Homework05　　malloclab　　李芳 10214602404

总结：

使用显式空闲链表+首次适配+按地址顺序维护完成本次实验，实现 init、malloc、free、realloc、coalesce，同时利用函数 extend_heap、find_fit、place 辅助实现，为了检查堆的连续性还有出错问题，设置 mm_check 函数。

mdriver 结果截图：

```
root@LAPTOP-L9FRK1MI:~/10214602404/malloclab-handout# ./mdriver -v
Team Name:10214602404
Member 1 :LiFang:10214602404@ecnu.edu.cn
Using default tracefiles in /root/10214602404/malloclab-handout/traces/
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.000156 36407
 1       yes   99%    5848  0.000136 42905
 2       yes   99%    6648  0.000179 37202
 3       yes   99%    5380  0.000160 33562
 4       yes   66%   14400  0.000137104956
 5       yes   92%    4800  0.002144  2238
 6       yes   92%    4800  0.002260  2123
 7       yes   55%   12000  0.053226   225
 8       yes   51%   24000  0.171239   140
 9       yes   80%   14401  0.000186 77633
10       yes   46%   14401  0.000082175195
Total          80%  112372  0.229907   489

Perf index = 48 (util) + 33 (thru) = 81/100
root@LAPTOP-L9FRK1MI:~/10214602404/malloclab-handout#
```

实验拆解以及画图演示：

（为了节省空间，所有空闲块中间没有画出空白块，每个函数图示能分情况的就分情况画了，情况比较多的，挑了较复杂的情况）

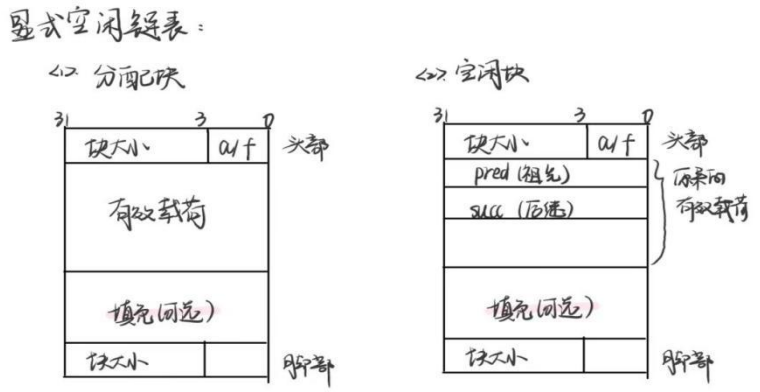
编译预处理设置、声明函数

部分参考课本已有的宏定义 MAX、GET、PUT、GET_SIZE、GET_ALLOC、HDRP、FTRP、NEXT_BLKP、PREV_BLKP，仿照课本格式新加了维护显式空闲链表需要的宏定义：GET_PTR_VAL、SET_PTR、 GET_PRED、GET_SUCC(n)、SET_PRED、SET_SUCC，还有静态指针 static char *free_list_headp、static char *free_list_tailp 来表示空闲链表的表头表尾。（各个宏定义的作用在代码注释中已写明）

声明所需要的函数：int mm_init(void); void *mm_malloc(size_t size); void mm_free(void *ptr); void *mm_realloc(void *ptr, size_t size); static void *coalesce(void *ptr); static void *extend_heap(size_t words); static void *find_fit(size_t asize); static void place(void *bp, size_t asize);

定义 debug 所需要的预处理宏定义#define DEBUG（确定是否需要 debug）、条件编译 #define VERBOSE 0　#ifdef DEBUG #define VERBOSE 1 #endif（VERBOSE=1，打印信息）、函数：static void mm_check(int verbose, const char* func);　static void mm_checkblock(int verbose, const char* func, void* bp);　static int mm_checkheap(int verbose, const char* func);

