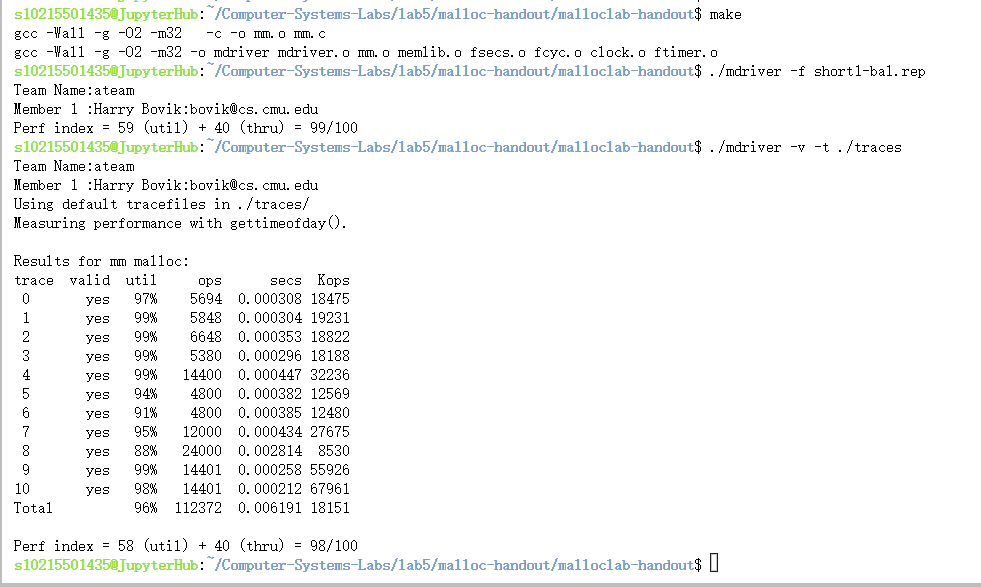
**Malloclab实验报告**

10215501435 杨茜雅



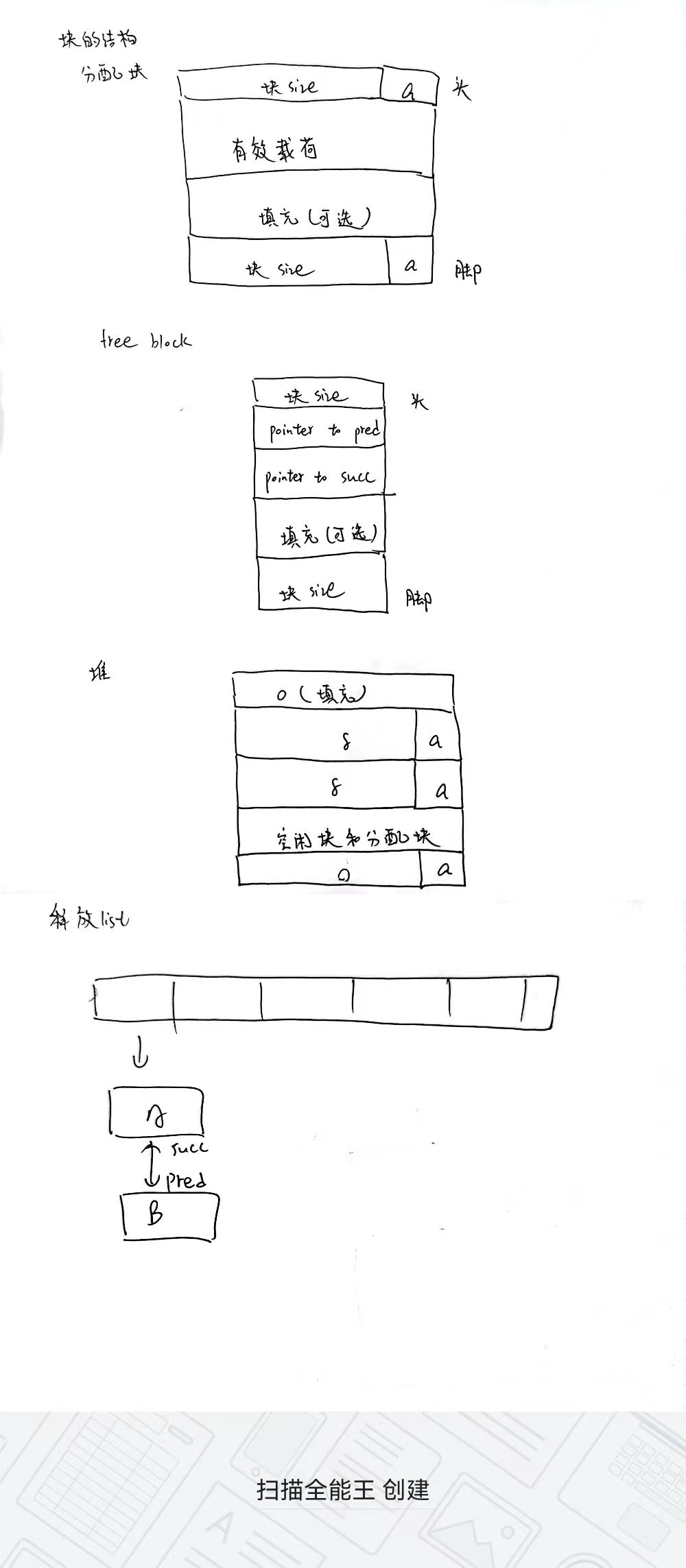
需要实现init、malloc、free、realloc

对于吞吐量thru而言，需要关注malloc、free和realloc每次操作的复杂度。

