

**数据库系统实验报告**

题 目 数据库系统算法的实现

学 院 计算学部

专 业 软件工程

学 号 120L022109

学 生 李世轩

任 课 教 师 战徳臣

哈尔滨工业大学计算机科学与技术学院

2023.5

##### 实验环境

操作系统 Windows 11

编程语言 C语言

编程IDE CLion 2023.1.2

##### 程序运行结果

这里先介绍一下，程序主函数。

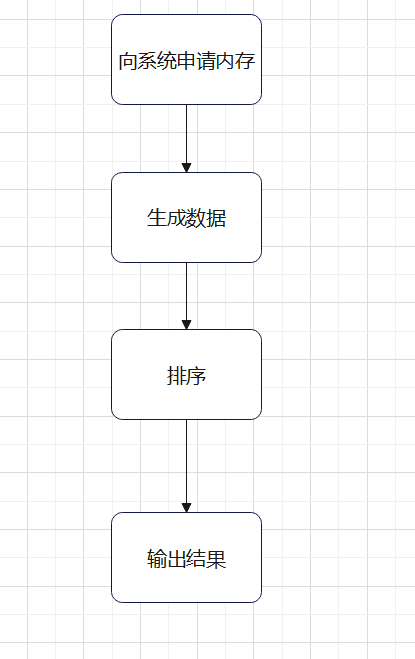


main函数会接受一个或零个参数，指明生成数据的数量，若没有这个参数则默认生成100条数据。接下来会输出一些基本信息，从上往下分别是，内存块的大小（按照字节计算），内存块数量，单条记录的内存大小，一个内存块可以存放的记录数量（为了进行对齐）。

然后会向系统申请一块前面定义好内存大小的内存，并将其置为0。

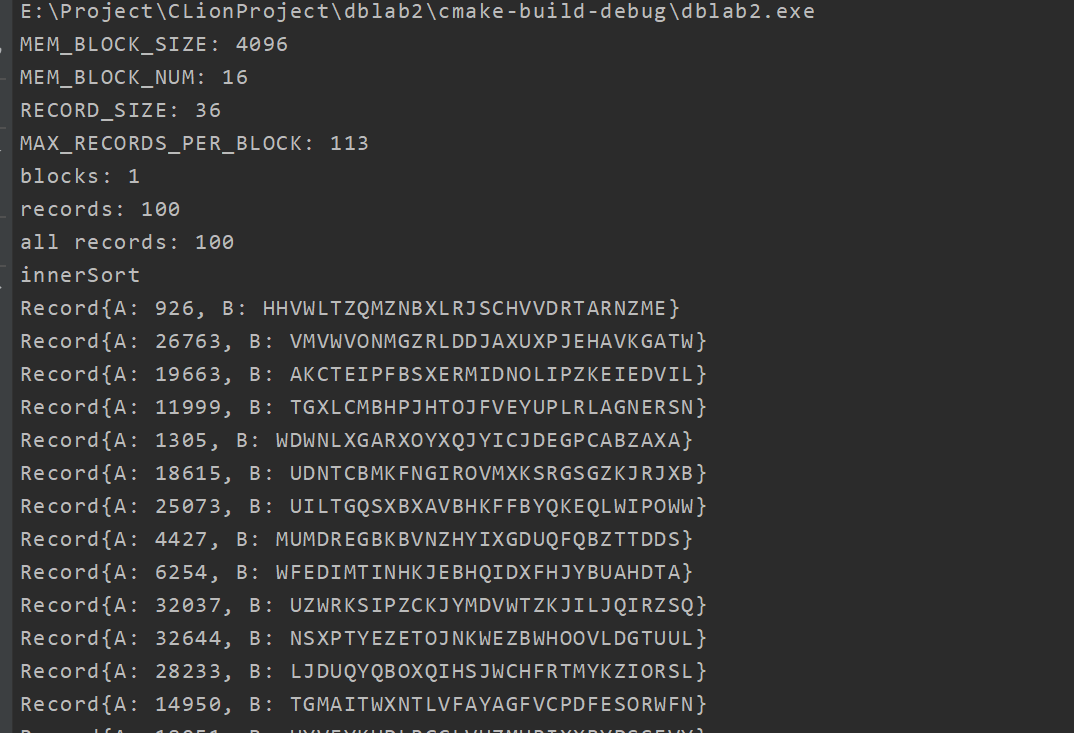
接下来调用generateData函数，向文件input.dat中输出随机生成的数据。然后调用mergeSort函数对input.dat中的数据进行排序并输出到output.dat中。然后使用printData将这两个文件中的数据输出到控制台。（这些函数也可以单独使用，但是不能处理内存块大小或记录大小改变的情况。）