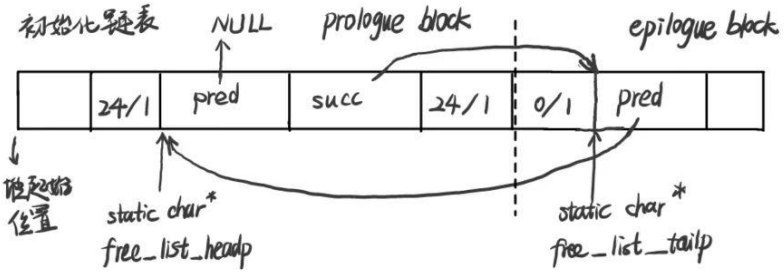
结构体图示：

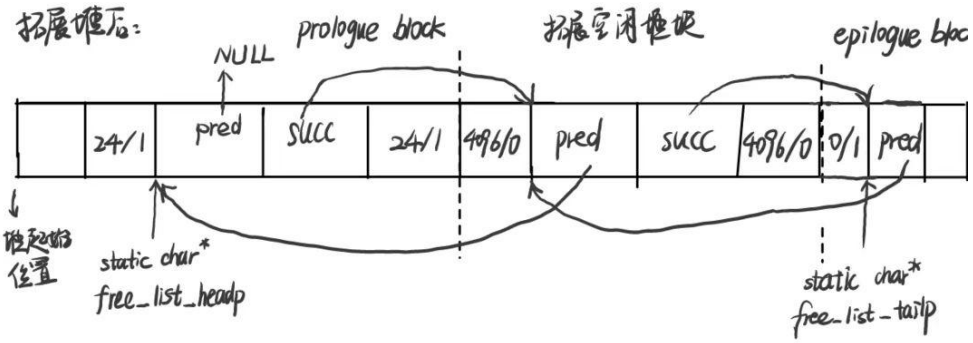


int mm_init(void);

init 实现堆初始化：如果成功返回 0，否则返回-1。首先利用 mem_sbrk 开辟一个大小为 40 字节的初始空堆，如果开辟出错则返回-1，然后进行序言块和结尾块的初始化，每个语句的操作已注释解释，完成 40 字节块初始化之后利用 extend_heap 进行堆开辟。

