Lập Trình C (cơ bản)

Chương 04. Danh sách liên kết (2 chiều)

Soạn bởi: TS. Nguyễn Bá Ngọc

2021

Nội dung

- Quản lý bộ nhớ động: Một số hàm cơ bản
- Danh sách liên kết 2 chiều: Giao diện khái quát

Nội dung

- Quản lý bộ nhớ động: Một số hàm cơ bản
- Danh sách liên kết 2 chiều: Giao diện khái quát

Một số hàm quản lý bộ nhớ động

stdlib.h

`void *malloc(size_t size);` - Cấp phát khối nhớ với kích thước = size.

`void *calloc(size_t nmemb, size_t size);` - Cấp phát khối nhớ cho mảng nmemb phần tử, kích thước mỗi phần tử = size, khởi tạo tất cả các bits = 0.

`void *realloc(void *ptr, size_t size);` - Giải phóng bộ nhớ ở địa chỉ ptr và cấp phát khối nhớ có kích thước size. Dữ liệu trong phạm vi cũ được bảo toàn. Nếu ptr == NULL thì realloc hoạt động giống như malloc.

Giá trị trả về: Các hàm malloc, calloc, realloc trả về con trỏ tới khối nhớ mới được cấp phát hoặc trả về NULL trong trường hợp không thể cấp phát bộ nhớ.

`void free(void *ptr);` - Giải phóng bộ nhớ đã được cấp phát ở địa chỉ ptr.

Ví dụ quản lý bộ nhớ động

- Cấp phát bộ nhớ cho 1 mảng 3 phần tử kiểu int:
  - int *a = malloc(3 * sizeof(int)); // Không khởi tạo
  - int *a = calloc(3, sizeof(int)); // Tất cả phần tử = 0
  - int *a = realloc(NULL, 3 * sizeof(int)); // Tương tự malloc
- Giả sử a[0] == 1, a[1] == 2, a[2] == 3. Cấp phát lại bộ nhớ cho mảng a để lưu 10 phần tử:
  - a = realloc(a, 10 * sizeof(int));
    - Giá trị a[0], a[1] và a[2] vẫn được giữ nguyên. Các giá trị còn lại chưa được khởi tạo (không xác định).
    - Bộ nhớ cho a có thể được cấp phát ở địa chỉ mới.
- Giải phóng bộ nhớ động đã được cấp phát:
  - free(a); // Có thể thực hiện ở bất kỳ thời điểm nào
  - Bộ nhớ động được cấp phát trong Heap, người lập trình có thể giải phóng bộ nhớ động ở bất kỳ thời điểm nào.

Nội dung

- Quản lý bộ nhớ động: Một số hàm cơ bản
- Danh sách liên kết 2 chiều: Giao diện khái quát

Bài tập. Triển khai danh sách liên kết 2 chiều với giao diện khái quát

Yêu cầu:

- Triển khai theo đặc tả được cung cấp trong các slides tiếp theo.
- Sử dụng tệp tiêu đề cùng với các kiểm thử đơn vị ở địa chỉ: https://github.com/bangoc/c-basic/tree/master/bkc

Cấu trúc khái quát

- Cấu trúc nút khái quát:

```
typedef struct dll_node_s {
  struct dll_node_s *next;
  struct dll_node_s *prev;
} *dll_node_t;
```

  - Con trỏ next trỏ tới phần tử đứng sau hoặc NULL;
  - Con trỏ prev trỏ tới phần tử đứng trước hoặc NULL.

