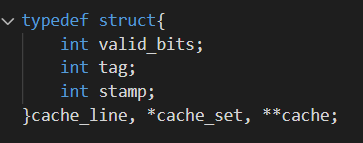
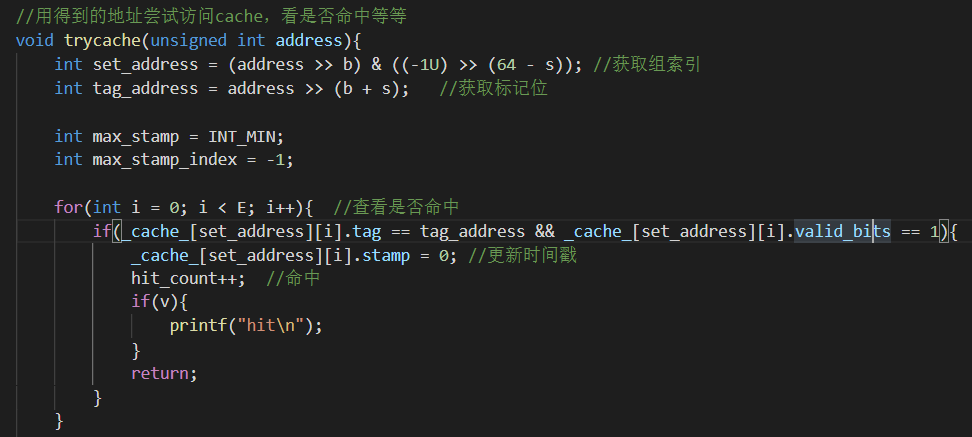
最终make生成的答案在cachelab-handout文件夹内

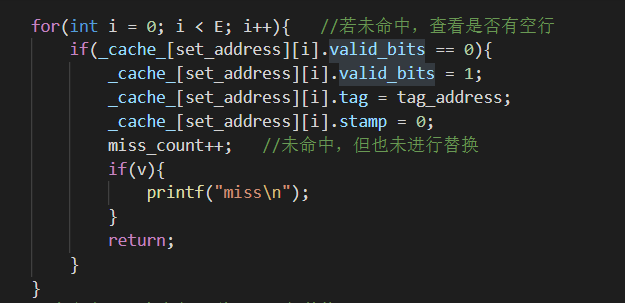
1.Part A

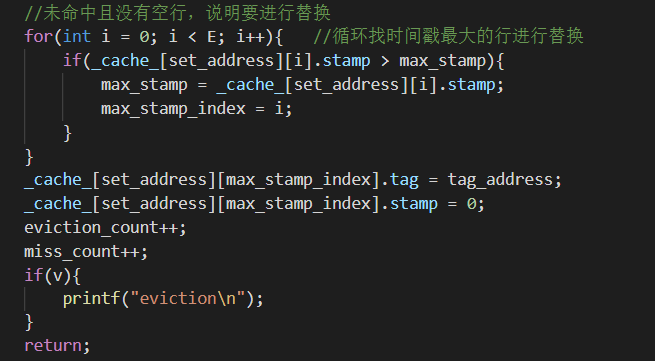
* 阅读实验说明，得知这一部分实际是要编写模拟一个cache。其中替换的操作选择使用LRU策略，为每行设置一个时间戳，时间戳越大表示最后访问时间距离现在越远，所以选择时间戳最大的元素进行替换即可。
* 首先定义cache的结构，因为每行cache要包括valid有效位、tag标记位和stamp时间戳（用于LRU算法决定覆盖哪一行cache），一个组中有多行，一个cache有多组。因此将cache定义为二维的结构体数组，声明一个对象并写出cache的初始化函数init\_cache（malloc动态分配空间）。读入参数等的操作不再赘述，利用说明中给出的getopt、fscanf函数进行读入即可。



