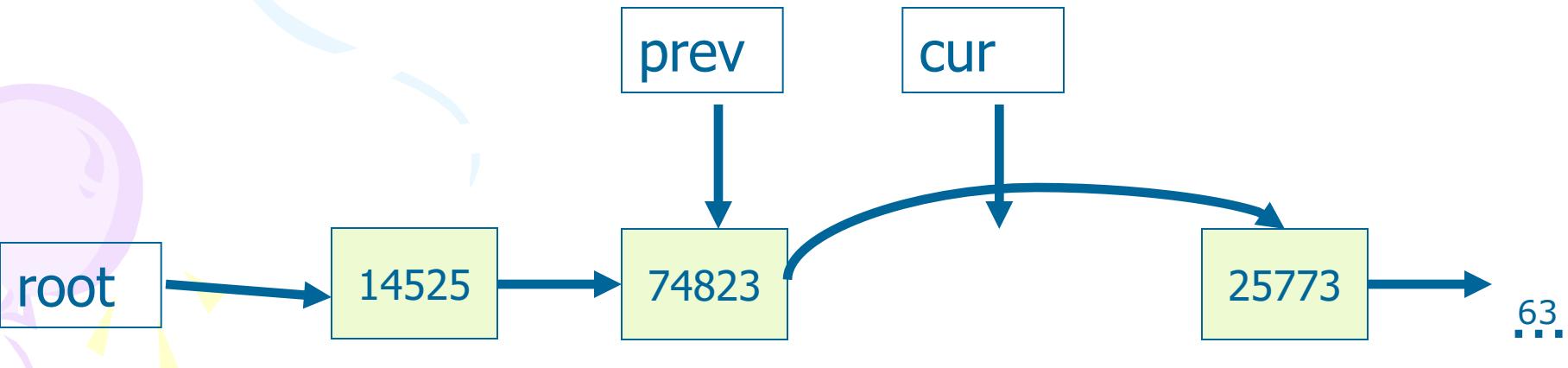
Removing a student - mid list

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

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

return root;
```

ID
53621



Exercise

- Sử dụng Student_Package3.c và thêm hàm change_grade(thay đổi điểm). Hàm này nên lấy tham số là root của list, ID của người có điểm (grade) ta muốn thay đổi và điểm mới.
- Gợi ý: tạo ra 1 new student với cùng tên, ID như cũ nhưng có điểm mới.Sau đó loại bỏ student cũ đi và thêm student mới này vào sử dụng các hàm đã có từ trước

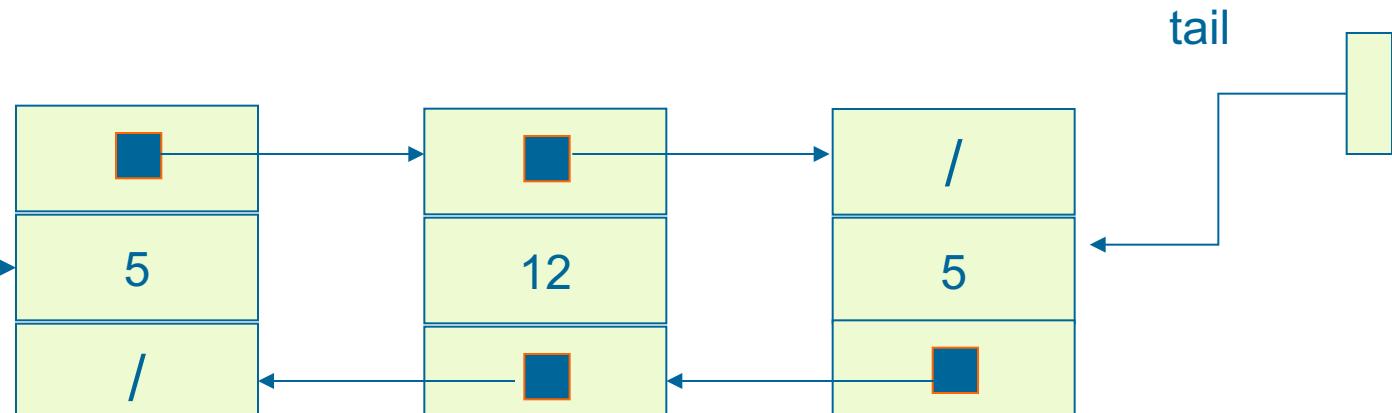
Câu hỏi

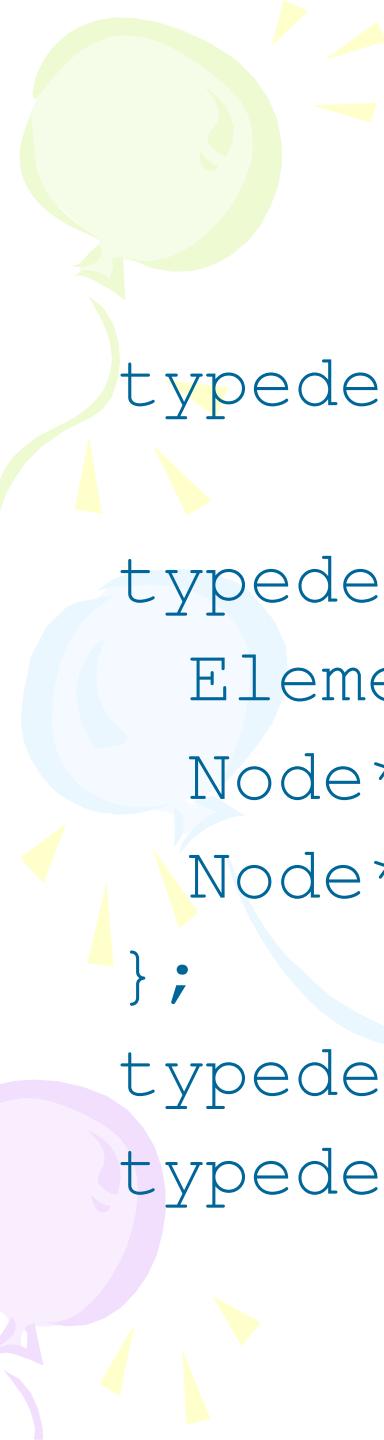
- Hiện tại, chúng ta đang thiết kế “danh bạ” cho điện thoại.
- Bạn phải khai báo một cấu trúc mà có thể lưu giữ tên, số điện thoại, và địa chỉ email. Bạn phải xây dựng chương trình có thể chứa số lượng dữ liệu bất kỳ.
- Gợi ý: bạn có thể tổ chức các thành phần và cấu trúc dữ liệu sử dụng cấu trúc bản ghi sau

```
struct AddressList {  
    struct AddressList *prev;  
    struct AddressList *next;  
    struct Address addr;  
};
```

Danh sách liên kết đôi

- Một phần tử có 2 trường con trỏ, ta có thể theo dõi phần tử trước và sau.





Khai báo

```
typedef ... ElementType;
```

```
typedef struct Node{  
    ElementType Element;  
    Node* Prev;  
    Node* Next;  
};
```

```
typedef Node* Position;  
typedef Position DoubleList;
```

Khởi tạo và kiểm tra danh sách rỗng

