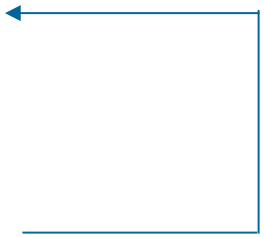
C Programming Basic – week 3

Chủ đề

- Danh sách liên kết đơn
  - Cài đặt danh sách
  - Duyệt danh sách
  - Cập nhật danh sách
- Danh sách liên kết kép
  - Cài đặt danh sách
  - Duyệt danh sách
  - Cập nhật danh sách

Danh sách liên kết đơn

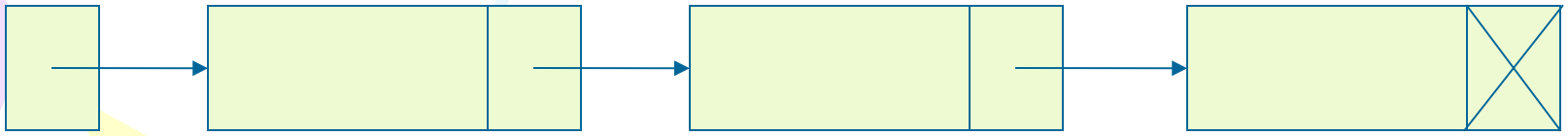
- Một hoặc một vài thành phần là con trỏ trỏ đến chính cấu trúc đó



```
struct list {
char data;
struct list *link;
};
list item1, item2, item3;
item1.data='a';
item2.data='b';
item3.data='c';
item1.link=item2;
item2.link=item3;
item3.link=NULL;
```

Cài đặt danh sách

- Cấu trúc lưu giữ thông tin để truy cập đến các phần tử khác trong danh sách
- Danh sách liên kết đơn chỉ chứa thông tin của phần tử tiếp theo
- Trong C, con trỏ được sử dụng để lưu trữ địa chỉ của phần tử tiếp theo
- Mảng: Có thể truy cập đến dữ liệu tức thì
- Danh sách liên kết: Có thể theo đổi số phần tử



root (or head)

NULL

Khai báo danh sách

```
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;
```

Cấp phát bộ nhớ cho một phần tử

- Cấp phát bộ nhớ cho mỗi phần tử thông qua con trỏ

```
struct node * new;
new = (struct node*) malloc(sizeof(struct node));
new->element = …
new->next = null;
```

- new->addr tương đương (*new).addr

VD

- Xây dựng danh bạ điện thoại
- Khai báo cấu trúc chứa name, phone number, và email
- Chương trình có thể lưu trữ danh sách với độ dài bất kỳ

VD (2)

- Tạo danh sách liên kết đơn để lưu trữ danh bạ
- Viết hàm để thêm một phần tử vào sau phần tử hiện tại và dùng hàm đó để tạo danh sách
- Viết hàm để in ra tất cả các phần tử của danh sách
- Viết hàm để xóa một nút trong danh sách

Gợi ý

- Có thể sử dụng khai báo danh sách liên kết sau:

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

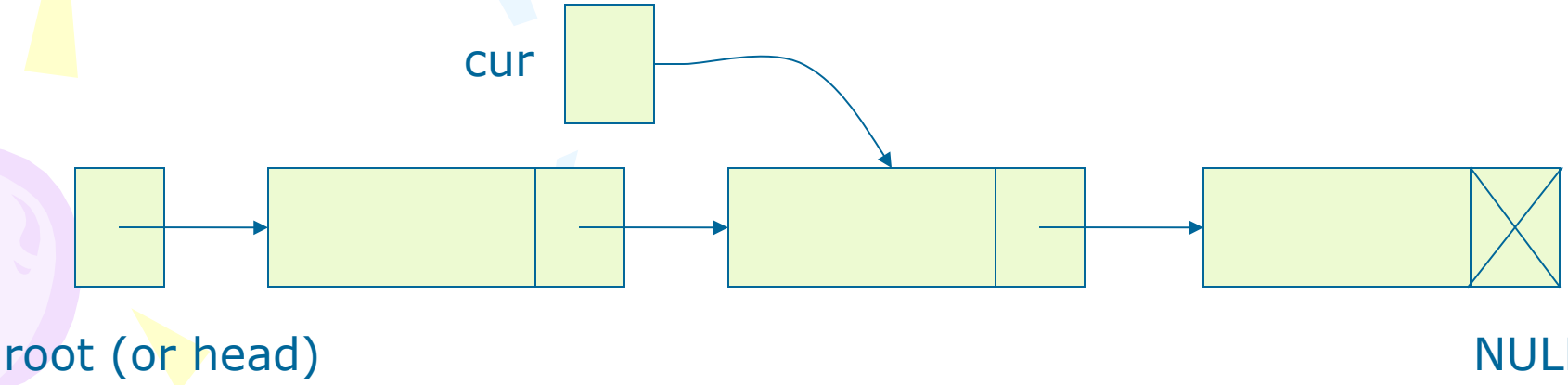
Khai báo cấu trúc

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

- "next" là con trỏ trỏ đến phần tử tiếp theo, có kiểu AddressList.
- "addr" chứa thông tin danh bạ

Các thông tin quan trọng của danh sách

- Root: Đầu của danh sách
- NULL: Con trỏ rỗng, báo hiệu kết thúc danh sách
- Cur: Con trỏ trỏ đến phần tử hiện tại của danh sách



Khởi tạo

```
struct AddressList {
struct AddressList *next;
struct Address addr;
};
struct AddressList * root, *cur;
```

Tạo nút mới

```
struct AddressList* makeNewNode(){
struct AddressList * new = (struct AddressList
  * ) malloc(sizeof(struct AddressList));
strcpy((new->addr).name, « Tran Van Thanh »);
….
new->next =NULL;
return new;
}
root = makeNewNode();
cur = root;
```

Tạo nút mới (2)

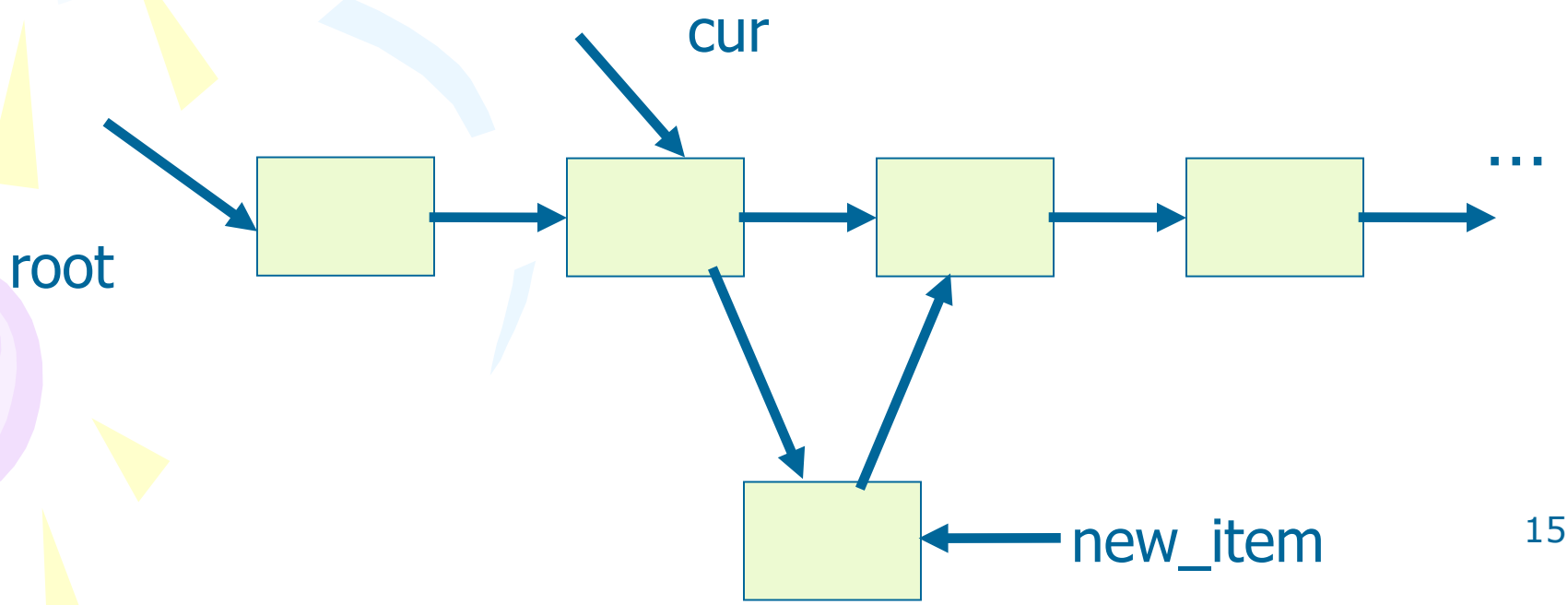
- Có thể nhận một bản ghi làm tham số

```
struct AddressList* makeNewNode(Address
  addr){
struct AddressList * new = (struct
  AddressList * ) malloc(sizeof(struct
  AddressList));
new->addr=addr;
new->next =NULL;
return new;


}
```

Chèn thêm phần tử

- Sau vị trí hiện tại

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



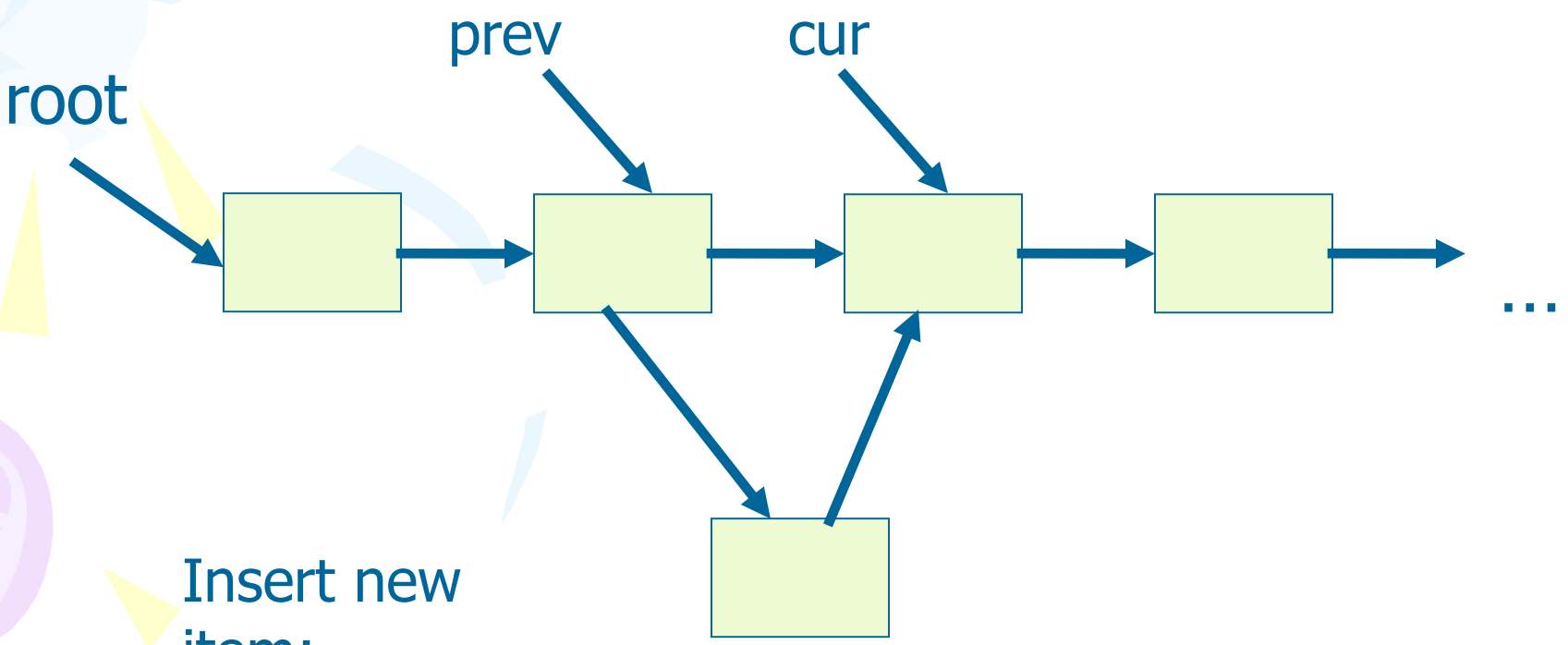
Chèn thêm phần tử (2)

- Sau vị trí hiện tại

```
new = ( struct AddressList * ) malloc( sizeof( struct AddressList
   ) );
new->addr = addr;
new->next = NULL;
// new = makeNewNode();
if ( root == NULL ) {
   /* if there is no element */
   root = new;
   cur = root;
} else {                                                    ...
    new->next=cur->next;
     cur->next = new;
     // prev=cur;
     cur = cur->next;
}
}
```

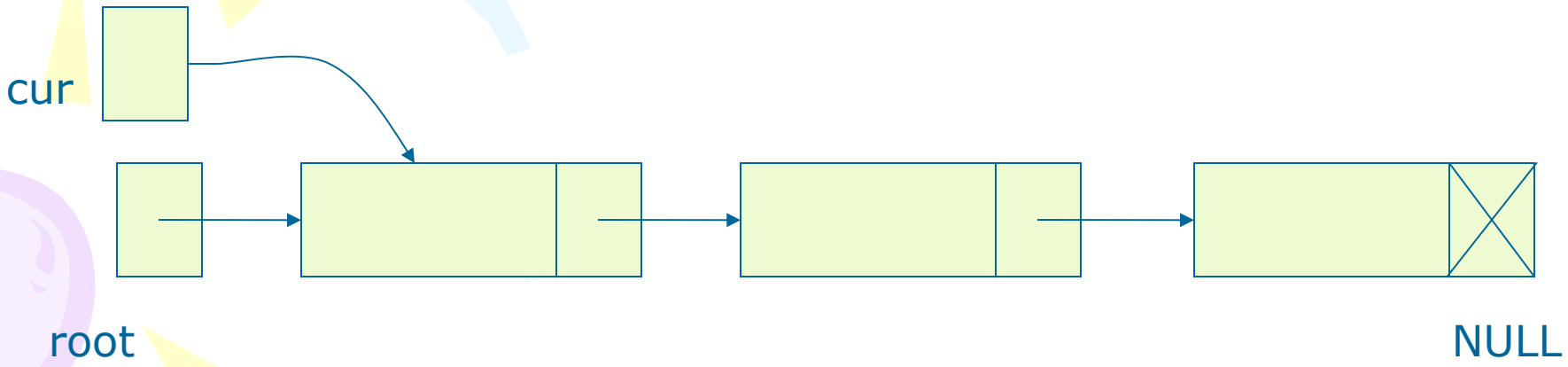
Chèn thêm phần tử (3)

- Trước vị trí hiện tại



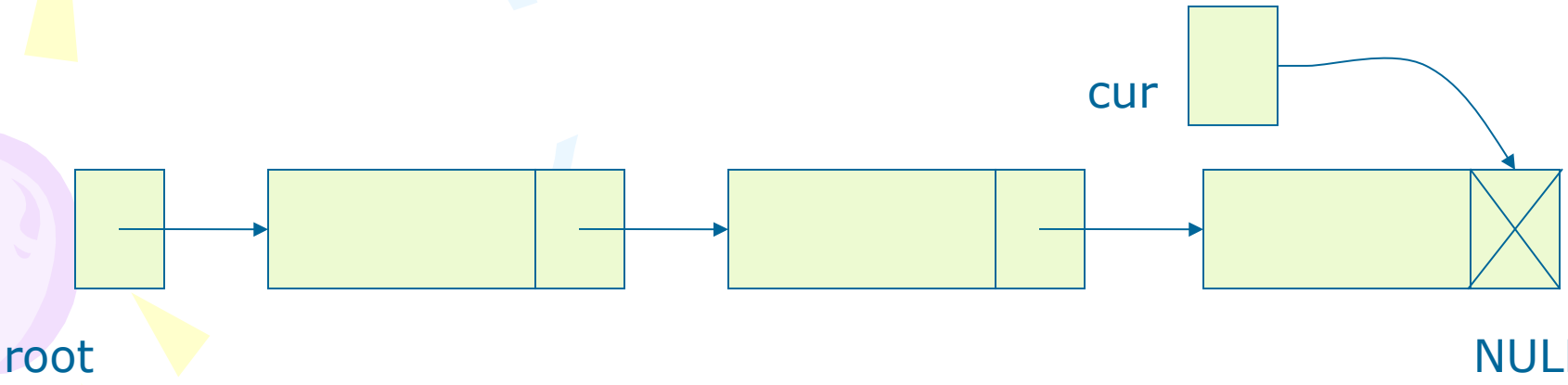
Duyệt danh sách

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



Duyệt danh sách (2)

