这道题要求的是，写一个`fastsort.c`文件，然后通过给出两个命令行参数来调用它，例如

```
./fastsort inputfile outputfile
```

其中，输入文件是一个由`generate.c`来创建的待排序文件

这个文件可能有若干行（`generate.c -n`参数可以指定输出的行数），每一行有100个字节，前4个字节是一个无符号的整数，表示`key`，后面96个字节是24个待排序的整数`records`

下面要求实现一个排序，对上述文件首先基于`key`做排序，之后对`key`后面的记录做排序，即对`key`后面24个数字做排序，要求排序是升序的、稳定的，然后保存到输出文件中

简单说，就是从上往下每一行的`key`值递增，且保证`key`对应的`records`的列从左到右每一列递增

具体的要求是

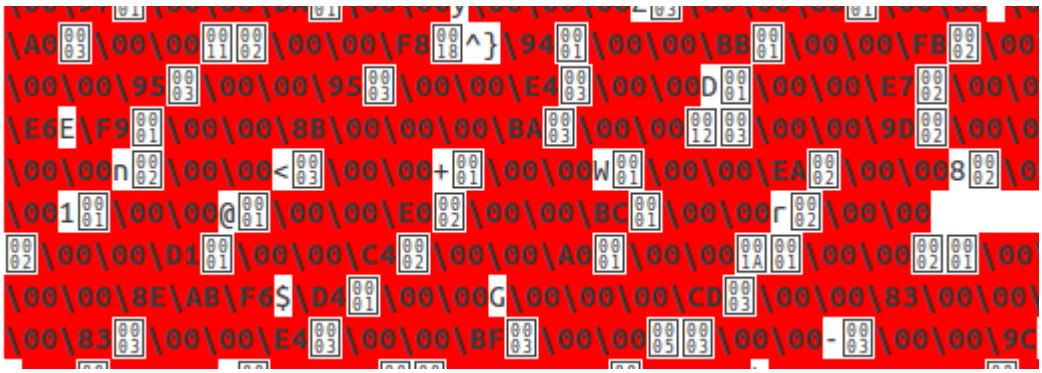
- 必须使用系统调用（`open()`，`read()`，`write()`，`close()`等），而不是标准库函数（`fopen()`，`fclos()`）
- 如果需要知道输入文件的大小，`stat()`或者`fstat()`可以提供帮助
- 排序需要是升序的、稳定的
- 可以使用任何习惯使用的排序，例如可以先尝试使用`qsort()`
- 可以使用`malloc()`，但是要记得`free()`
- 要退出，需要使用`exit()`
- `key`是无符号32位整数
- 不需要实现双通道排序，不需要考虑外部排序，数据能完整导入内存
- 无论出于何种原因的文件失效和非法、以及命令行参数错误，必须给出错误信息，且必须将错误信息输出到标准错误输出，而不是标准输出

另外给出了一些提示，例如

- 从小处着手，例如可以先获得一个简单读取输入文件的程序，一次一行，并打印出它所读取的内容，然后，慢慢添加功能并随时测试它们
- 保留程序旧版本的副本（版本控制）

由于输入文件是二进制文件

gen
10.0 KB



所以期望的输出文件应该也是一个二进制文件

但是二进制文件没有办法排序呀，因此这意味着我需要用二进制模式打开文件

例如在C/C++中常见的

```
freopen("in.txt","rb",stdin); // 使用二进制模式只读打开in.txt并重定向到标准输入
```

但是题目要求不允许使用标准库函数，只能使用系统调用

这反而更加方便了，毕竟这不需要我们自己去做二进制的转化和字节的处理，直接系统调用读进来，然后我们的业务逻辑直接当做是整数来做就可以了

已经在`sort.h`定义了结构体

```
#ifndef __sort_h__
#define __sort_h__

// DO NOT EDIT THIS FILE

#define NUMRECS (24)

typedef struct __rec_t {
    unsigned int key;
    unsigned int record[NUMRECS];
} rec_t;



#endif // __sort_h__
```

直接按照结构体读入，然后直接系统调用`read()`

系统调用会直接读好，省掉了二进制读进来，拼凑出一个整数的过程，也就不需要自己去做字节的拼接了

```
read(INPUT_FILE_NAME, varName, sizeof(rec_t))
```

