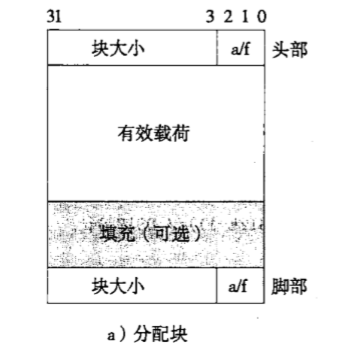
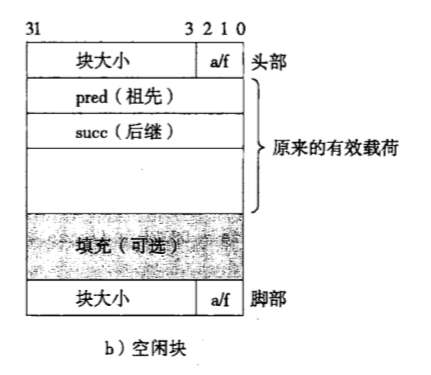
课本9.9 P561

存储器实现的必要技术：

* 针对空闲块的组织方法有以下三种：   
  a.隐式空闲链表(implicit free list)

遍历所有的块，根据头部的最后一位是否为0来判断是否是未分配的块（即空闲块）

  
b.显式空闲链表(explicit free list)



增加pred和succ，来存储前后空闲块的位置，只需要遍历空闲块就好  
c.分离空闲链表(segregated free list)

在常数时间内就可以完成

**吞吐量上：隐式<显式<分离**

* 查找空闲块的三个方法：   
  a.首次适应(first fit)   
  b.最佳适配(best fit)   
  c.下一次适配(next fit)

