lab9 1. `struct Page* page = le2page(le, page_link);` `le2page(le, page_link)` -> `to_struct((le), struct Page, page_link)` -> `((struct Page *)((char *)(le) - offsetof(struct Page, page_link)))` -> `((struct Page *)((char *)(le) - ((size_t)(&((struct Page *)0)->page_link))))`

First use the address of le, minus the offset of page_link in struct Page. Then we find the base address of the struct Page that contains le. Finally, we cast the base address to a struct Page pointer.

2.

请详细描述default_pmm.c中的default_alloc_pages和default_free_pages的功能与实现方式。

default_alloc_pages： (1). check total free lists

```
    assert(n > 0);
    if (n > nr_free) {
        return NULL;
    }
```

(2). find the first n blocks contains n pages through the linked list:

```
        struct Page *page = NULL;
        list_entry_t *le = &free_list;
        while ((le = list_next(le)) != &free_list) {
            struct Page *p = le2page(le, page_link);
            if (p->property >= n) {
                page = p;
                break;
            }
        }
```

(3). remove the found page from the list. if the remaining property of the page is larger than needed, tranfer it to next page

```
    if (page != NULL) {
    list_entry_t* prev = list_prev(&(page->page_link));
    list_del(&(page->page_link));
    if (page->property > n) {
        struct Page *p = page + n;
        p->property = page->property - n;
        SetPageProperty(p);
        list_add(prev, &(p->page_link));
    }
    nr_free -= n;
    ClearPageProperty(page);
    }
```

default_free_pages (1). assert page frame status, and clear flags

```
    assert(n > 0);
    struct Page *p = base;
    for (; p != base + n; p ++) {
        assert(!PageReserved(p) && !PageProperty(p));
        p->flags = 0;
        set_page_ref(p, 0);
    }
```

(2). restore property to base Page

```
    base->property = n;
    SetPageProperty(base);
    nr_free += n;
```

(3). append the base page to linked list

```
    if (list_empty(&free_list)) {
        list_add(&free_list, &(base->page_link));
    } else {
        list_entry_t* le = &free_list;
        while ((le = list_next(le)) != &free_list) {
            struct Page* page = le2page(le, page_link);
            if (base < page) {
                list_add_before(le, &(base->page_link));
                break;
            } else if (list_next(le) == &free_list) {
                list_add(le, &(base->page_link));
            }
        }
    }
```