```
void MakeNull_List (DoubleList *DL) {  
    (*DL) = NULL;
```

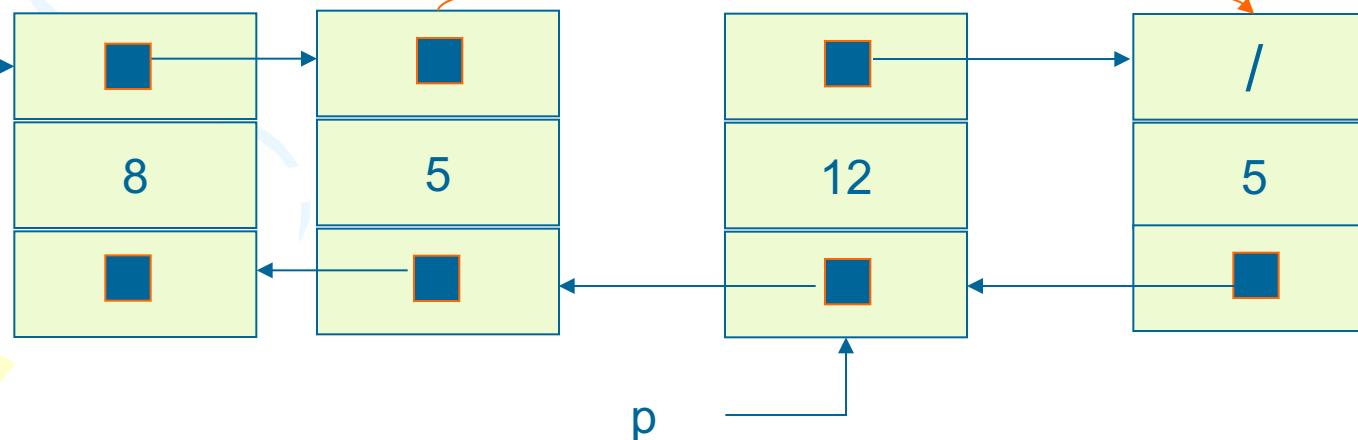
```
}
```