流程图如下：

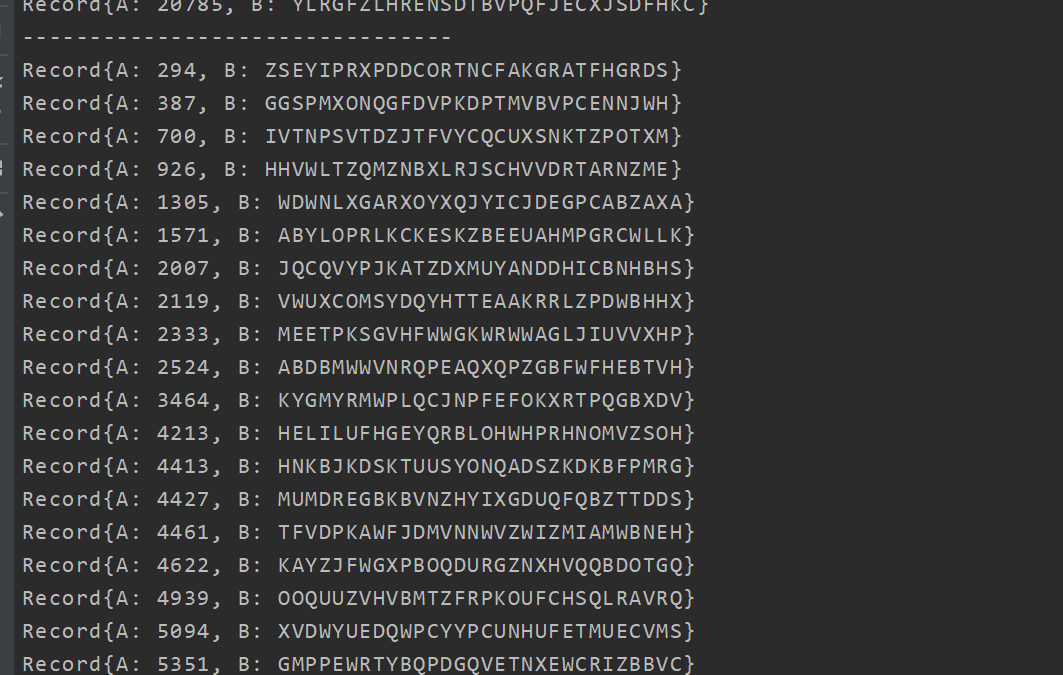


运行结果如下：

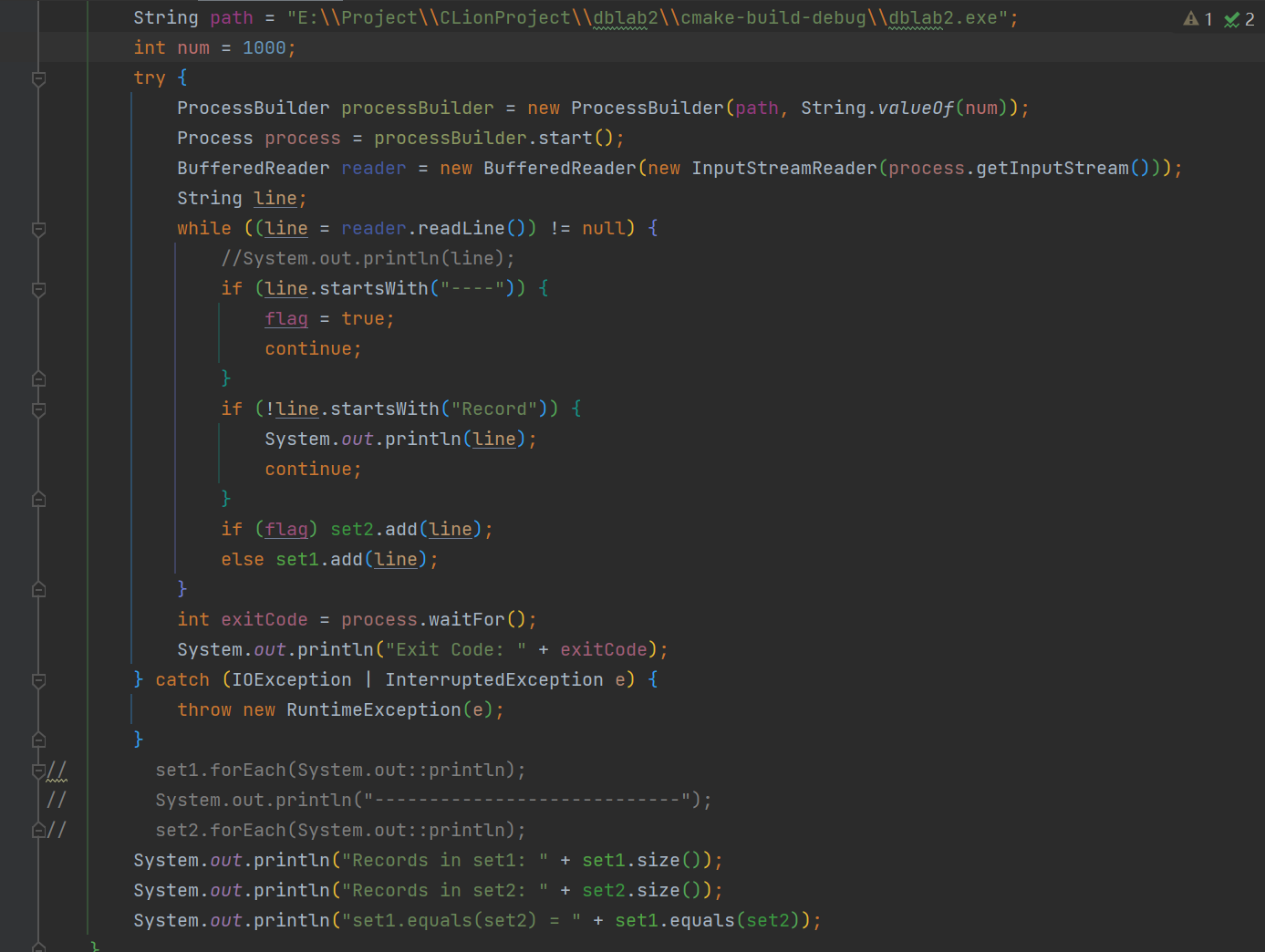
前面输出一些基本信息和未排序文件中的内容。



在分割线后输出排好序的文件中的内容。

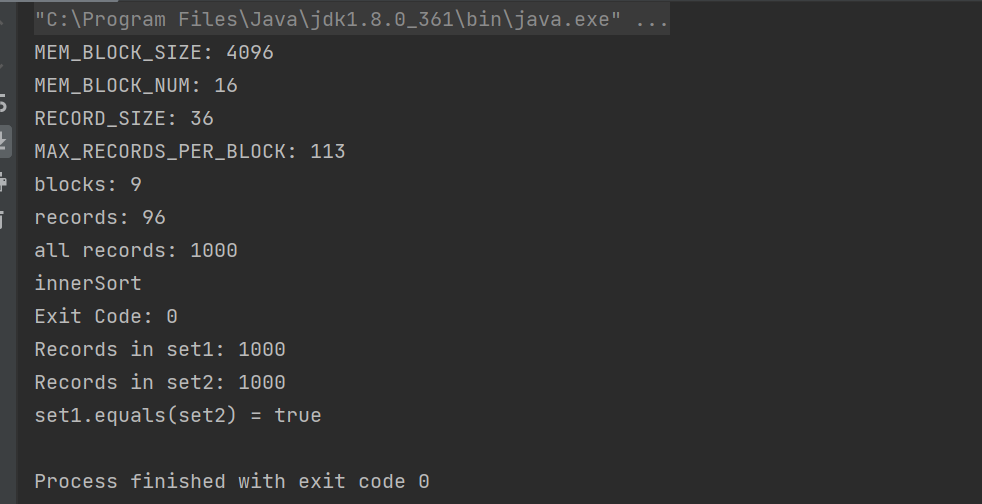


另外，这里我还编写了一份java脚本，用来运行这个c语言程序。

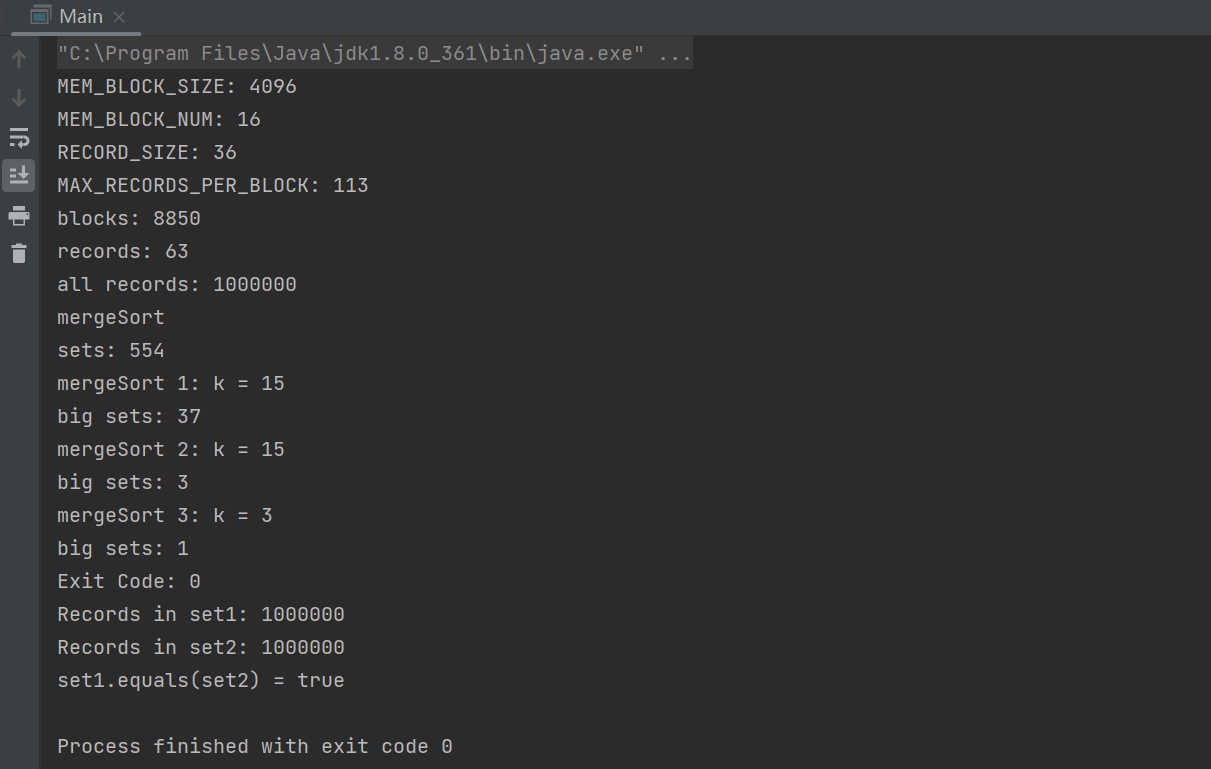


其思路为，运行程序，并读取其输出到控制台中的文字。当读取到的一行开头不为Record时，将其输出到自己的控制台，当读取到Record时，对一个boolean标记判断，若为false将其放到Set1中，否则，放入到Set2中。这个标记初始化为false，当读取到“------”分割时，将标记置为true，这样就可以借助java的set类判断，排序前后的文件中记录的集合是否一致。