对于空间利用率util而言需要关注块内损失和由于块是连续不分散的，无法整体利用而产生的开销。所以在free和malloc的时候要尽量大块整体利用。在free块合并和realloc的时候需要判断一下接下来的块是否是空闲的，如果是空闲的就可以整体利用。

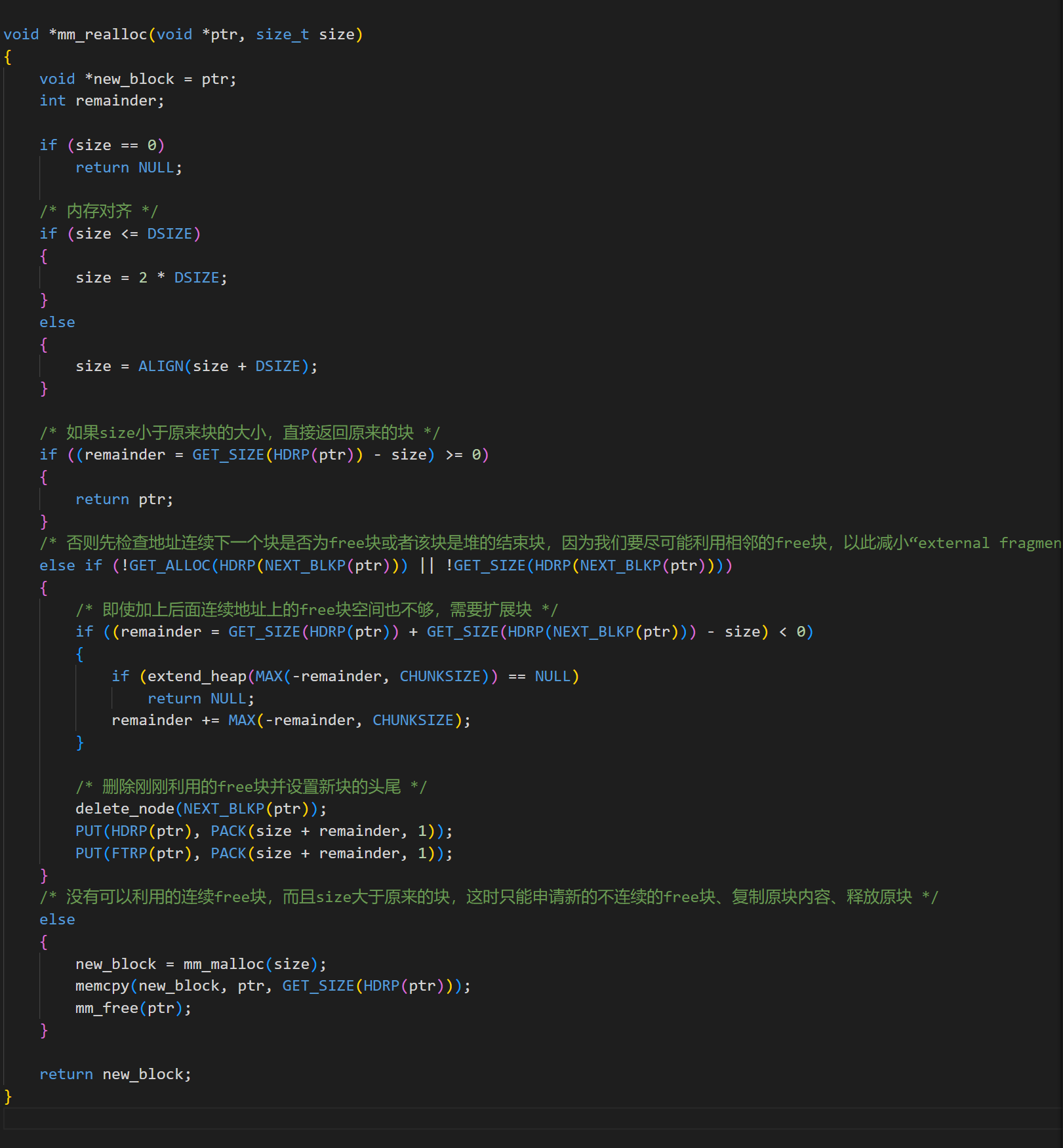
代码中有用到书上p603-p605页之间显示空闲链表、分离的空闲链表和分离适配的内容。