图示：

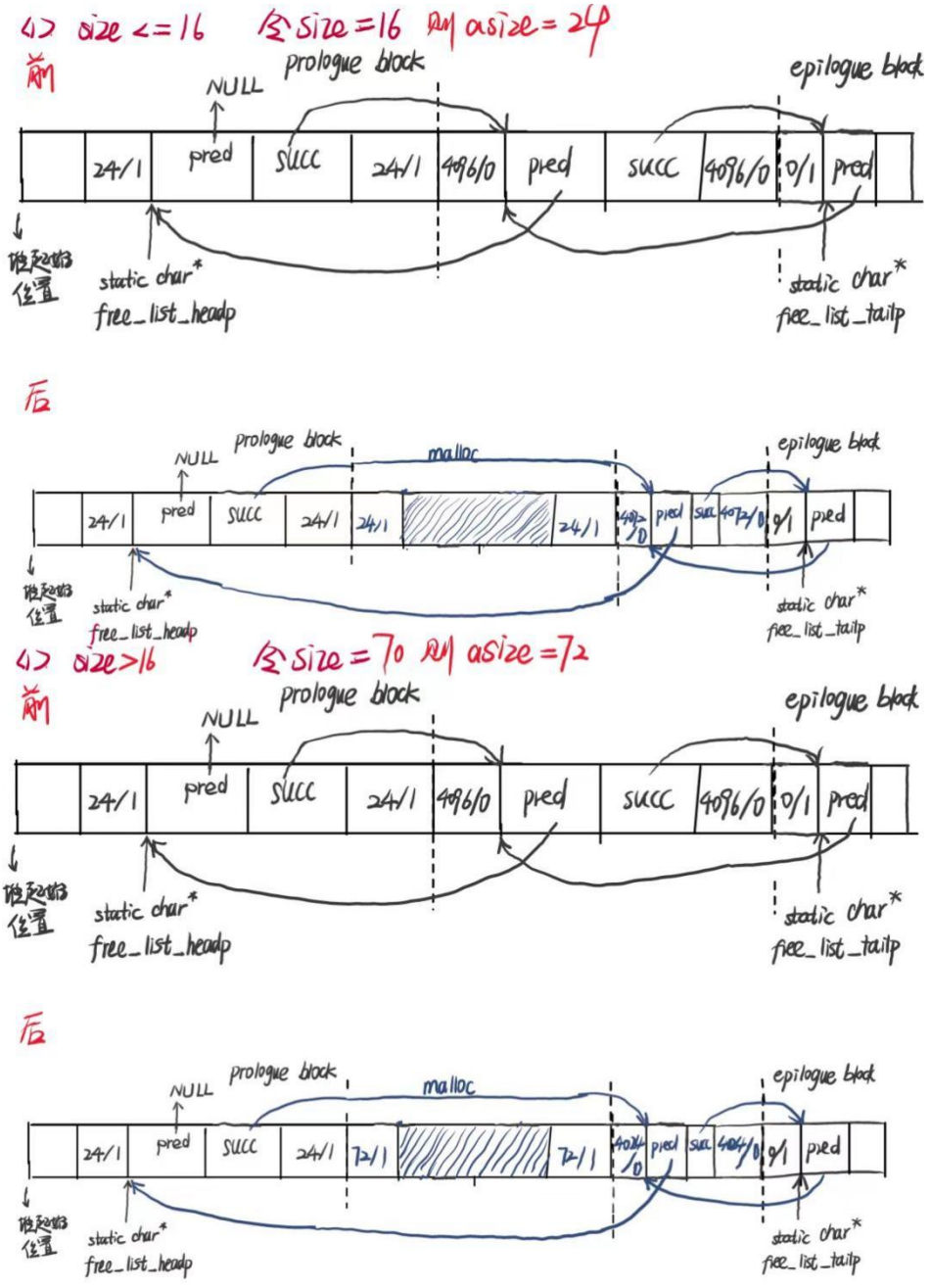




void *mm_malloc(size_t size);

malloc 仿照课本上隐式空闲链表的写法，更改了 asize 的设定、find_fit 函数和 place 函数需要修改空闲链表。首先，设定 asize（最小为 24Bytes），如果 size<16bytes，那么用最小块大小 24Bytes 代替；否则，调整头部尾部块大小和分配位，然后在空闲链表使用首次适配的方式顺序搜索，找到第一个适配的空闲块，调用 place 函数放置即可；假设没有合适空闲块就在 asize 和 CUNKSIZE 中选较大者去开辟新堆，再去放置。分配成功分配块地址，不成功返回 NULL。

