1. **Dòng 1: struct Node\*\* head\_ref**
   * **Giải thích**: head\_ref là con trỏ đến con trỏ (Node\*\*), trỏ đến biến head (con trỏ đến nút đầu tiên của danh sách).
   * **Tại sao**:
     + Dùng Node\*\* vì hàm cần thay đổi head (đặt về NULL) trong hàm gọi. Nếu chỉ dùng Node\* head, thay đổi head trong hàm không ảnh hưởng đến head bên ngoài (do truyền tham trị).
     + **Ví dụ**: Nếu danh sách là [5 -> 10 -> 15], head\_ref chứa địa chỉ của head, và \*head\_ref là head, trỏ đến nút 5.
2. **Dòng 2: struct Node\* current = \*head\_ref;**
   * **Giải thích**: Tạo con trỏ current trỏ đến nút đầu tiên của danh sách (\*head\_ref là head).
   * **Tại sao**:
     + Cần một con trỏ để duyệt qua từng nút và giải phóng chúng. Bắt đầu từ nút đầu tiên để đảm bảo không bỏ sót nút nào.
     + **Ví dụ**: Với danh sách [5 -> 10 -> 15], current trỏ đến nút 5.
3. **Dòng 3-6: Vòng lặp giải phóng**

c

CollapseWrap

Copy

while (current != NULL) {

struct Node\* temp = current;

current = current->next;

free(temp);

}

* + **Giải thích**:
    - Vòng lặp chạy khi current không phải NULL (tức còn nút trong danh sách).
    - Lưu current vào temp để giữ địa chỉ nút hiện tại.
    - Di chuyển current đến nút tiếp theo (current->next).
    - Giải phóng bộ nhớ của nút temp bằng free(temp).
  + **Tại sao**:
    - Trong C, mỗi nút được tạo bằng malloc (trong create\_node) phải được giải phóng bằng free để tránh rò rỉ bộ nhớ (memory leak).
    - Dùng temp để lưu current trước khi di chuyển, vì sau khi gọi free(temp), truy cập current->next sẽ gây lỗi (truy cập bộ nhớ đã giải phóng).
    - Kiểm tra current != NULL để đảm bảo không truy cập bộ nhớ không hợp lệ khi danh sách rỗng hoặc đã duyệt hết.
    - **Ví dụ**: Với [5 -> 10 -> 15]:
      * Lần 1: temp = 5, current = 10, giải phóng nút 5.
      * Lần 2: temp = 10, current = 15, giải phóng nút 10.
      * Lần 3: temp = 15, current = NULL, giải phóng nút 15.

1. **Dòng 7: \*head\_ref = NULL;**
   * **Giải thích**: Đặt head (thông qua \*head\_ref) về NULL để biểu thị danh sách rỗng.
   * **Tại sao**:
     + Yêu cầu của hàm là đặt head về NULL sau khi giải phóng để đảm bảo danh sách được đánh dấu là rỗng, tránh truy cập nhầm vào bộ nhớ đã giải phóng.
     + Điều này cũng giúp các hàm khác (như print\_list) dễ dàng kiểm tra danh sách rỗng.
     + **Ví dụ**: Sau khi giải phóng [5 -> 10 -> 15], head trở thành NULL.

**Hình ảnh minh họa**

Danh sách: [5 -> 10 -> 15]:

* Trước: head -> [5 -> 10 -> 15]
* Lần 1: temp -> [5], current -> [10 -> 15], giải phóng nút 5
* Lần 2: temp -> [10], current -> [15], giải phóng nút 10
* Lần 3: temp -> [15], current = NULL, giải phóng nút 15
* Sau cùng: head = NULL

**Tại sao cần giải phóng bộ nhớ?**

* Trong C, bộ nhớ động (cấp phát bằng malloc) không tự động được thu hồi. Nếu không giải phóng, chương trình sẽ chiếm dụng bộ nhớ, gây rò rỉ (memory leak), đặc biệt khi danh sách lớn.
* Đặt head = NULL giúp chương trình biết danh sách đã rỗng, tránh lỗi truy cập bộ nhớ không hợp lệ.

**2. Hàm main**

c

CollapseWrapRun

Copy

int main() {

struct Node\* head = NULL;

*// Tạo danh sách [5, 10, 15]*

insert\_at\_head(&head, 15);

insert\_at\_head(&head, 10);

insert\_at\_head(&head, 5);

printf("Danh sach ban dau: ");

print\_list(head);

*// Giải phóng danh sách*

free\_list(&head);

printf("Danh sach sau khi giai phong: ");

print\_list(head);

return 0;

}

**Giải thích từng bước và lý do**

1. **Dòng 2: struct Node\* head = NULL;**
   * **Giải thích**: Khởi tạo head trỏ đến NULL, biểu thị danh sách rỗng.
   * **Tại sao**:
     + Ban đầu, danh sách chưa có nút nào, nên head phải là NULL để các hàm như insert\_at\_head xử lý đúng trường hợp danh sách rỗng.
     + **Ví dụ**: Trước khi chèn, danh sách là rỗng.
2. **Dòng 4-6: Tạo danh sách [5, 10, 15]**

c

CollapseWrap

Copy

insert\_at\_head(&head, 15);

insert\_at\_head(&head, 10);

insert\_at\_head(&head, 5);

* + **Giải thích**: Gọi insert\_at\_head 3 lần để tạo danh sách. Vì chèn vào đầu, thứ tự gọi ngược lại tạo ra [5 -> 10 -> 15].
  + **Tại sao**:
    - insert\_at\_head đơn giản và nhanh (không cần duyệt danh sách), phù hợp để tạo danh sách theo yêu cầu.
    - Truyền &head để hàm có thể cập nhật head khi chèn nút mới.
    - **Ví dụ**:
      * Sau insert\_at\_head(&head, 15): [15]
      * Sau insert\_at\_head(&head, 10): [10 -> 15]
      * Sau insert\_at\_head(&head, 5): [5 -> 10 -> 15]

1. **Dòng 8-9: In danh sách ban đầu**

c

CollapseWrap

Copy

printf("Danh sach ban dau: ");

print\_list(head);

* + **Giải thích**: In danh sách để xác nhận: 5 10 15.
  + **Tại sao**:
    - Yêu cầu in danh sách ban đầu để kiểm tra xem nó được tạo đúng trước khi giải phóng.
    - Hàm print\_list duyệt và in từng giá trị, giúp người dùng thấy cấu trúc danh sách.

1. **Dòng 11-13: Giải phóng và in lại**

c

CollapseWrap

Copy

free\_list(&head);

printf("Danh sach sau khi giai phong: ");

print\_list(head);

* + **Giải thích**:
    - Gọi free\_list(&head) để giải phóng toàn bộ danh sách và đặt head = NULL.
    - In danh sách để xác nhận nó rỗng (print\_list sẽ in Danh sach rong).
  + **Tại sao**:
    - Yêu cầu giải phóng bộ nhớ và kiểm tra danh sách rỗng sau khi giải phóng.
    - Truyền &head để free\_list có thể đặt head = NULL.
    - In lại để xác nhận danh sách đã được giải phóng hoàn toàn.

**Kết quả chạy**

text

CollapseWrap

Copy

Danh sach ban dau: 5 10 15

Danh sach sau khi giai phong: Danh sach rong

**3. Hàm hỗ trợ: print\_list (đã được cập nhật)**

c

CollapseWrap

Copy

void print\_list(struct Node\* head) {

if (head == NULL) {

printf("Danh sach rong\n");

return;

}

struct Node\* current = head;

while (current != NULL) {

printf("%d ", current->data);

current = current->next;

}

printf("\n");

}

**Giải thích ngắn gọn và lý do**

* **Cập nhật**: Thêm kiểm tra head == NULL để in Danh sach rong khi danh sách rỗng.
* **Tại sao**: Yêu cầu in danh sách sau khi giải phóng để xác nhận nó rỗng. Kiểm tra này giúp hiển thị thông báo rõ ràng thay vì không in gì.
* **Quy trình**: Nếu danh sách không rỗng, duyệt và in từng giá trị; nếu rỗng, in thông báo.

**Lời khuyên cho người mới**

* **Hiểu con trỏ**: Con trỏ (Node\*) là mũi tên chỉ đến một nút. Node\*\* là mũi tên chỉ đến mũi tên khác, cho phép thay đổi head.
* **Vẽ sơ đồ**: Vẽ danh sách trên giấy, đánh dấu head, current, temp, next để thấy cách giải phóng từng nút.
* **Tại sao giải phóng bộ nhớ?**:
  + Mỗi nút được tạo bằng malloc chiếm bộ nhớ. Nếu không giải phóng, chương trình sẽ gây rò rỉ bộ nhớ, đặc biệt với danh sách lớn.
  + Đặt head = NULL giúp tránh lỗi truy cập bộ nhớ đã giải phóng.
* **Kiểm tra trường hợp rỗng**: Luôn kiểm tra head == NULL để xử lý danh sách rỗng một cách an toàn.

**Tóm tắt với lý do**

* **free\_list**:
  + Giải phóng từng nút bằng free để tránh rò rỉ bộ nhớ.
  + Dùng Node\*\* để đặt head = NULL, đáp ứng yêu cầu.
  + Dùng temp để tránh lỗi truy cập bộ nhớ đã giải phóng.
* **main**:
  + Tạo danh sách [5, 10, 15] bằng insert\_at\_head vì nhanh và đơn giản.
  + In trước và sau khi giải phóng để xác nhận kết quả.
* **Tại sao cần như vậy**:
  + Đảm bảo chương trình an toàn (không lỗi truy cập bộ nhớ).
  + Đáp ứng yêu cầu tạo, in, giải phóng, và xác nhận danh sách rỗng.
  + Tránh rò rỉ bộ nhớ, một vấn đề quan trọng trong C.