这样，系统调用会直接读一个`sizeof(rec_t)`大小的东西进来，然后按照要求存放到`varName`中

所以这个时候，变量`varName`已经可以使用了

例如可以

```
printf("%u\n",varName.key);
varName.record[i] += varName.record[j];
```

另外提供了一个工具

`dump`可以把二进制文件中的内容变成可阅读的整数输出

首先看看输出出来的结果

这是用`genrate.c`得到的结果用`dump`输出

```
key: 1804289383 rec:886 777 915 793 335 386 492 649 421 362 27 690 59 763 926 540 426 172 736 211 368 567 429 782
key: 861021530 rec:862 123 67 135 929 802 22 58 69 167 393 456 11 42 229 373 421 919 784 537 198 324 315 370
key: 1129566413 rec:526 91 980 956 873 862 170 996 281 305 925 84 327 336 505 846 729 313 857 124 895 582 545 814
key: 1548233367 rec:434 364 43 750 87 808 276 178 788 584 403 651 754 399 932 60 676 368 739 12 226 586 94 539
key: 1036140795 rec:570 434 378 467 601 97 902 317 492 652 756 301 280 286 441 865 689 444 619 440 729 31 117 97
key: 2007905771 rec:481 675 709 927 567 856 497 353 586 965 306 683 219 624 528 871 732 829 503 19 270 368 708 715
key: 1373226340 rec:149 796 723 618 245 846 451 921 555 379 488 764 228 841 350 193 500 34 764 124 914 987 856 743
key: 387346491 rec:227 365 859 936 432 551 437 228 275 407 474 121 858 395 29 237 235 793 818 428 143 11 928 529
key: 2103318776 rec:404 443 763 613 538 606 840 904 818 128 688 369 917 917 996 324 743 470 183 490 499 772 725 644
```

那么意思就很简单了

我们期望得到的是一个排序后的结果——首先是对`key`进行排序，从上到下每一行都是递增的`key`，然后是对`record`排序，从左到右每一列（除了第一列的`key`）都是递增的`record`

所以仿照`dump.c`的方法，先把数据读入

先不考虑操作非法的错误信息，先把要做的事情做好，再徐徐图之为上

```
while (1) {
    read(inputFile, &r, sizeof(rec_t));
    printf("key: %u rec:", r.key);
    for (int j = 0; j < NUMRECS; j++)
        printf("%u ", r.record[j]);
    printf("\n");
}
```

输出是直接输出到屏幕的，所以很容易可以观察得到结果

下面首先解决`rec`的排序

`rec_t`中的`record`是数组的形式，数组的长度是宏定义的24，也就是$100-4=96$个字节

那么数组的排序可以先试试`qsort()`嘛

```
int cmp(const void * pa, const void * pb){
    int a = *(int*)pa;
    int b = *(int*)pb;
    return a - b;
}
```

```
while (1) {
    read(inputFile, &r, sizeof(rec_t));
    qsort(r.record, NUMRECS, sizeof(r.record[0]), cmp); // qsort排序
    printf("key: %u rec:", r.key);
    for (int j = 0; j < NUMRECS; j++)
        printf("%u ", r.record[j]);
    printf("\n");
}
```

观察到输出结果已经正确了

但是要求排序是稳定的，所以不能是快速排序，快排是不稳定的，所以不能直接调用`qsort`，又因为不能用C++，所以`std::sort()`自然也是不能用的

不过，如果用新版本的G++编译器，那么`qsort()`已经被实现成了稳定的排序，也就是说，对于新版本的编译器来说，其实用`qsort()`也是可以完成稳定排序这一目标的