```
int Empty (DoubleList DL) {  
    return (DL==NULL);
```

```
}
```

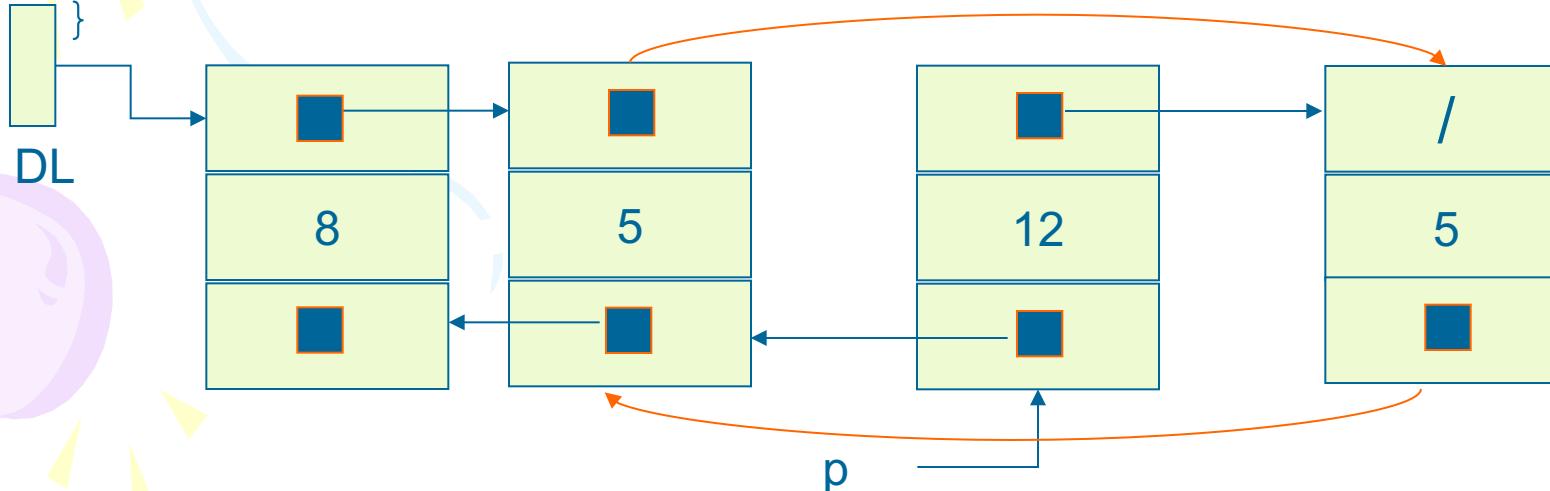
Xóa một nốt trỏ bởi p

```
void Delete_List (Position p, DoubleList *DL) {  
    if (*DL == NULL) printf("Empty list");  
    else {  
        if (p==*DL) (*DL)=(*DL)->Next;  
        //Delete first element  
        else p->Previous->Next=p->Next;  
        if (p->Next!=NULL) p->Next->Previous=p->Previous;  
        free(p);  
    }  
}
```



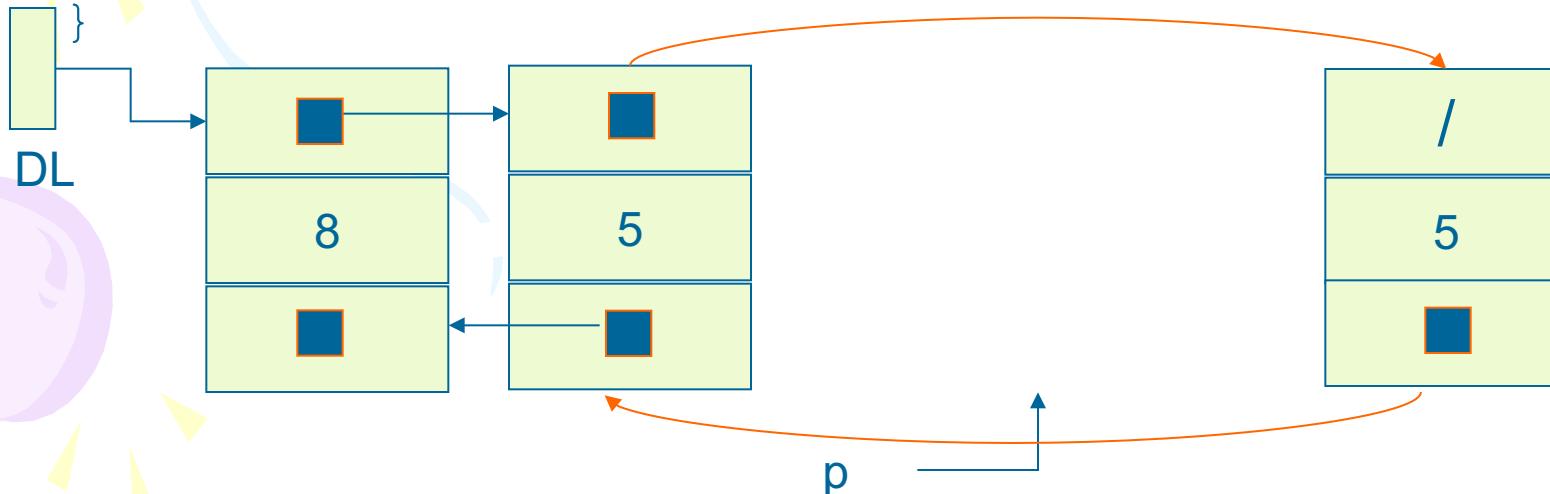
Xóa một nốt trỏ bởi p

```
void Delete_List (Position p, DoubleList *DL) {  
    if (*DL == NULL) printf("Empty list");  
    else {  
        if (p==*DL) (*DL)=(*DL)->Next;  
        //Delete first element  
        else p->Previous->Next=p->Next;  
        if (p->Next!=NULL) p->Next->Previous=p->Previous;  
        free(p);  
    }  
}
```



Xóa một nốt trỏ bởi p

```
void Delete_List (Position p, DoubleList *DL) {  
    if (*DL == NULL) printf("Empty list");  
    else {  
        if (p==*DL) (*DL)=(*DL)->Next;  
        //Delete first element  
        else p->Previous->Next=p->Next;  
        if (p->Next!=NULL) p->Next->Previous=p->Previous;  
        free(p);  
    }  
}
```



Chèn

```
void Insert_List (ElementType X,Position p, DoubleList *DL){  
    if (*DL == NULL){ // List is empty  
        (*DL)=(Node*)malloc(sizeof(Node));  
        (*DL)->Element = X;  
        (*DL)->Previous =NULL;  
        (*DL)->Next =NULL;  
    }  
    else{  
        Position temp;  
        temp=(Node*)malloc(sizeof(Node));  
        temp->Element=X;  
        temp->Next=p;  
        temp->Previous=p->Previous;  
        if (p->Previous!=NULL)  
            p->Previous->Next=temp;  
        p->Previous=temp;  
    }  
}
```