面向对象程序设计期末实验报告

学生：郭书廷（3170104871）**指导老师：**许威威

1.内存池设计与工作原理

由于 vector 申请的大小是随机的，且最大最小值差距很大，所以我并没有采用固定块大小的按块分配的内存池，我设计的内存池工作原理如下：

1.申请内存

当用户需要申请大小 N 的内存时，首先检查 N 是否大于阈值，**如果大于阈值**，则直接调用 malloc；其次检查**是否有被回收的块**可以使用，如果没有，**分配一块新的内存**给用户。内存池在空间不足的时候会自动扩容，新申请的大小为上一次申请大小的两倍，也就是说，内存池的总容量成倍增长。

2.释放内存

当用户需要释放一块大小为 N 的内存时， 检查 N 是否大于阈值，如果大于阈值，则直接调用 free；否则，按如下方法回收内存。设置一个参数 P，把最大允许的容量分为 P 块，按指数 K 增长。长度为 N 的块被放入第 Indx(N)个链表中。

$$K = MaxSize^{\frac{1}{P}}$$

$$Indx(N) = \lfloor \log_K N \rfloor$$

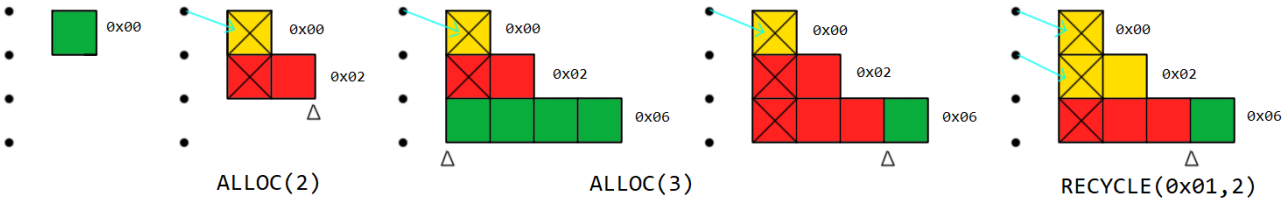
如果一个被回收的块，大小为 N，设 i=Indx(N)，那么，它会被放入第 i 个链表，因为第 i 个链表中，每个块的大小都在区间 $[K^i, K^{i+1})$内。此时，如果我们需要申请一块大小为 N1 的内存，则编号大于 Indx(N1)的链表中的任意一块都可以被使用。假设我们找到了一块足够大的内存，那么我们把前 N1 个位置分配给用户，把剩下的部分重新插入对应的链表以供继续使用。

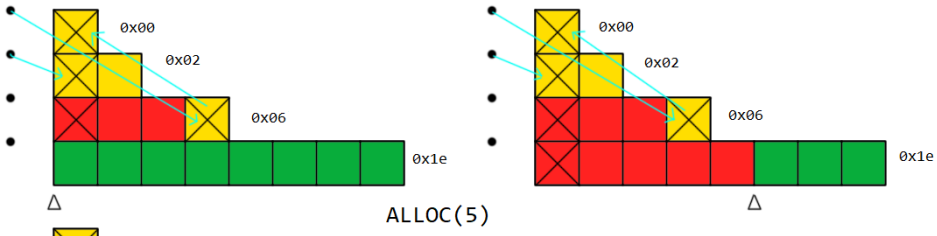
3.重新合并

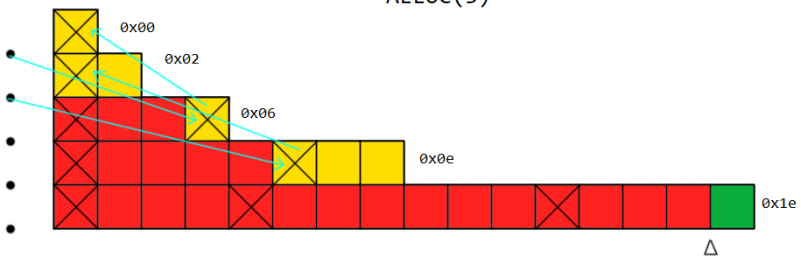
当申请和释放次数变多的时候，内存可能会被拆成小块，这时我们需要重新把连在一起的空闲块拼成一个。