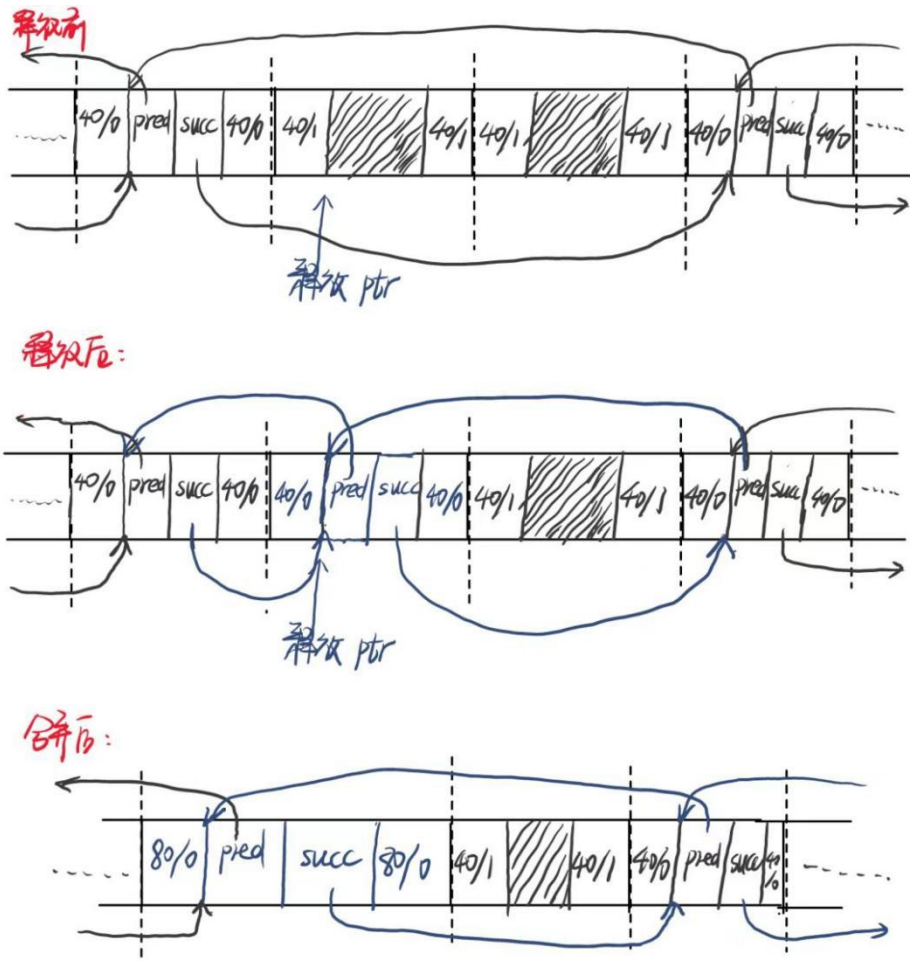
图示：



void mm_free(void *ptr);

free 实现释放块：为了让空闲块按照地址顺序排序，那么除了将这个块的头部和尾部未分配设置为 0 之外，还需要将其插入空闲链表。我采用了地址顺序排序：在空闲链表中顺序查找，找到离被释放块最近的第一个空闲的块，设置好被释放块的头脚部，然后更新被释放块和找到空闲块的 PRED 和 SUCC 值。接着调用 coalesce 进行合并。

图示：



void *mm_realloc(void *ptr, size_t size);

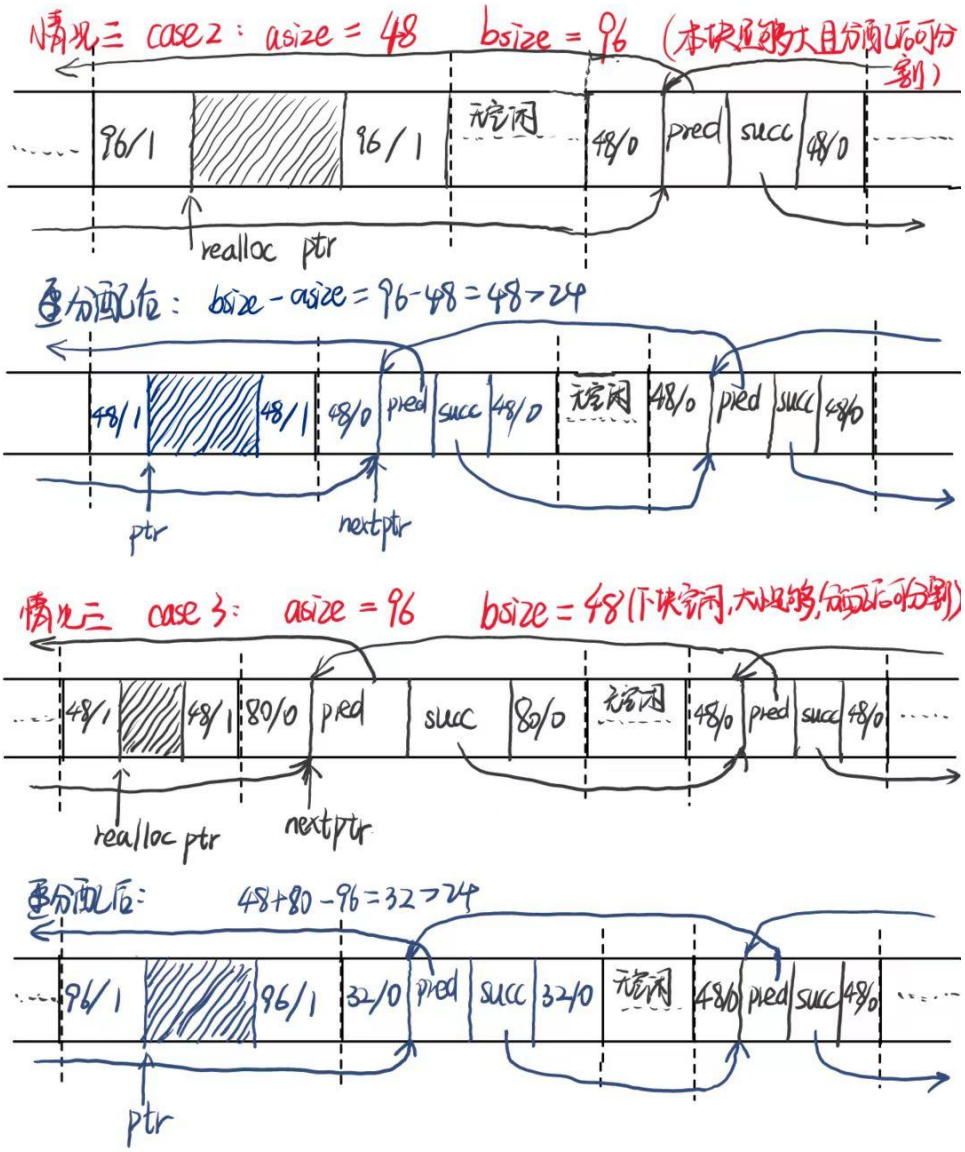
realloc 实现重新分配，和课本上的隐式链表区别在于：如果 newptr 可以等于 oldptr，跟 place 的操作类似，但不能直接调用 place 函数，因为 place 的实现是假设这个块是空闲块，来进行相应的操作的，对于显式空闲块的结构，place 需要用到空闲块的 PRED、SUCC 指针值，但是这里是已分配的块，没有 PRED、SUCC 这两个指针，因此不能调用 place。

