Lab2-queue.h 补充讲解

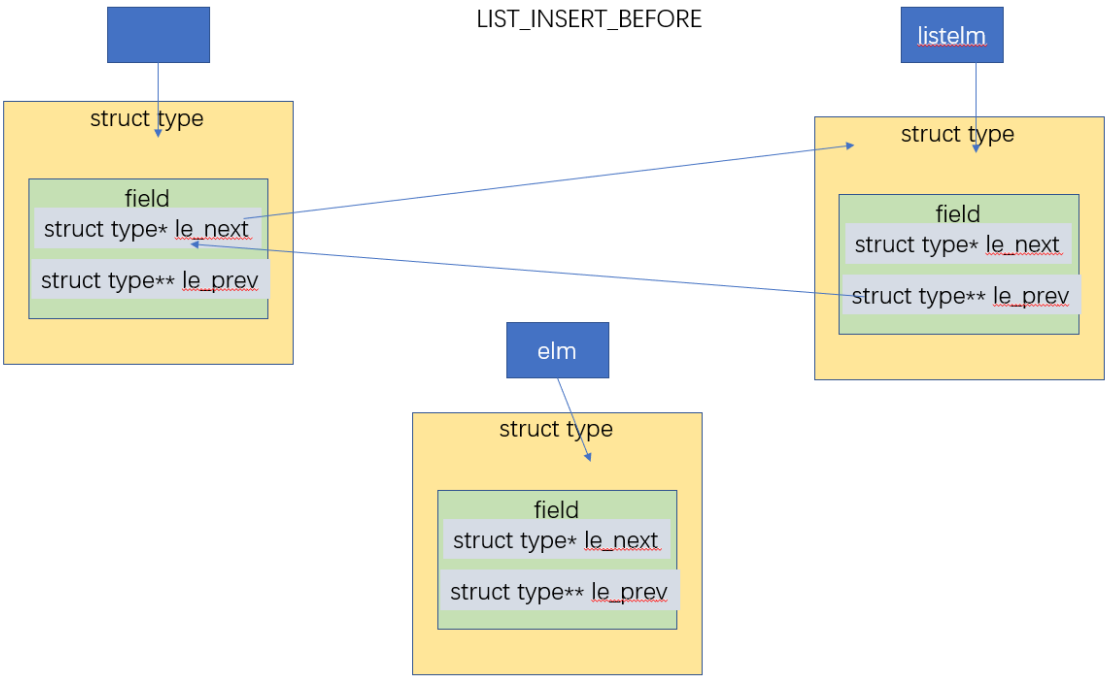
MOS操作系统在queue.h中定义了一系列宏函数来实现链表操作。在Lab2的exercise3.1中，各位同学需要补充实现`LIST_INSERT_AFTER`与`LIST_INSERT_QUEUE`。根据我们的经验，这一部分内容确实不太容易理解，因此我们给出了一些适当的提示，帮助大家更好地理解这种链表的组织形式。

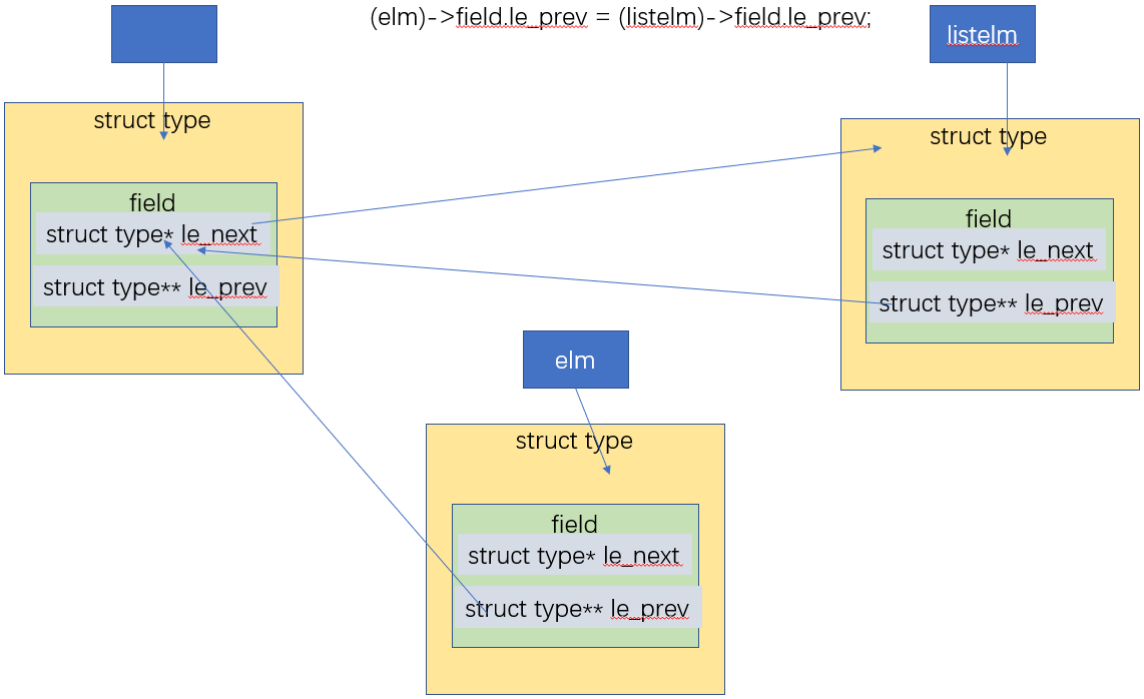
MOS中实现的链表，核心在于`LIST_ENTRY`所定义的链表项结构体，这一结构体主要是两个成员：`struct type *le_next`与`struct type **le_prev`，一般的双向链表中，两个指针将分别指向前一项和后一项结构体，但是MOS实现的链表在此处有一些差异，代码的注释给出了一些提示，你也可以通过观察它们的类型来推测它们大概的用途。