**下一次适配比第一次适配要块，首次适配的利用率比下一次适配要高**

**最佳适配的存储率比上面两种情况都高，但是时间花费的比较多一些**

**其实分离空闲链表就最接近于最佳适配策略了，不需要进行彻底的堆搜寻**

* **块的合并**

普通合并 || **带边界标记的合并**

何时合并？直接合并+推迟合并

分配器编写者的目标：

存储器利用率（有效载荷比率，碎片最小化）

吞吐量（单位时间内完成分配和释放请求的数目）

在二者之间寻找一个平衡

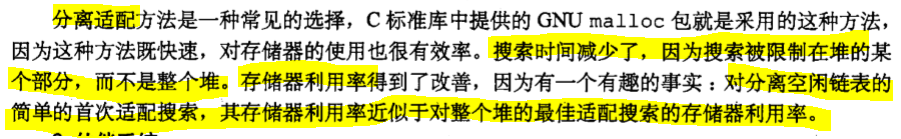
课本上给出的例子是：

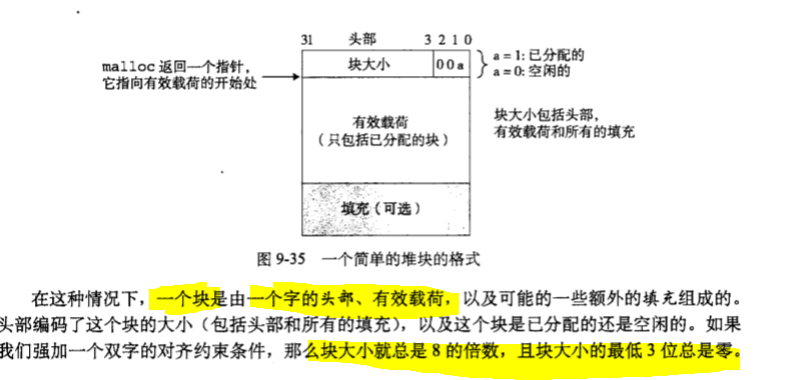
隐式空闲链表+首次适配

**综合来看的话：我准备采取**

**分离适配（最佳适配）+带边界标记的合并+最佳适配中的首次适配**

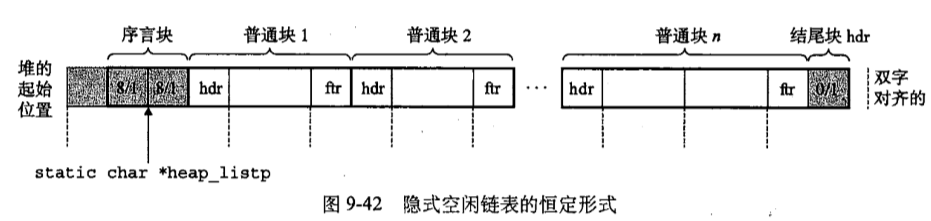
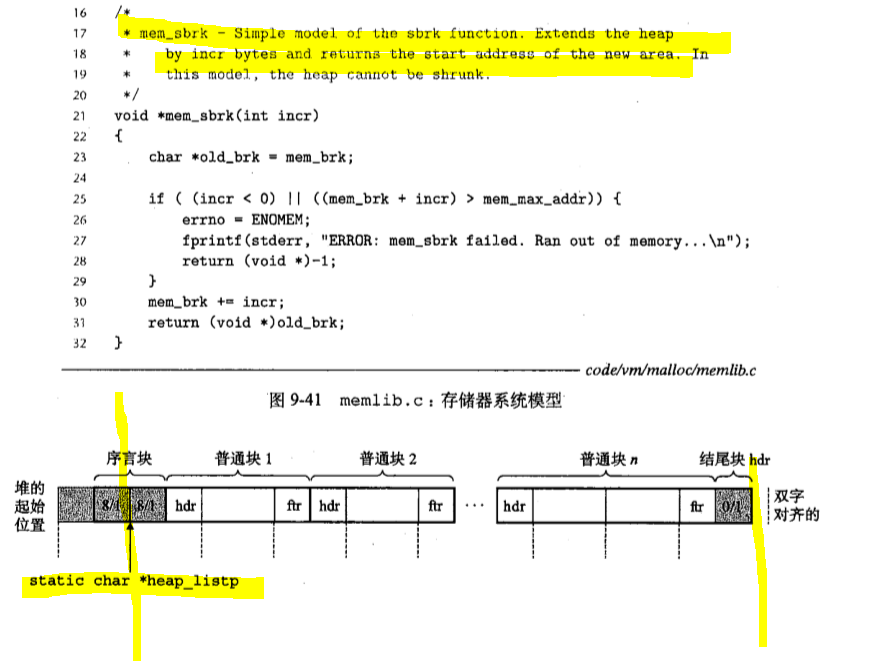
**（因为空闲链表不会合并空闲块）**

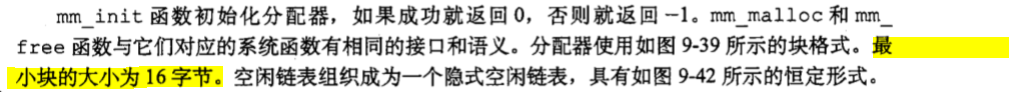


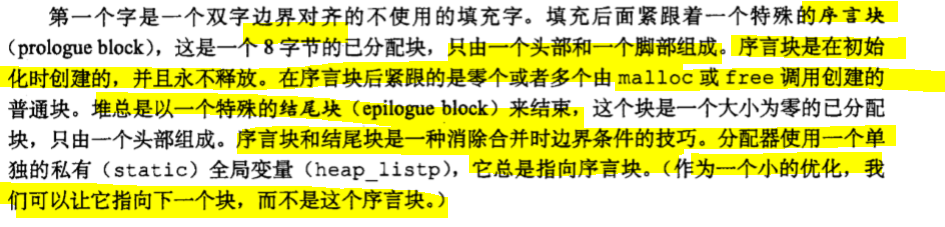


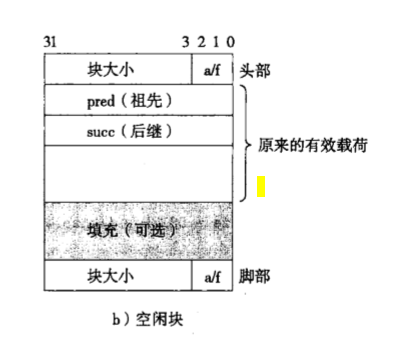
**#define PACK(size,alloc) ((size) | (alloc))**

**就是那个头部第一行的数值，size是块的大小，alloc用0/1来表示是否被分配**









byte是字节,也就是8位。用来储存char或者char类型指针。

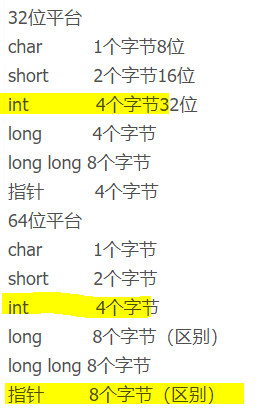
word是字, 2B

dword是双字，4B

一个字占得字节数和机器有关，一般而言说的都是两字节

一个汉字为2个字节，一个字节是8位，所以一个汉字就是16位

因为是 64 位机器，所以指针的大小是 8 字节，sizeof(size\_t) == 8，返回的指针必须是 8-byte 对齐的



* 空闲块的组织方法-segregated free list   
  segregated free list 中大小类的分类方法如下，并且将该list表放在heap的头部，通过序言块将它与数据隔离。在每一个大小类中，空闲块按照size由大到小排序。

空闲块查找方法 - best fit   
因为同一大小类中空闲块由小到大排序，所以查找是第一个合适的就是最适配的

mm\_realloc 改进   
对于请求的newsize>原始的oldsize这种情况，我们将运用类似coalesce中的方法，先去检查前后是否有空闲块，并是否满足前后空闲块和当前已分配的空闲块size相加大于newsize，如果是则合并，不需要再重新请求空闲块。如果不行，则需要重新mm\_malloc一块新的空间