但是这里，我还是默认快排不稳定

快排不让用，但是又要尽可能快（不能是冒泡排序、选择排序……），所以最好的就是归并排序

归并排序真的是好东西，是稳定的，是$O(n \log n)$，用了分治的思想，效率极高

所以下面用归并排序来完成`record`的排序

归并排序不是重点，就不详细说明了

```
void myMergeSort(int a[], int left, int right) {
    int mid;
    if (left < right) {
        mid = (left + right) / 2;
        myMergeSort(a, left, mid);
        myMergeSort(a, mid + 1, right);
        myMergeArray(a, left, right);
    }
}

void myMergeArray(int a[], int left, int right) {
    int begin1 = left;
    int mid = (left + right) / 2;
    int begin2 = mid + 1;
    int k = 0;
    int newArrayLen = right - left + 1;
    int *b = (int*) malloc(newArrayLen * sizeof(int)); // 用malloc，因为数组长度要运行时才能确定
    while (begin1 <= mid && begin2 <= right) {
        if (a[begin1] <= a[begin2])
            b[k++] = a[begin1++];
        else
            b[k++] = a[begin2++];
    }
    while (begin1 <= mid)
        b[k++] = a[begin1++];
    while (begin2 <= right)
```

```
        b[k++] = a[begin2++];
    copyArray(b, a, newArrayLen, left);
    free(b); // 记得free
}

void copyArray(int source[], int dest[], int len, int left) {
    int j = left;
    for (int i = 0; i < len; i++) {
        dest[j] = source[i];
        j++;
    }
}
```

由于数组的长度是需要在运行是确定的，所以用了`malloc()`，用了动态内存申请的话，一定要记得释放掉

（事实上，数组的长度是24，是固定的，直接开一个`NUMRECS`大小的数组就OK了）

```
while (1) {
    read(inputFile, &r, sizeof(rec_t));
    myMergeSort(r.record, 0, NUMRECS - 1);
    printf("key: %u rec:", r.key);
    for (int j = 0; j < NUMRECS; j++)
        printf("%u ", r.record[j]);
    printf("\n");
}
```

之后考虑`key`的排序

由于`rec`是固定的，所以很显然宏定义一个固定长度的数组，然后对数组进行排序即可，也可以像我一样无需考虑数组大小，直接在归并排序的现场去计算大小然后动态分配

但是，一共有多少行是无从得知的

当然，方法肯定是有的

例如，HINT部分给出了一个方法，那就是查文件的大小

> If you need to know the size of an input file, `stat()` or `fstat()` could help.

然后通过每行100个字节这个已知条件，就可以算出一共有多少行，然后就可以把`key`开在一个数组里面

试想一下，一共多少行是可以被计算出来的，一共多少列（$1\ key + 24\ record$）也是已知的

那后面的做法就千奇百怪了，随便怎么搞都可以

最简单的相当于是一个二维数组了

首先按照行来排序，也就是基于`key`排序，排序以后的每一行的`key`数值递增，然后在每一行内部完成排序，即每一列也递增

二维数组的排序是比较容易实现的

甚至可以每一行的`record`在读入的时候就排序结束，然后直接对二维数组的第一列排序，保持后面的跟着一起动就可以了

或者，开一个一维数组，只保留`key`或者指向`rec_t`的指针

然后一样的，可以使用`qsort()`或者类似的排序，对关键字`key`或者`rec_t -> key`排序

但是这样不太好，我不太喜欢

因为很有可能数组是开不下的

这相当于先要开一个$ROW \times 25$的空间把数据读入，然后$ROW \times 25$的二维数组保存需要排序的数据，如果是归并排序的话，还要一个$ROW \times 25$的辅助空间

当然解决办法总归是比问题多的，例如可以在线处理，读一个处理一个，那么，读入的空间只需要25，而不使用归并排序，使用其他的省空间的排序，就可以把辅助空间省下来

所以总的来说，问题还是能够解决的

至于如何处理排序的时间和空间复杂度，不在我的考虑范围之内

但是我不选择这种方法

一个是因为计算文件大小本来就会导致奇奇怪怪的问题

既然数据进来了，就尽可能不要再去统计文件大小啊这种事情

另一个，统计大小，也就是意味着数组的大小是在运行时确定的

对于Java来说，数组的长度是动态的这没有关系，但是对C来说，数组的长度最好是在编译的时候确定，而不是等到传入文件了再动态计算数组的大小

当然也可以使用`malloc()`来做

不过呢，反正都已经这样了，不如就做成链表好了

链表包含一个键值`val`（也就是每一行的`key`），以及一个指向`rec_t`的指针，另外有一个`next`指向链表的下一个结点

```
typedef struct listnode {
    rec_t* rec;
    int val;
    struct listnode* next;
} ListNode;
```