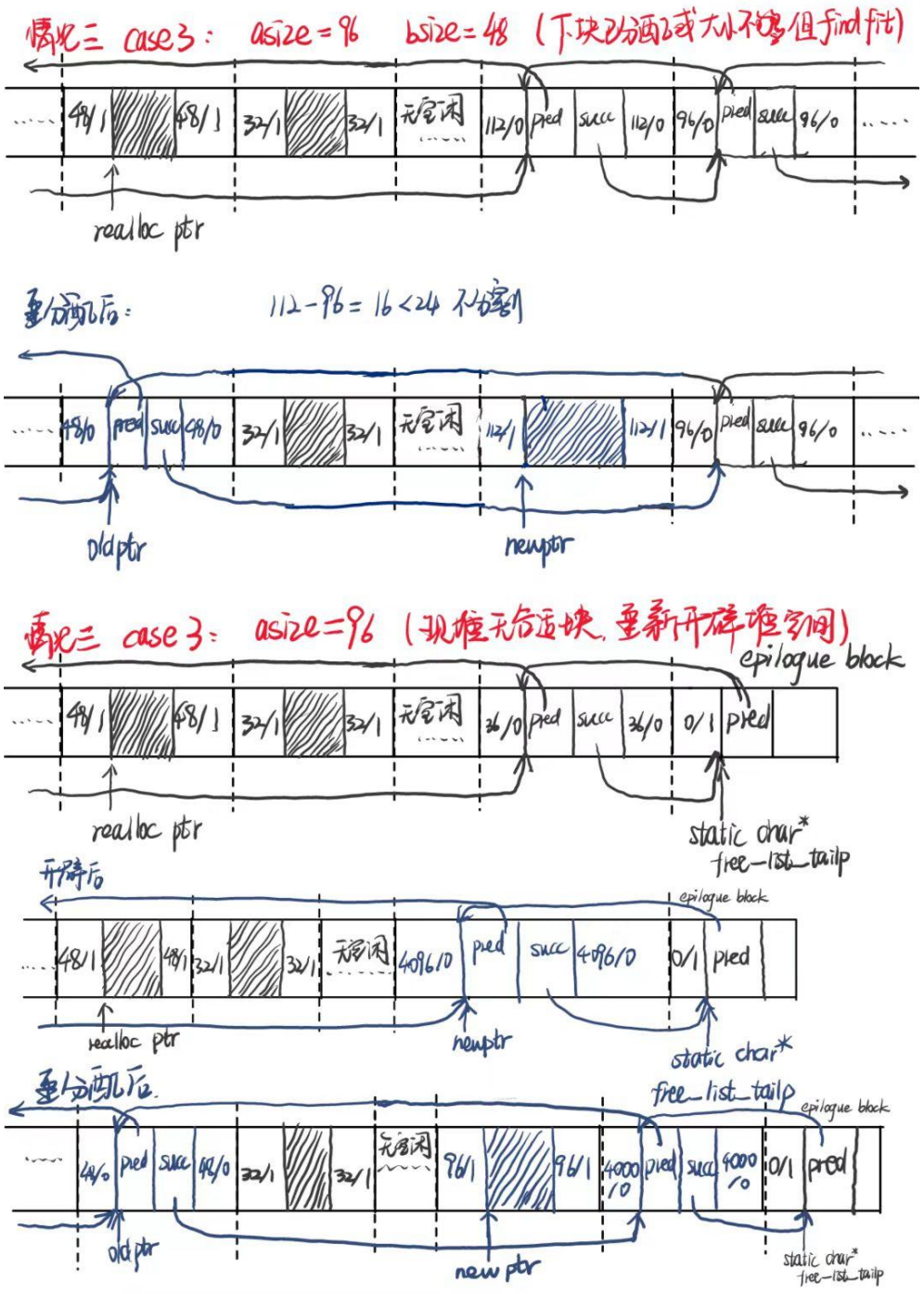
主要分三种情况进行：情况一：ptr 等于 NULL，等价于调用 mm_malloc(size)；情况二：size 等于 0，等价于调用 mm_free(ptr)；情况三：满足重新分配，需要像 malloc 一样调整一下块大小。