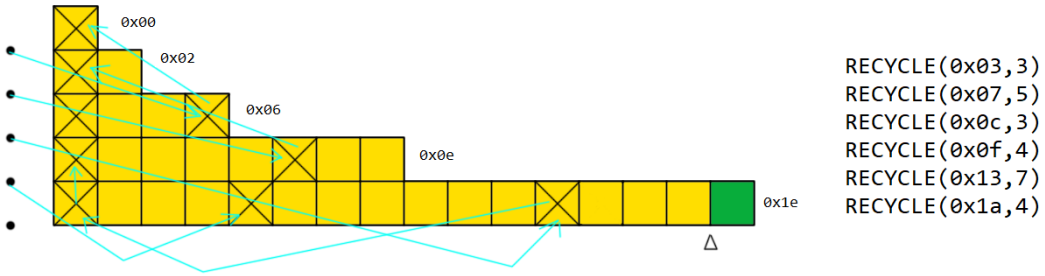
设置一个阈值 Q，每当空闲块的个数增加 Q 个时，把所有的块放入一个数组，进行排序、合并。从而提高内存利用率。

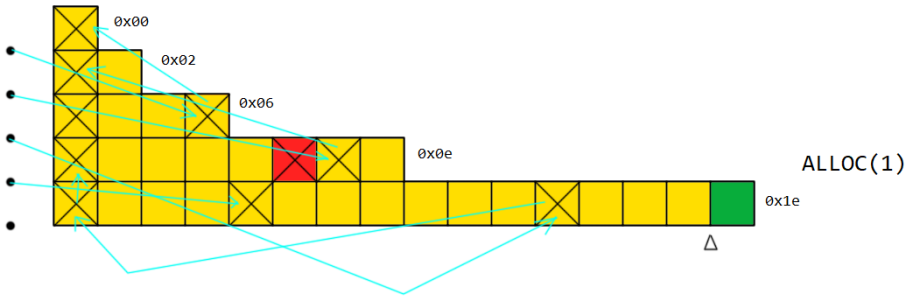
下面是一个例子。在这个例子中，K=2。

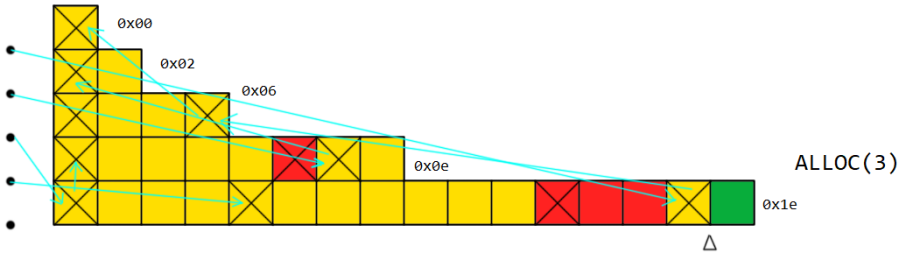

0x00
0x00
0x02
ALLOC(2)
0x00
0x02
0x06
ALLOC(3)
0x00
0x02
0x06
0x00
0x02
0x06
RECYCLE(0x01,2)

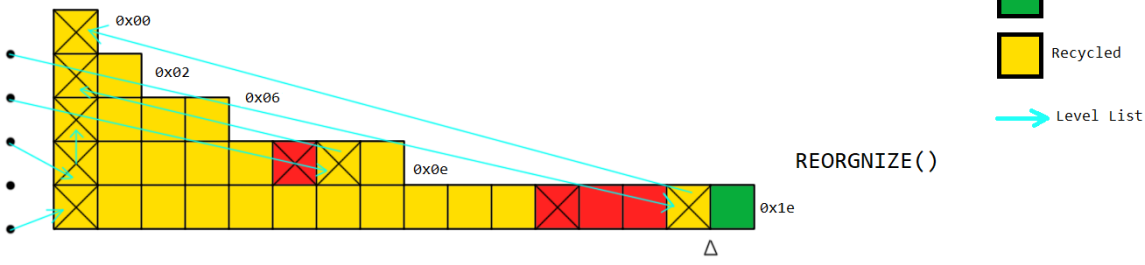

0x00
0x02
0x06
0x1e
ALLOC(5)
0x00
0x02
0x06
0x1e


0x00
0x02
0x06
0x0e
0x1e
ALLOC(4)
ALLOC(7)
ALLOC(4)