当然，既然用链表了，而且读入的时候也确实是一行一行读入的

所以可以考虑一种在线处理的思想，直接读一个进来，然后调用归并排序完成`rec`数组的排序

然后直接把新读入进来的结点（已经完成读入，且`rec`已经排序）直接挂在属于他自己的位置上

最简单的方法就是遍历链表，找到第一个大于自己的`key`，然后，修改链表的指针

```
p->next = cur->next
cur->next = p
```

这是插入排序了，相当于是$O(n^2)$的复杂度了

那么链表排序有没有可能实现$O(n \log n)$的复杂度呢

答案当然是有的，例如可以使用快慢指针，主要是用到了快慢指针来找到中间点，快指针每次走2个`next`，慢指针每次走1个`next`，那么当快指针走到链表结束的时候，慢指针就处于链表中间的位置

所以其实也是一种分治和归并的思想

下面给出链表排序的代码

```
ListNode* listSort(ListNode* head) {
    if (head == NULL || head->next == NULL)
        return head;

    // 快指针每次走2个next，慢指针每次走1个next
    ListNode* fast = head;
    ListNode* slow = head;

    //通过快慢指针法寻找中间结点
    while (fast->next != NULL && fast->next->next != NULL) {
        fast = fast->next;
        fast = fast->next;
        slow = slow->next;
    }

    ListNode* mid = slow->next;
    slow->next = NULL; //把链表拆分开为两个链表

    ListNode* list1 = listSort(head);
    ListNode* list2 = listSort(mid);

    ListNode* sorted = listMerge(list1, list2);
    return sorted;
}

ListNode* listMerge(ListNode* list1, ListNode* list2) {
    if (list1 == NULL)
        return list2;
    if (list2 == NULL)
        return list1;

    ListNode* head;
    ListNode* tmp;

    if (list1->val <= list2->val) {
        head = list1;
        list1 = list1->next;
    } else {
        head = list2;
        list2 = list2->next;
    }

    tmp = head;

    while (list1 && list2) {
        if (list1->val <= list2->val) {
            tmp->next = list1;
            tmp = tmp->next;
            list1 = list1->next;
```

```
        } else {
            tmp->next = list2;
            tmp = tmp->next;
            list2 = list2->next;
        }
    }

    if (list1 == NULL) {
        tmp->next = list2;
    }
    if (list2 == NULL) {
        tmp->next = list1;
    }

    return head;
}
```

好的，接下来就可以在主函数里面创建链表了

首先读入的时候要`malloc()`了，不能后一次读就覆盖掉前一次读入的内容，要全部留下了

```
rec_t *r = (rec_t*) malloc(sizeof(rec_t));
```

然后系统调用，读入一个结构体 把结构体的地址和`key`值保留到一个链表结点中（`malloc`一个链表结点，设置`rec`和`val`，并设置`next`为`NULL`）

```
read(inputFile, r, sizeof(rec_t));
ListNode* node = (ListNode*) malloc(sizeof(ListNode));
node->rec = r;
node->val = r->key;
node->next = NULL;
```

然后就是链表的操作，注意头指针不要丢了

```
if (head == NULL) {
    head = node;
    cursor = head;
} else {
    cursor->next = node;
    cursor = cursor->next;
}
```