其运行结果为如下：



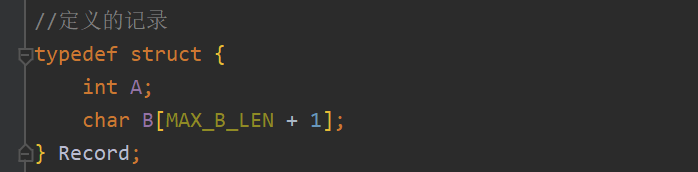
这一次的运行中，生成记录数为1000，因为其占用内存块比拥有的内存块少，直接进行内排序（innerSort）。



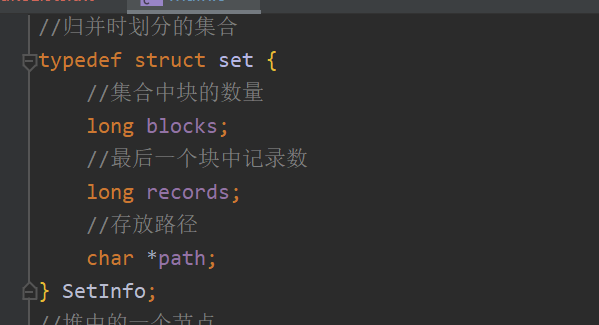
这一次的运行中，生成记录数为1000000，因为其占用内存块比拥有的内存块多，进行归并排序（mergeSort）。首先，将所有记录分为554个集合，第一遍，进行15路归并（因为当前内存块一共只有15块，必须剩余一个块作为输出缓存），完成后生成37个集合，第二遍，进行15路归并，完成后剩余3个集合。最后进行3路归并。

