最小的已分配块大小是一个 4 字节头部和一个 1 字节有效载荷，舍入到 8 字节。而最小空闲块的大小是一个 4 字节的头部和一个 4 字节的脚部，加起来是 8 字节，已经是 8 的倍数，就不需要再舍入了。所以，这个分配器的最小块大小就是 8 字节。

对齐要求	已分配块	空闲块	最小块大小（字节）
单字	头部和脚部	头部和脚部	12
单字	头部，但是没有脚部	头部和脚部	8
双字	头部和脚部	头部和脚部	16
双字	头部，但是没有脚部	头部和脚部	8

9.8　这里没有特别的技巧。但是解答此题要求你理解简单的隐式链表分配器的剩余部分是如何工作的，是如何操作和遍历块的。

code/vm/malloc/mm.c

```
1  static void *find_fit(size_t asize)
2  {
3      /* First-fit search */
4      void *bp;
5
6      for (bp = heap_listp; GET_SIZE(HDRP(bp)) > 0; bp = NEXT_BLKP(bp)) {
7          if (!GET_ALLOC(HDRP(bp)) && (asize <= GET_SIZE(HDRP(bp)))) {
8              return bp;
9          }
10     }
11     return NULL; /* No fit */
12 #endif
13 }
```

code/vm/malloc/mm.c

9.9　这又是一个帮助你熟悉分配器的热身练习。注意对于这个分配器，最小块大小是 16 字节。如果分割后剩下的块大于或者等于最小块大小，那么我们就分割这个块（第 6～10 行）。这里唯一有技巧的部分是要意识到在移动到下一块之前（第 8 行），你必须放置新的已分配块（第 6 行和第 7 行）。

code/vm/malloc/mm.c

```
1  static void place(void *bp, size_t asize)
2  {
3      size_t csize = GET_SIZE(HDRP(bp));
4
5      if ((csize - asize) >= (2*DSIZE)) {
6          PUT(HDRP(bp), PACK(asize, 1));
7          PUT(FTRP(bp), PACK(asize, 1));
8          bp = NEXT_BLKP(bp);
9          PUT(HDRP(bp), PACK(csize-asize, 0));
10         PUT(FTRP(bp), PACK(csize-asize, 0));
11     }
12     else {
13         PUT(HDRP(bp), PACK(csize, 1));
14         PUT(FTRP(bp), PACK(csize, 1));
15     }
16 }
```

code/vm/malloc/mm.c

9.10　这里有一个会引起外部碎片的模式：应用对第一个大小类做大量的分配和释放请求，然后对第二个大小类做大量的分配和释放请求，接下来是对第三个大小类做大量的分配和释放请求，以此类推。对于每个大小类，分配器都创建了许多不会被回收的存储器，因为分配器不会合并，也因为应用不会再向这个大小类再次请求块了。