显示空闲链表：将空闲块组织成显示的数据结构（堆->双向空闲链表），每个空闲块都有一个前驱和后继指针。但是由于它需要包含所有的指针和头部脚部，所以需要足够大，内部碎片程度高。

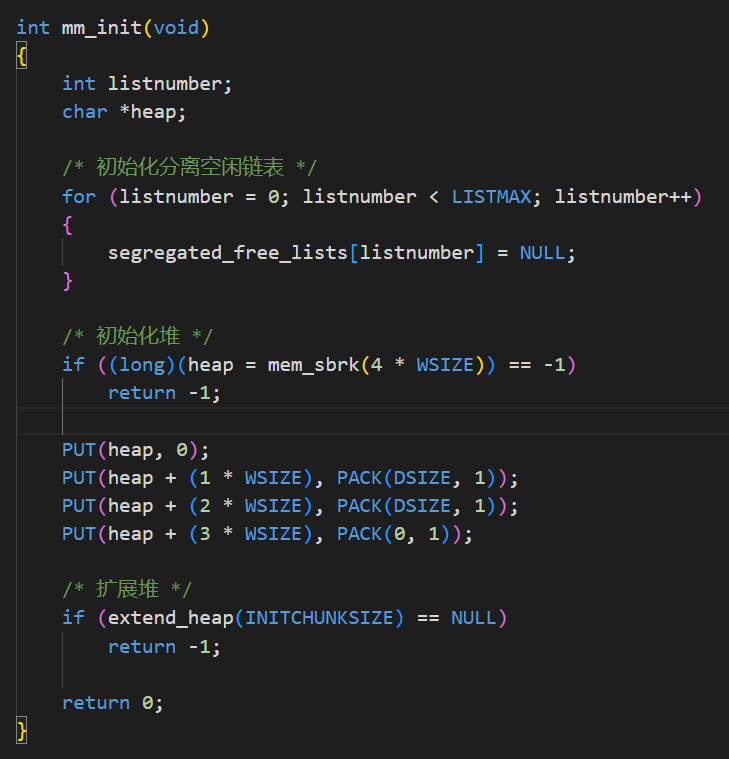


分离适配：为了malloc一个块，确定大小类之后对空闲链表做首次适配，如果找到了一个合适的块就分割，剩下的部分插入到空闲链表中，找不到合适的块就搜索下一个更大的空闲链表。要是一直没有找到就申请从新的堆内存里分配一个块。