此时情况三又分为 3case:case1:调整后大小等于块原本大小，无需操作；case2:调整后大

小小于块原本大小，去查二者差值满不满足分割（满足的话就要去更新分割后两块的头脚部，并且把分割后空闲块插入到空闲链表中，不能直接用 place，不满足就可以返回了）；case3: 调整后大小大于块大小，去查下一块（要是下一块为空，并且大小相加后满足重分配就更新块信息即可，这中间还要看满不满足分割；要是下块是已分配的或者相加后大小不满足，就去搜索满足块，找不到就要去拓展堆，然后把重分配这一块放到合适的空闲块里）。

图示：





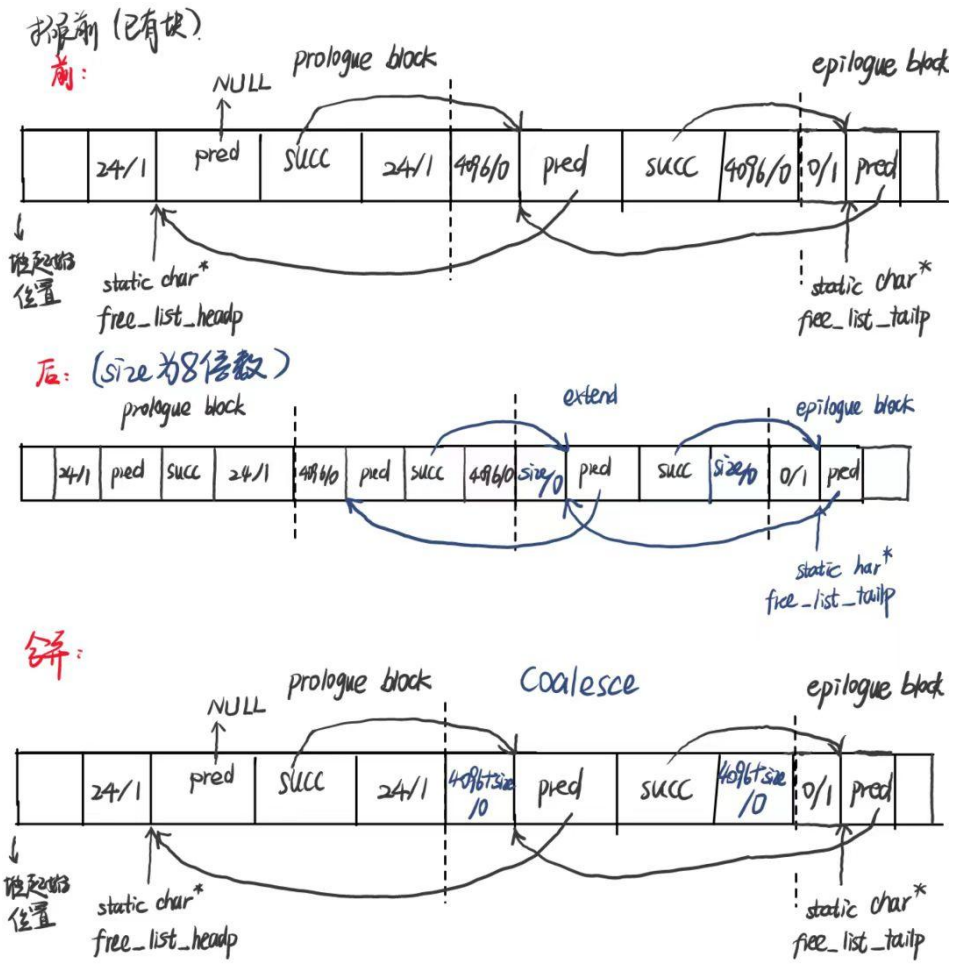
static void *coalesce(void *ptr);