##### 程序实现

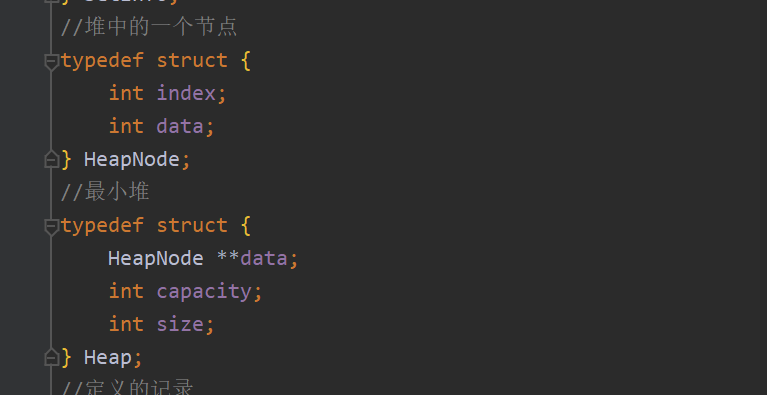
先介绍一下，程序中用到的数据结构。



这是定义好的记录的格式，A为整数，B为字符串，其长度由宏定义指明。

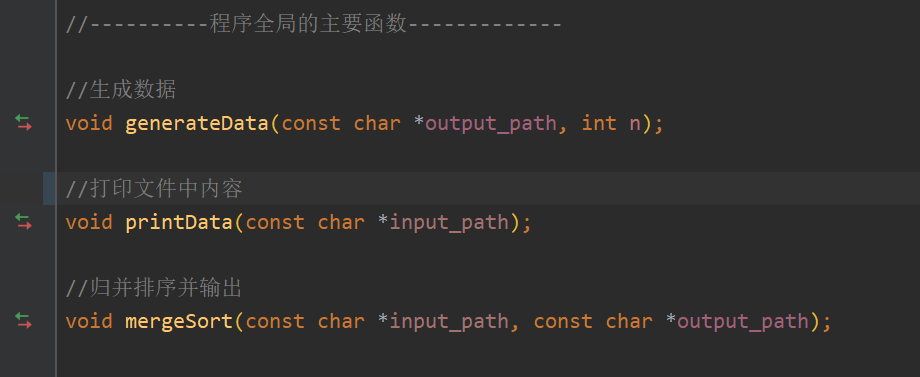


这个数据结构用来记录归并排序中划分的集合的信息。



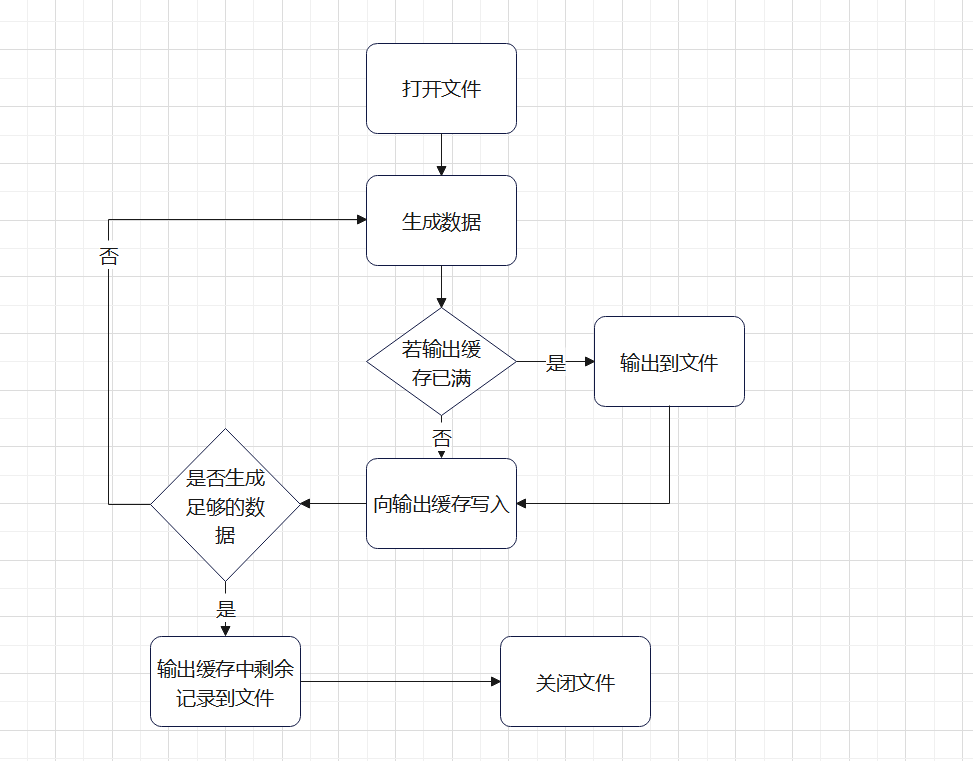
这两个数据结构是一个最小堆，用来在归并排序过程中进行局部的排序。

接下来介绍程序的主要函数。

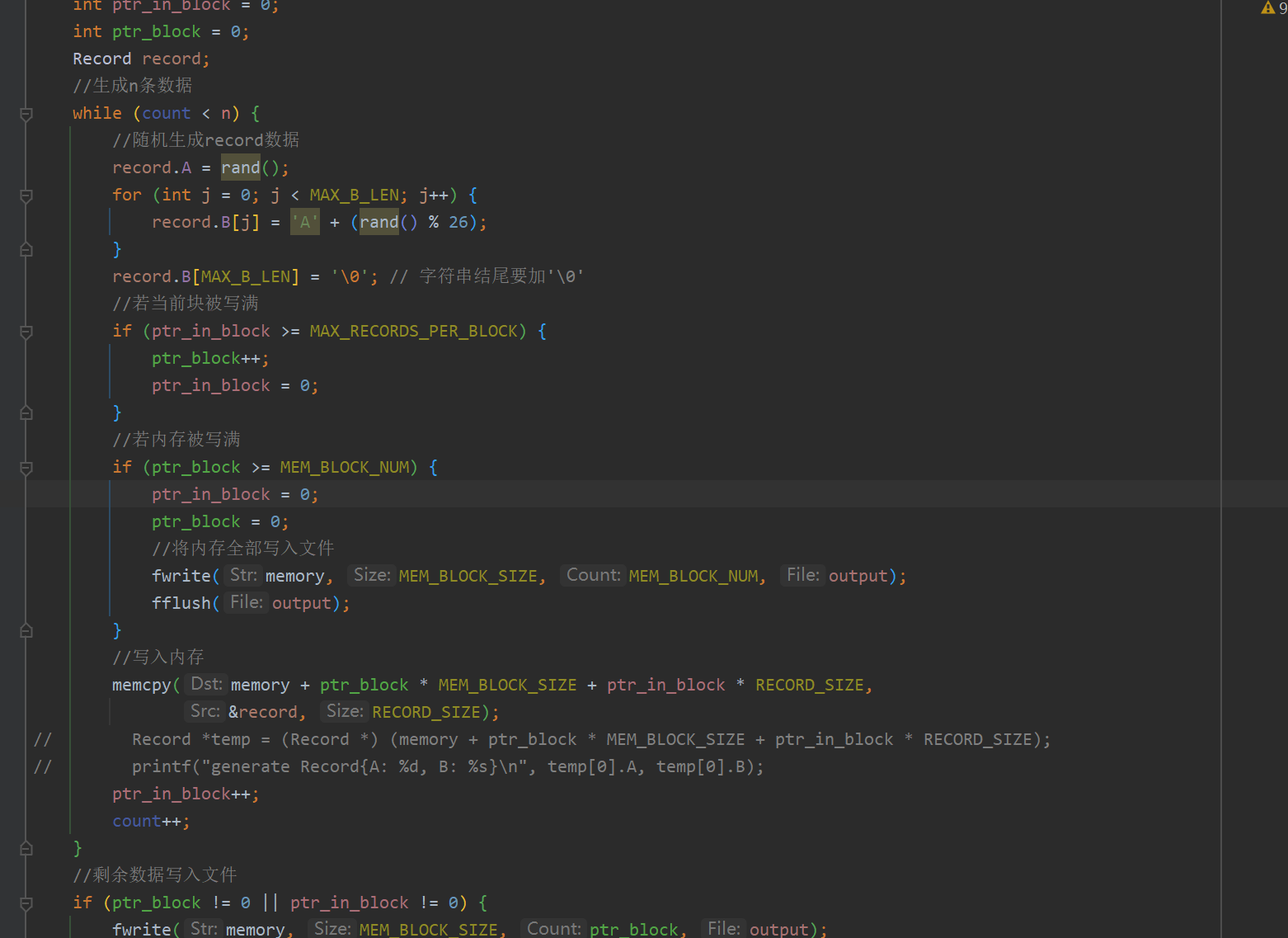


**void generateData(const char \*output\_path, int n)**

这个函数用来随机生成数据，流程图如下：



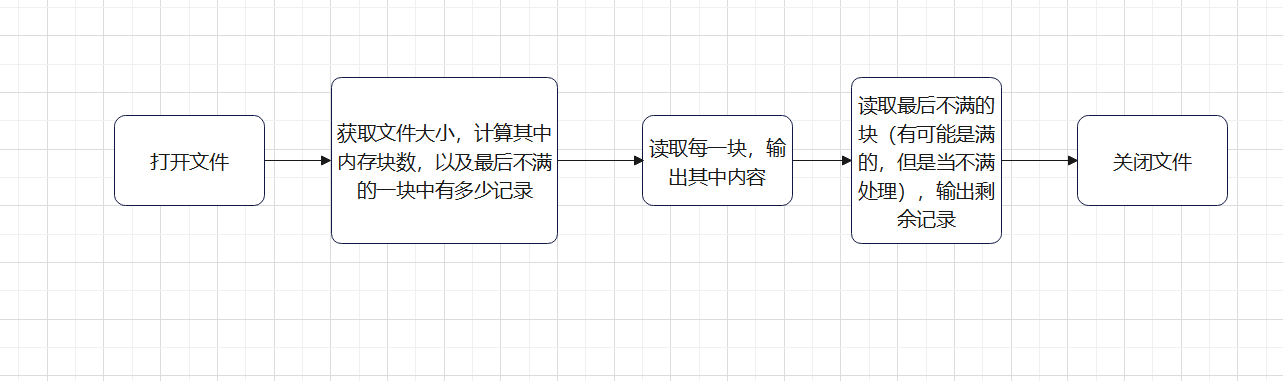
具体代码如下：



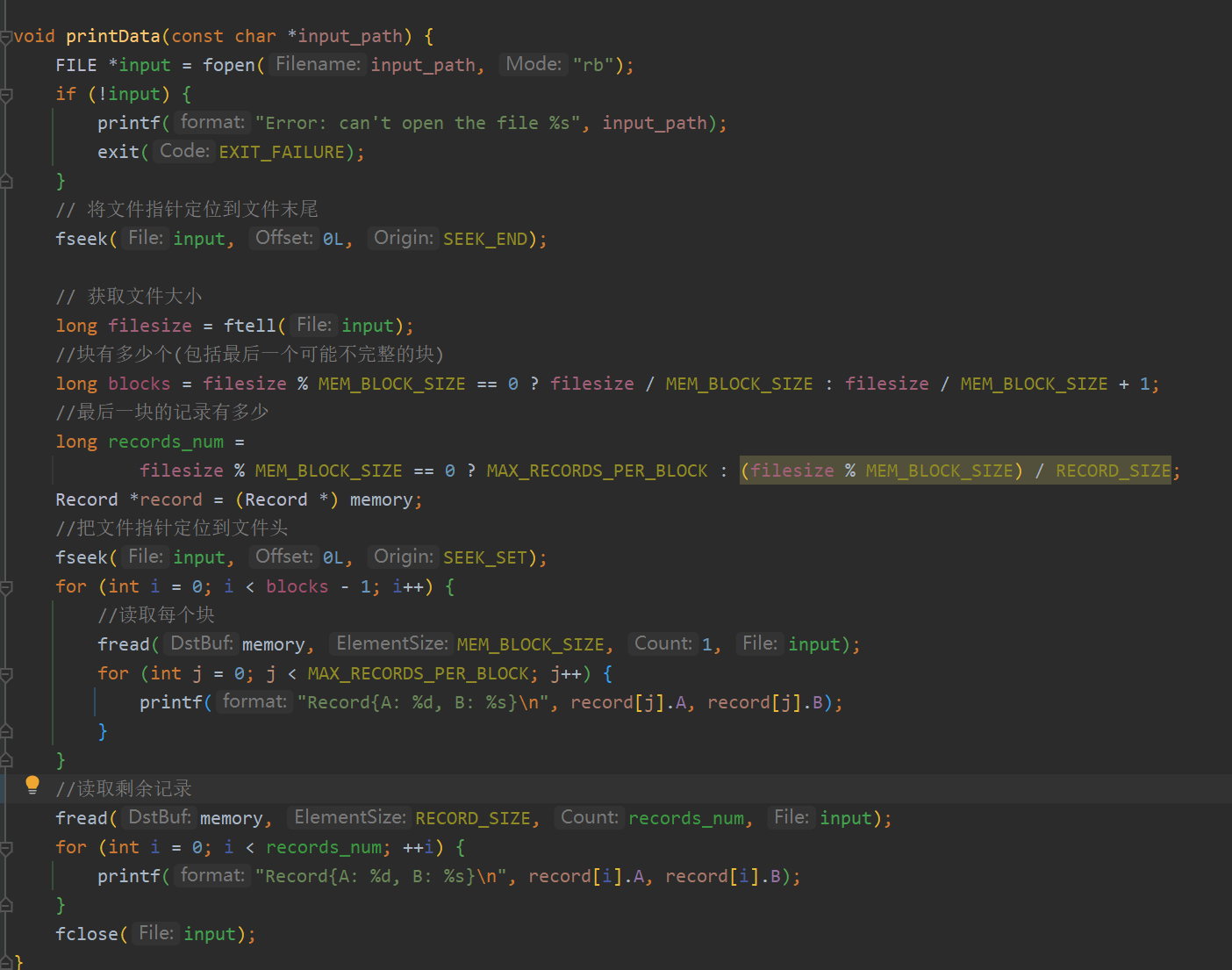
这个函数的实现思路为，先生成一条记录，如果将输出缓存写满，那么就将输出缓存中的内容输出到文件中，如果生成数量足够，那么将缓存区中的内容都写出到文件中。