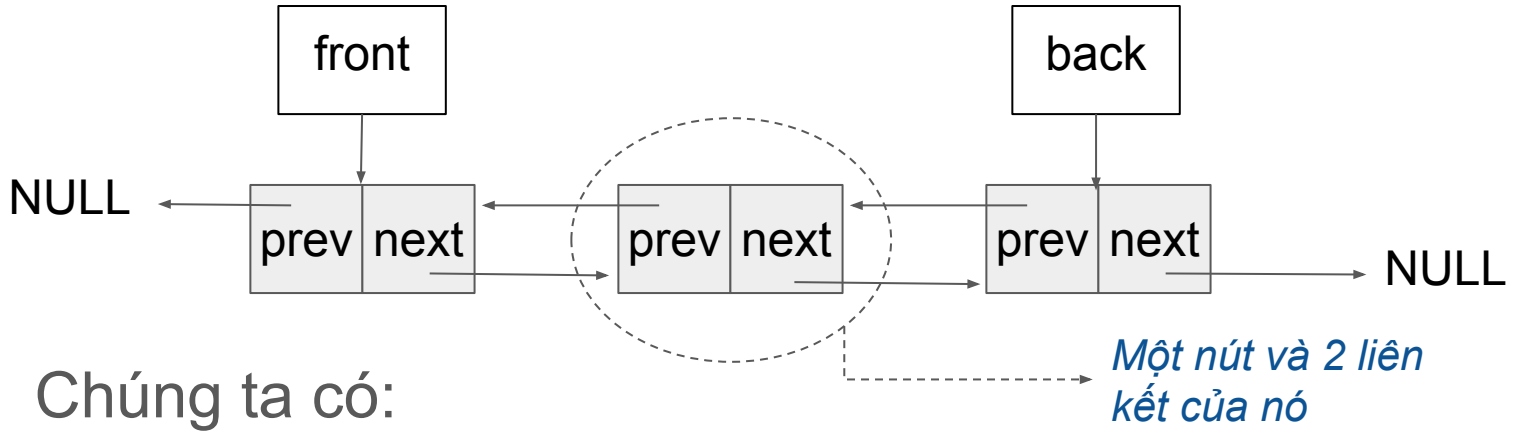
- Cấu trúc danh sách khái quát

```
typedef struct dll_s {
  dll_node_t front;
  dll_node_t back;
} *dll_t;
```

  - Con trỏ front trỏ tới phần tử đầu tiên hoặc NULL;
  - Con trỏ back trỏ tới phần tử cuối cùng hoặc NULL.

Biểu diễn dạng sơ đồ

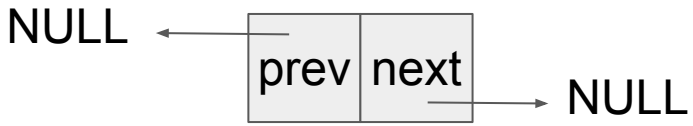
- Ở trạng thái bình thường danh sách liên kết 2 chiều có cấu trúc tương tự như trong sơ đồ:



- Chúng ta có:
  - Với một nút node trong danh sách:
    - node->next == NULL <=> node == back
    - node->prev == NULL <=> node = front
  - Trong trường hợp danh sách rỗng (ví dụ mới được tạo):
    - front == back == NULL
  - Trong trường hợp danh sách chỉ có 1 nút (node):
    - front == back == node

Giao diện khái quát

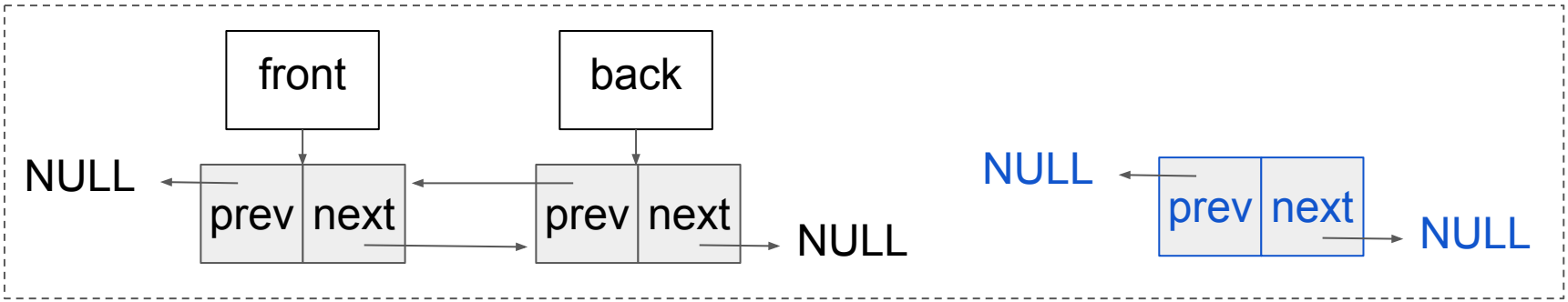
- `dll_node_t dll_create_node()`
  - Cấp phát bộ nhớ động cho một nút và khởi tạo các liên kết bằng NULL.