coalesce 实现合并操作。和课本中的隐式空闲链表相类似，但是需要更改显式空闲链表的状态，使得相邻空闲块达到合并的效果。分四种情况：情况一：上下块已分配，无需操作；情况二：上块已分配，下块空闲（合并该块和下块）；情况三：上块空闲，下块已分配（合并该块和上块）；情况四：上下块空闲（合并三块）。只要是需要进行合并的情况。基本都是先更改块大小，再更新头脚部，然后更新 pred 和 succ，更新指向返回空闲块的地址指针。图示包含在 mm_free 和 extend_heap 中了。

static void *extend_heap(size_t words)：

拓展堆大小：当堆初始化或者 malloc 找不到合适空闲块时被调用。首先就是要将拓展堆大小更改为请求大小向上舍入最接近的 8 字节倍数，再利用 mem_sbrk 请求空间，目的为了对齐。申请空间成功后，初始化新分配块头部和脚部，再把这一块插入到空闲链表的中去，插入位置是链表尾节点前，然后更新结尾块。

图示：



static void *find_fit(size_t asize);

顺序从头到尾遍历空闲链表，然后采用首次适配，找到合适的第一个空闲块，找到返回块地址，没找到返回 NULL。

static void place(void *bp, size_t asize);

分两种情况放置，如果剩余部分大于 24 字节就分割：先设置当前块的头尾部大小 asize 和已分配，然后获取剩余部分块指针，把原本位于空闲链表中的空闲块指针删掉，插入入分割剩余空闲块；否则不分割放置：设置好头尾部大小 size 和已分配，然后从空闲链表中删除这块空闲块。place 的图示都包含在 malloc 和 realloc 里面了。

static void mm_check(int verbose, const char* func);

只有当#define DEBUG 时会使 VERBOSE=1，启动 mm_check 进行 debug；否则 VERBOSE=0。mm_check 会调用两个函数：每次检查，会先去调用 mm_checkheap 来检查序言块和结尾块的错误情况，然后再去遍历每块并调用 mm_checkblock 来检查每个块的错误情况。一旦出现错误，就会 printf 出相应的错误信息，可以辅助我去调试。

mm_checkheap 检查堆的序言块和结尾。序言块：如果序言块头部不等于脚部，发生错误；如果序言块头部不为 24/1，发生错误。结尾块：如果遍历空闲链表直至大小为 0 块，这个块不是结尾块，发生错误；如果结尾块头部不是 0/1，发生错误。

mm_checkblock 检查块：头脚部是否匹配相同、有效载荷区域是否对齐、每块大小是否双字对齐。如果错误会打印错误信息。

实验时间复杂度分析

mm_init()：先初始化先开辟的 24 字节，复杂度 O(1)；后面调用 extend_heap()，这个函数进行空间开辟的时间复杂度也是 O(1)。因此总的时间复杂度是 O(1)。

mm_malloc()：搜索到合适的进行放置时，复杂度 O(1)；没有合适需要先拓展，再放置，复杂度也是 O(1)。因此总的时间复杂度是 O(1)。

mm_free()：因为要从空闲链表头开始进行遍历，利用地址顺序查找加首次适配查找来维护空闲列表，所以时间复杂度为 O(L)，L 为空闲链表长度，简化为 O(n)；后面又调用 coalesce 进行合并块时，时间复杂度为 O(1)。因此总的时间复杂度是 O(n)。

mm_realloc()：情况一要调用 mm_malloc()，时间复杂度是 O(1)；情况一要调用 mm_free()，时间复杂度是 O(1)；情况三 case1 无需操作，时间复杂度是 O(1)；情况三 case2 满足空闲分割放置，要去遍历空闲链表，时间复杂度为 O(L)，L 为空闲链表长度，简化为 O(n)；情况三 case3 首先去查下一块是否满足，然后看是否调用 extend_heap,memcpy,mm_free 这些都需要时间复杂度是 O(1)。因此总的时间复杂度是 O(n)或 O(1)。