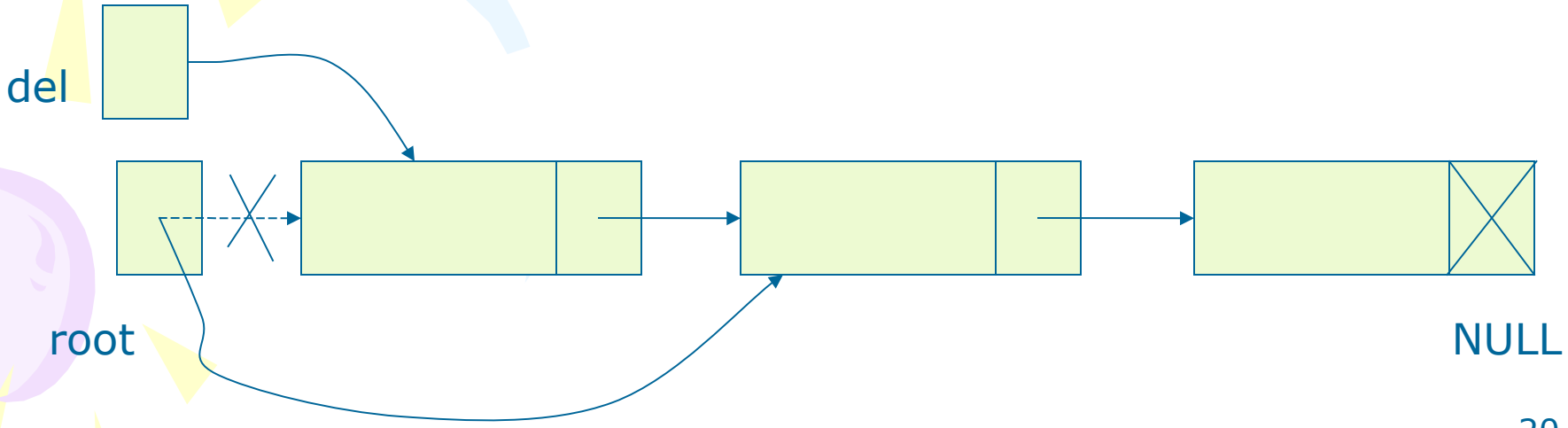
- Thay đổi giá trị của cur trong danh sách
- Con trỏ cur được gọi là một iterator
- Kết thúc khi gặp con trỏ NULL



Xóa phần tử

- Xóa phần tử đầu tiên

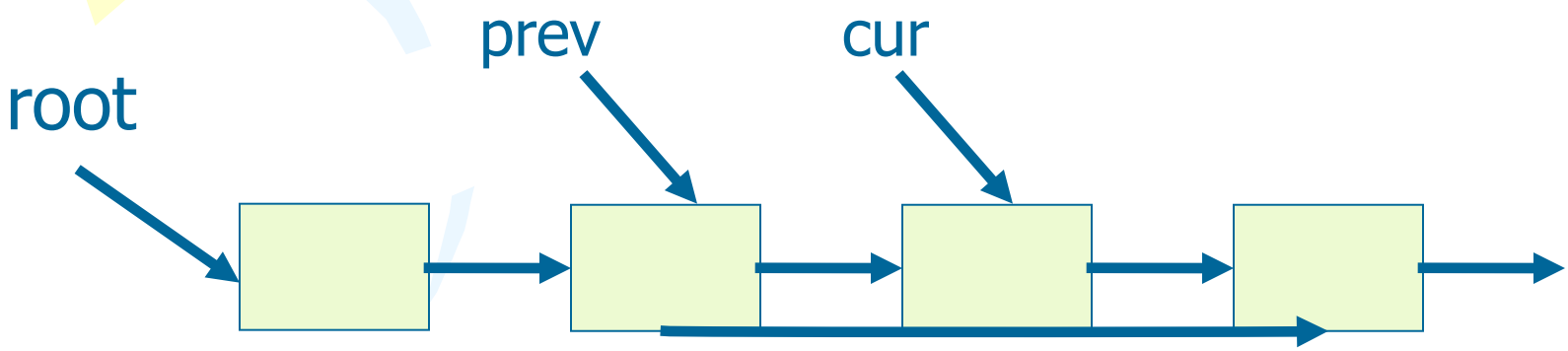
  root = del->next; free(del);
- Thay đổi giá trị của "root" thành giá trị của "next" (được trỏ tới bởi "del")



Xóa phần tử (2)

- Xóa phần tử ở giữa danh sách
- Xóa phần tử trỏ tới bởi *cur*
- Xác định *prev* là con trỏ trỏ tới phần tử liền trước *cur*

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



Exercise 3.1

- Đọc dữ liệu từ tệp ‘phone.dat’ (từ bài tập trước) và tải vào danh sách liên kết
- Sử dụng các hàm đã học
- In ra danh sách
- Yêu cầu người dùng nhập vào thông tin và cập nhật/thêm vào danh sách

Exercise 3.2

- Cài đặt các hàm insert, delete với tham số n (integer) là vị trí trong danh sách
  - N=0 : Thêm/xóa vị trí đầu tiên
  - N=1 : Thêm vào sau phần tử đầu tiên ; xóa phần tử đầu tiên
  - ...

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

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

Exercise 3.3

- Thêm hàm vào Exercise 3.1 cho phép cập nhật và xóa ở vị trí *n* cho trước
- Sử dụng hàm xóa phần tử đã học
- Viết chương trình dạng menu cho phép tìm kiếm
  - theo phone number
  - theo name
- Bắt các lỗi có thể xảy ra
- Thêm hàm đảo ngược danh sách

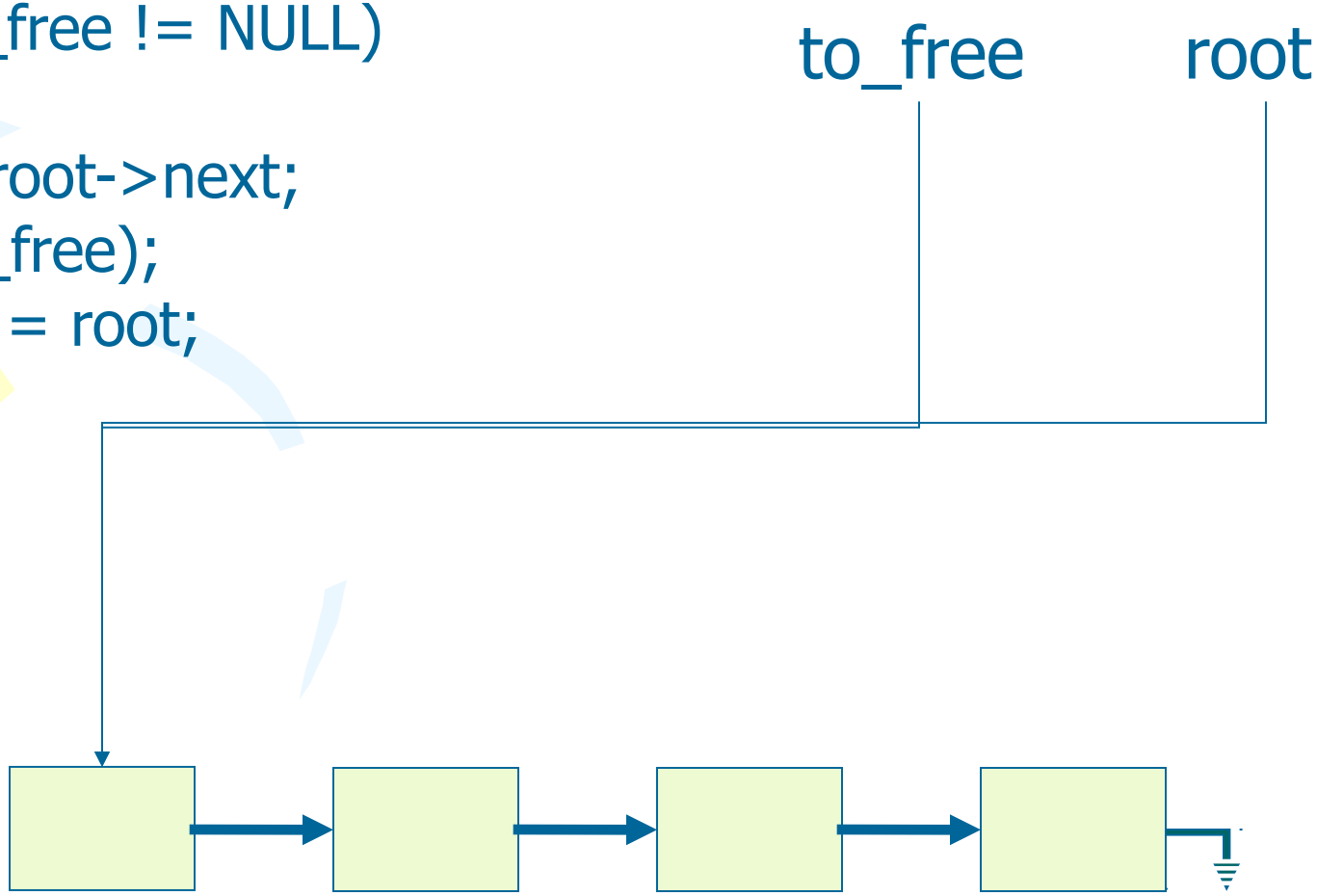
Exercise 3.4

a) Viết hàm chia danh sách
   - Split: Chia danh sách ra làm đôi tại vị trí *n* (bắt đầu từ 0)
   - Extract: Trích rút danh sách con từ vị trí n1 tới vị trí n2
   - Thêm chức năng vào menu chương trình

b) Viết hàm in danh sách ra tệp văn bản. Tham số là root của danh sách và đường dẫn tệp. Sử dụng để lưu kết quả của hàm split/extract nêu trên

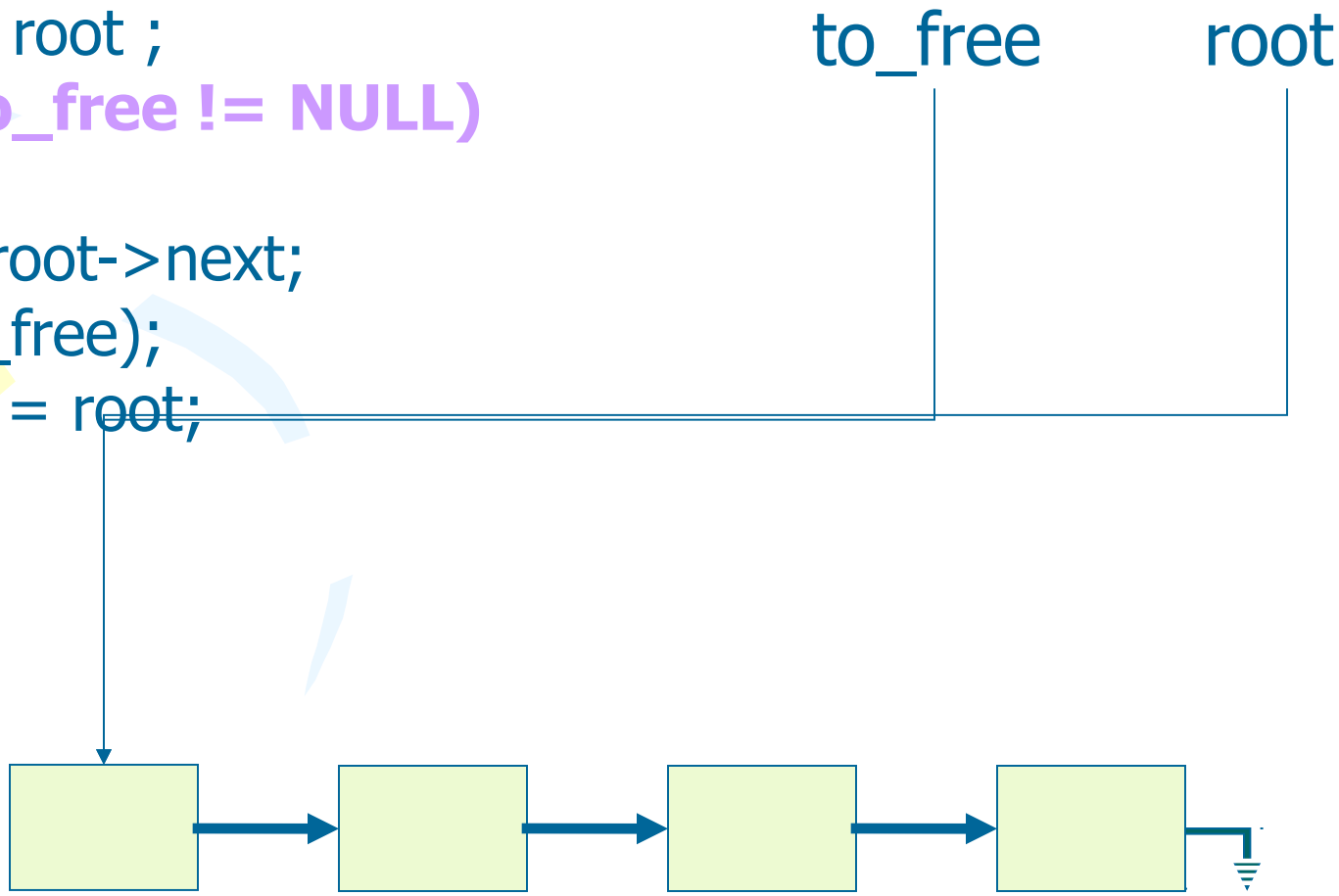
Xóa danh sách

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



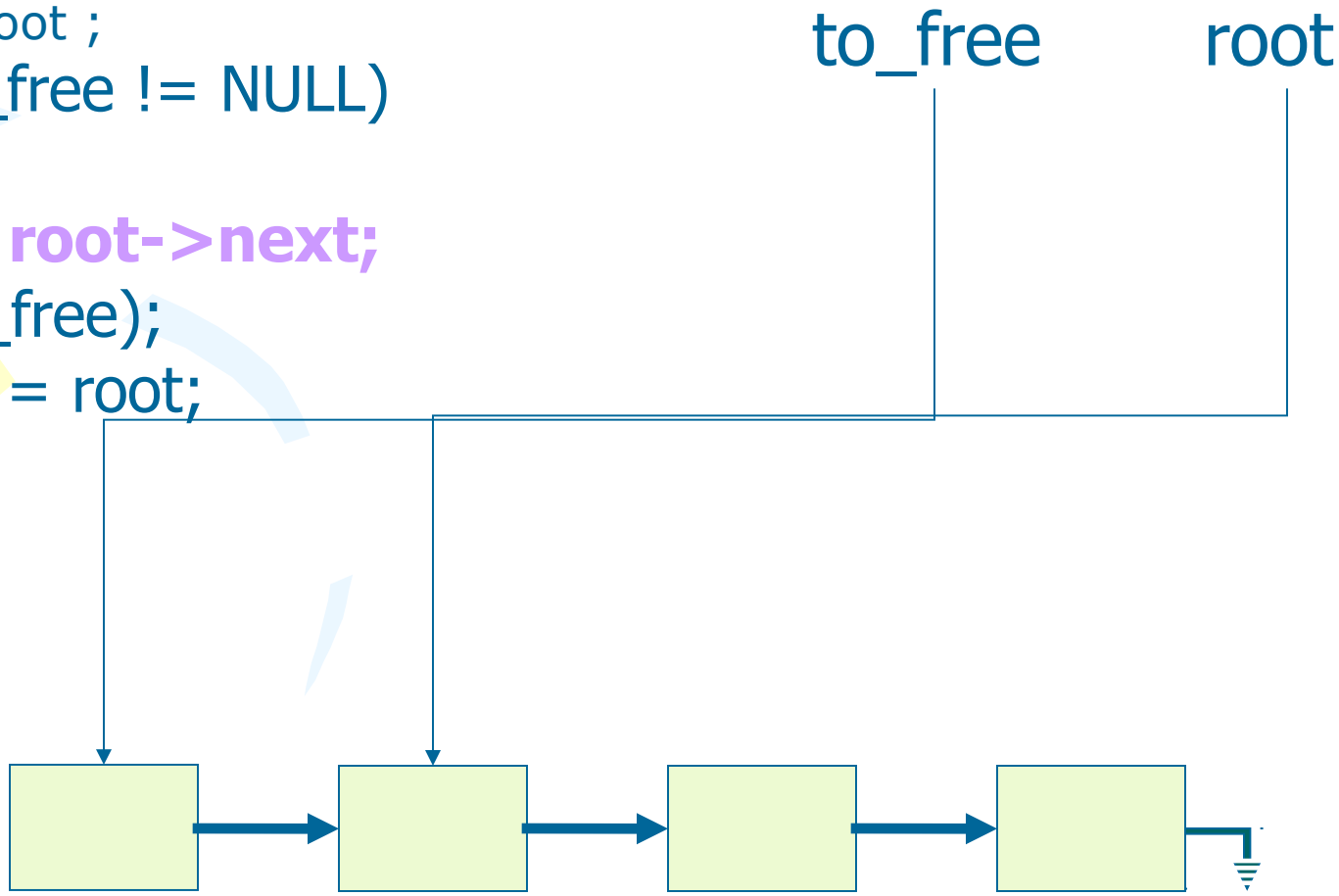
Xóa danh sách (2)