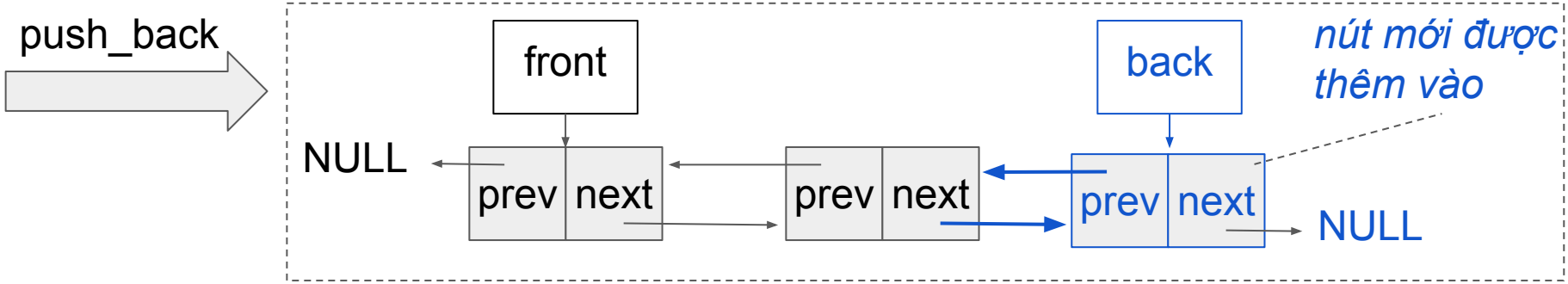


- `void dll_free_node(dll_node_t node);`
  - Giải phóng bộ nhớ được cấp phát cho node.
- `dll_t dll_create_list();`
  - Cấp phát bộ nhớ động cho một danh sách rỗng.
- `void dll_free_list(dll_t list);`
  - Giải phóng bộ nhớ cho danh sách list (bao gồm cả các nút của nó).

Giao diện khái quát(2)

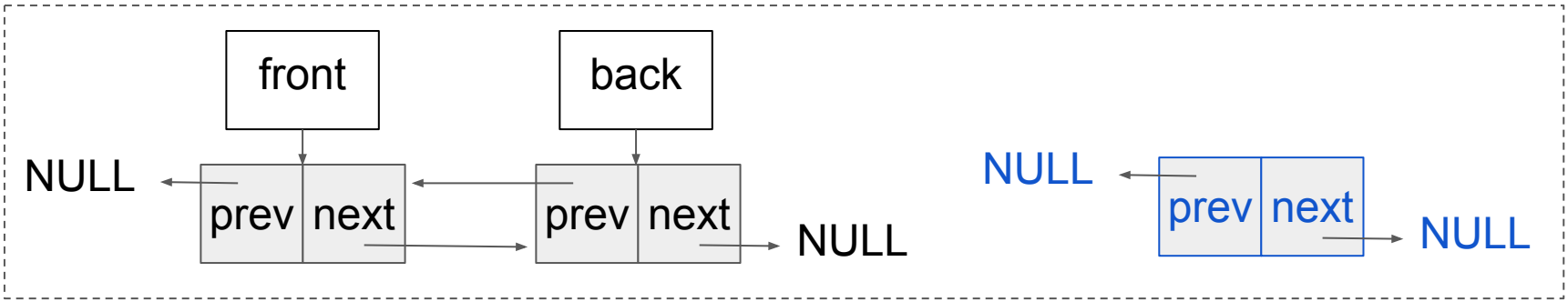
- `void dll_push_back(dll_t list, dll_node_t nn);`
  - Trường hợp list là danh sách rỗng: sau khi thêm nn vào danh sách chúng ta có front == back == nn
  - Trong các trường hợp còn lại: nn được thêm vào sau nút back (cũ) và nút back được cập nhật thành nn.
  - => nn là nút cuối cùng sau khi được thêm vào (back == nn).

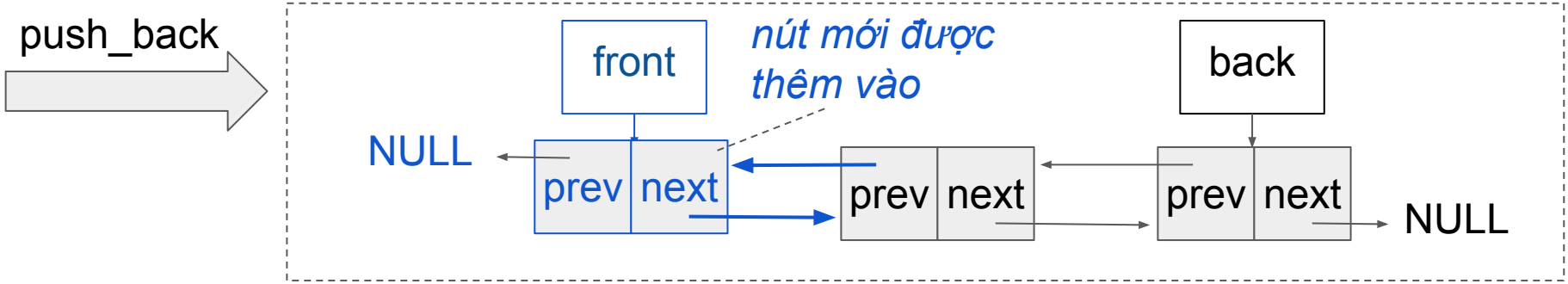




Giao diện khái quát(3)