**void printData(const char \*input\_path)**

这个函数用来输出文件中的内容到控制台，代码流程图如下：



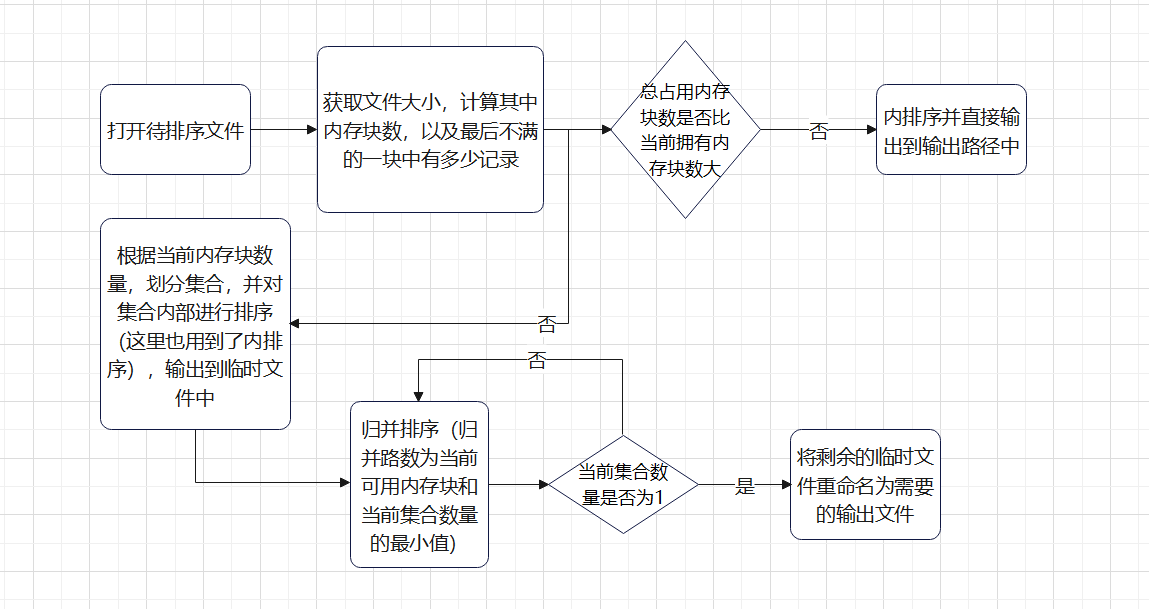
具体代码如下：



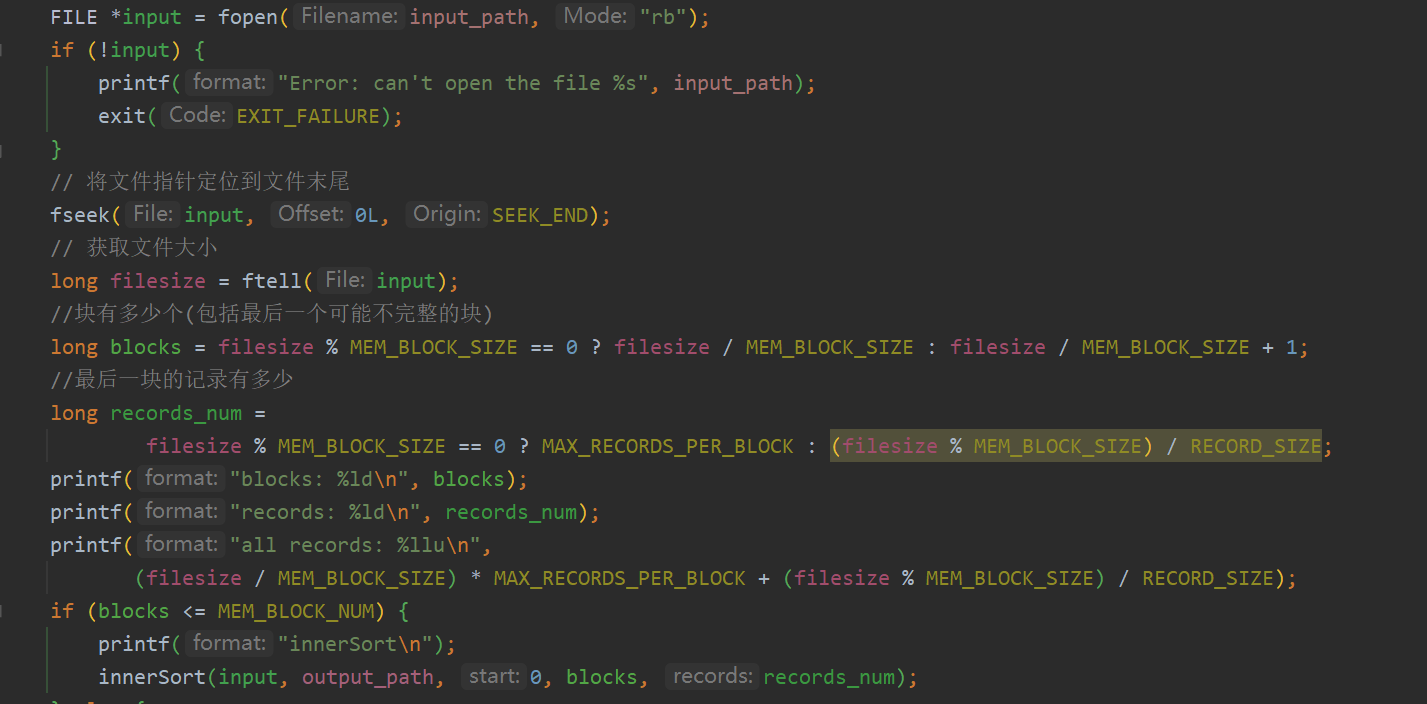
这个函数的实现思路为，先获取文件大小，并计算其中中有多少内存块，以及最后一个内存块中的记录数（有可能最后一块没有存满）。不断读取所有内存块输出其中内容，然后读取最后不满的块（有可能是满的，但是当不满处理），输出剩余记录。