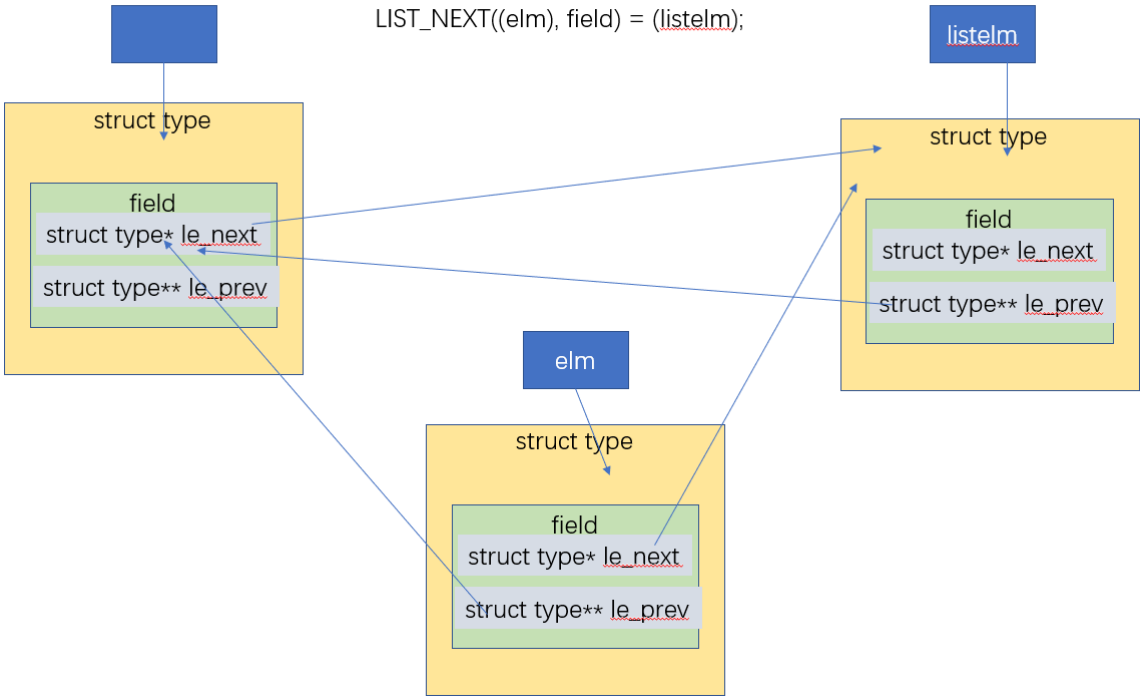
```
#define LIST_ENTRY(type)                                                \
        struct {                                                        \
                struct type *le_next;   /* next element */              \
                struct type **le_prev;  /* address of previous next element */ \
        }
```