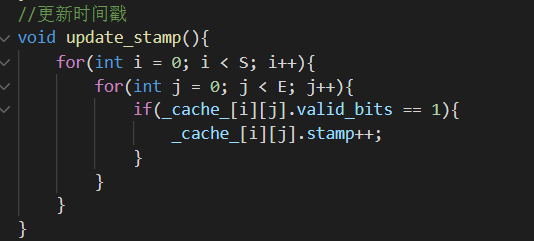
* 由于仅仅是模拟cache，不同的操作指令的实际操作很相似，所以共同写为一个函数trycache，L和S指令相当于调用一次，M指令相当于调用两次。
* 首先判断是否命中，需要组索引地址处的有效位为1且tag与请求的标记位一致，hit\_count计数。然后判断是否有空行，需要看是否存在有效位为0的行，miss\_count计数。若未命中且没有空行，则需要进行替换，根据LRU策略，寻找时间戳最大的进行替换，miss\_count和evition\_count计数。



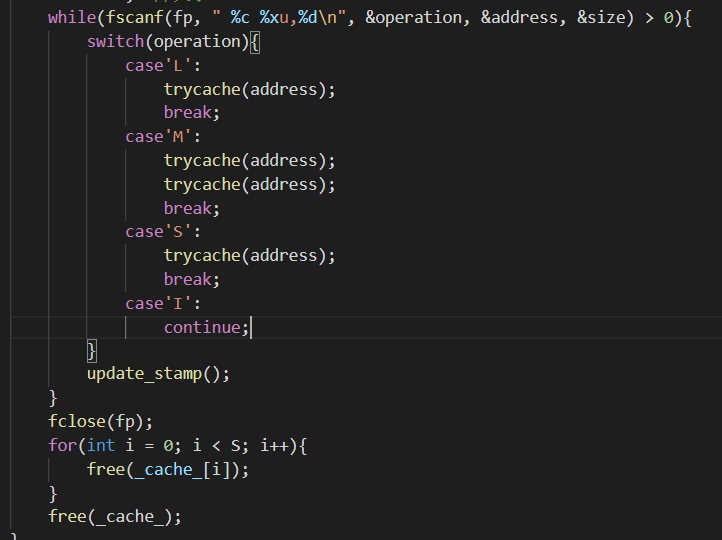




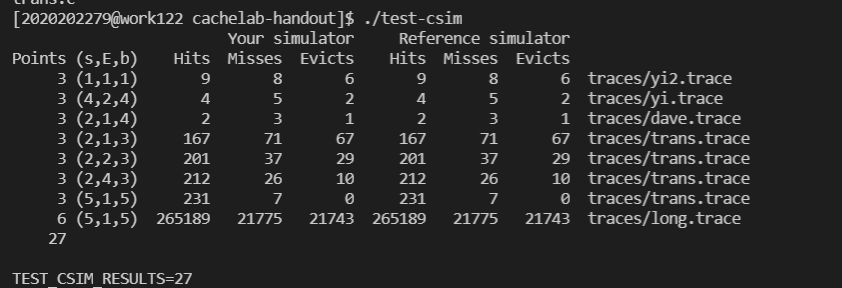
* 更新时间戳的函数update\_stamp，将所有有效的行的时间戳+1



* 对不同的指令(L、M、S)分类进行操作，每次操作完都更新时间戳，还要注意在最后将malloc申请的空间释放（free）



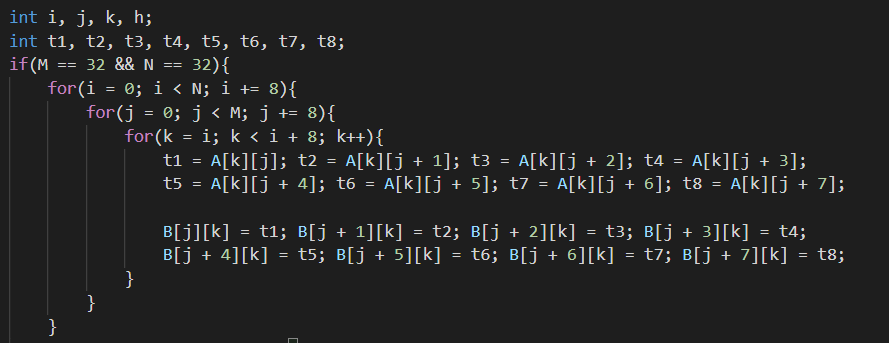
* 进行验证，结果正确



2.Part B

（1）32x32

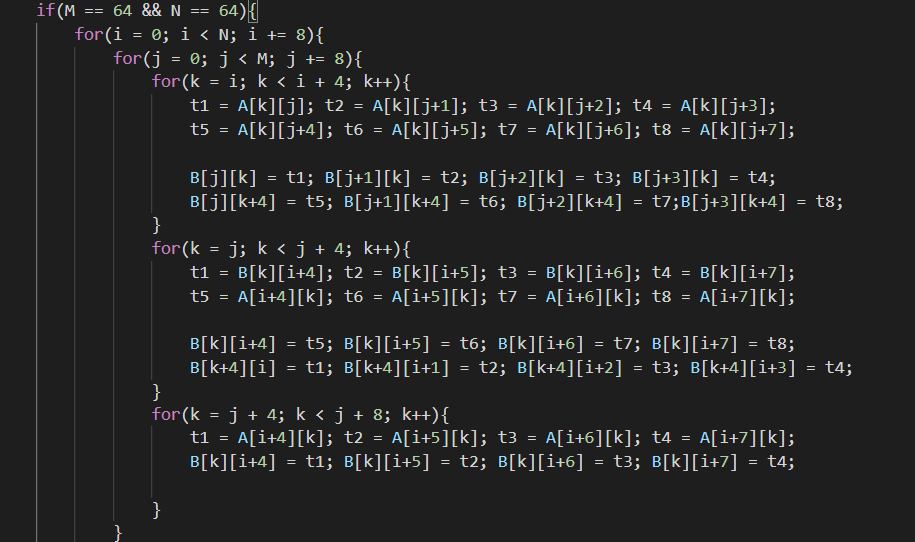
