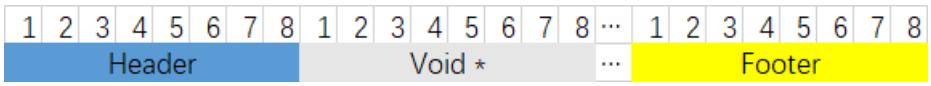
MallocLab

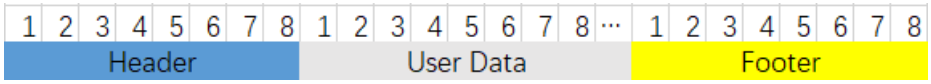
主算法

使用*Segregated free list*算法

- 基础数据结构
  创建25个桶，第i个桶中储存总长度（包括*header*和*footer*）在区间$(2^{i-1},2^i]$的空闲块，同时要求每个块大小至少为24。由于堆的大小一共只有$20MB$，所有空闲块都能被放到其中的一个桶中。
- 内存块的空间分配
  空闲块中的分配情况如下图所示。前8个字节*Header*储存当前块的长度，由于长度需要对其，所以最后的3个二进制位一定是0，最后一个二进制位为标记位（0表示空闲，1表示占用），标记为0。最后8个字节*Footer*的储存内容与*Header*相同。由于是空闲块，所以内部的空间是可以使用的，用来储存一个指针，表示在*free list*上的下一块空间在哪一个。



  占用块的分配情况如下图所示。其中，*Header*和*Footer*的储存内容与空闲块相似，但是标记位需要标记为1。由于占用块不用在链表上出现，所以中间的指针位可以给用户使用，以节省空间。



- *mm_malloc*

  现在*free list*中查找可能存在可以使用的块的桶（即$2^i \geq len$，i为桶的标号，len为对齐且加上*Header*和*Footer*后所需要的长度），然后再桶中使用*Best fit*查找可以使用的空闲块。如果当前桶中没有适合的块，则到下一个桶中找。如果所有的桶中都没有需要的块，则调用*mm_sbrk*向系统申请所需大小的内存块。

  找到一个内存块后，如果这个内存块的大于所需要空间加上24（保证切割后剩余块也是可以使用的），那么将这个内存块进行切割，把对应大小的块返回，剩余块放回*free list*上。否则，直接把整个内存块返回给用户。

- *mm_free*

  释放内存的时候，将标记位记为0，然后找到对应的桶，将这个内存块接到对应桶的链表上。

- *mm_realloc*

使用mm_malloc申请一块新的大小，然后使用$memcpy$将数据复制到新的内存块上，然后释放旧的内存块，把新的内存块返回给用户。

算法优化

- $free$内存块时向前合并、向后合并。

  在mm_free中，要把空闲块挂到链表上之前，使用$Header$查看后一个内存块是否是空闲块，如果也是空闲块，则将其从对应桶的链表上取下来（由于链表为单向链表，所以只能通过遍历链表进行删除），和当前块合并，并且循环至后一块不是空闲块，或者超出当前申请的堆大小时停止。

  向前合并类似，使用前一块的$Footer$进行查找。需要主义当前块是否是地址最小的块，如果是则不能向前找。

- $realloc$是判断前续、后续块是否能够合并，且合并后大小足够。

  在$mm_realloc$中，先判断前一块是否是空闲的，如果是则判断前一块和当前块放在一起能否达到需要的大小。如果可以，则将这两块合并，并且分割出相应大小的块进行数据拷贝，返回给用户。

  向后合并类似，但是合并后不必进行数据拷贝，直接返回当前指针即可。

- $malloc$向系统申请堆空间时，如果需要的空间与2的整数次幂相近，则申请2的整数次幂的空间。

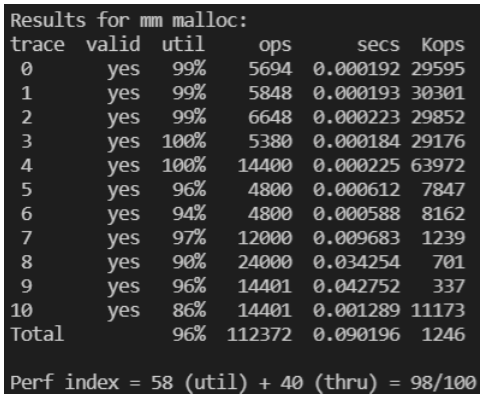
  向系统申请一块稍微大一点的内存块，当这个块被释放之后，可以使比较接近的大小都可以使用这个块，以提高空间利用率。

- 使用int代替$size_t$。

  由于堆空间的总大小只有$20MB$，所以$Header$和$Footer$中储存长度可以使用int，而不用$size_t$。同时，在初始化的时候申请一个大小为4字节的块，当作哨兵节点，可以解决地址不对齐的问题。

评测结果

- 该算法的内存利用率较高，速度较快，有不错的性能。



```
Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.000192 29595
 1       yes   99%    5848  0.000193 30301
 2       yes   99%    6648  0.000223 29852
 3       yes  100%    5380  0.000184 29176
 4       yes  100%   14400  0.000225 63972
 5       yes   96%    4800  0.000612  7847
 6       yes   94%    4800  0.000588  8162
 7       yes   97%   12000  0.009683  1239
 8       yes   90%   24000  0.034254   701
 9       yes   96%   14401  0.042752   337
10       yes   86%   14401  0.001289 11173
Total          96%  112372  0.090196  1246

Perf index = 58 (util) + 40 (thru) = 98/100
```