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



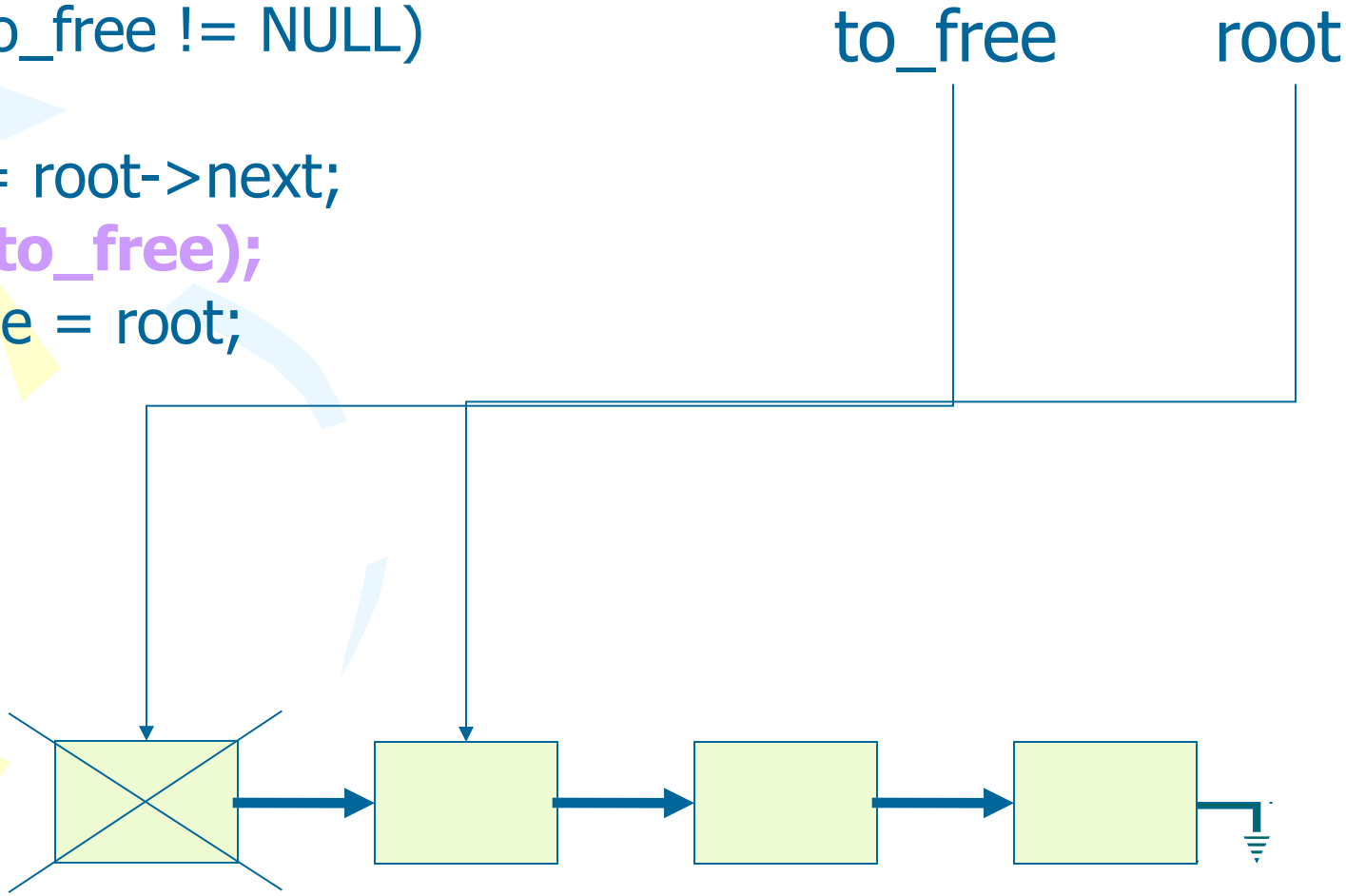
Xóa danh sách (3)

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



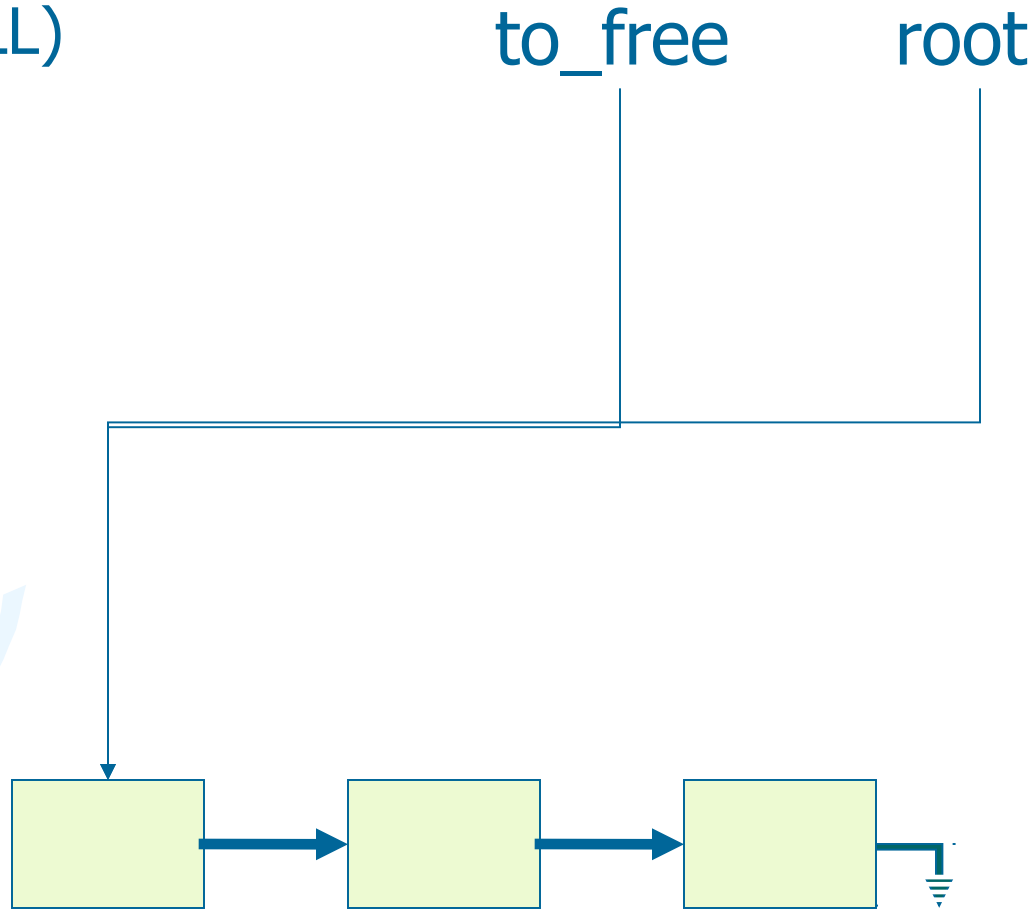
Xóa danh sách (4)

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



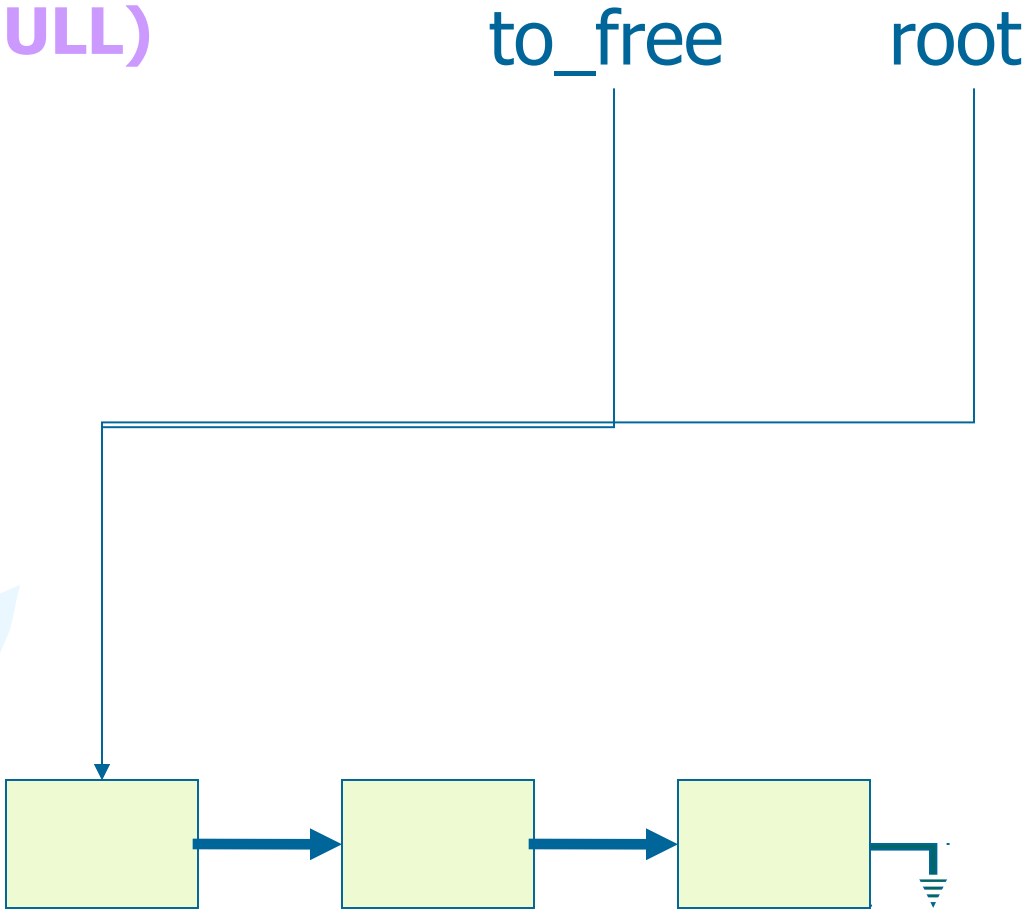
Xóa danh sách (5)

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



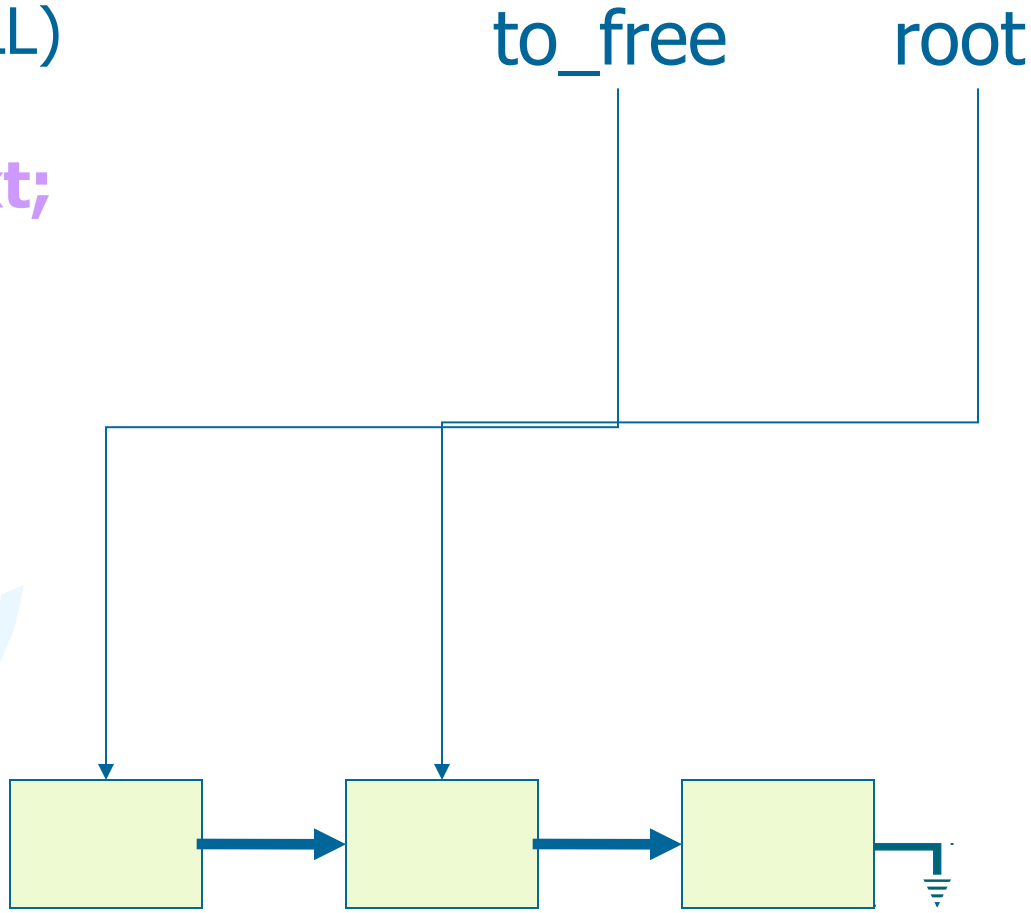
Xóa danh sách (6)

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



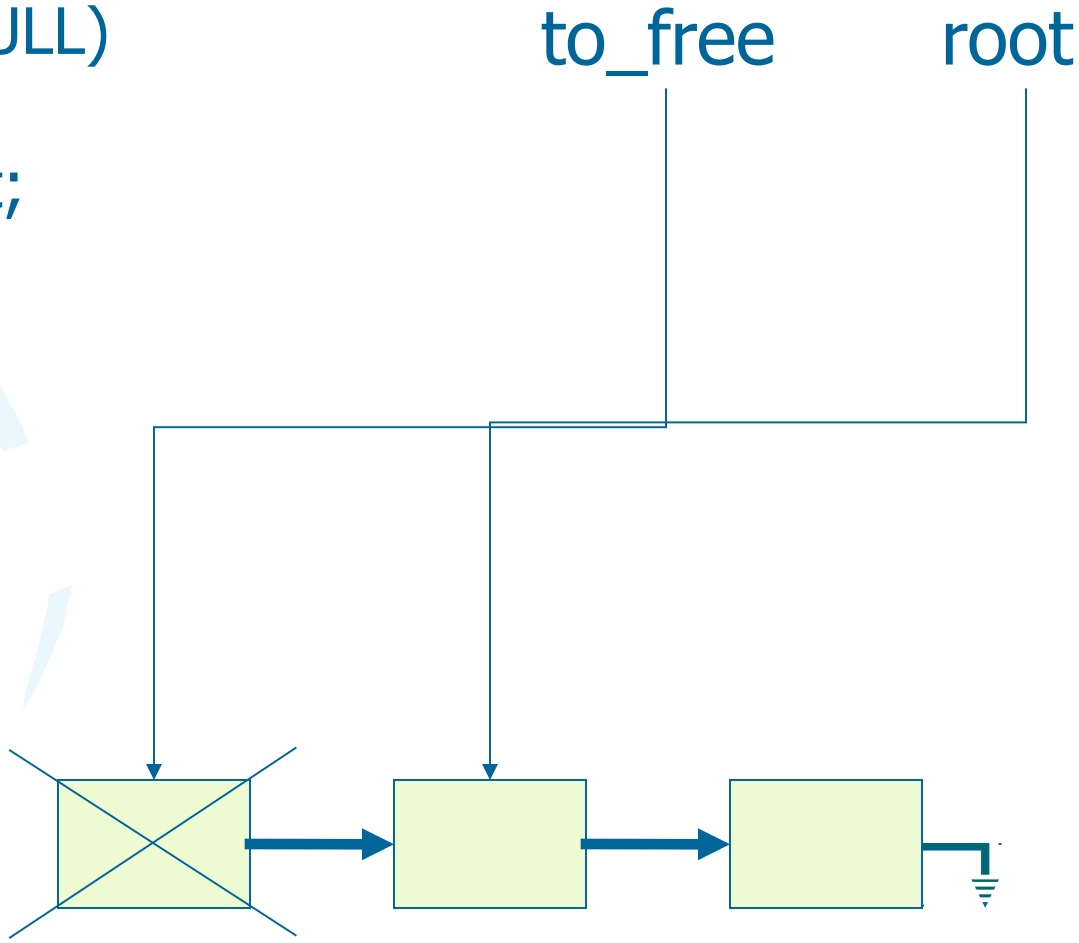
Xóa danh sách (7)