* 根据说明，s = 5, E = 1, b = 5，所以cache共有32组，每组一行，每行存储32字节 （8个int）。整个cache一共可以存放矩阵的8行。采取分块策略，用8x8分块进行优化。但由于AB相同位置的元素是映射在同一个cache line上的，所以对于处在对角线上的区域，会产生大量冲突不命中，比如A的第四行被B的第四行覆盖后，又要再次访问A的第四行的情况。所以利用8个局部变量t1-t8，一次性取出一个块中所有元素，之后不在访问这个块，以此减少冲突不命中的情况。



* 经验证，miss数为287，满足要求

（2）64x64

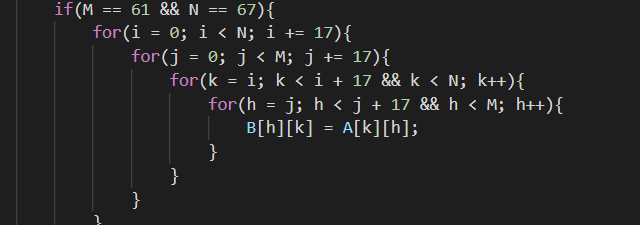
* 对于64x64的矩阵，cache只能存放矩阵的4行。所以如果还使用8x8的分块，会因为映射到相同的块，在按列访问B时产生大量的冲突不命中。因此，希望在同样的miss次数下，把更多的数据先暂存到B中的其他位置，在之后的步骤中再进行移动。考虑如下步骤：1.先把A的前四行（4x8）转置存入B的前四行，这样前四行前四列就已转置完成，而后四列存放在B的右上角，不必再访问A的前四行；2.按列依次读取A的左下角，按行读取B的右上角，都暂存在变量t1-8中，然后分别按行填入B的右上角和B的左下角；3.把A的右下角填入B的右下角。按照这个顺序访问，B数组每块的元素中不命中数可以明显减少，达到要求。



* 经验证，miss数为1219，满足要求

（3）61x67

* 由于矩阵规模不再是8的倍数，所以需要多次尝试不同的分块策略。经过尝试各种大小规模的分块，发现17x17的分块可以使miss数达到2000以下，满足要求，所以使用该分块策略。



* 经验证，miss数为1950，满足要求