全部做完以后，当场就可以对`record`数组排序，调用`myMergeSort()`

这些都是在一个大的`while`循环里面，循环退出的条件是读到文件结束，这个可以在`read()`系统调用的返回值出现`EOF`的时候直接`break`出来

出了`while`循环，关掉读入文件，然后调用`listSort()`基于`key`关键字排序

注意排序是会返回一个头结点指针的

因为有可能排序以后是后半部分的第一个跑到了头结点

```
head = listSort(head);
```

之后遍历链表，输出

```
ListNode* p = head;
while (p != NULL) {
    printf("key: %u rec: ", p->rec->key); // 输出当前结点的rec的key
    rec_t r = *(p->rec);
    for (int j = 0; j < NUMRECS; j++) // 遍历输出record数组
        printf("%u ", r.record[j]);
    printf("\n");
    p = p->next; // 遍历链表
}
```

```
key:34740865 rec:36 40 63 72 172 196 211 234 270 283 321 342 347 379 398 427 518 532 560 770 830 950 955 984
key:38649718 rec:29 30 62 125 141 153 183 216 305 377 393 418 472 474 485 487 496 573 726 797 933 948 957 980
key:61101360 rec:8 71 97 126 225 269 310 337 350 437 570 582 618 621 643 657 722 783 815 827 853 888 928 955
key:75245562 rec:45 89 205 361 378 403 443 447 522 528 557 579 605 610 700 704 787 816 821 829 882 892 911 960
key:78417603 rec:13 174 216 267 279 292 294 295 330 339 344 519 542 596 635 650 661 757 825 832 860 876 953 990
key:96617457 rec:25 43 52 132 140 147 220 255 260 292 335 339 349 422 523 529 629 653 690 766 936 949 957 968
key:116423768 rec:10 18 66 98 98 98 142 179 187 320 394 425 438 463 472 486 505 639 643 680 695 752 978 988
key:180785147 rec:81 122 131 179 181 181 223 231 254 281 350 433 487 494 532 590 603 720 787 813 850 857 875 982
key:184794536 rec:8 37 96 127 136 212 260 277 316 367 435 486 567 570 693 697 719 767 839 906 946 966 986 999
key:198619204 rec:127 169 170 205 221 223 227 267 287 292 382 520 540 563 616 660 721 728 798 826 878 883 983 997
key:203845520 rec:34 87 94 142 157 308 320 422 492 492 522 538 545 554 602 644 685 708 735 780 784 802 925 930
key:207026272 rec:84 122 154 171 218 269 308 334 335 365 457 653 701 703 720 728 747 776 797 806 821 907 933 959
key:209359415 rec:32 57 134 154 181 296 297 297 309 404 415 483 506 551 587 595 708 722 776 825 892 962 977 999
key:221713886 rec:39 109 152 155 183 198 264 289 295 313 355 393 535 548 621 631 673 735 790 793 833 877 949 969
key:241909610 rec:59 189 205 217 237 260 274 284 338 355 417 433 518 566 606 620 641 770 849 856 888 945 969 987
key:255789528 rec:36 46 133 189 248 249 303 333 368 498 500 529 567 648 746 753 754 788 797 808 872 890 908 958
key:260401255 rec:81 84 127 150 224 228 269 292 396 422 599 630 658 667 705 721 811 819 904 920 939 940 972 984
```

看着一切正常

记得free

```
p = head;
while (p != NULL){
    p = p->next;
    free(head);
    head = p;
}
```

现在解决一下把数据输出到二进制文件这件事

因为最后是保持了`rec_t`这个结构没有变化，而`rec_t`的`record`已经是排序的

所以直接遍历链表，对链表的每一个结点，取出`rec_t`的地址，然后系统调用，直接写入

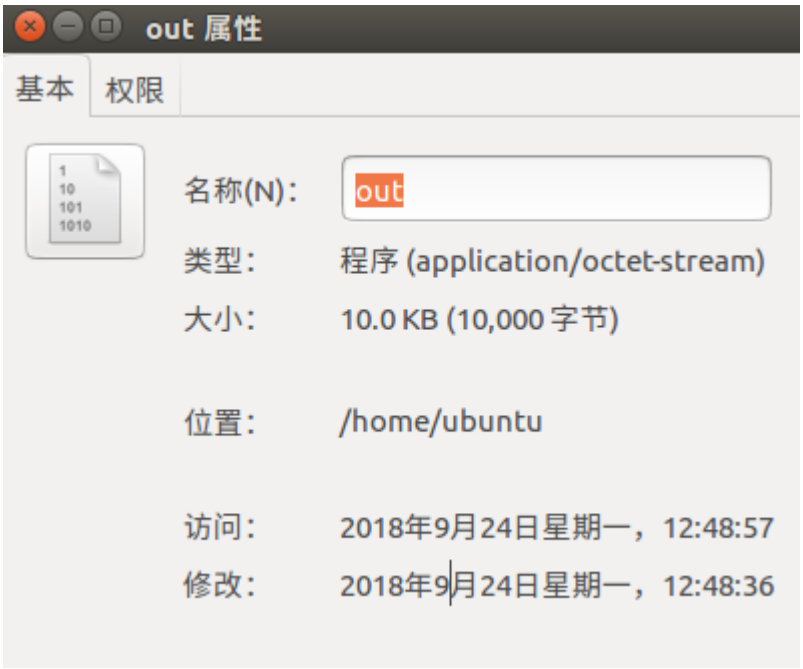
直接写入就是二进制的，不需要手动转化什么的

```
write(outputFile, &r, sizeof(rec_t));
```