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



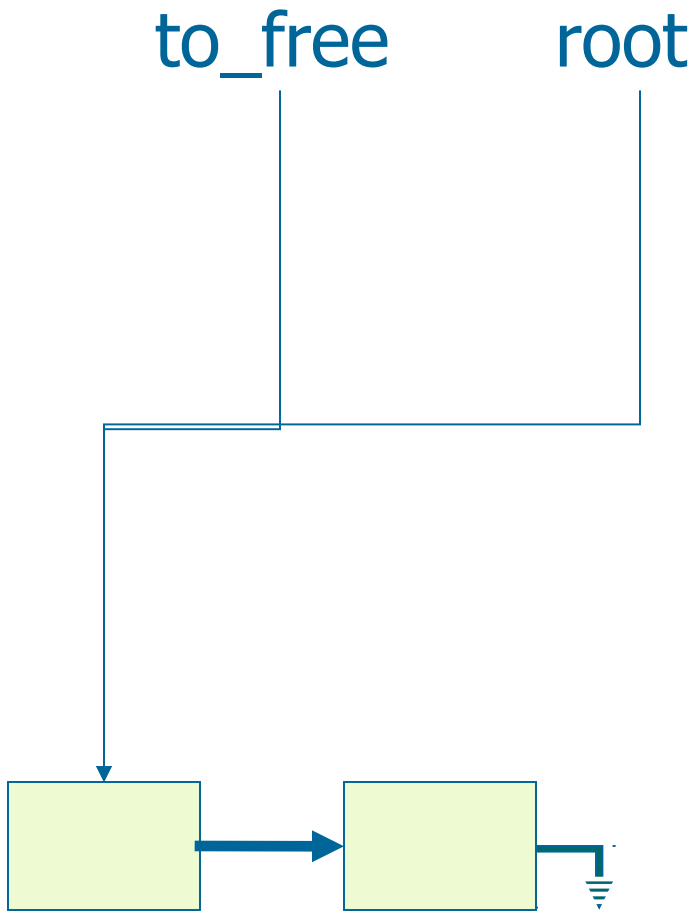
Xóa danh sách (8)

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



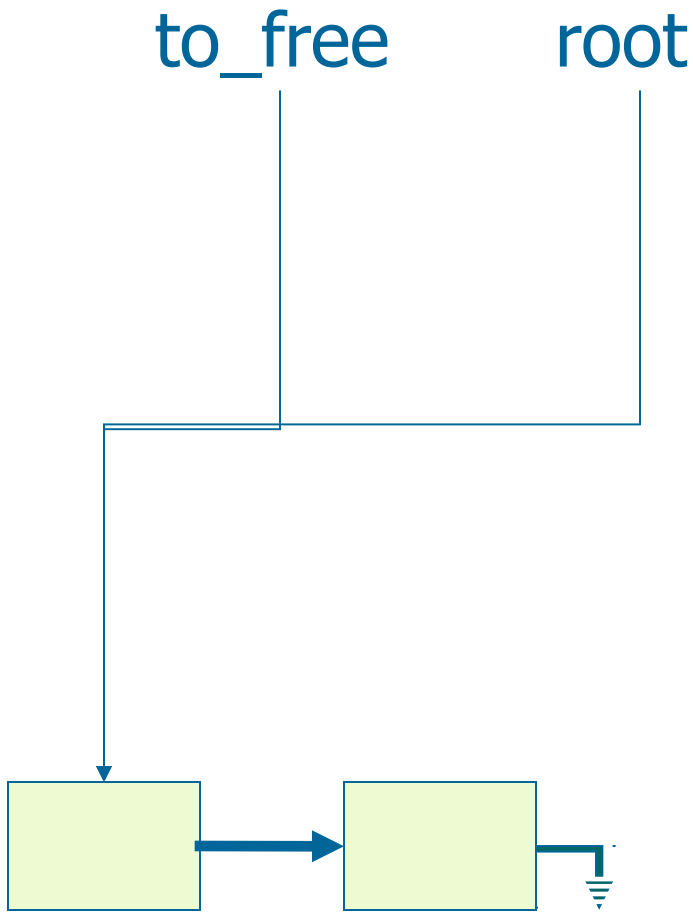
Xóa danh sách (9)

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



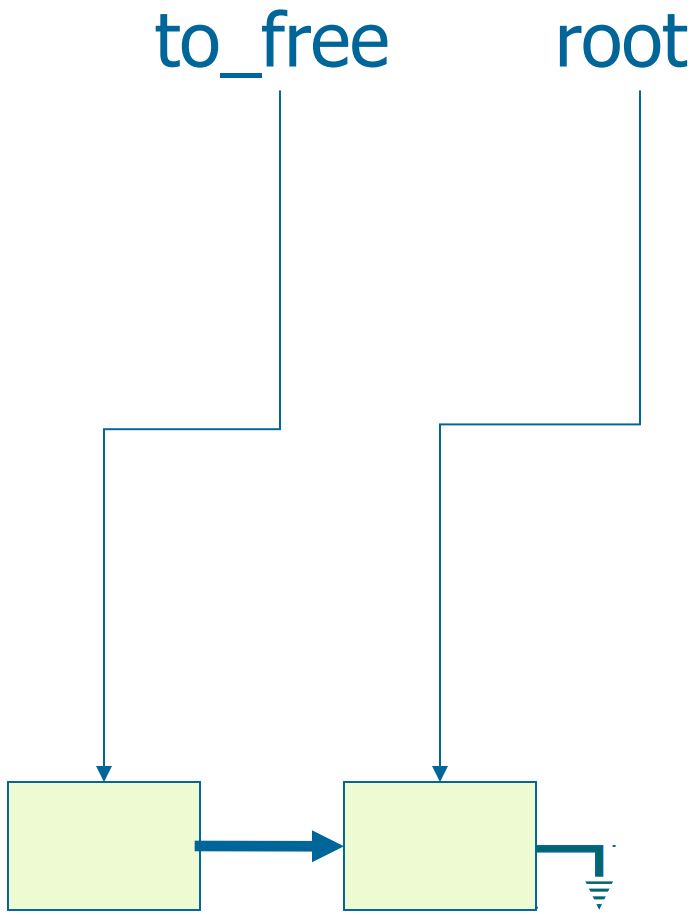
Xóa danh sách (10)

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



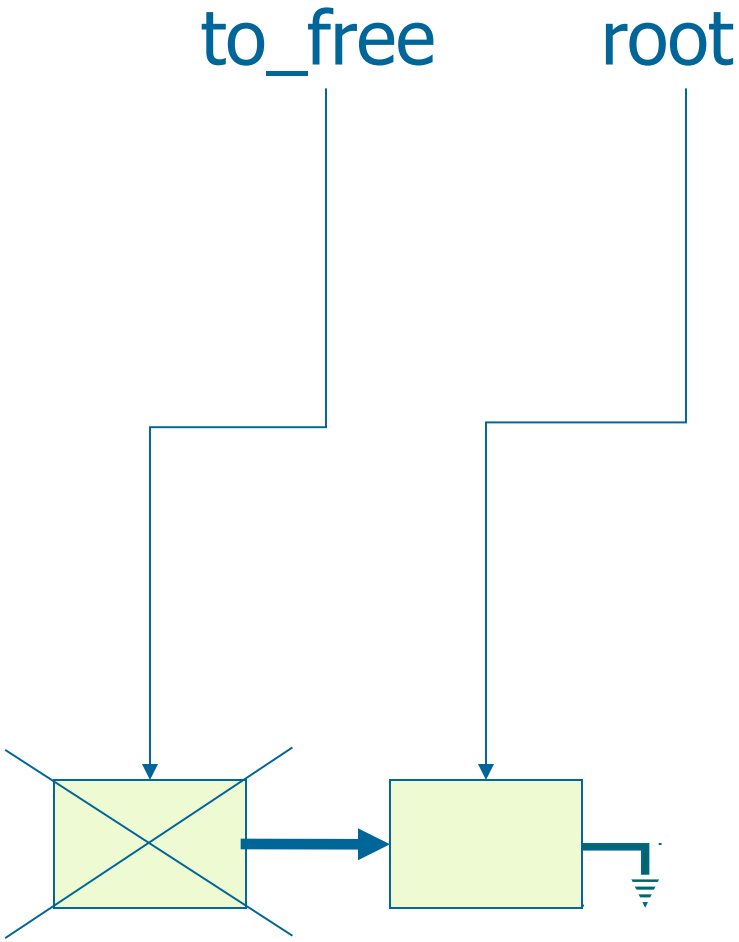
Xóa danh sách (11)

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



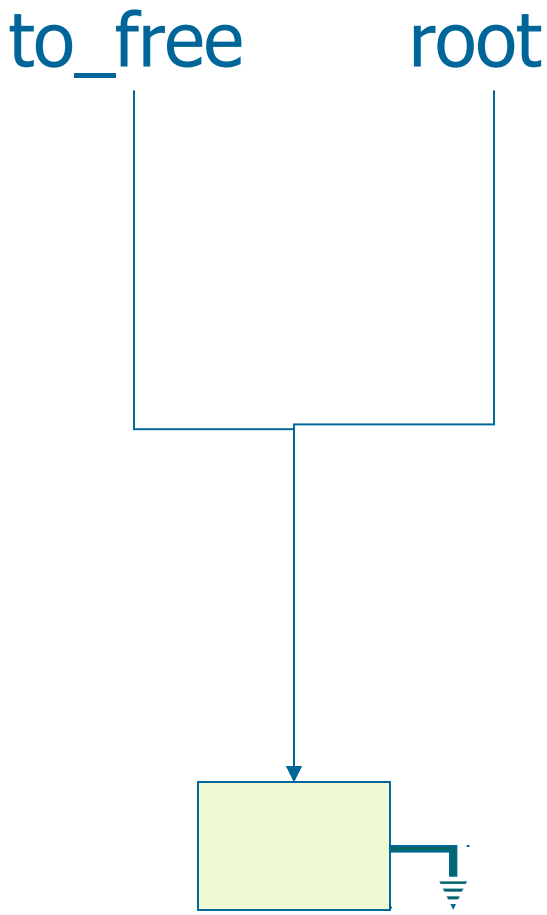
Xóa danh sách (12)

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



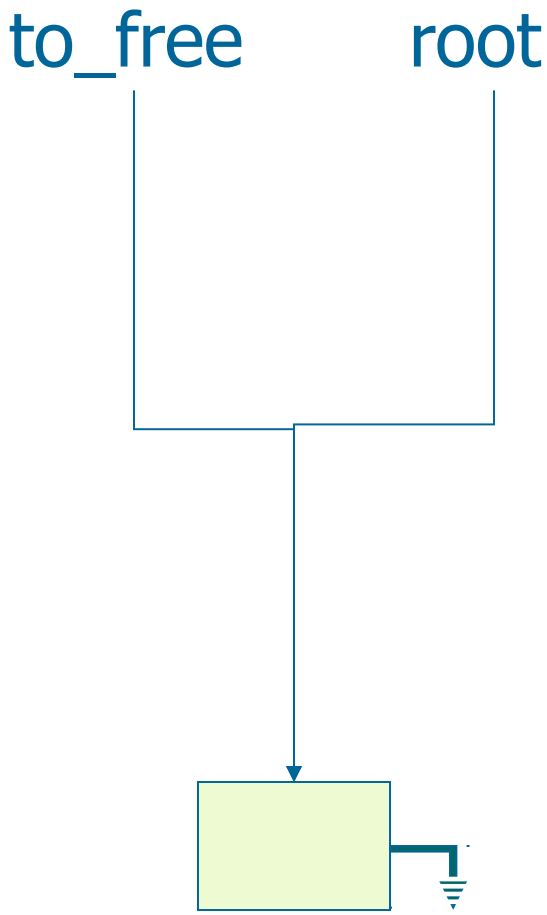
Xóa danh sách (13)

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



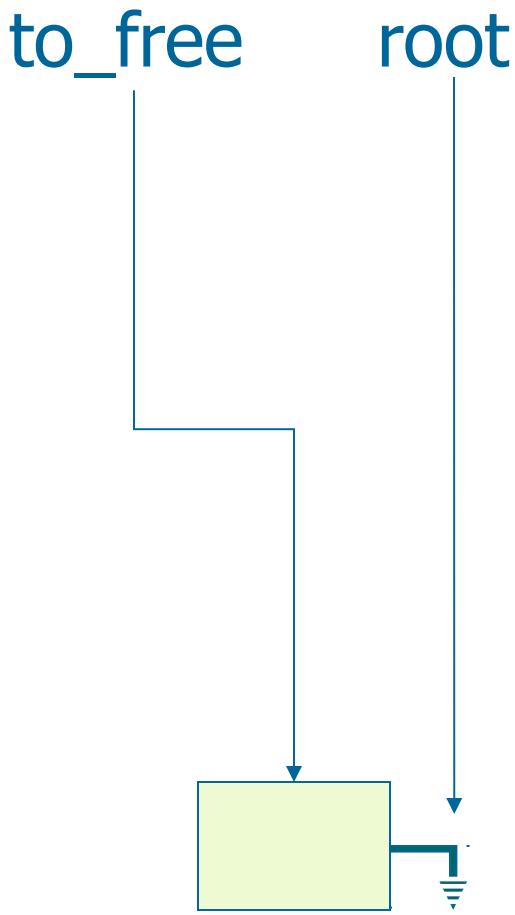
Xóa danh sách (14)

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



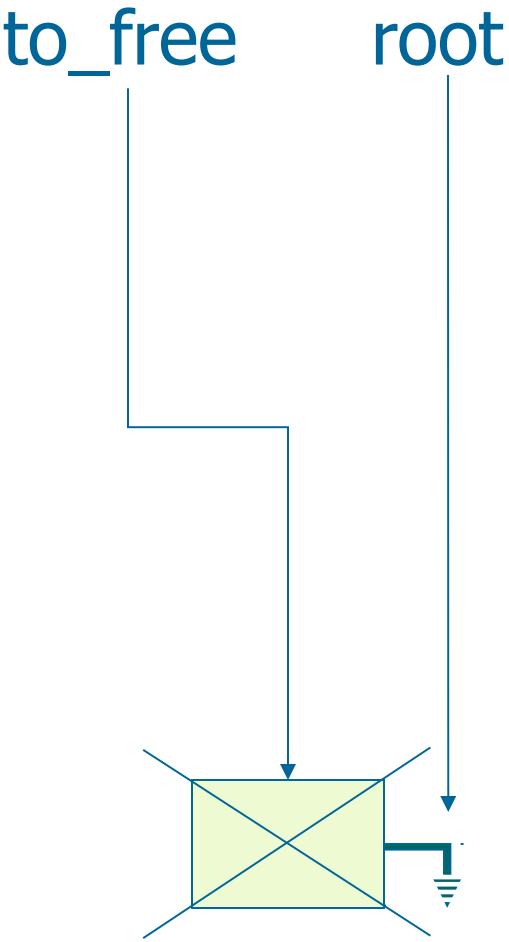
Xóa danh sách (15)