**void mergeSort(const char \*input\_path, const char \*output\_path)**

这个函数将input的文件排序，并输出到output文件中。代码流程图如下：



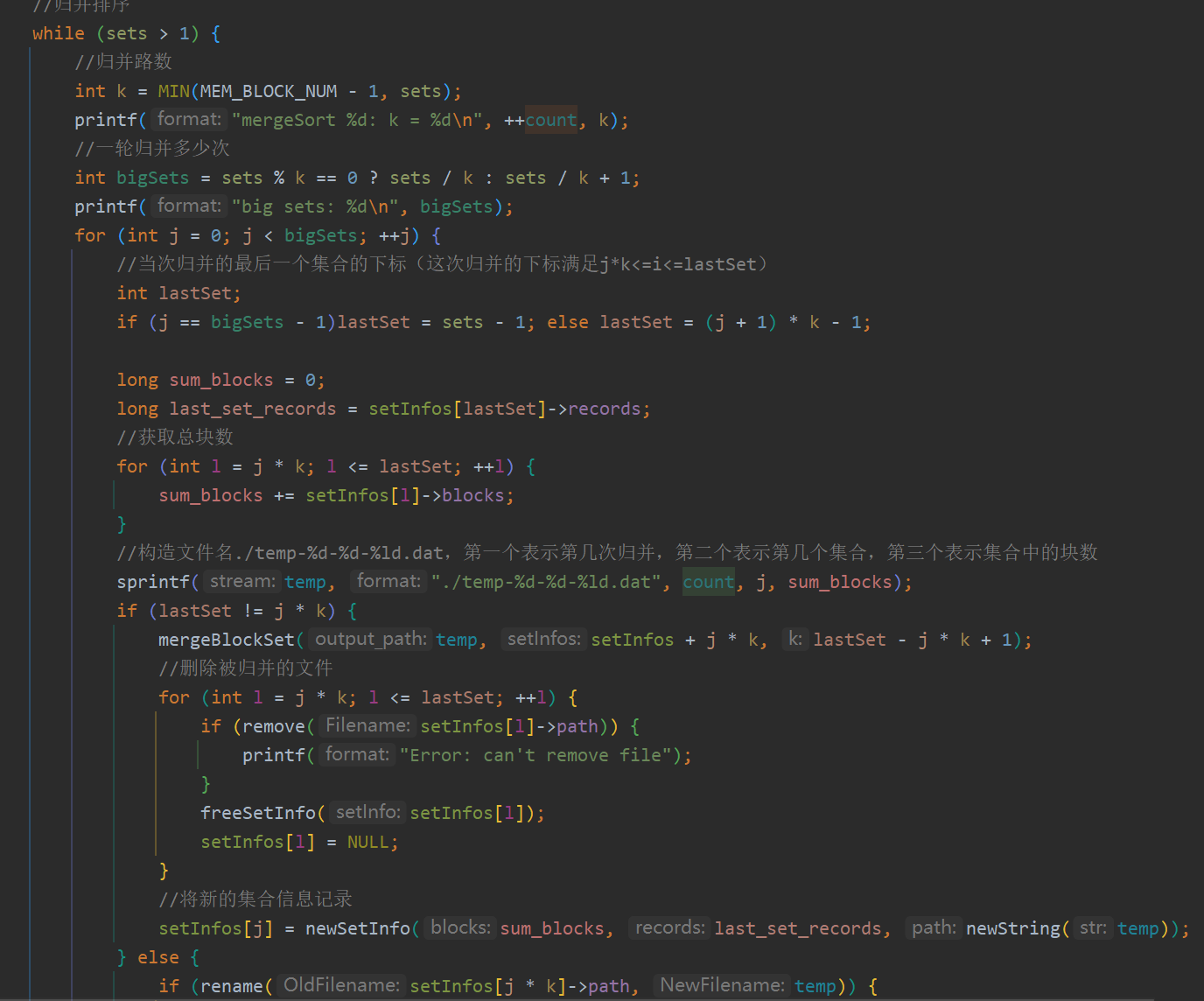
这个函数可以被划分为以下几个模块。



首先，获取文件大小，计算其中内存块数，以及最后不满的一块中有多少记录。然后，判断总占用内存块数是否比当前拥有内存块数大，若否，进行内排序并直接输出到输出路径中。

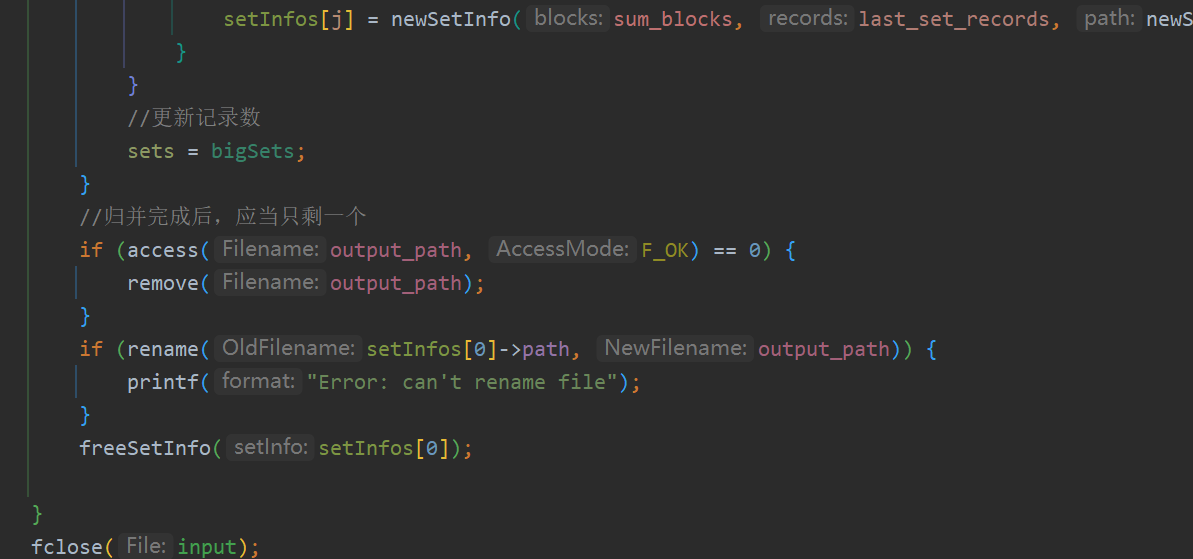


若总占用内存块数是比当前拥有内存块数大，就划分集合，对每个集合内排序，并输出到临时文件中。



完成集合划分后，进行归并排序，我们将归并排序分为若干趟，首先计算这一趟的归并路数，归并路数为当前可用内存块（出去输出缓存外还剩余的内存块数量）和当前集合数量的最小值。

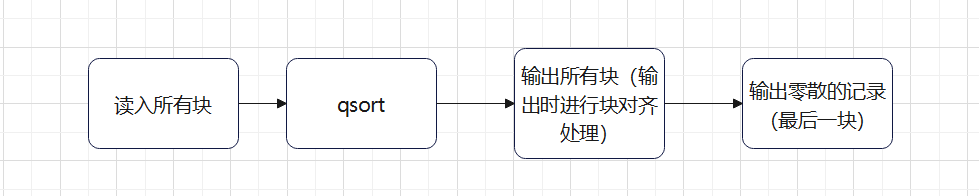
这一趟归并若干次，每一次将k（归并路数）个集合进行归并，并生成新的临时文件。每一趟完成删除上一轮得到临时文件。