选择的参数是

```
O_WRONLY|O_CREAT|O_TRUNC, S_IRWXU
// O_WRONLY 只写打开
// O_CREAT 若此文件不存在，则创建它
// O_TRUNC 如果此文件存在，而且为只读或读写成功打开，则将其长度截短为0
// S_IRWXU: 00700 拥有者读写执行权限
```

这时候就输出了一个二进制文件



尝试`dump`，能够输出结果，说明输出部分实现正确，且输出结果符合预期，说明程序实现成功

```
key: 34740865 rec:36 40 63 72 172 196 211 234 270 283 321 342 347 379 398 427 518 532 560 770 830 950 955 984
key: 38649718 rec:29 30 62 125 141 153 183 216 305 377 393 418 472 474 485 487 496 573 726 797 933 948 957 980
key: 61101360 rec:8 71 97 126 225 269 310 337 350 437 570 582 618 621 643 657 722 783 815 827 853 888 928 955
key: 75245562 rec:45 89 205 361 378 403 443 447 522 528 557 579 605 610 700 704 787 816 821 829 882 892 911 960
key: 78417603 rec:13 174 216 267 279 292 294 295 330 339 344 519 542 596 635 650 661 757 825 832 860 876 953 990
key: 96617457 rec:25 43 52 132 140 147 220 255 260 292 335 339 349 422 523 529 629 653 690 766 936 949 957 968
key: 116423768 rec:10 18 66 98 98 98 142 179 187 320 394 425 438 463 472 486 505 639 643 680 695 752 978 988
key: 180785147 rec:81 122 131 179 181 181 223 231 254 281 350 433 487 494 532 590 603 720 787 813 850 857 875 982
key: 184794536 rec:8 37 96 127 136 212 260 277 316 367 435 486 567 570 693 697 719 767 839 906 946 966 986 999
key: 198619204 rec:127 169 170 205 221 223 227 267 287 292 382 520 540 563 616 660 721 728 798 826 878 883 983 997
key: 203845520 rec:34 87 94 142 157 308 320 422 492 492 522 538 545 554 602 644 685 708 735 780 784 802 925 930
key: 207026272 rec:84 122 154 171 218 269 308 334 335 365 457 653 701 703 720 728 747 776 797 806 821 907 933 959
```

最后做一些细节上的优化

- 记得输出错误信息
  - 对于文件权限问题，输出`Error: Cannot open file foo\n`
  - 对于命令行参数问题，输出`Usage: fastsort inputfile outputfile`
  - 输出都是输出到标准错误输出
- 改成命令行参数传入文件名
- 对文件的操作必须做检查，例如通过系统调用的返回值
- 文件处理完成必须关掉文件
- 加入`exit()`