- `void dll_push_front(dll_t list, dll_node_t nn);`
  - Trường hợp list là danh sách rỗng: sau khi thêm nn vào danh sách chúng ta có front == back == nn
  - Trong các trường hợp còn lại: nn được thêm vào trước nút front (cũ) và nút front được cập nhật thành nn.
  - => nn là nút đầu tiên sau khi được thêm vào (front == nn).



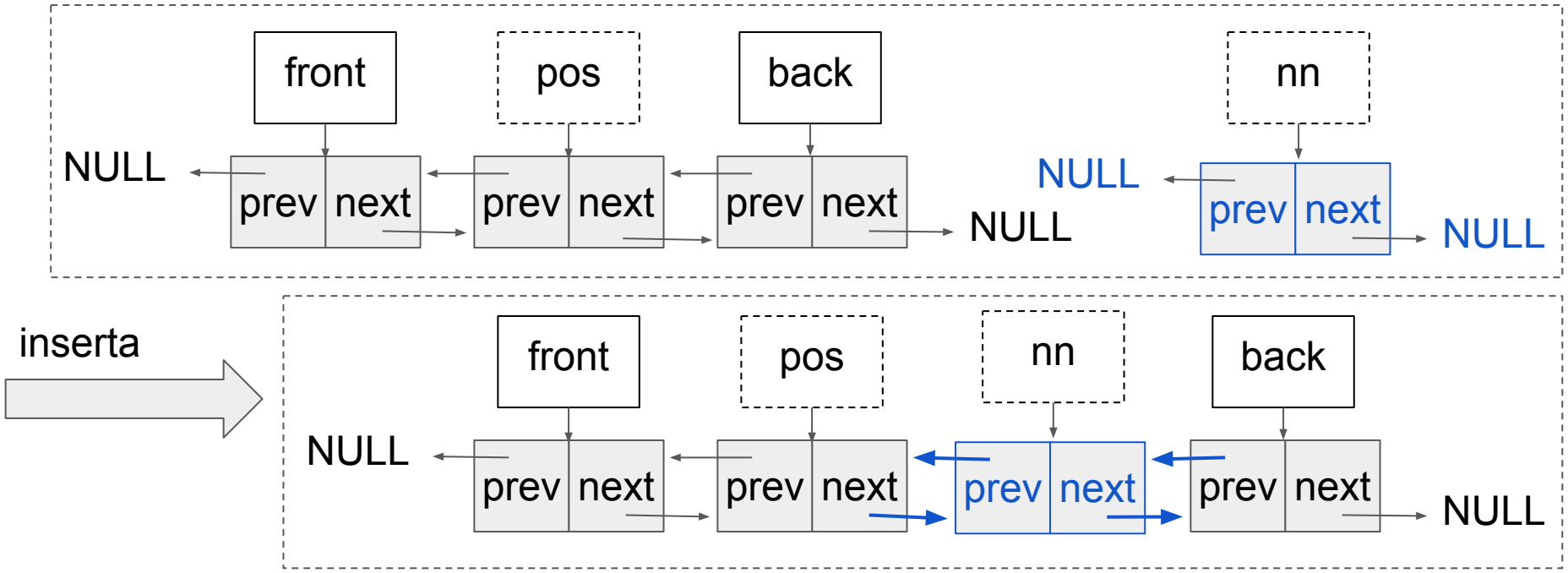


Giao diện khái quát(3)

- `void dll_pop_back(dll_t list)`
  - Xóa phần tử cuối cùng trong danh sách nếu có, nếu ngược lại (danh sách rỗng) thì không làm gì.
- `void dll_pop_front(dll_t list)`
  - Xóa phần tử đầu tiên trong danh sách nếu có, nếu ngược lại (danh sách rỗng) thì không làm gì.
- `dll_node_t dll_front(dll_t list)`
  - Trả về con trỏ tới phần tử đầu tiên trong danh sách, hoặc NULL nếu list là danh sách rỗng.
- `dll_node_t dll_back(dll_t list)`
  - Trả về con trỏ tới phần tử cuối cùng trong danh sách, hoặc NULL nếu list là danh sách rỗng.