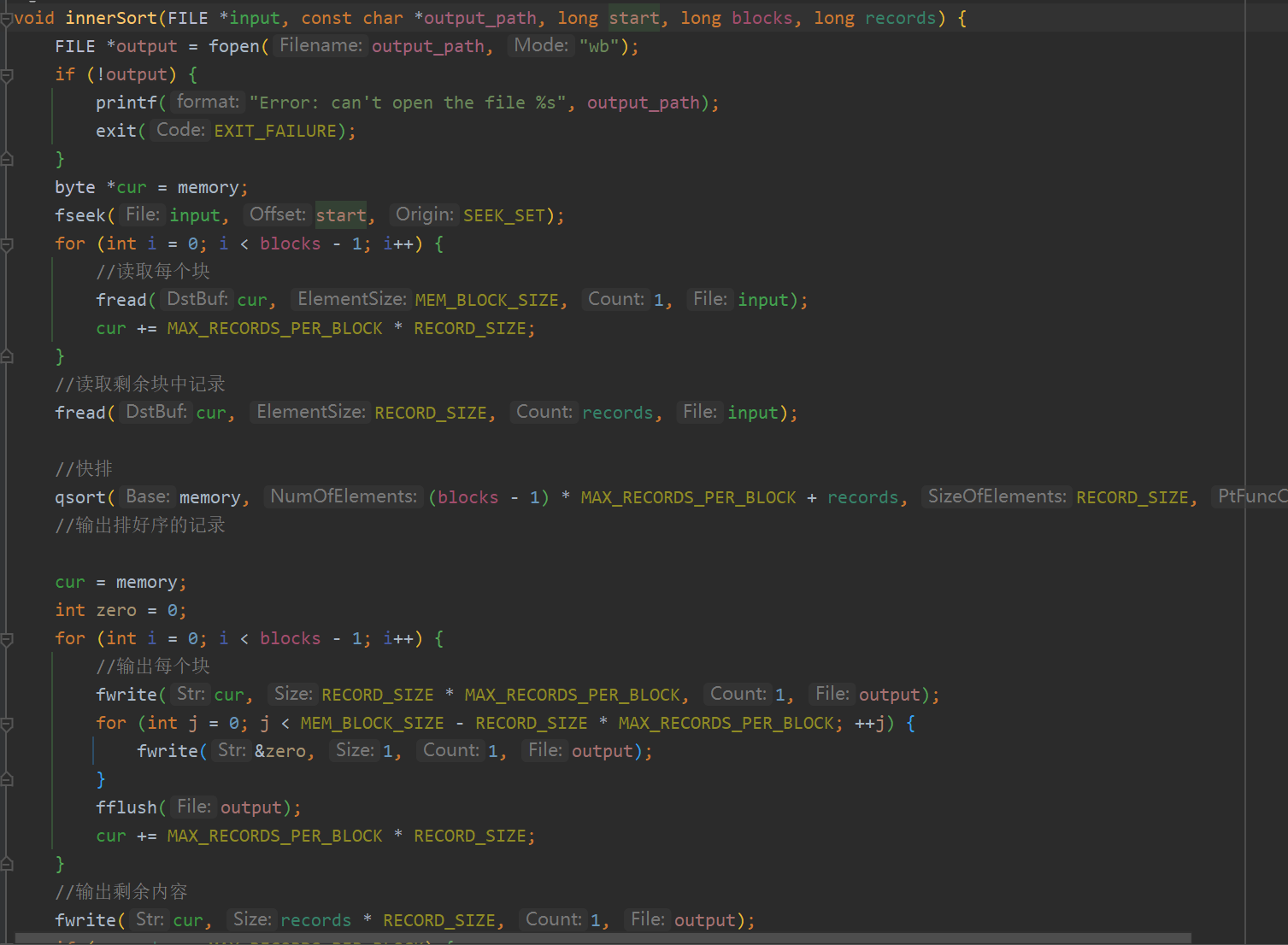
当集合数为1时，此时应该已完成了归并，将最后一个生成的临时文件重命名为需要的输出文件。