最后完整的程序如下

```
#include <stdio.h>
```

```c
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <assert.h>
#include <ctype.h>
#include <string.h>
#include "sort.h"

void myMergeArray(int[], int, int);
void myMergeSort(int[], int, int);
void copyArray(int[], int[], int, int);

typedef struct listnode {
    rec_t* rec;
    int val;
    struct listnode* next;
} ListNode;

ListNode* listSort(ListNode*);
ListNode* listMerge(ListNode*, ListNode*);

int main(int argc, const char* argv[]) {

    if (argc != 3){
        fprintf(stderr, "Usage: fastsort inputfile outputfile\n");
        exit(1); // 非0意味着非正常退出
    }

    const char* inFile = argv[1]; // 输入文件
    const char* outFile = argv[2]; // 输出文件

    // 打开输入文件，权限只读
    int fin = open(inFile, O_RDONLY);
    if (fin < 0) {
        fprintf(stderr, "Error: Cannot open file %s\n", inFile);
        exit(2);
    }
    // 打开输出文件，权限只写，文件不存在则创建并置权限IRWXU
    int fout = open(outFile, O_WRONLY | O_CREAT | O_TRUNC, S_IRWXU);
    if (fout < 0){
        fprintf(stderr, "Error: Cannot open (or create) file %s\n", outFile);
        exit(3);
    }

    ListNode* head = NULL; // 链表头
    ListNode* cursor; // 链表指针

    while (1) {
        // 开始读入
        rec_t* r = (rec_t*)malloc(sizeof(rec_t)); // 分配一个rec_t的地址
        int readValidate = read(fin, r, sizeof(rec_t));
        if (readValidate == 0)
            break; // EOF，读到文件结束，break
```