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



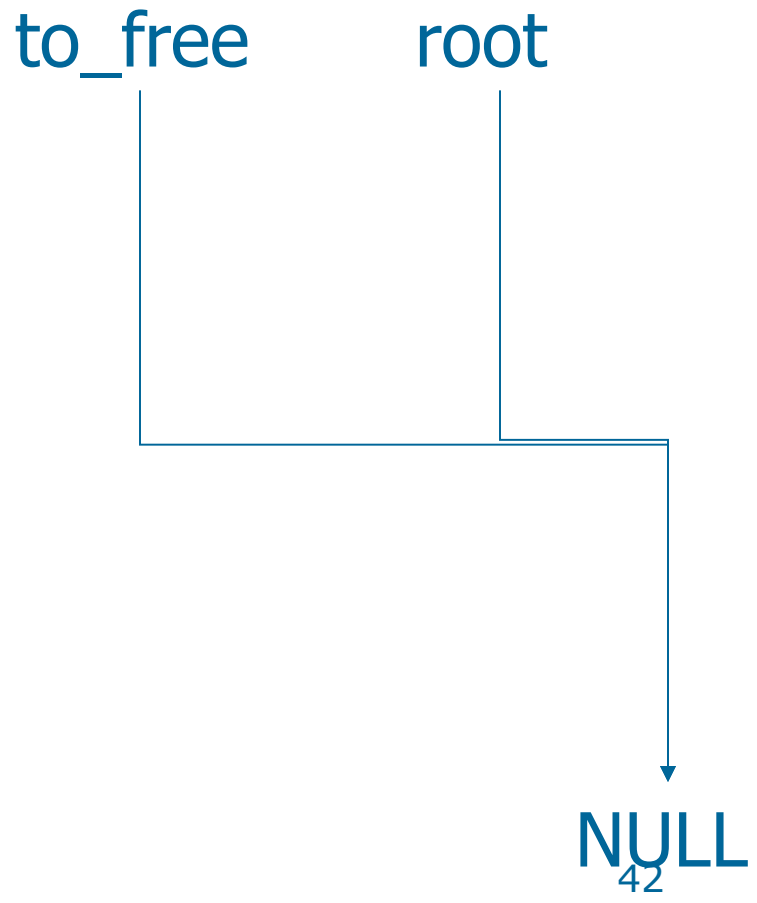
Xóa danh sách (16)

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



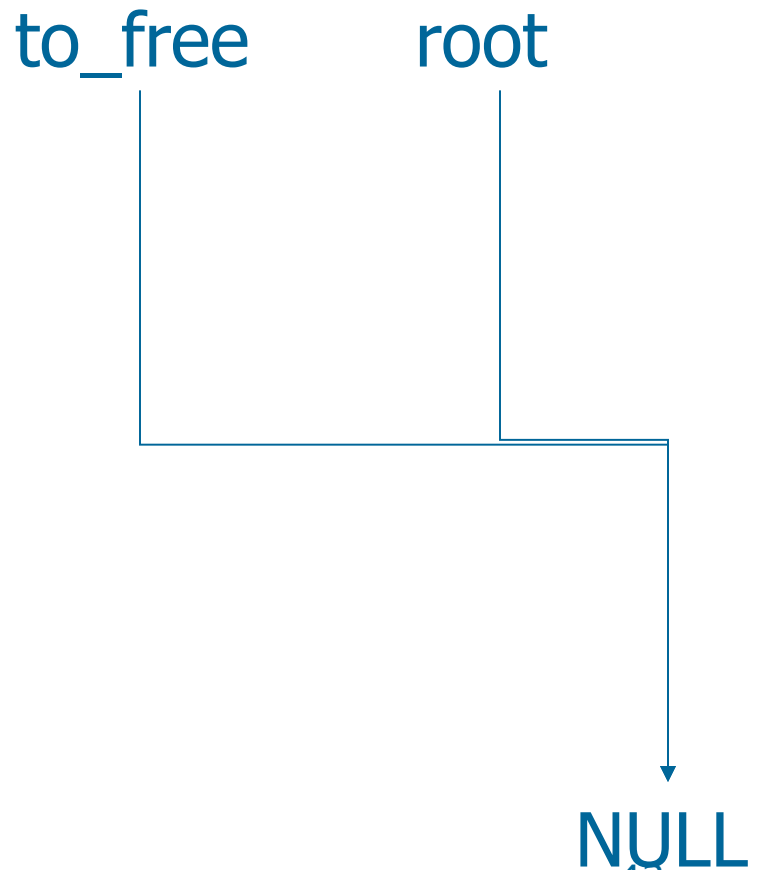
Xóa danh sách (17)

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



Xóa danh sách (18)

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



Exercise 3.5

- 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

Exercise 3.5 (2)

- Thêm vào menu chương trình
- Kiểm tra bằng hàm traverse()

Tổng kết các chức năng của Bài tập

- 1. Import from NokiaDB.dat (insertafter)
- 2. Display (traverse)
- 3. Add new phone (insertbefore)
- 4. Insert at Position
- 5. Delete at Position
- 6. Delete current
- 7. Delete first
- 8. Reverse List
- 9. Save to File
- 10. Quit(Free)

Exercise 3.6

- Viết chương trình quản lý sinh viên sử dụng danh sách liên kết

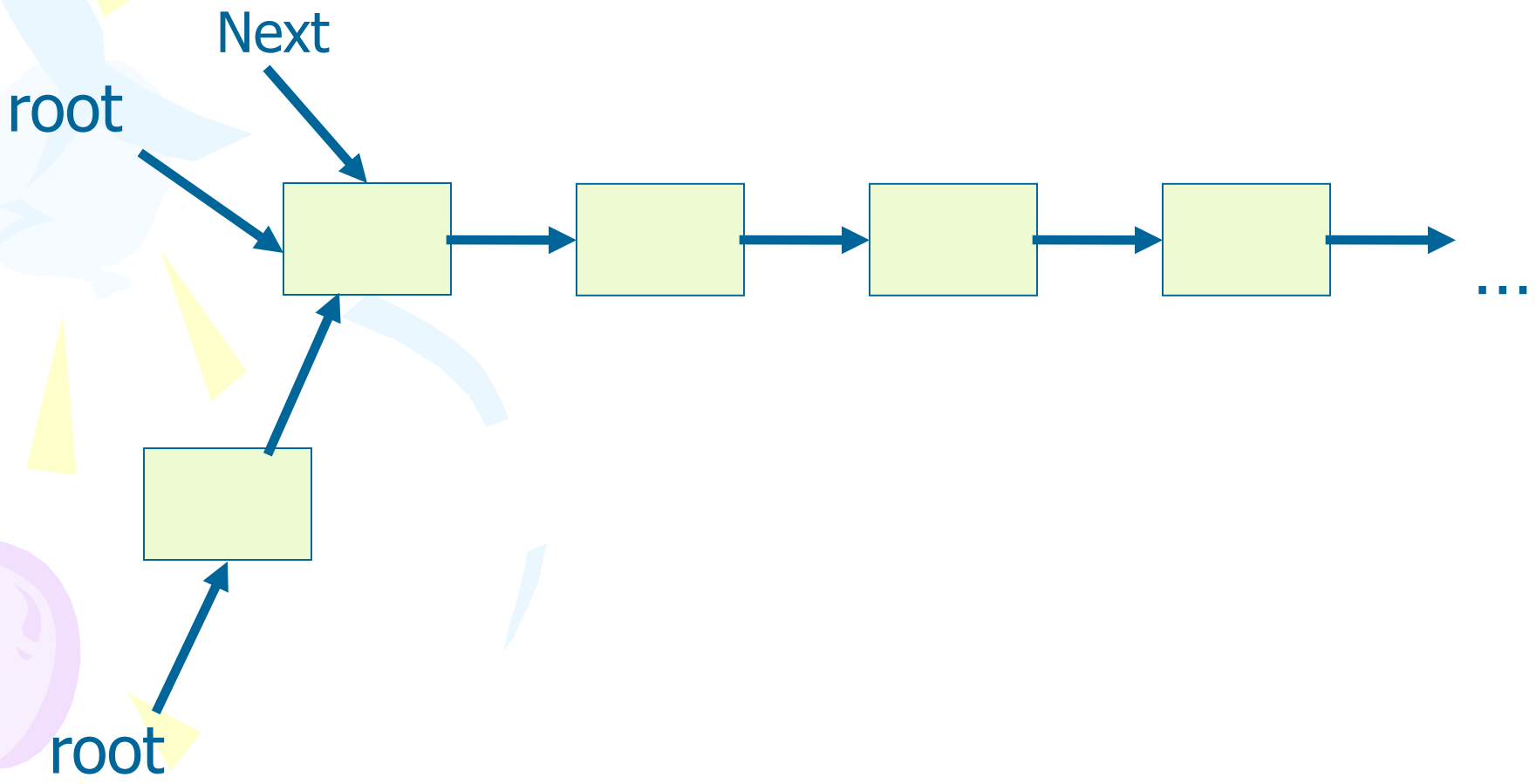
```
typedef struct Student_t {
  char    id[ID_LENGTH];
  char    name[NAME_LENGTH];
  int     grade;

  struct Student_t *next;
} Student;
```

Exercise 3.6 (2)

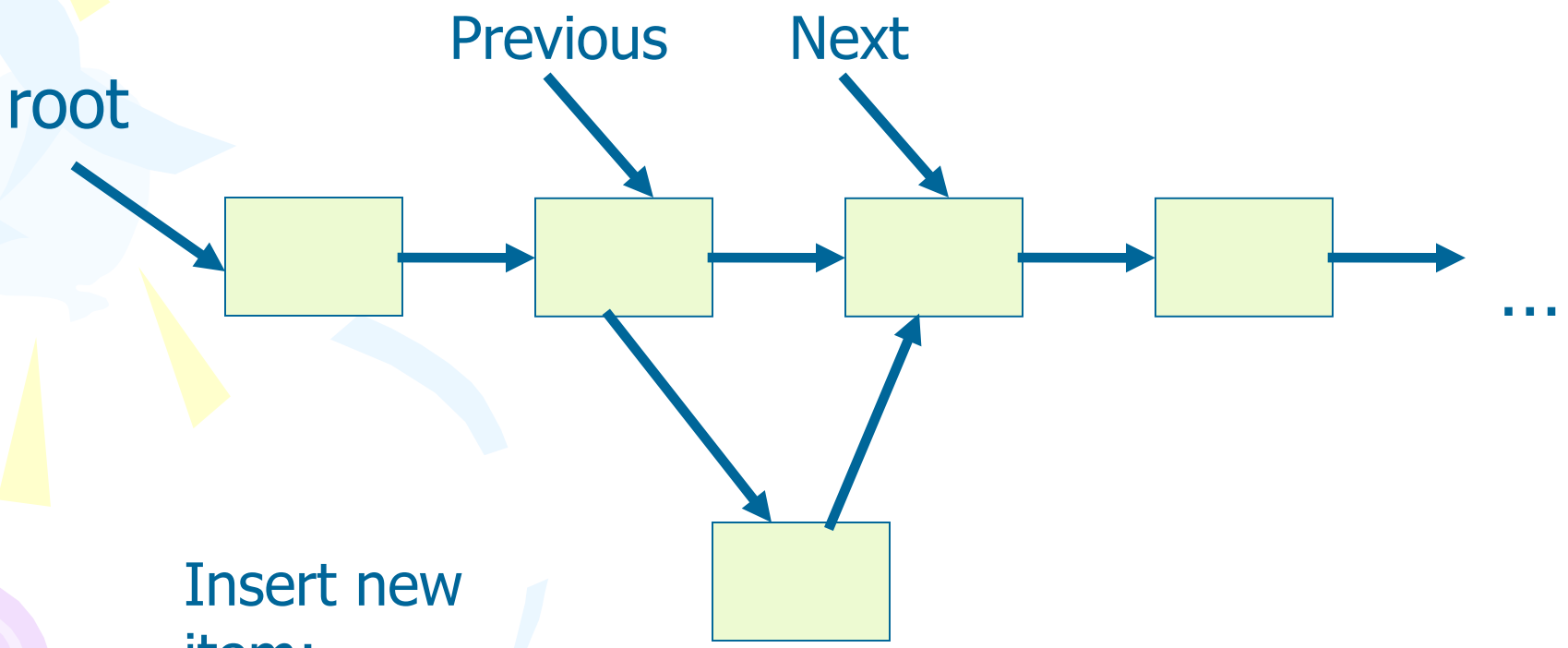
sao cho:

- Danh sách được lưu trữ theo chiều giảm dần của điểm
- Chương trình cung cấp các chức năng:
  - Thêm sinh viên (cần tìm đúng vị trí thêm)
  - Tìm kiếm theo ID: trả về con trỏ
  - Xóa sinh viên với ID cho trước

Thêm sinh viên



Thêm sinh viên (2)



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

    if (root == NULL)
        return to_add;

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

    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;
}
```

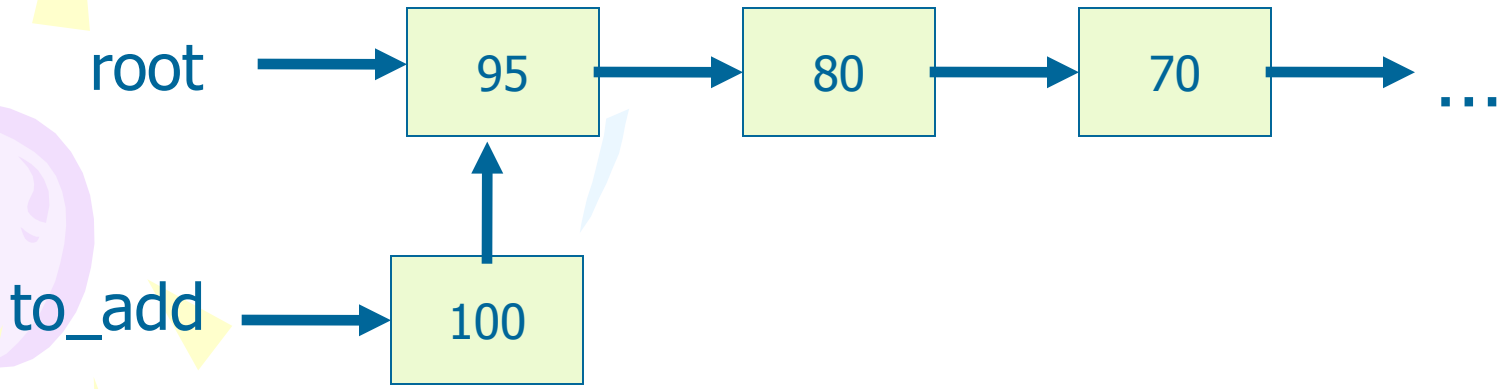
handle empty list

handle beginning

the rest

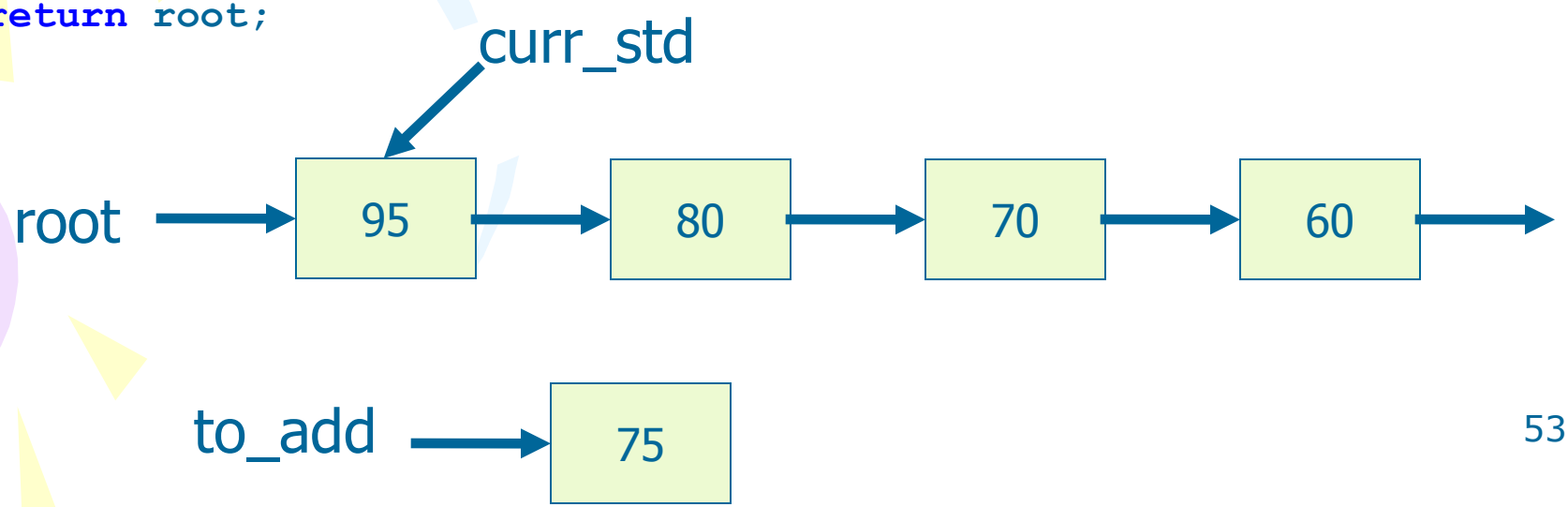
Thêm sinh viên (3)