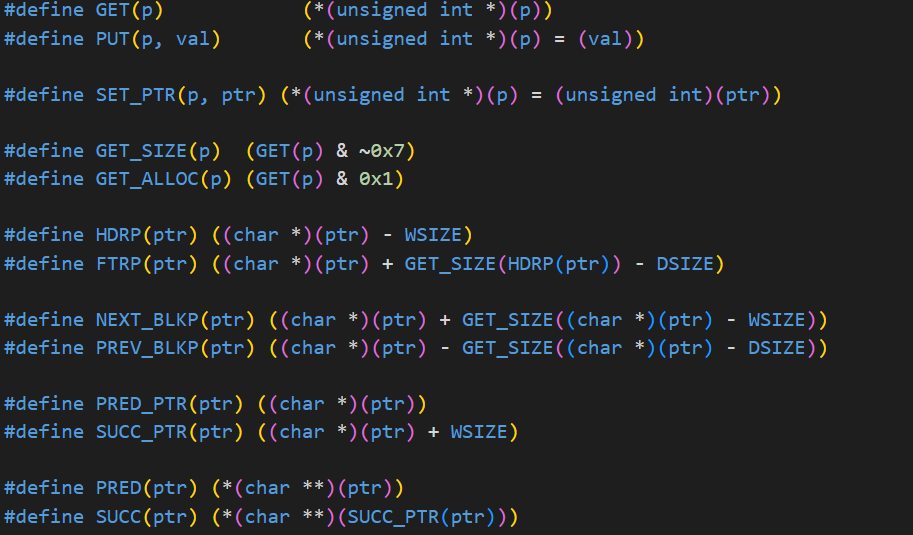
对应代码：



分离链表与堆的结构（起始和结束）



对指针所在的内存赋值时进行类型转换，避免warning



一些辅助函数 ：

static void \*extend\_heap(size\_t size)扩展堆

static void \*coalesce(void \*ptr)合并相邻空闲块

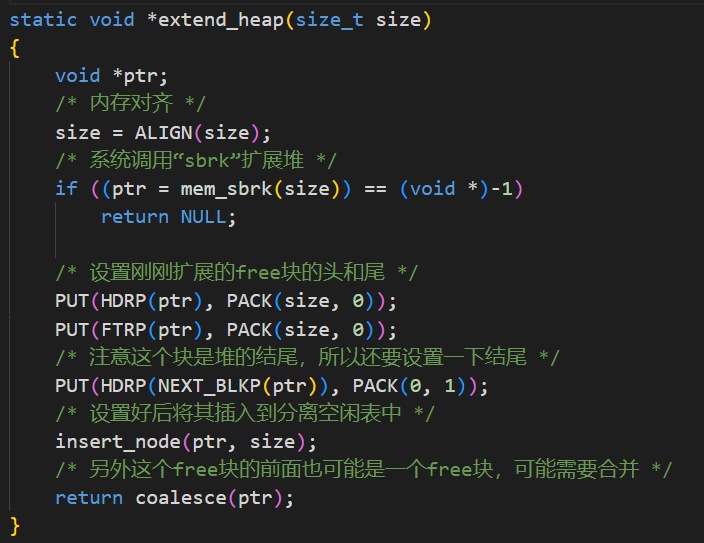
static void \*place(void \*ptr, size\_t size)在prt所指向的空闲块中分配size大小的块，如果剩下的空间大于2\*DWSIZE，则分离放入释放链表

static void insert\_node(void \*ptr, size\_t size)