**void innerSort(FILE \*input, const char \*output\_path, long start, long blocks, long records)**

这个函数进行内排序，几个参数为，输入文件，输出路径，文件起始位置，待排序块数，最后一块中的记录数。代码流程图如下：



具体代码如下

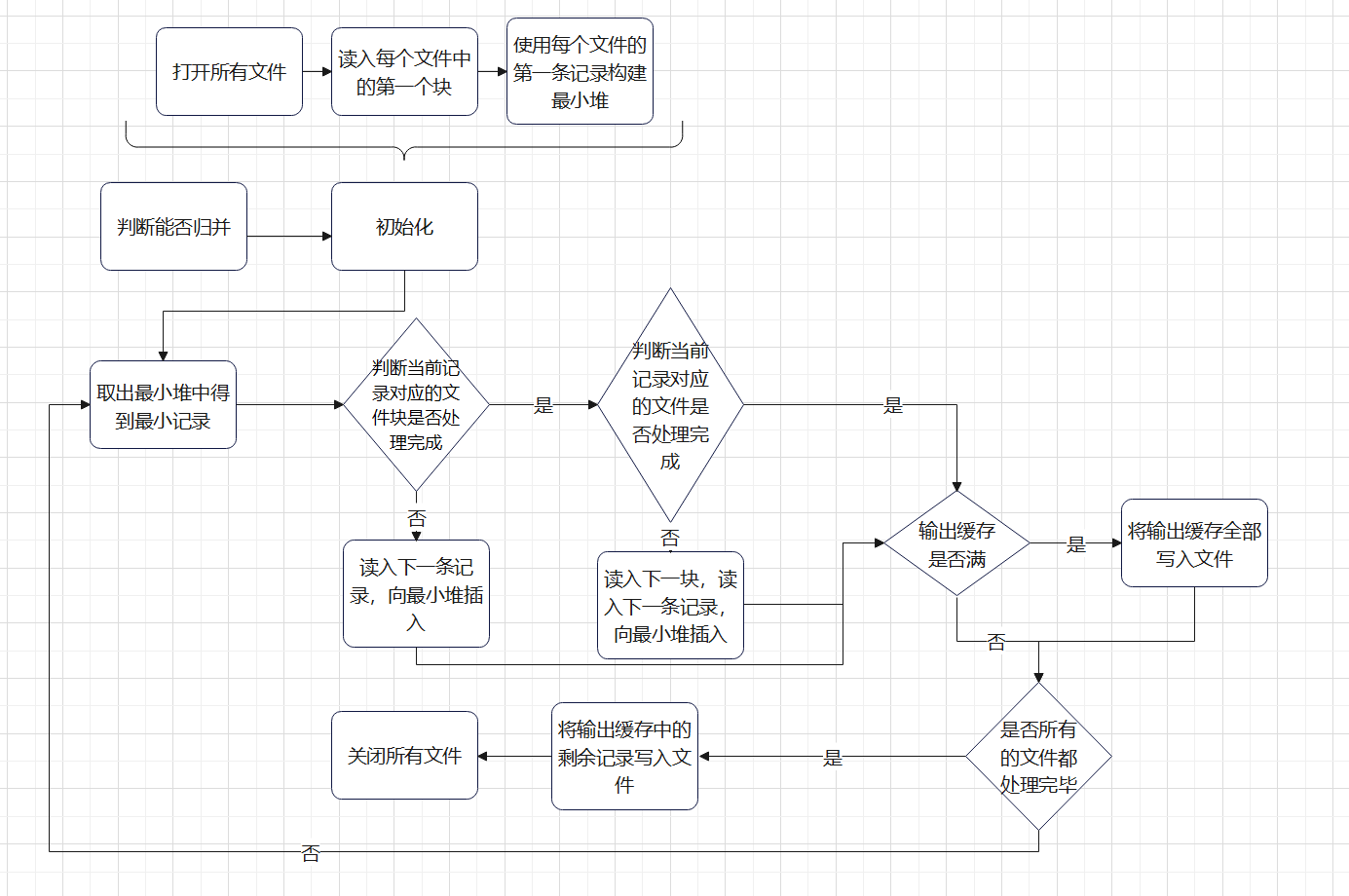


实现思路为，读入所有块（包括最后一块可能不满的块），调用c标准库中的qsort，输出所有块（输出时进行块对齐处理），输出零散的记录（最后一块）。

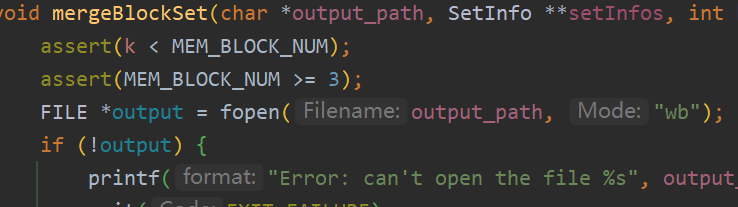
除了上述三个主要函数外，再介绍两个比较重要的函数。

**void mergeBlockSet(char \*output\_path, SetInfo \*\*setInfos, int k)**

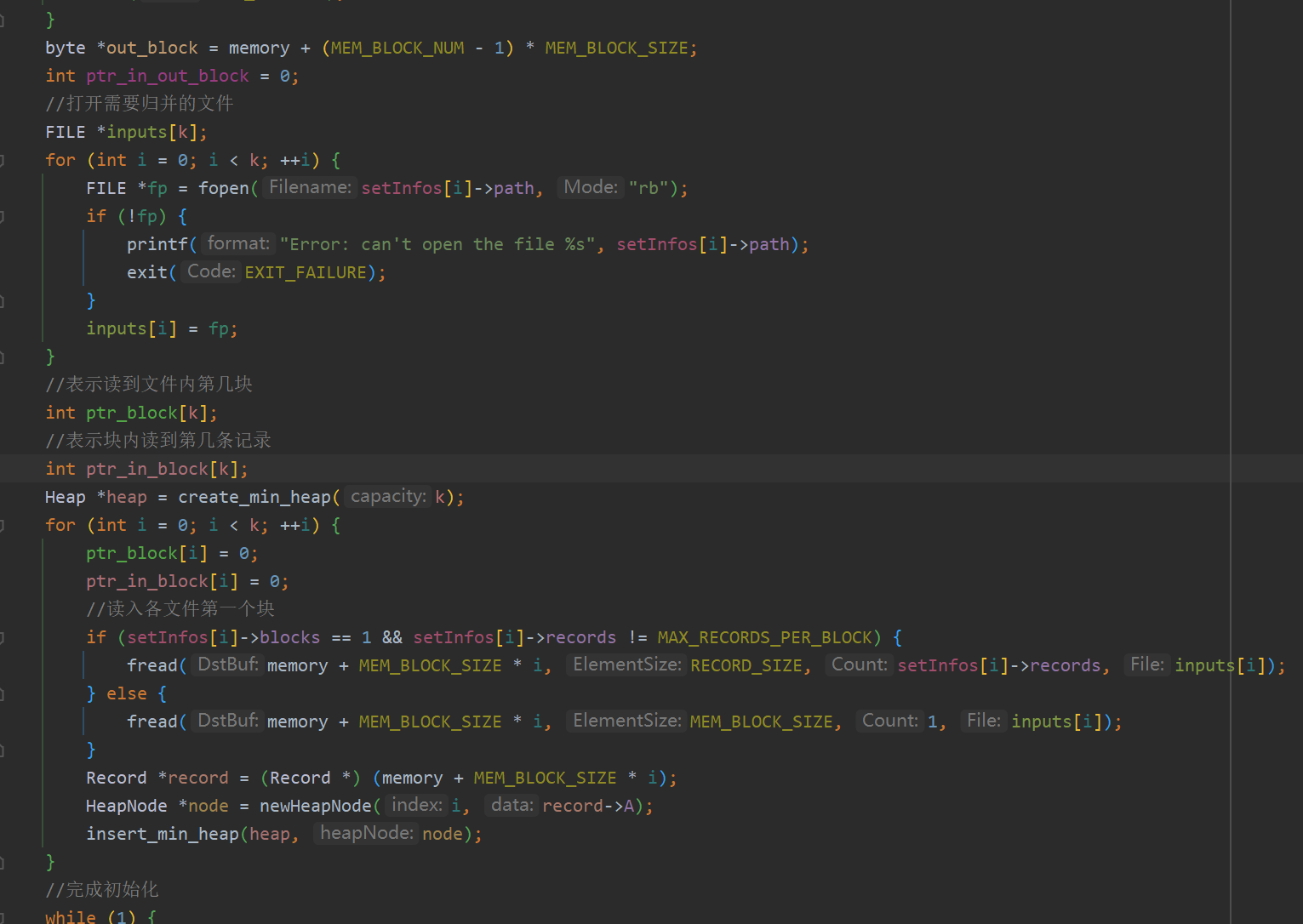
这个函数归并k个集合中的记录，并将排序结果输出到output中，这些集合信息存放在setInfos中。代码流程图如下：



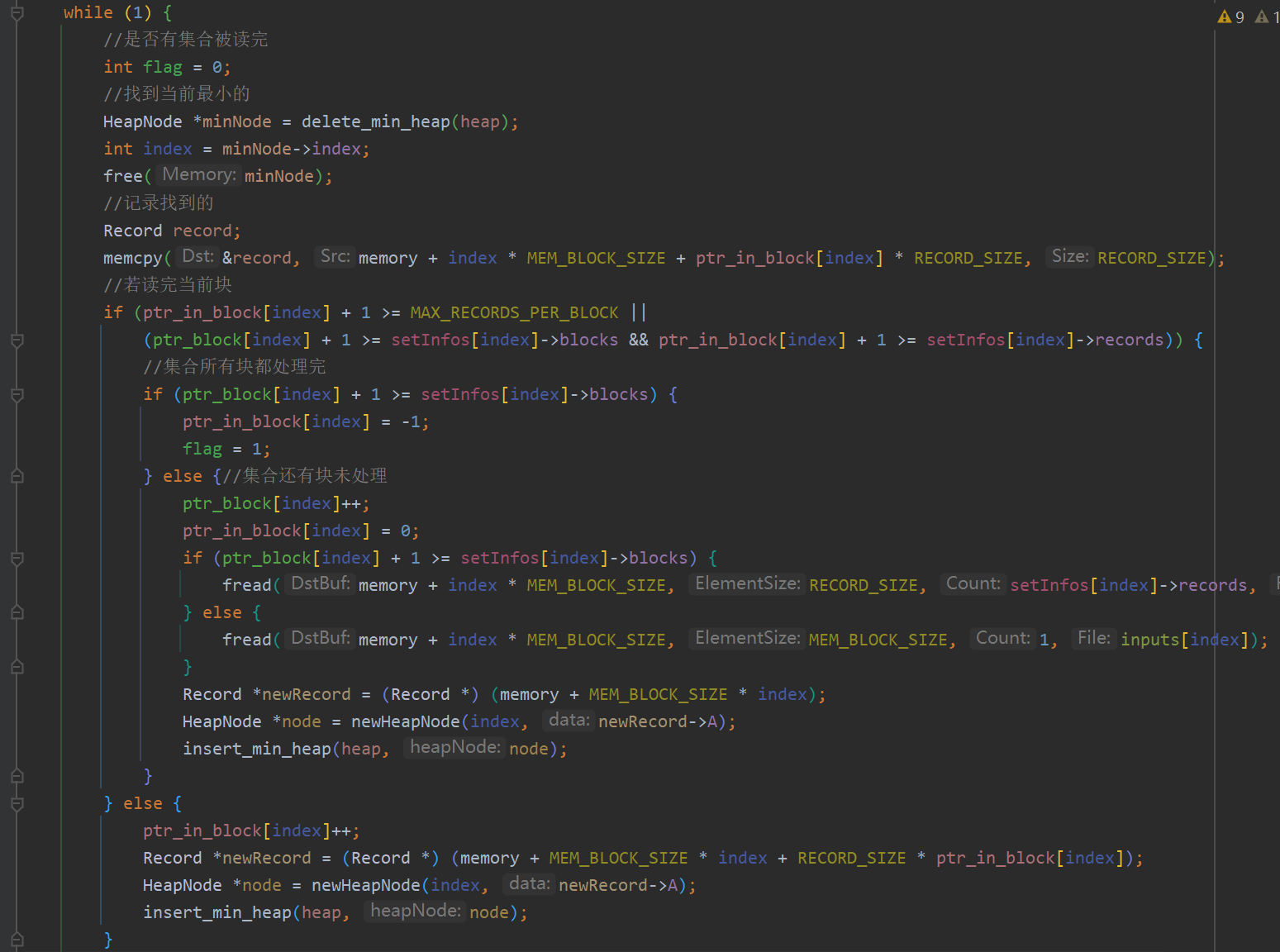
这个函数可以划分为几个模块：



首先，对一些条件进行判读，确保归并路数小于内存块数，因为要保留一个块当作输出缓存，然后内存块数要大于等于3，否则如果内存块数小于等于2的话，减去输出缓存后无法进行归并（只有少于一个可用输入缓存）。



首先是初始化，打开所有文件，读入每个文件中的第一个块，使用每个文件的第一条记录构建最小堆。



取出最小堆中得到最小记录，判断当前记录对应的文件块（已在内存中的）是否处理完成。若是，进一步判断判断当前记录对应的文件是否处理完成，若处理完，则直接进行下一步，若未处理完，读入文件的下一块，将新的记录插入最小堆。若否，生成新纪录并插入最小堆。



判断输出缓存是否满，若是，将输出缓存全部写入文件。判断是否所有的文件都处理完毕，若是，则将输出缓存中的剩余记录写入文件，关闭所有文件。

其余函数，比如对SetInfo创建删除，对最小堆操作的函数，都比较常规，这里不再赘述。