```
if (root == NULL)
    return to_add;

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



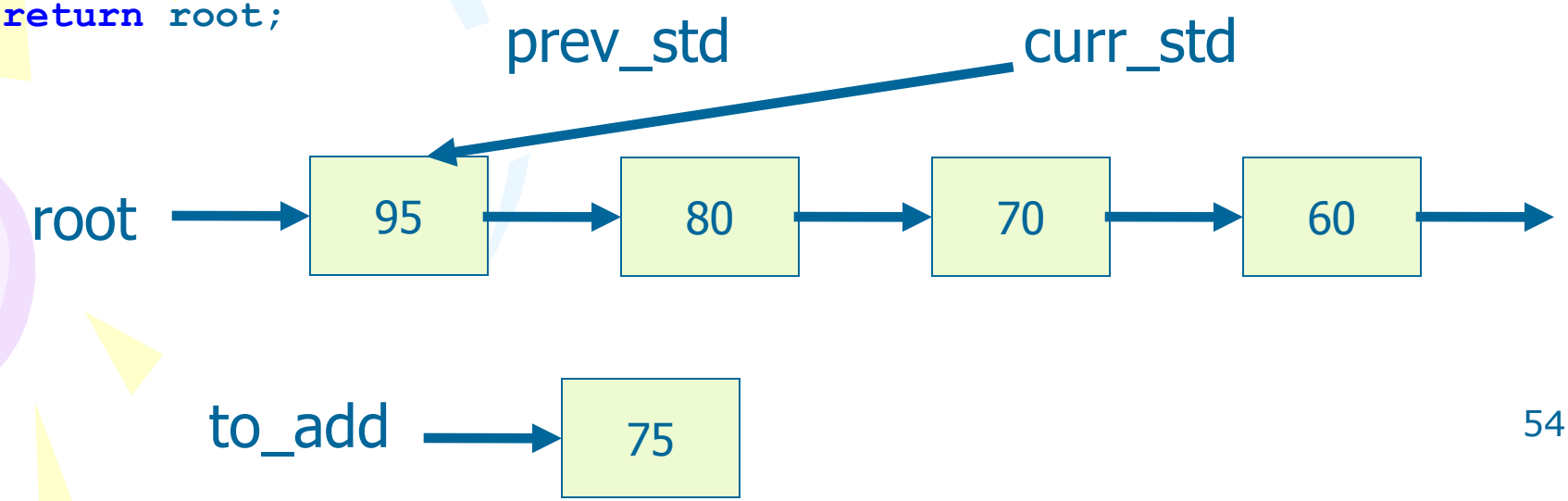
Thêm sinh viên (4)

```
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;
```



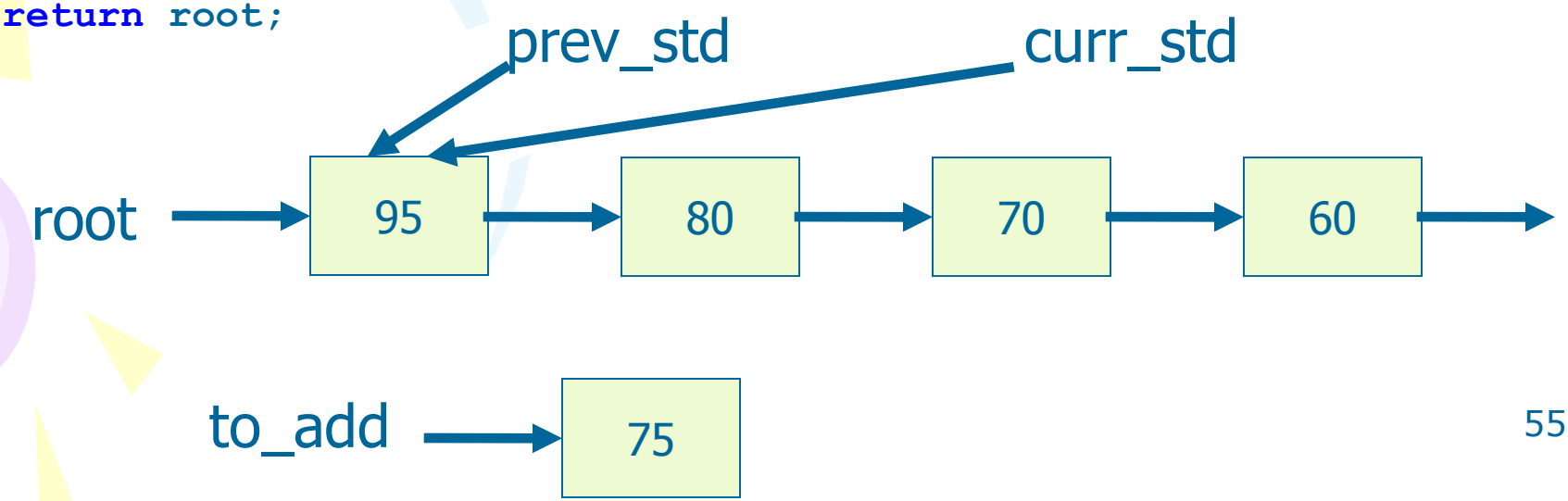
Thêm sinh viên (5)

```
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;
```



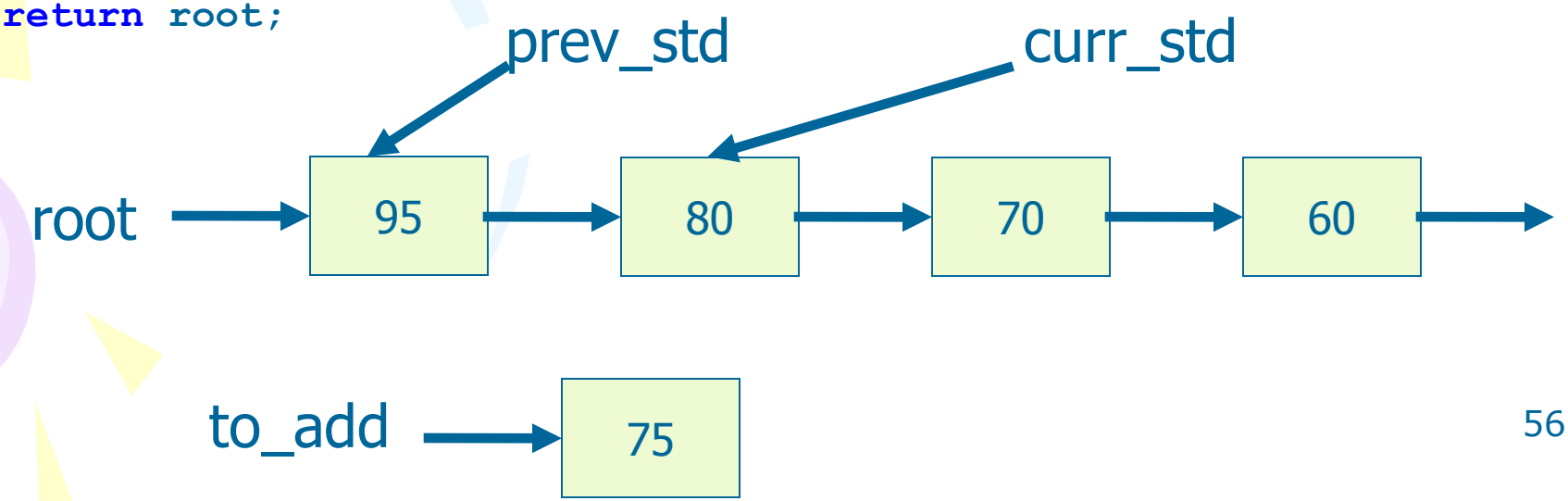
Thêm sinh viên (6)

```
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;
```



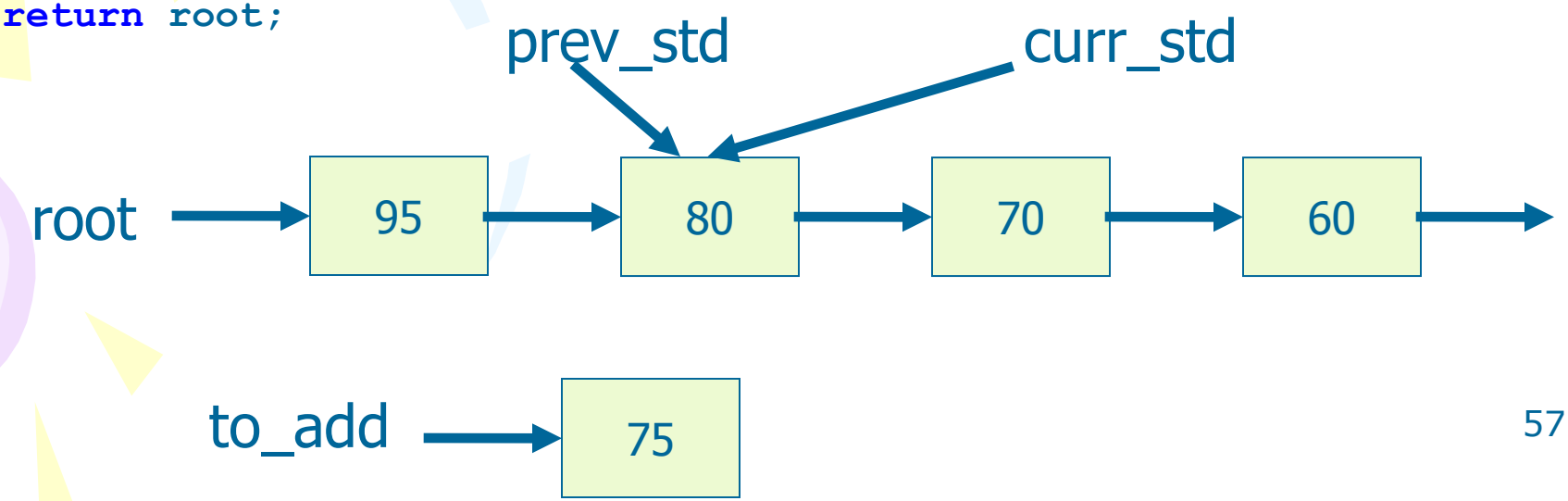
Thêm sinh viên (7)

```
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;
```



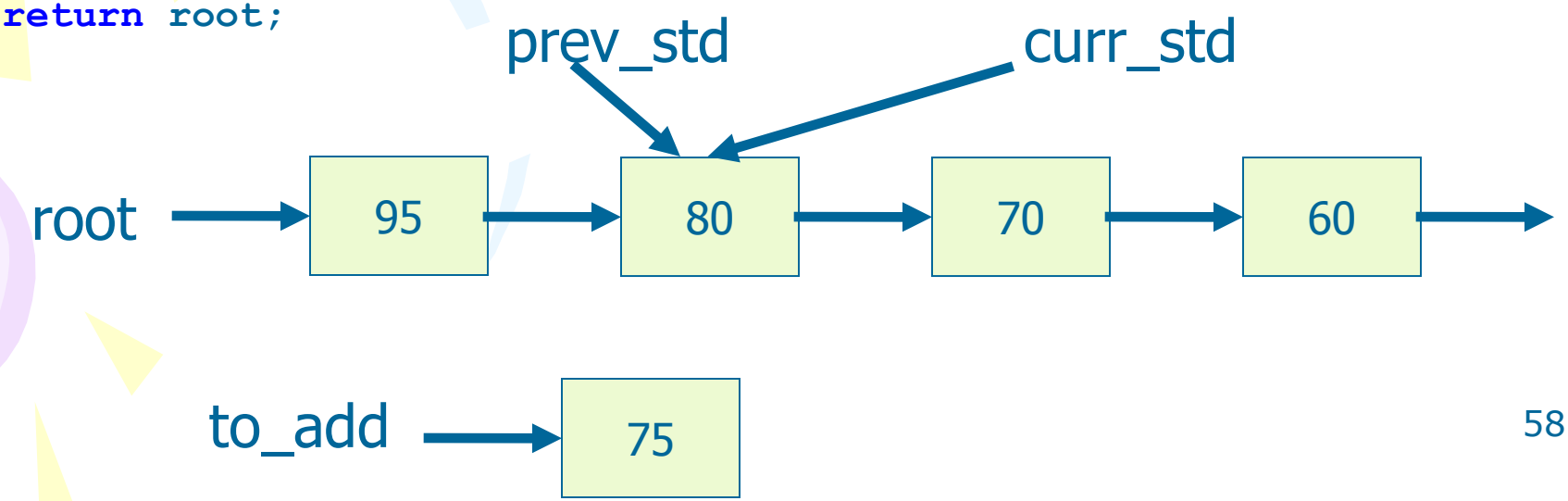
Thêm sinh viên (8)

```
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;
```



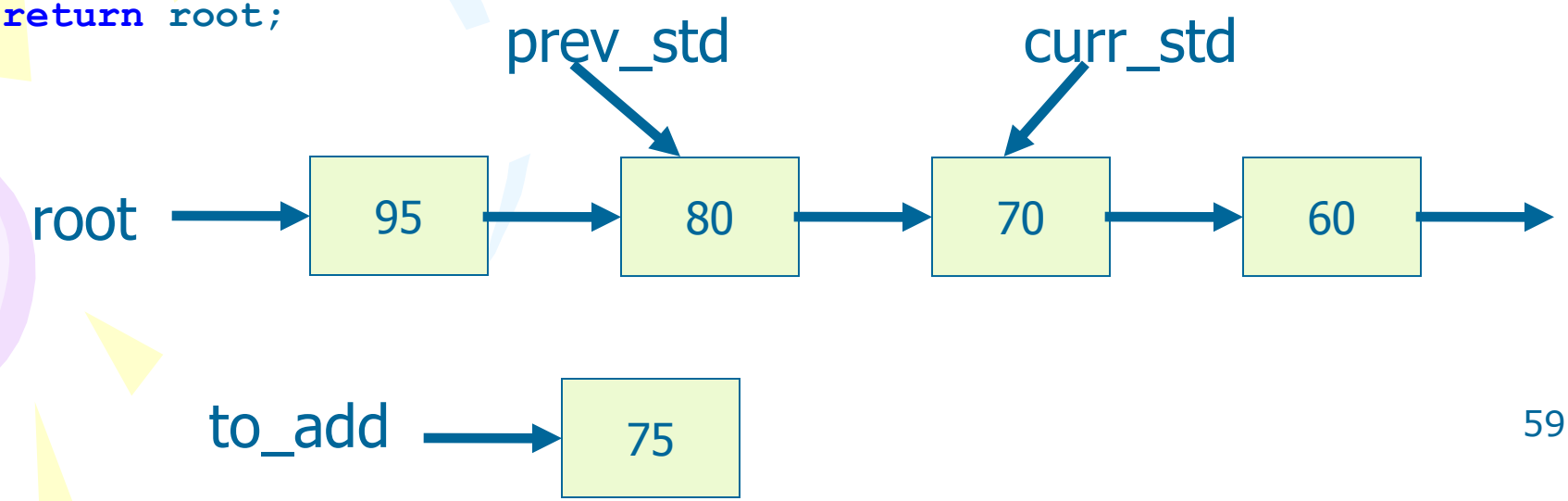
Thêm sinh viên (9)

```
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;
```