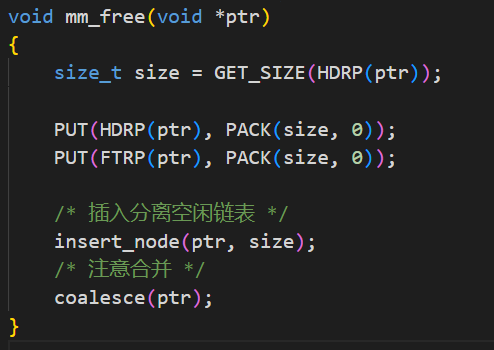
将ptr指向的空闲块插入到分离链表中

static void delete\_node(void \*ptr)将ptr指向的块从分离链表中删除

extend\_heap：



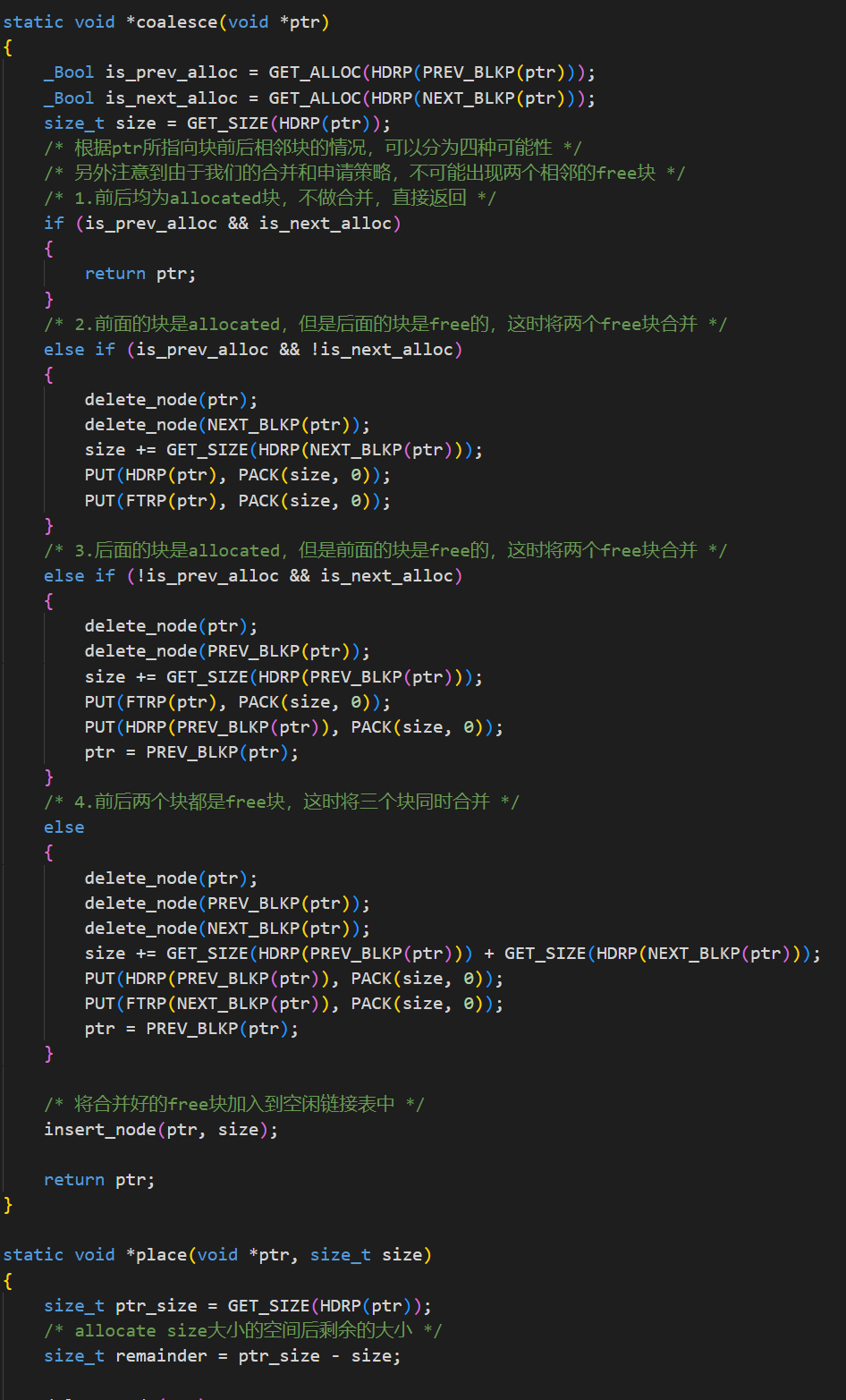
Free:



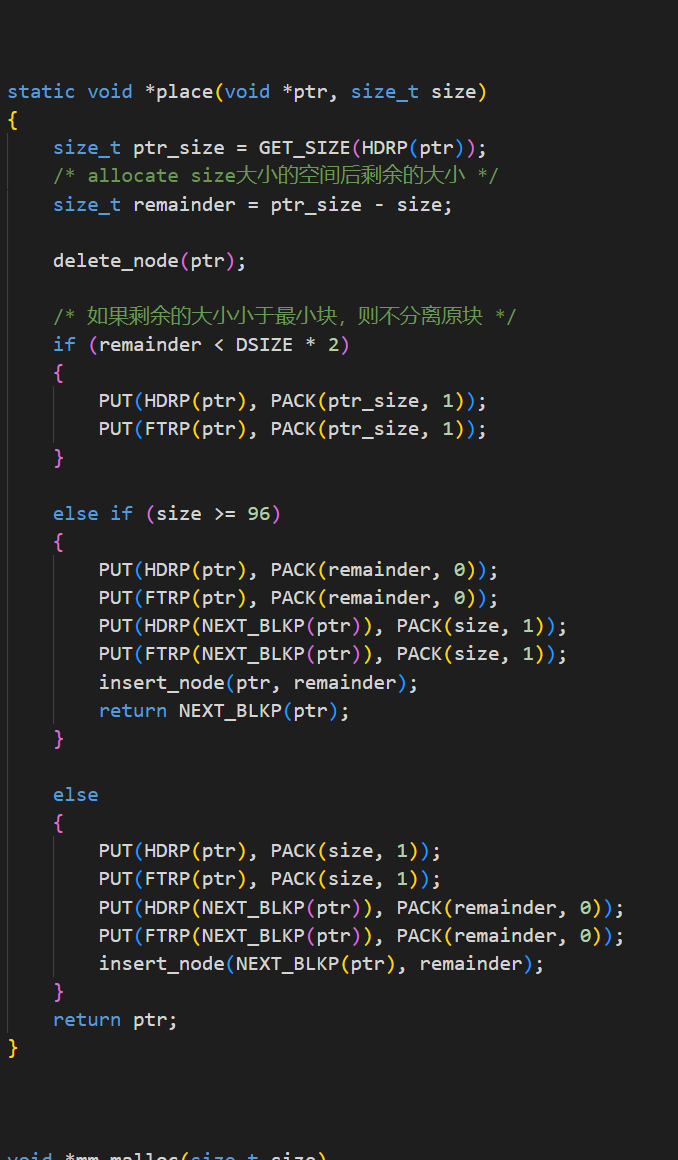
Coalesce:

注意四种情况：

1. 前后都是分配好的，不合并
2. 前面分配好了后面是空闲的，两个空闲块合并
3. 后面是分配好的前面是空闲的，两个空闲块合并
4. 前后都是空闲的，三个块合并



Place:



如果剩余的大小小于最小块则不分离原块，否则就要分离。

但在分离原块的时候有一种特殊情况：

如果每次分配的块都是small、big、small、big的顺序，如果在释放的时候不是按照同样的顺序来的话就会出现外部碎片（比如所有大的块都释放了，小的没有被释放）。这样即使有很多空闲的块但都不是连续的所以也不能合并，如果下次来了一个比较大的分配请求，还需要去重新找一个空闲块。所以应该小的放在一起，大的放在一起，以“small、small、small、……、big、big、big……”的顺序，这样即使没有以small、big、small、big的顺序释放，也可以合并空闲块，减少重新找空闲块的次数。

Insert\_node:

需要保持链表中“small、small、small、……、big、big、big……”的顺序。

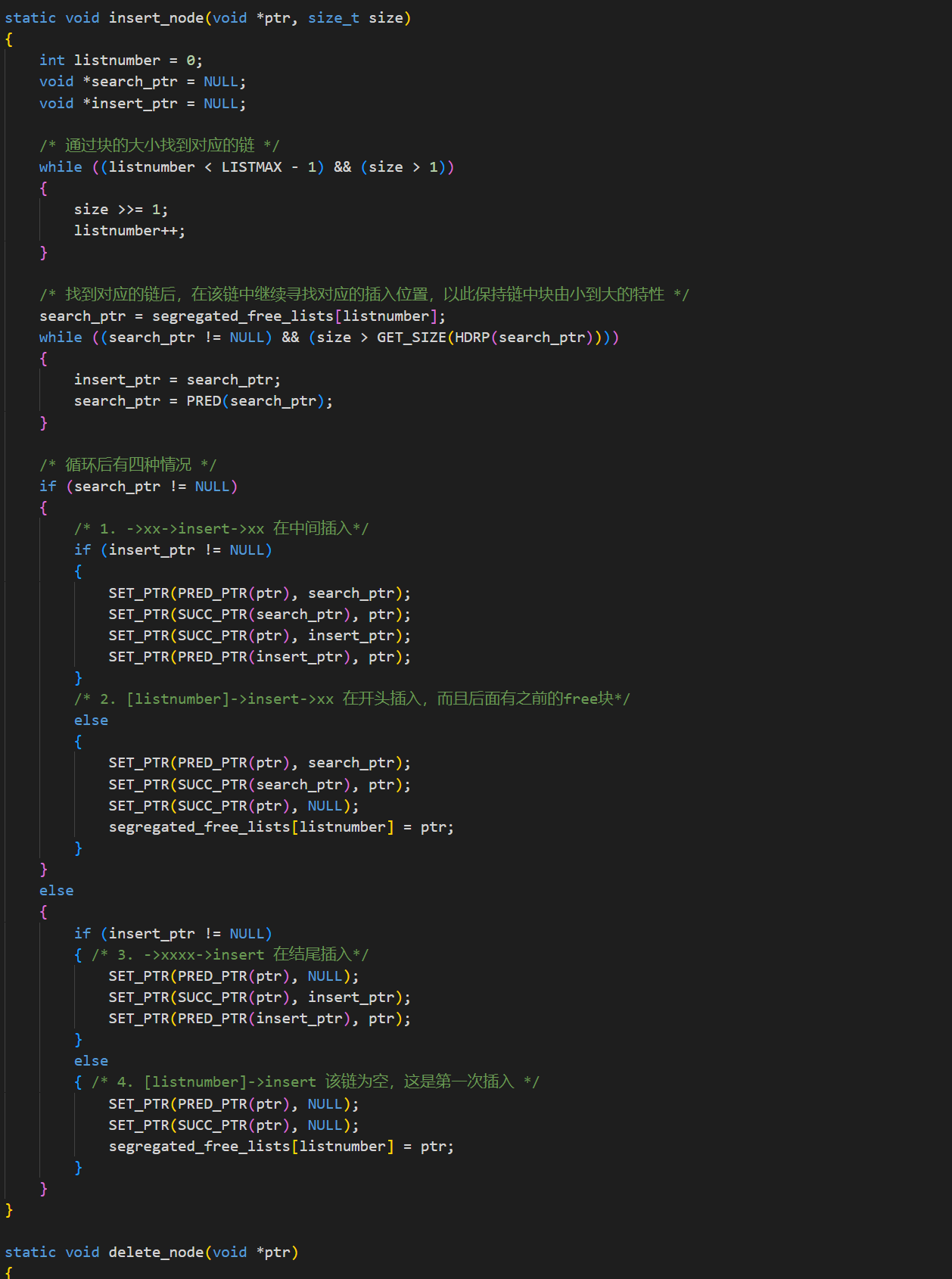
有四种插入情况：

1、 /\* 1. ->xx->insert->xx 在中间插入\*/

2、/\* 2. [listnumber]->insert->xx 在开头插入，而且后面有之前的free块\*/

3、/\* 3. ->xxxx->insert 在结尾插入\*/

4、/\* 4. [listnumber]->insert 该链为空，这是第一次插入 \*/



delete\_node:

通过块的大小找到对应的链表

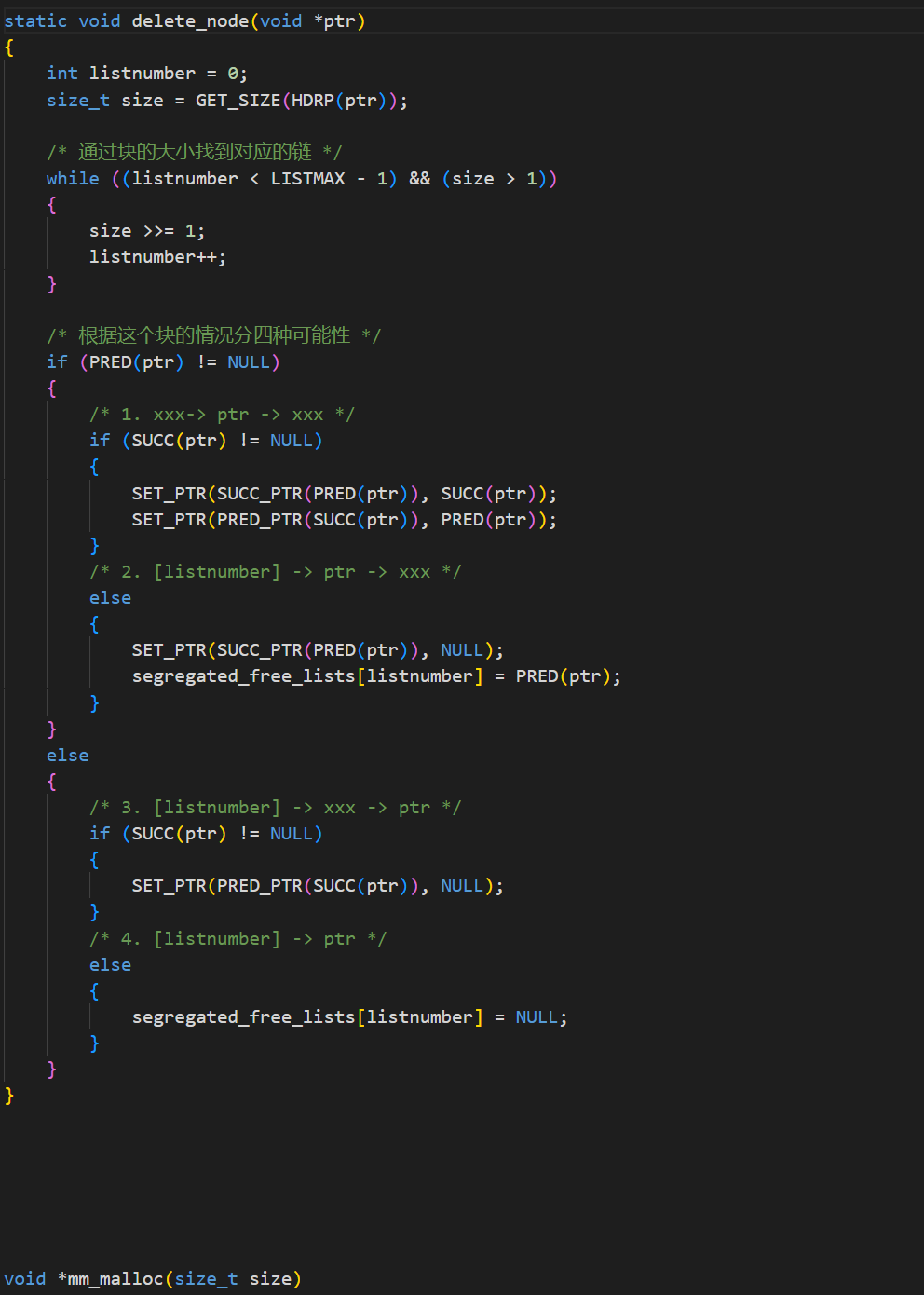
有四种情况：

1、/\* 1. xxx-> ptr -> xxx \*/

2、/\* 2. [listnumber] -> ptr -> xxx \*/

3、/\* 3. [listnumber] -> xxx -> ptr \*/

4、/\* 4. [listnumber] -> ptr \*/



mm\_malloc:

