**Cấu trúc Node**

typedef struct Node {

int data;

struct Node\* next;

} Node;

* **Giải thích**:
  + data: Lưu giá trị số nguyên của nút (ví dụ: 1, 2, 3).
  + next: Con trỏ trỏ tới nút tiếp theo trong danh sách, hoặc NULL nếu là nút cuối.
  + typedef: Đặt tên alias là Node để viết code ngắn gọn.
* **Tại sao**:
  + Danh sách liên kết đơn cần mỗi nút lưu trữ một giá trị và một liên kết tới nút tiếp theo để tạo thành chuỗi.
  + next cho phép điều chỉnh con trỏ để hợp nhất hai danh sách.
* **Ví dụ**:
  + Với danh sách [1 -> 3 -> 5]:
    - Nút đầu: data = 1, next trỏ tới nút có data = 3.
    - Nút cuối: data = 5, next = NULL.

**2. Hàm create\_node**

Node\* create\_node(int data) {

Node\* new\_node = (Node\*)malloc(sizeof(Node));

new\_node->data = data;

new\_node->next = NULL;

return new\_node;

}

* **Giải thích**:
  1. **Dòng 1**: Cấp phát bộ nhớ động cho một nút mới bằng malloc.
  2. **Dòng 2**: Gán giá trị data vào trường data của nút mới.
  3. **Dòng 3**: Đặt next về NULL, vì nút mới chưa liên kết.
  4. **Dòng 4**: Trả về con trỏ tới nút mới.
* **Tại sao**:
  1. Hàm này tạo nút mới khi nhập danh sách từ người dùng.
  2. malloc cấp phát bộ nhớ động trên heap, phù hợp cho danh sách liên kết.
  3. Đặt next = NULL để tránh lỗi truy cập bộ nhớ ngẫu nhiên.
* **Ví dụ**:
  1. Gọi create\_node(1): Tạo nút với data = 1, next = NULL.

**3. Hàm insert\_at\_tail**

void insert\_at\_tail(Node\*\* head\_ref, int data) {

Node\* new\_node = create\_node(data);

if (\*head\_ref == NULL) {

\*head\_ref = new\_node;

return;

}

Node\* current = \*head\_ref;

while (current->next != NULL) {

current = current->next;

}

current->next = new\_node;

}

* **Giải thích**:
  1. **Dòng 1**: Nhận head\_ref để cập nhật head nếu danh sách rỗng.
  2. **Dòng 2**: Tạo nút mới với data.
  3. **Dòng 3-5**: Nếu danh sách rỗng, đặt head trỏ tới nút mới.
  4. **Dòng 6-9**: Duyệt tới nút cuối, liên kết nút cuối với nút mới.
* **Tại sao**:
  1. Giữ thứ tự nhập của người dùng (ví dụ: nhập 1, 3, 5 tạo [1 -> 3 -> 5]).
  2. Dùng Node\*\* để xử lý danh sách rỗng.
  3. Độ phức tạp O(n) thời gian do duyệt tới nút cuối.
* **Ví dụ**:
  1. Ban đầu: head = NULL.
  2. Gọi insert\_at\_tail(&head, 1): [1].
  3. Gọi insert\_at\_tail(&head, 3): [1 -> 3].

**4. Hàm print\_list**

void print\_list(Node\* head) {

if (head == NULL) {

printf("Danh sach rong\n");

return;

}

Node\* current = head;

while (current != NULL) {

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

if (current->next != NULL) {

printf(" -> ");

}

current = current->next;

}

printf("\n");

}

* **Giải thích**:
  1. **Dòng 1**: Nhận head trỏ tới nút đầu.
  2. **Dòng 2-4**: Nếu danh sách rỗng, in thông báo và thoát.
  3. **Dòng 5-10**: Duyệt danh sách, in giá trị data, thêm " -> " nếu có nút tiếp theo.
  4. **Dòng 11**: In ký tự xuống dòng.
* **Tại sao**:
  1. In danh sách để kiểm tra hai danh sách nhập vào và danh sách hợp nhất.
  2. Kiểm tra head == NULL để xử lý danh sách rỗng.
  3. Dấu " -> " giúp hình dung liên kết giữa các nút.
* **Ví dụ**:
  1. Với danh sách [1 -> 3 -> 5]: In 1 -> 3 -> 5.
  2. Với danh sách rỗng: In Danh sach rong.

**5. Hàm merge\_lists**

Node\* merge\_lists(Node\* head1, Node\* head2) {

*// Nếu một trong hai danh sách rỗng, trả về danh sách còn lại*

if (head1 == NULL) return head2;

if (head2 == NULL) return head1;

Node\* merged\_head = NULL;

Node\*\* tail\_ref = &merged\_head; *// Con trỏ đến con trỏ để cập nhật đuôi*

*// So sánh và hợp nhất*

while (head1 != NULL && head2 != NULL) {

if (head1->data <= head2->data) {

\*tail\_ref = head1;

head1 = head1->next;

} else {

\*tail\_ref = head2;

head2 = head2->next;

}

tail\_ref = &((\*tail\_ref)->next);

}

*// Gắn danh sách còn lại (nếu có)*

if (head1 != NULL) \*tail\_ref = head1;

if (head2 != NULL) \*tail\_ref = head2;

return merged\_head;

}

* **Giải thích từng dòng**:
  1. **Dòng 1**: Nhận head1 và head2, trả về con trỏ tới đầu danh sách hợp nhất.
  2. **Dòng 2-3**: Nếu một danh sách rỗng, trả về danh sách còn lại.
  3. **Dòng 5**: Khởi tạo merged\_head = NULL cho danh sách hợp nhất.
  4. **Dòng 6**: tail\_ref là con trỏ đến con trỏ, dùng để cập nhật liên kết cuối của danh sách hợp nhất.
  5. **Dòng 8-14**: So sánh và hợp nhất:
     + Nếu head1->data <= head2->data, lấy nút từ head1, di chuyển head1.
     + Ngược lại, lấy nút từ head2, di chuyển head2.
     + Cập nhật tail\_ref để trỏ tới next của nút vừa thêm.
  6. **Dòng 16-17**: Gắn danh sách còn lại (nếu có) vào cuối danh sách hợp nhất.
  7. **Dòng 19**: Trả về merged\_head.
* **Tại sao**:
  1. Chỉ điều chỉnh con trỏ, không tạo nút mới, đáp ứng yêu cầu đề bài.
  2. Dùng tail\_ref để theo dõi nút cuối của danh sách hợp nhất, giúp thêm nút mà không cần duyệt lại.
  3. Độ phức tạp O(n1 + n2) thời gian (duyệt mỗi nút một lần), O(1) không gian (không cấp phát bộ nhớ mới).
* **Ví dụ** (với [1 -> 3 -> 5] và [2 -> 4 -> 6]):
  1. Ban đầu: head1 -> [1 -> 3 -> 5], head2 -> [2 -> 4 -> 6], merged\_head = NULL.
  2. Lần 1: 1 <= 2, lấy nút 1, merged\_head -> [1], head1 -> [3 -> 5].
  3. Lần 2: 3 > 2, lấy nút 2, [1 -> 2], head2 -> [4 -> 6].
  4. Lần 3: 3 <= 4, lấy nút 3, [1 -> 2 -> 3], head1 -> [5].
  5. Lần 4: 5 > 4, lấy nút 4, [1 -> 2 -> 3 -> 4], head2 -> [6].
  6. Lần 5: 5 <= 6, lấy nút 5, [1 -> 2 -> 3 -> 4 -> 5], head1 = NULL.
  7. Lần 6: Gắn head2 -> [6], kết quả: [1 -> 2 -> 3 -> 4 -> 5 -> 6].

**6. Hàm free\_list**

void free\_list(Node\*\* head\_ref) {

Node\* current = \*head\_ref;

while (current != NULL) {

Node\* temp = current;

current = current->next;

free(temp);

}

\*head\_ref = NULL;

}

* **Giải thích**:
  1. **Dòng 1**: Nhận head\_ref để thay đổi head.
  2. **Dòng 2-6**: Duyệt danh sách, lưu current vào temp, giải phóng temp, di chuyển current.
  3. **Dòng 7**: Đặt head về NULL.
* **Tại sao**:
  1. Giải phóng bộ nhớ của danh sách hợp nhất để tránh rò rỉ.
  2. Dùng temp để tránh truy cập bộ nhớ đã giải phóng.
  3. Độ phức tạp O(n) thời gian, O(1) không gian.
* **Ví dụ**:
  1. Với danh sách [1 -> 2 -> 3 -> 4 -> 5 -> 6]:
     + Lần 1: Giải phóng nút 1, current = 2.
     + Lần 6: Giải phóng nút 6, current = NULL.
     + \*head\_ref = NULL.

**7. Hàm main**

int main() {

Node\* head1 = NULL;

Node\* head2 = NULL;

int n1, n2, value;

*// Nhập danh sách thứ nhất*

printf("Nhap so luong phan tu cua danh sach thu nhat: ");

scanf("%d", &n1);

if (n1 < 0) {

printf("So luong phan tu khong hop le!\n");

return 1;

}

printf("Nhap %d gia tri cho danh sach thu nhat:\n", n1);

for (int i = 0; i < n1; i++) {

scanf("%d", &value);

insert\_at\_tail(&head1, value);

}

*// Nhập danh sách thứ hai*

printf("Nhap so luong phan tu cua danh sach thu hai: ");

scanf("%d", &n2);

if (n2 < 0) {

printf("So luong phan tu khong hop le!\n");

return 1;

}

printf("Nhap %d gia tri cho danh sach thu hai:\n", n2);

for (int i = 0; i < n2; i++) {

scanf("%d", &value);

insert\_at\_tail(&head2, value);

}

*// In hai danh sách*

printf("Danh sach thu nhat: ");

print\_list(head1);

printf("Danh sach thu hai: ");

print\_list(head2);

*// Hợp nhất danh sách*

Node\* merged\_list = merge\_lists(head1, head2);

printf("Danh sach sau khi hop nhat: ");

print\_list(merged\_list);

*// Giải phóng danh sách hợp nhất*

free\_list(&merged\_list);

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

print\_list(merged\_list);

return 0;

}

* **Giải thích từng phần**:
  1. **Dòng 2-4**: Khởi tạo head1, head2, và biến nhập liệu.
     + **Tại sao**: Bắt đầu với hai danh sách rỗng, các biến hỗ trợ nhập liệu.
  2. **Dòng 6-14**: Nhập danh sách thứ nhất:
     + Nhập n1, kiểm tra n1 < 0, nhập n1 giá trị, dùng insert\_at\_tail.
     + **Tại sao**: Giữ thứ tự nhập, xử lý lỗi input.
  3. **Dòng 16-24**: Nhập danh sách thứ hai:
     + Tương tự danh sách thứ nhất, nhập n2 và các giá trị.
     + **Tại sao**: Đảm bảo hai danh sách được nhập độc lập.
  4. **Dòng 26-29**: In hai danh sách:
     + **Tại sao**: Xác nhận hai danh sách được tạo đúng.
  5. **Dòng 31-33**: Hợp nhất và in kết quả:
     + Gọi merge\_lists, in danh sách hợp nhất.
     + **Tại sao**: Đáp ứng yêu cầu hợp nhất, kiểm tra kết quả.
  6. **Dòng 35-37**: Giải phóng danh sách hợp nhất:
     + **Tại sao**: Tránh rò rỉ bộ nhớ, xác nhận giải phóng.
  7. **Dòng 39**: Kết thúc chương trình.
* **Tại sao cấu trúc này**:
  1. Nhập danh sách bất kỳ thay vì cố định [1, 3, 5] và [2, 4, 6].
  2. Giao diện thân thiện, xử lý lỗi input.
  3. In danh sách sau mỗi thao tác để kiểm tra.

**Kết quả chạy**

**Trường hợp 1: Nhập [1, 3, 5] và [2, 4, 6]**

Nhap so luong phan tu cua danh sach thu nhat: 3

Nhap 3 gia tri cho danh sach thu nhat:

1 3 5

Nhap so luong phan tu cua danh sach thu hai: 3

Nhap 3 gia tri cho danh sach thu hai:

2 4 6

Danh sach thu nhat: 1 -> 3 -> 5

Danh sach thu hai: 2 -> 4 -> 6

Danh sach sau khi hop nhat: 1 -> 2 -> 3 -> 4 -> 5 -> 6

Danh sach sau khi giai phong: Danh sach rong

**Trường hợp 2: Một danh sách rỗng**

text

CollapseWrap

Copy

Nhap so luong phan tu cua danh sach thu nhat: 0

Nhap 0 gia tri cho danh sach thu nhat:

Nhap so luong phan tu cua danh sach thu hai: 3

Nhap 3 gia tri cho danh sach thu hai:

2 4 6

Danh sach thu nhat: Danh sach rong

Danh sach thu hai: 2 -> 4 -> 6

Danh sach sau khi hop nhat: 2 -> 4 -> 6

Danh sach sau khi giai phong: Danh sach rong

**Trường hợp 3: Danh sách có độ dài khác nhau**

text

CollapseWrap

Copy

Nhap so luong phan tu cua danh sach thu nhat: 2

Nhap 2 gia tri cho danh sach thu nhat:

1 3

Nhap so luong phan tu cua danh sach thu hai: 4

Nhap 4 gia tri cho danh sach thu hai:

2 4 6 8

Danh sach thu nhat: 1 -> 3

Danh sach thu hai: 2 -> 4 -> 6 -> 8

Danh sach sau khi hop nhat: 1 -> 2 -> 3 -> 4 -> 6 -> 8

Danh sach sau khi giai phong: Danh sach rong

**Hình ảnh minh họa**

**Danh sách ban đầu:**

* Danh sách 1: head1 -> [1] -> [3] -> [5] -> NULL
* Danh sách 2: head2 -> [2] -> [4] -> [6] -> NULL

**Quá trình hợp nhất:**

1. Lần 1: 1 <= 2, lấy nút 1, merged\_head -> [1], head1 -> [3 -> 5].
2. Lần 2: 3 > 2, lấy nút 2, [1 -> 2], head2 -> [4 -> 6].
3. Lần 3: 3 <= 4, lấy nút 3, [1 -> 2 -> 3], head1 -> [5].
4. Lần 4: 5 > 4, lấy nút 4, [1 -> 2 -> 3 -> 4], head2 -> [6].
5. Lần 5: 5 <= 6, lấy nút 5, [1 -> 2 -> 3 -> 4 -> 5], head1 = NULL.
6. Lần 6: Gắn head2 -> [6], [1 -> 2 -> 3 -> 4 -> 5 -> 6].

**Danh sách sau hợp nhất:**

text

CollapseWrap

Copy

merged\_head -> [1] -> [2] -> [3] -> [4] -> [5] -> [6] -> NULL

**Sau khi giải phóng:**

merged\_head -> NULL

**Notes**

**Hiểu con trỏ**:

* + Node\*: Con trỏ tới một nút.
  + Node\*\*: Con trỏ tới con trỏ Node\*, dùng trong tail\_ref để cập nhật liên kết động.
  + **Ví dụ**: tail\_ref trỏ tới next của nút cuối, \*tail\_ref là nút được thêm.

1. **Hiểu hợp nhất**:
   * Vẽ hai danh sách trên giấy, theo dõi head1, head2, và tail\_ref qua mỗi bước.
   * tail\_ref giúp nối nút mà không cần duyệt lại danh sách hợp nhất.
2. **Xử lý trường hợp đặc biệt**:
   * Kiểm tra danh sách rỗng trong merge\_lists để xử lý input rỗng.
   * Kiểm tra n1 < 0, n2 < 0 trong main để tránh lỗi input.
3. **Debug**:
   * Thêm printf trong merge\_lists để in head1->data, head2->data, và (\*tail\_ref)->data.
   * In danh sách sau mỗi bước hợp nhất để kiểm tra liên kết.
4. **Tại sao không tạo nút mới?**:
   * Chỉ điều chỉnh con trỏ để tái sử dụng các nút hiện có, tiết kiệm bộ nhớ.
   * Yêu cầu đề bài đảm bảo không cần cấp phát bộ nhớ mới.
5. **Tại sao dùng insert\_at\_tail?**:
   * Giữ thứ tự nhập, phù hợp với input đã sắp xếp tăng dần.
   * Dùng insert\_at\_head sẽ đảo ngược thứ tự, không phù hợp.