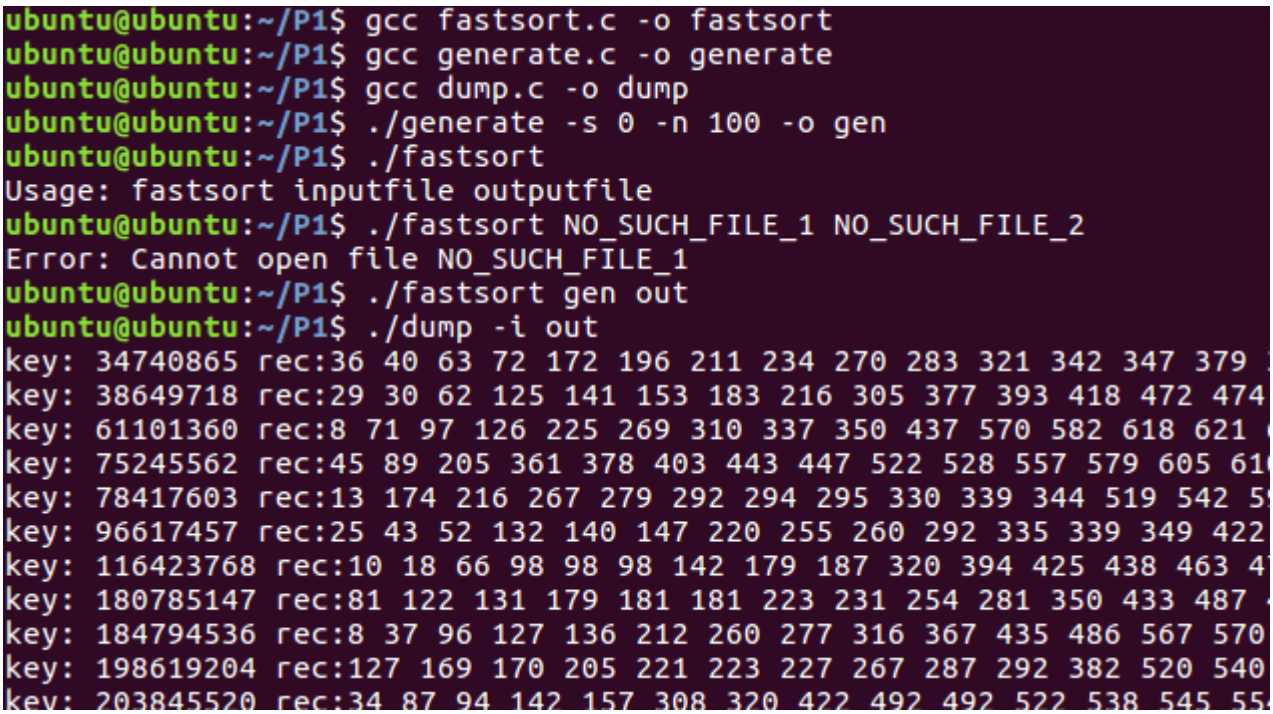
```
        if (readValidate < 0) {
            fprintf(stderr, "Error: Cannot read file %s\n", inFile);
            exit(4);
        }

        // 读入完成，下面创建链表结点
        ListNode* node = (ListNode*) malloc(sizeof(ListNode));
        node->rec = r;
        node->val = r->key;
        node->next = NULL;

        // 设置头指针
        if (head == NULL) {
            head = node;
            cursor = head;
        } else {
            cursor->next = node;
            cursor = cursor->next;
        }
        // 先对record排序
        myMergeSort(r->record, 0, NUMRECS - 1);
    }
    (void) close(fin); // 关闭读入文件

    // 基于key排序
    head = listSort(head);

    // 遍历，输出
    ListNode* p = head;
    while (p != NULL) {
#ifdef DEBUG_USE_ONLY
        // 调试时候输出到标准输出，以便观察
        printf("key: %u rec: ", p->rec->key);
        rec_t r = *(p->rec);
        for (int j = 0; j < NUMRECS; j++)
            printf("%u ", r.record[j]);
        printf("\n");
#endif
        rec_t r = *(p->rec);
        int writeValidate = write(fout, &r, sizeof(rec_t));
        if (writeValidate < 0){
            fprintf(stderr, "Error: Cannot write file %s\n", outFile); // 写文件异常
            exit(5);
        }
        p = p->next; // 指针指向下一个
    }

    // 开始释放内存
    p = head;
    while (p != NULL){
        p = p->next;
        free(head);
        head = p;
```

```
    }

    (void) close(fout); // 关闭文件


    return 0; // exit(0);


}


void myMergeSort(int a[], int left, int right) {
    int mid;
    if (left < right) {
        mid = (left + right) / 2;
        myMergeSort(a, left, mid);
        myMergeSort(a, mid + 1, right);
        myMergeArray(a, left, right);
    }
}

void myMergeArray(int a[], int left, int right) {
    int begin1 = left;
    int mid = (left + right) / 2;
    int begin2 = mid + 1;
    int k = 0;
    int newArrayLen = right - left + 1;
    int *b = (int*) malloc(newArrayLen * sizeof(int)); // 用malloc，因为数组长度要运行时才能确定
    while (begin1 <= mid && begin2 <= right) {
        if (a[begin1] <= a[begin2])
            b[k++] = a[begin1++];
        else
            b[k++] = a[begin2++];
    }
    while (begin1 <= mid)
        b[k++] = a[begin1++];
    while (begin2 <= right)
        b[k++] = a[begin2++];
    copyArray(b, a, newArrayLen, left);
    free(b); // 记得free
}

void copyArray(int source[], int dest[], int len, int left) {
    int j = left;
    for (int i = 0; i < len; i++) {
        dest[j] = source[i];
        j++;
    }
}


ListNode* listSort(ListNode* head) {
```

```
    if (head == NULL || head->next == NULL)
        return head;

    // 快指针每次走2个next，慢指针每次走1个next
    ListNode* fast = head;
    ListNode* slow = head;

    //通过快慢指针法寻找中间结点
    while (fast->next != NULL && fast->next->next != NULL) {
        fast = fast->next;
        fast = fast->next;
        slow = slow->next;
    }

    ListNode* mid = slow->next;
    slow->next = NULL; //把链表拆分开为两个链表

    ListNode* list1 = listSort(head);
    ListNode* list2 = listSort(mid);

    ListNode* sorted = listMerge(list1, list2);
    return sorted;
}

ListNode* listMerge(ListNode* list1, ListNode* list2) {
    if (list1 == NULL)
        return list2;
    if (list2 == NULL)
        return list1;

    ListNode* head;
    ListNode* tmp;

    if (list1->val <= list2->val) {
        head = list1;
        list1 = list1->next;
    } else {
        head = list2;
        list2 = list2->next;
    }

    tmp = head;

    while (list1 && list2) {
        if (list1->val <= list2->val) {
            tmp->next = list1;
            tmp = tmp->next;
            list1 = list1->next;
        } else {
            tmp->next = list2;
            tmp = tmp->next;
            list2 = list2->next;
        }
```

```
    }

    if (list1 == NULL) {
        tmp->next = list2;
    }
    if (list2 == NULL) {
        tmp->next = list1;
    }

    return head;
}
```

看一下输出的结果



好棒!