0x00
0x02
0x06
0x0e
0x1e
RECYCLE(0x03,3)
RECYCLE(0x07,5)
RECYCLE(0x0c,3)
RECYCLE(0x0f,4)
RECYCLE(0x13,7)
RECYCLE(0x1a,4)


0x00
0x02
0x06
0x0e
0x1e
ALLOC(1)


0x00
0x02
0x06
0x0e
0x1e
ALLOC(3)


0x00
0x02
0x06
0x0e
0x1e
REORGNIZE()
Allocated
Unused
Recycled
Level List

2.性能（测试结果为 100 次测试的平均值）

Built by vs2017 Releasex64.

Run on 2*8G RAM, Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz 2.80 GHz, Windows 10.

TestSize/PickSize/SmallSize	Allocate Time(s)		Memory(MB)	
	MyAllocator	std::allocator	MyAllocator	std::allocator
1000/1000/1000	0.009	0.011	10.1	8.6
10000/1000/1000	0.044	0.055	87.6	91.1
10000/10000/1000	0.051	0.062	101.2	98.0
15000/1000/1000	0.073	0.085	150.6	159.1
10000/100000/1000	0.074	0.077	113.0	101.2
15000/10000/1000	0.079	0.09	163.6	164.2
15000/100000/1000	0.101	0.107	169.5	168.1
20000/1000/1000	0.114	0.117	227.8	235.9
20000/10000/1000	0.115	0.128	243.2	241.6
20000/100000/1000	0.136	0.137	247.5	247.8
10000/1000000/1000	0.256	0.236	113.2	102.0
15000/1000000/1000	0.284	0.269	169.1	167.8
20000/1000000/1000	0.314	0.297	248.6	245.8
1000/1000/100000	0.009	0.011	16.8	15.2
10000/1000/100000	0.253	0.266	616.9	606.3
10000/10000/100000	0.387	0.393	927.8	780.7
15000/1000/100000	0.532	0.564	1348.7	1339.3
15000/10000/100000	0.776	0.847	1836.8	1617.9
20000/1000/100000	0.937	0.964	2358.4	2358.1
10000/100000/100000	0.986	1.505	2057.5	1254.1
20000/10000/100000	1.289	1.482	3023.5	2748.4
15000/100000/100000	1.858	5.835	4113.6	2594
10000/1000000/100000	2.784	3.954	2542.2	1459.8
20000/100000/100000	2.935	11.55	6647	4333
15000/1000000/100000	4.902	12.774	5589.4	3213.7
20000/1000000/100000	7.867	25.774	9763	5638.3

对于时间效率：

在大多数情况下，MyAllocator 比 std::allocator 快。

但可能是由于 stl::allocator 对小内存的分配做了优化，在**申请的块较小**的时候，MyAllocator **反而表现较差（红色标出）**，而在**申请的块较大**时能够体现出**明显的优势**（绿色标出）。

对于空间效率：

相比 std::allocator，MyAllocator 在空间利用率上有一定劣势。相比 std::allocator，MyAllocator 在数据非常大时，可以说是用空间换取了时间。但在实际测试中，MyAllocator 的空间利用率通常不会低于 40%。

3. 复杂度

时间复杂度：由于每次申请和释放时需要进行 logN 次运算，用来找到 Indx(N)，所以，申请和释放的时间复杂度为 O(logN)，即使有些时候需要重新花费 NlogN 的时间来重新排序、重新合并内存块，但只要阈值设置合适，仍能保证申请和释放操作的摊还复杂度为 O(logN)。

空间复杂度：由于内存池容量成倍增长，所以总耗费的空间通常在 N~4N 之间，空间复杂度为 O(N)。

郭书廷(3170104871)

2018/06/26