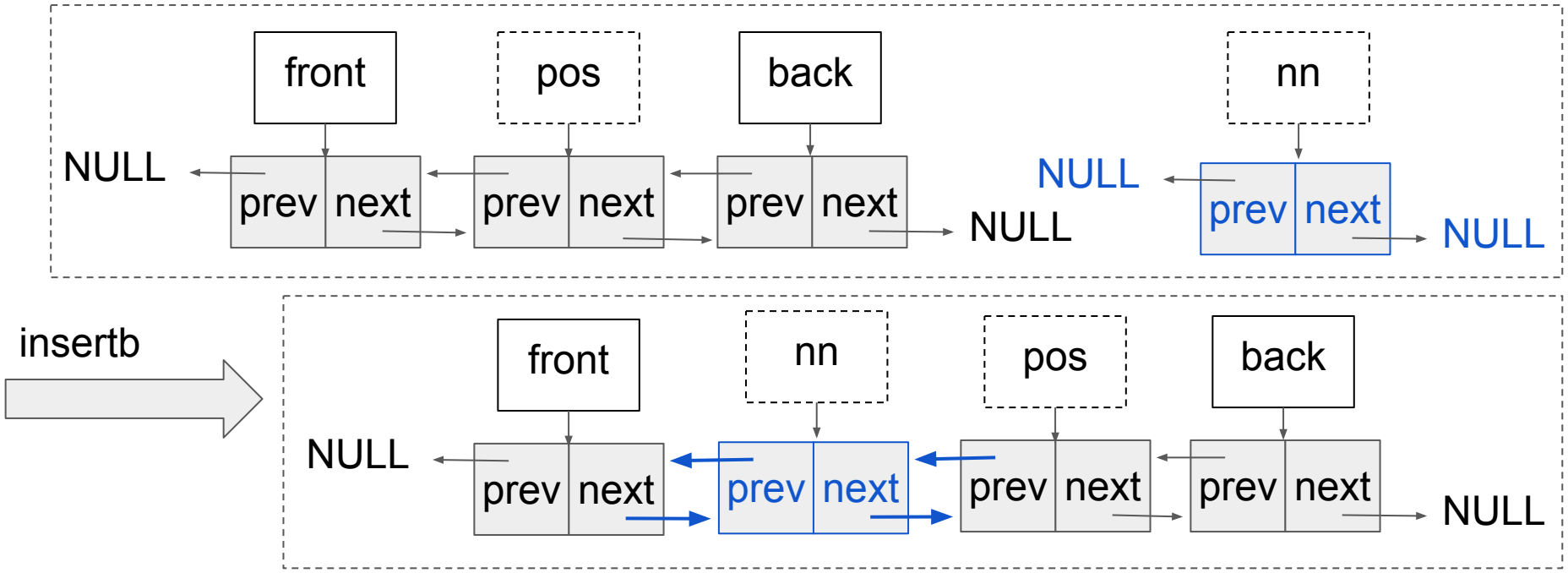
Giao diện khái quát(4)

- `dll_node_t dll_inserta(dll_t list,`
  `dll_node_t pos, dll_node_t nn);`
  - Nếu pos != NULL thì thêm nn vào sau pos, nếu ngược lại (pos == NULL) thì giống như dll_push_back.
  - Hàm trả về con trỏ nn.



Giao diện khái quát(5)

- `dll_node_t dll_insertb(dll_t list,`
  `dll_node_t pos, dll_node_t nn);`
  - Nếu pos != NULL thì thêm nn vào trước pos, nếu ngược lại (pos == NULL) thì giống như dll_push_front.
  - Hàm trả về con trỏ nn.



Giao diện khái quát(6)

- `int dll_is_empty(dll_t list)`
  - Trả về 1 nếu list là danh sách rỗng, trả về 0 nếu ngược lại.
- `long dll_length(dll_t list)`
  - Trả về số lượng phần tử trong list (trả về 0 nếu list là danh sách rỗng).
- `void dll_clear(dll_t list)`
  - Làm rỗng list.

Giao diện khái quát(7)

- `void dll_erase(dll_t list, dll_node_t pos);`
  - Xóa một phần tử bất kỳ trong list được trỏ tới bởi pos.
  - Nếu pos == dll->front thì tương tự dll_pop_front,
  - Nếu ngược lại, nếu pos == dll->back thì tương tự dll_pop_back
  - Nếu ngược lại thì kết nối pos->prev và pos->next bỏ qua pos