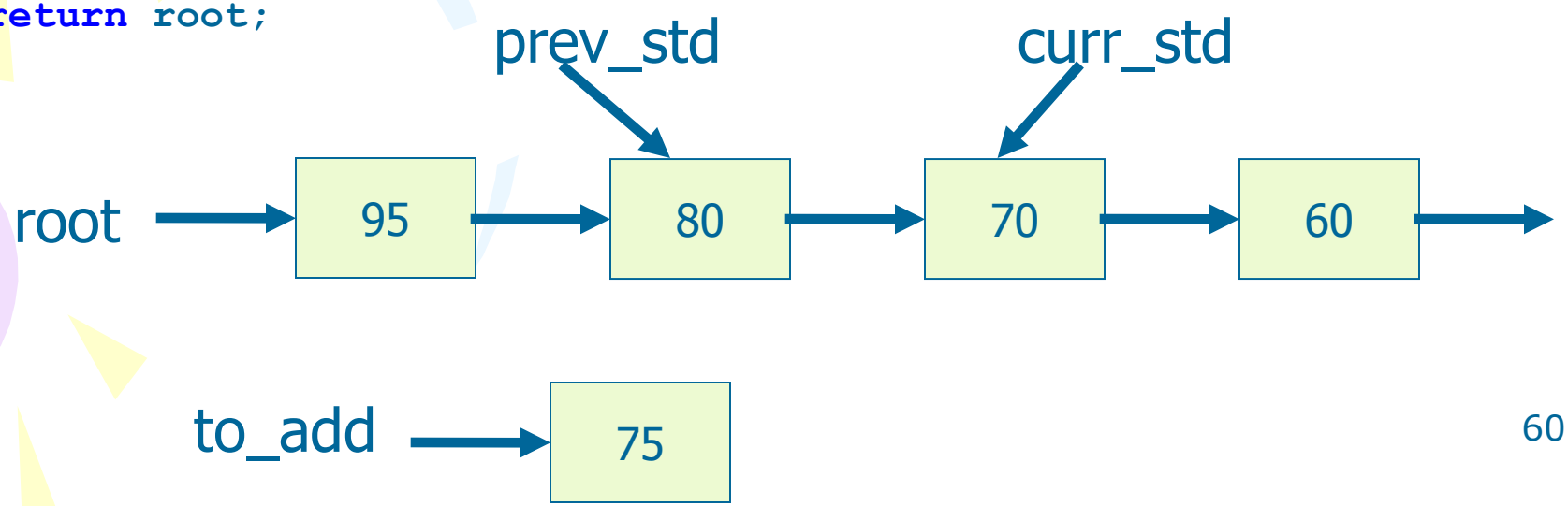
Thêm sinh viên (10)

```
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;
```



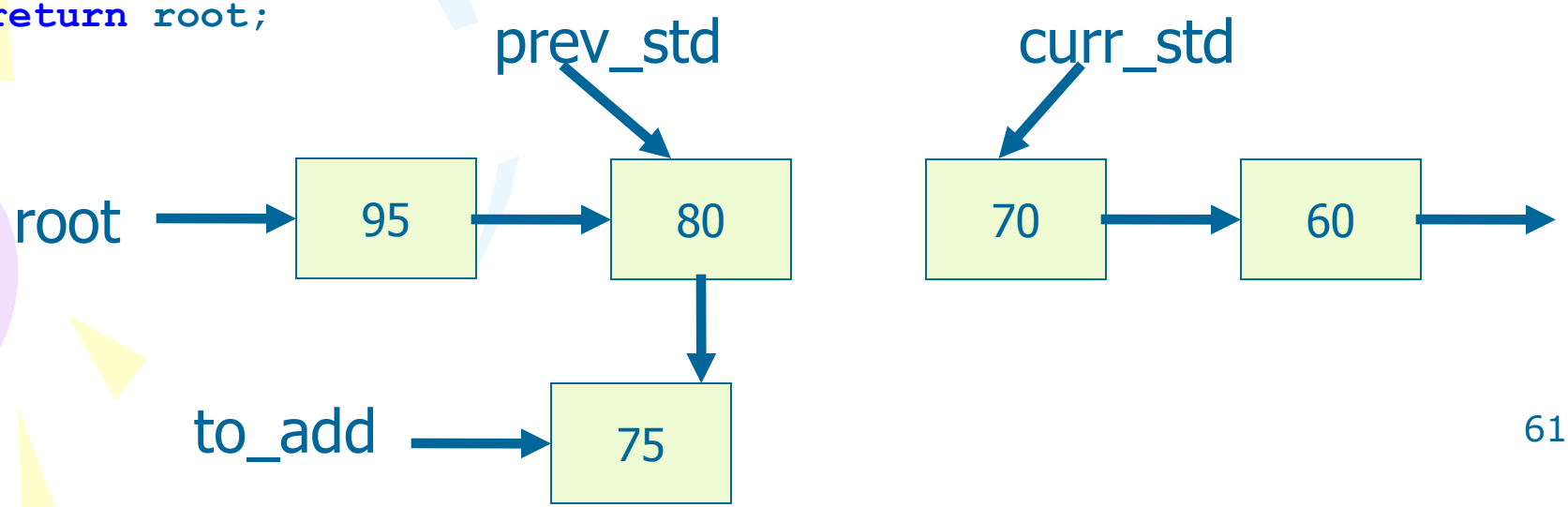
Thêm sinh viên (11)

```
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;
```



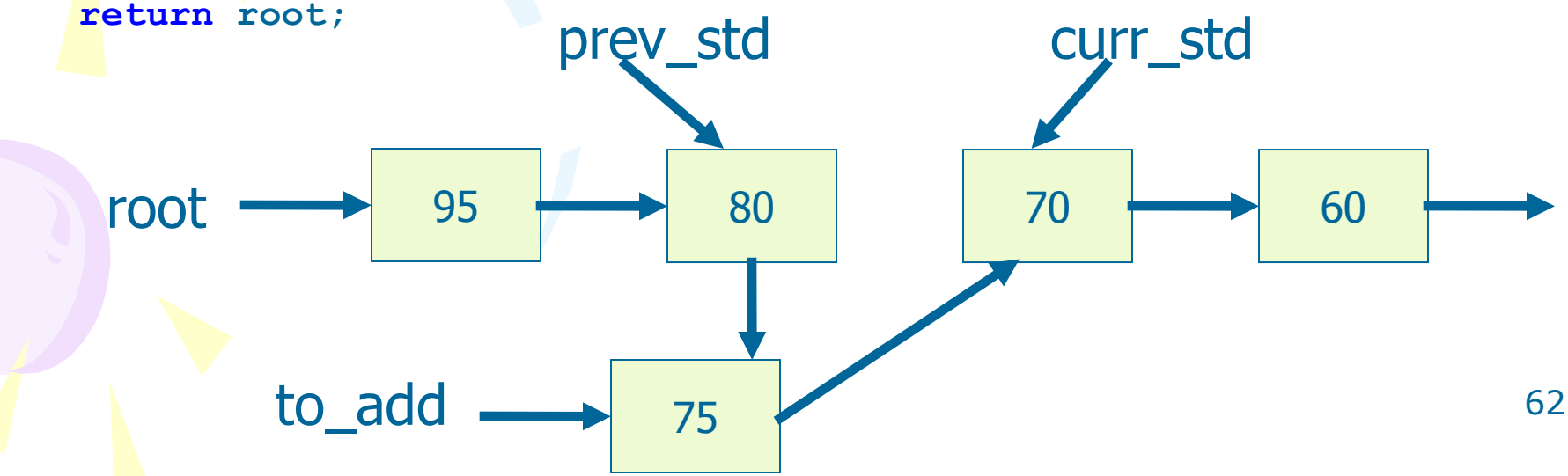
Thêm sinh viên (12)

```
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;
```



Thêm sinh viên (13)

```
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 3.7

- Cài đặt **`find_student`**, nhận vào danh sách sinh viên và ID cần tìm, trả về con trỏ trỏ tới sinh viên tương ứng, trả về NULL nếu không tìm thấy

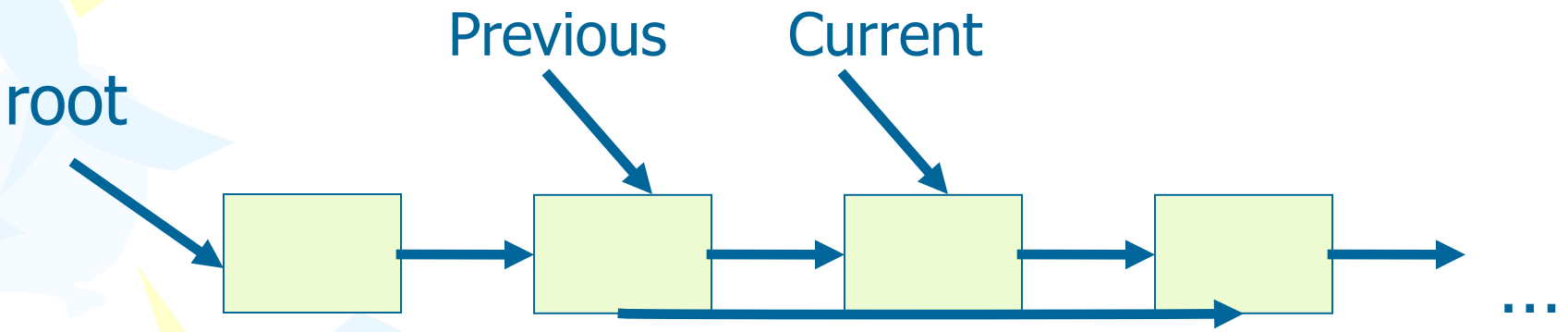
```
Student *find_student(Student *root,
                      char* id);
```

Gợi ý: sử dụng **`strcmp(s1, s2)`**

Xóa sinh viên

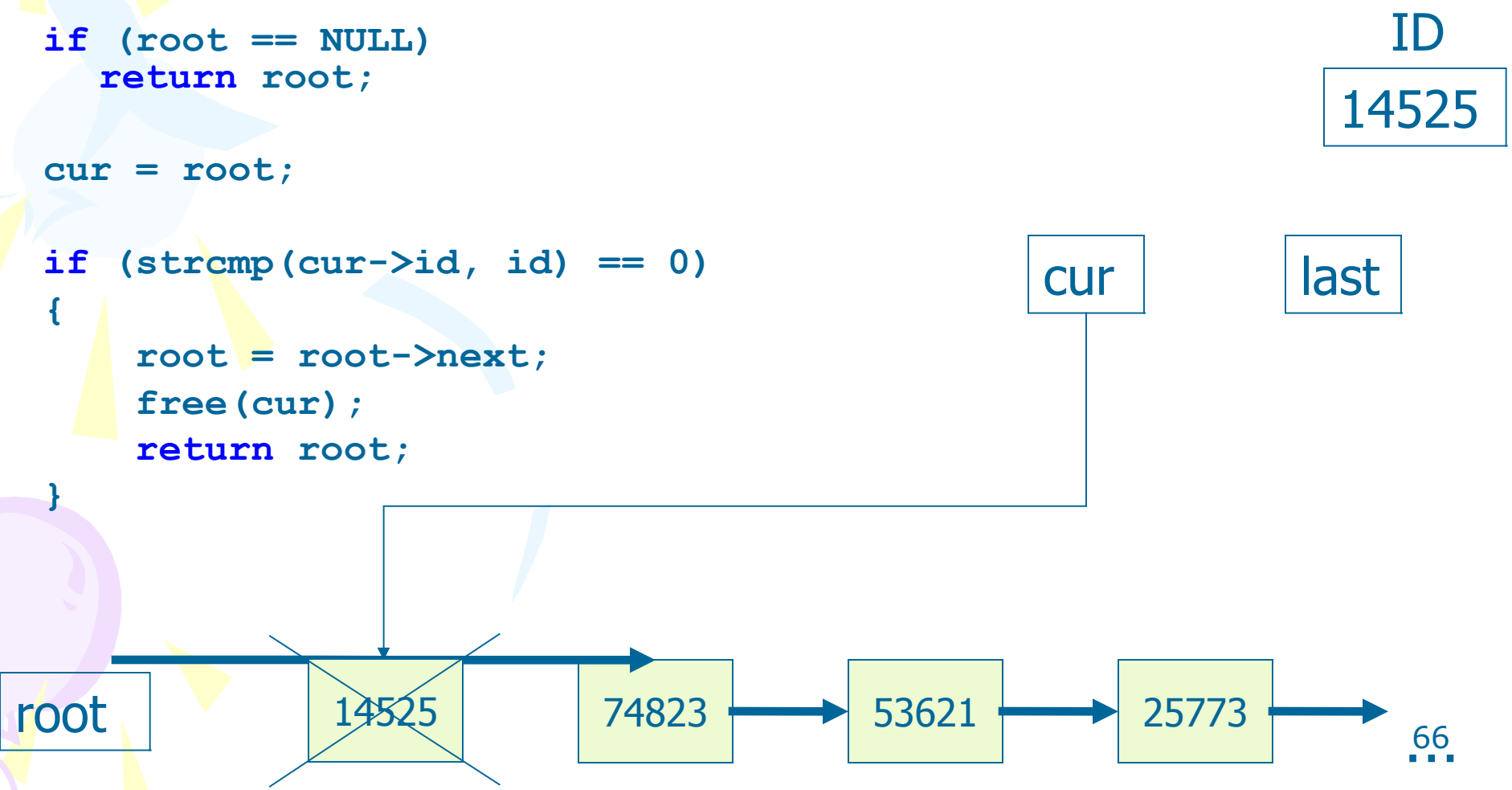
- Xóa sinh viên theo ID
- Sử dụng hàm **remove_student**

Xóa sinh viên (2)



Xóa sinh viên (3)

```
if (root == NULL)
   return root;

cur = root;

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

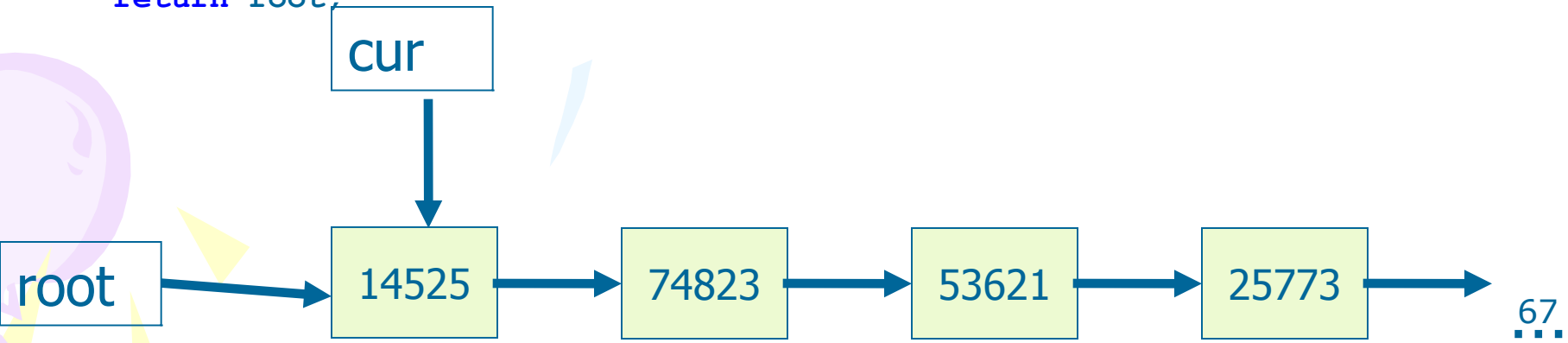


Xóa sinh viên (4)

```
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

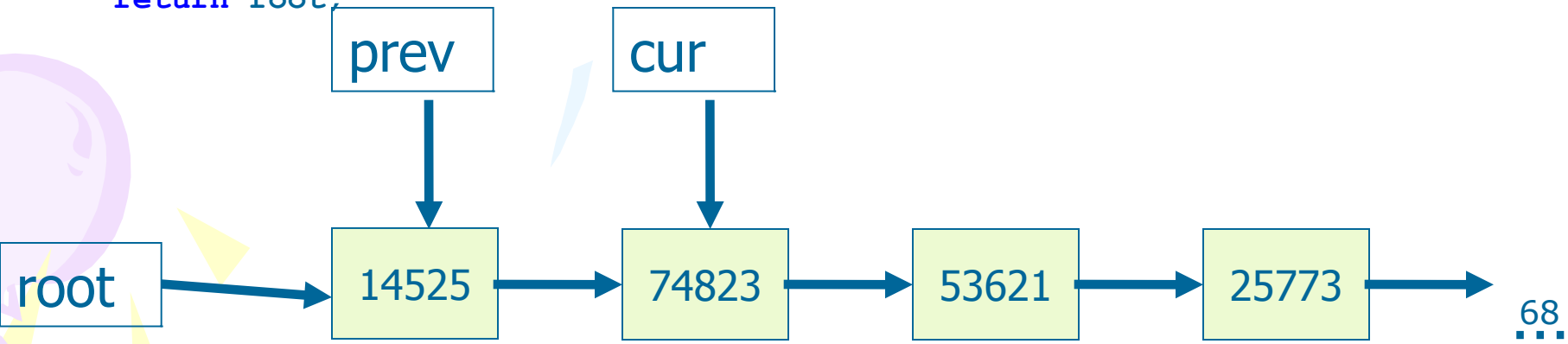


Xóa sinh viên (5)

```
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

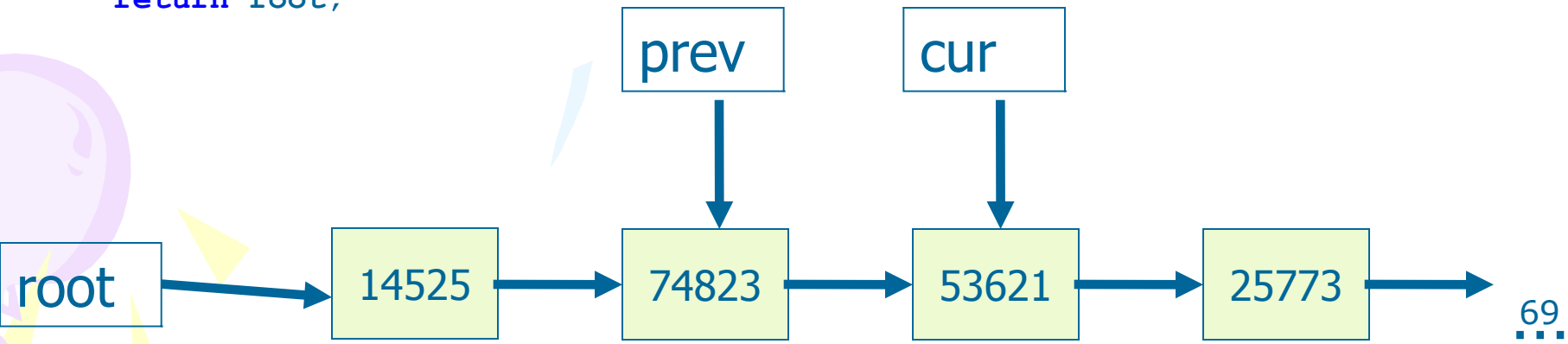


Xóa sinh viên (6)

```
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

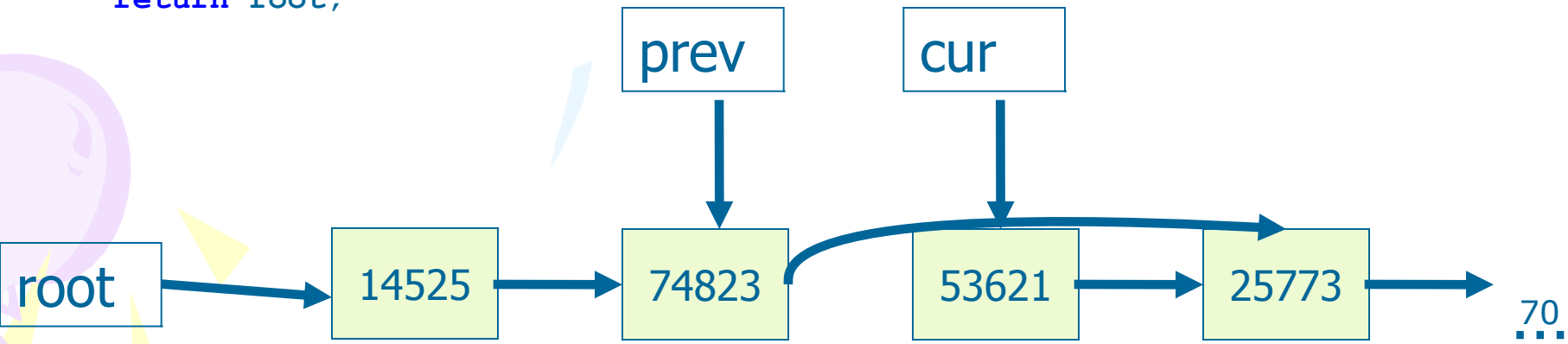


Xóa sinh viên (7)

```
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

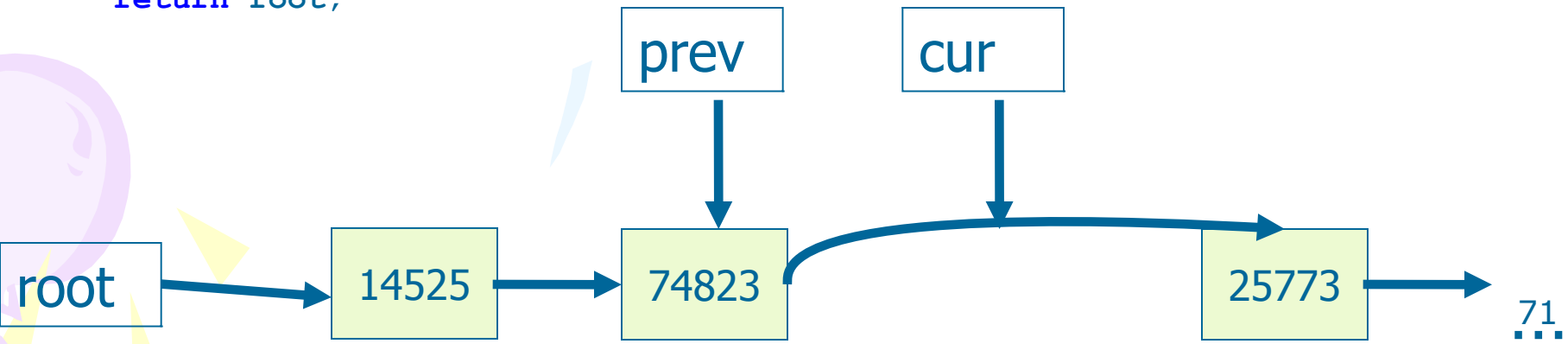


Xóa sinh viên (8)

```
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 3.8

- Thêm hàm **change_grade**. Nhận các tham số là danh sách sinh viên, ID của sinh viên cần thay đổi, và điểm số mới

- Gợi ý – Thêm một sinh viên có ID và điểm số mới. Xóa sinh viên có ID trong danh sách sau đó thêm sinh viên mới này vào.

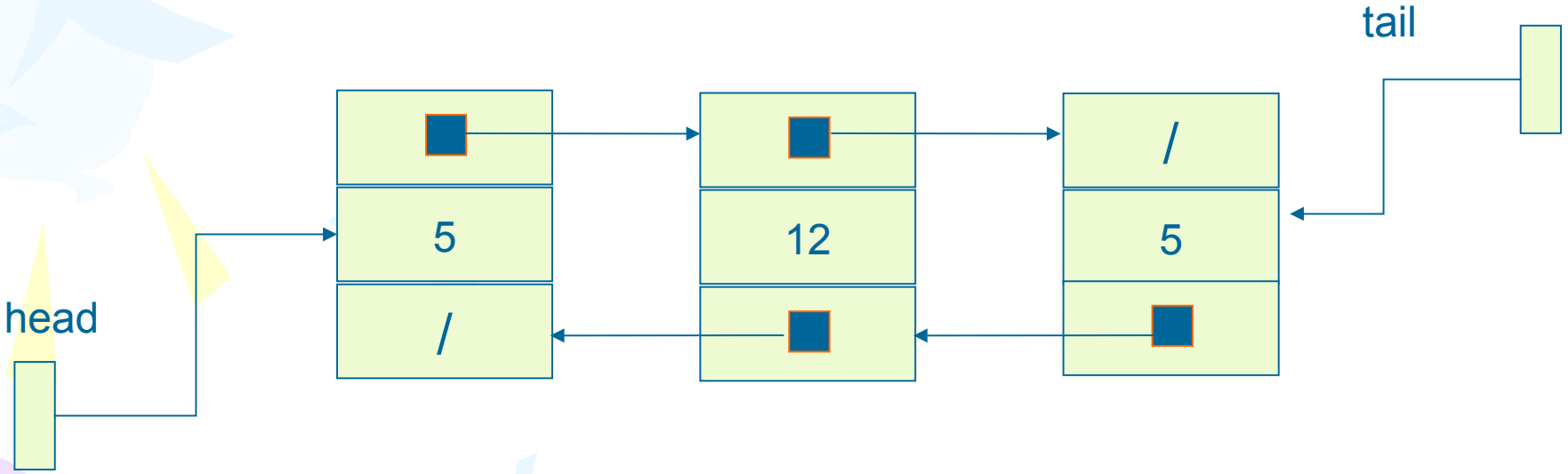
Homework 1

- Xây dựng danh bạ điện thoại
- Khai báo cấu trúc chức các thông tin name, phone number, và e-mail. Chương trình có khả năng lưu trữ danh sách với độ dài bất kỳ
- Gợi ý, sử dụng cấu trúc sau:

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

Danh sách liên kết đôi

- Mỗi phần tử có hai con trỏ, trỏ đến phần tử liền sau và liền trước trong danh sách



Khai báo

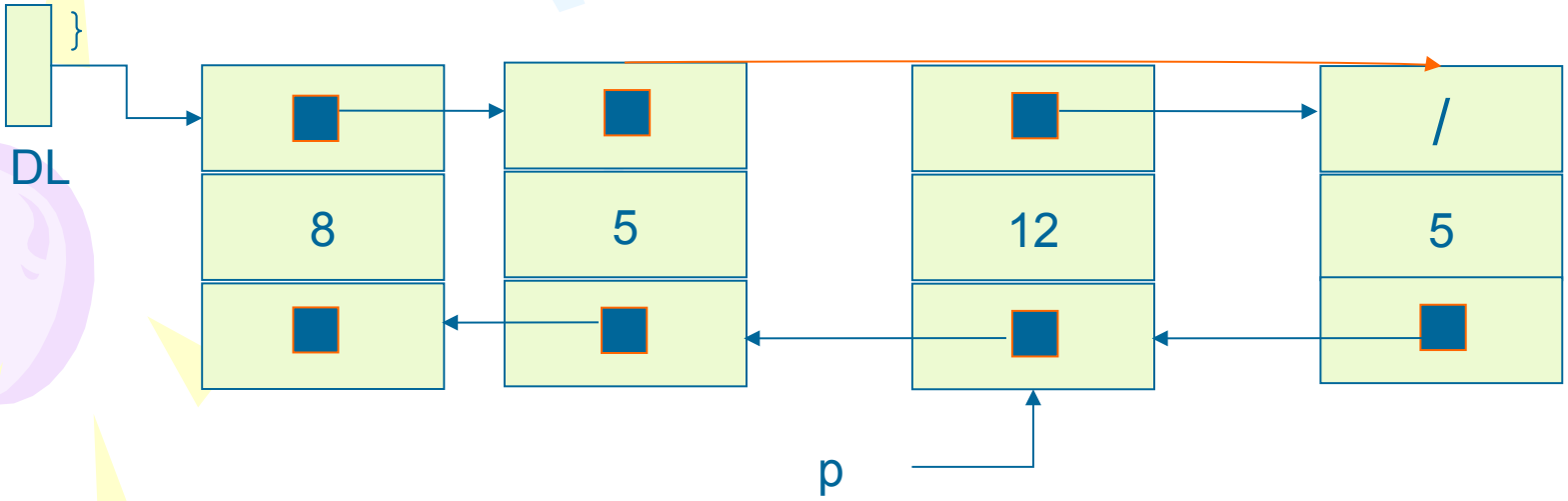
```
typedef ... ElementType;

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

Khởi tạo

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

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

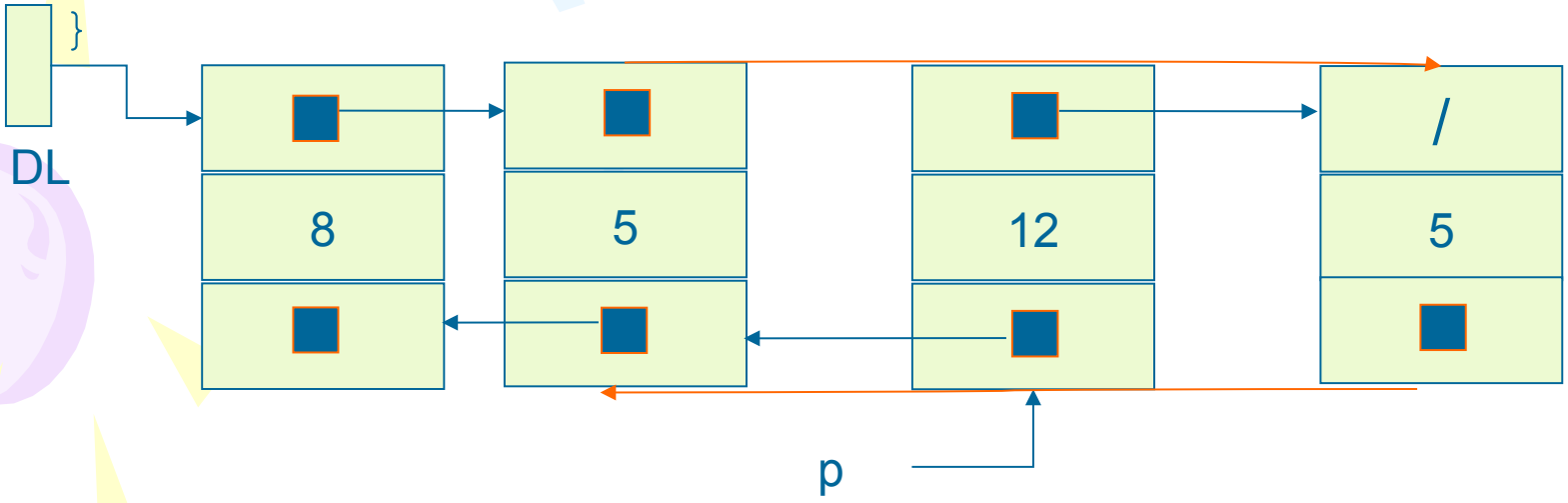
Xóa nút trong danh sách

```
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);
 }
}
```



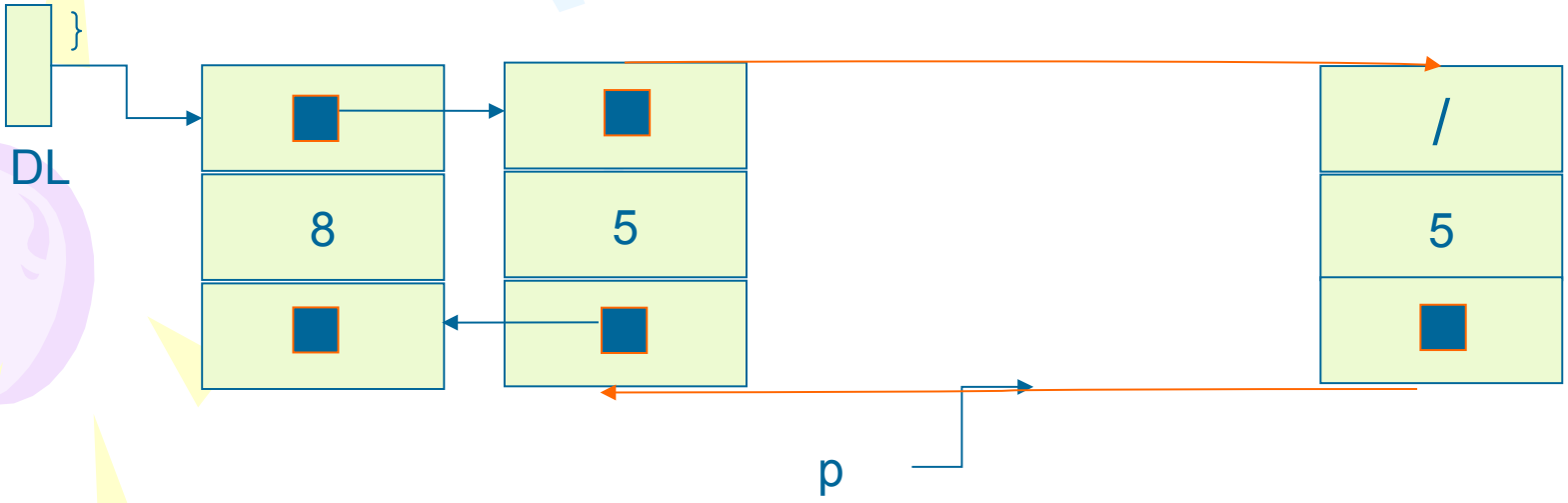
Xóa nút trong danh sách (2)

```
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 nút trong danh sách (3)

```
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);
 }
}
```



Thêm nút vào danh sách

```
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;
    }
}
```

Tạo thư viện

- Create lib.h
  - Type declaration
  - Function prototype
- Create lib.c
  - #include "lib.h"
  - Function Implementation
- Main Program: pro.c
  - #include "lib.h"

- Compile
- gcc – o ex pro.c lib.c

Another way:

gcc –c lib.c

gcc –c pro.c

gcc –o lib.o pro.o ex

Homework 2

- Sử dụng thư viện danh sách liên kết đôi và danh bạ điện thoại, thực hiện các chức năng sau:
  - Tính số các số điện thoại duy nhất trong danh bạ
  - Chia danh sách thành 2 danh sách con