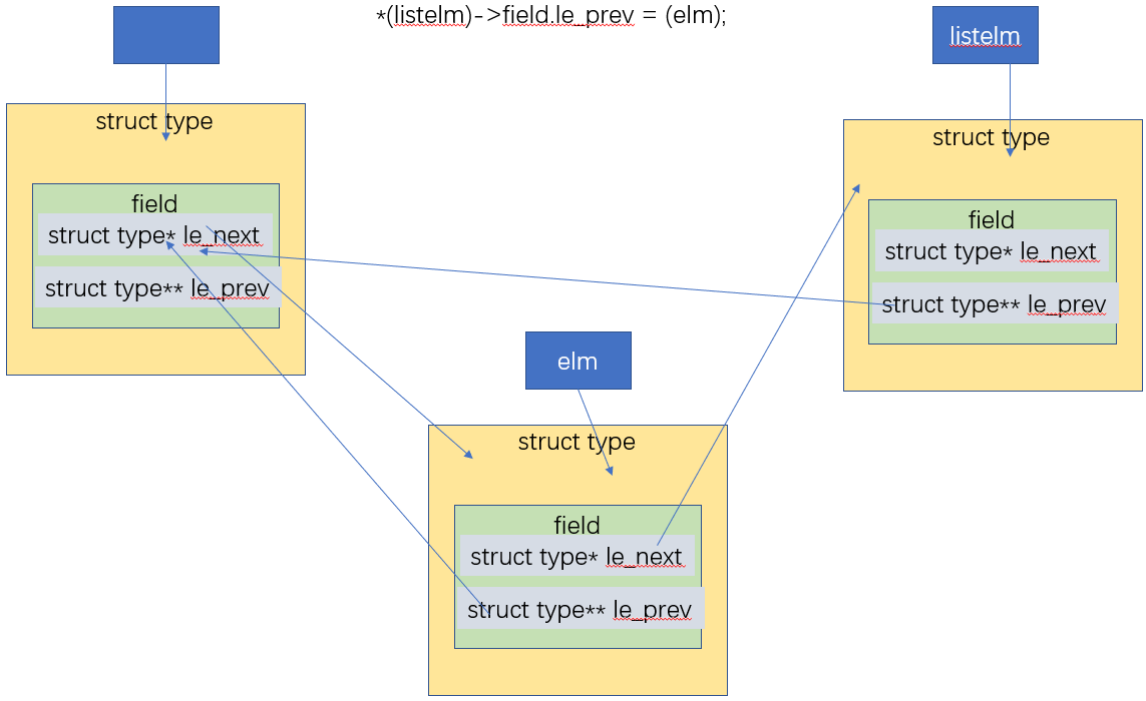
对于比较复杂的宏，可以尝试将其在原来的代码中展开，也可以尝试绘制链表的组织形式，用图形帮助理解。这里我们给出了`LIST_INSERT_BEFORE`的实现过程，你可以在实验报告中给出这一练习所要求的`LIST_INSERT_AFTER`与`LIST_INSERT_QUEUE`的图示。

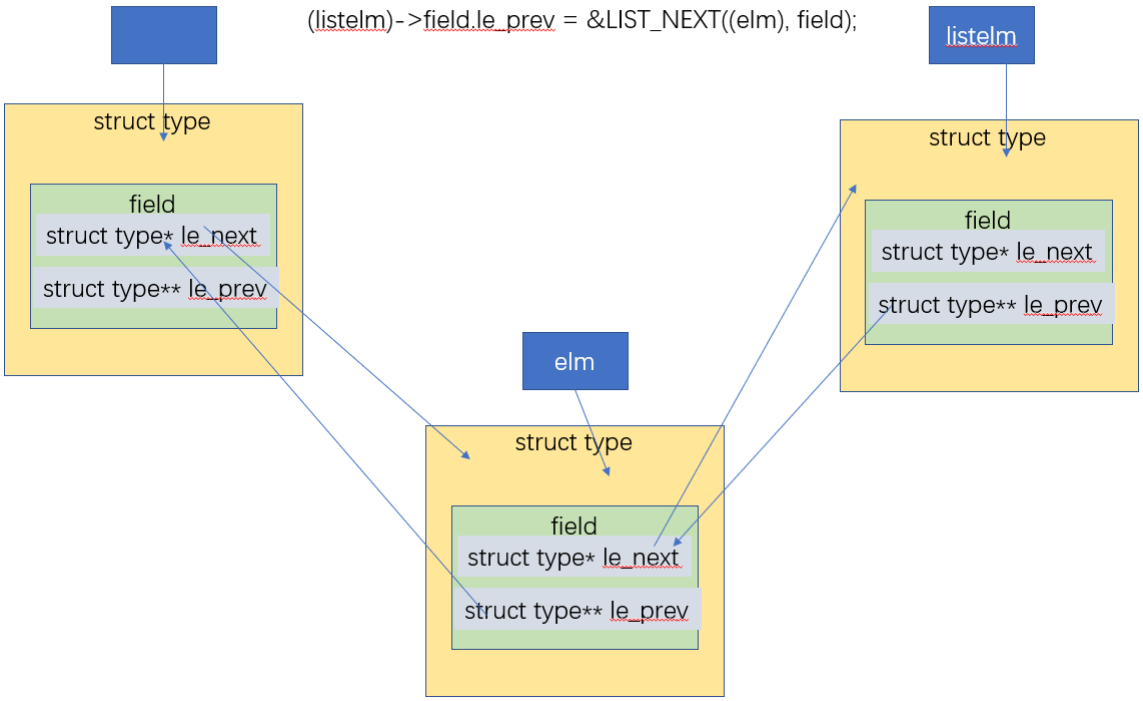
HINT：注意观察已经实现的宏能否用在这两个宏的实现中。




(elm)->field.le_prev = (listelm)->field.le_prev;
listelm
struct type
field
struct type* le_next
struct type** le_prev
struct type
field
struct type* le_next
struct type** le_prev
elm
struct type
field
struct type* le_next
struct type** le_prev


LIST_NEXT((elm), field) = (listelm);
listelm
struct type
field
struct type* le_next
struct type** le_prev
struct type
field
struct type* le_next
struct type** le_prev
elm
struct type
field
struct type* le_next
struct type** le_prev





不同于C++， Java， C语言没有提供泛型的机制，但我们实现的链表提供了一个可以供不同类型数据使用的数据结构。

PS.

对于lab2的其他练习，指导书已经给出了非常详尽的提示，happy coding!