```
    1: /*
    2: Name: Zhenjiang Tian
    3: BlazerID: ztian
    4: Project: Sort listings by host_name and price using qsort and output new files.
    5: Compile: gcc -Wall -Wextra -O2 listing.c -o listing
    6: Run: ./listing_sort listings.csv
    7: */
    8: #include <stdio.h>
    9: #include <stdlib.h>
   10: #include <string.h>
   11:
   12: #define LINESIZE 1024
   13: #define MAX_ROWS 30000
   14: struct listing {
   15:         int id, host_id, minimum_nights, number_of_reviews, calculated_host_listings
_count,availability_365;
   16:         char *host_name, *neighbourhood_group, *neighbourhood, *room_type;
   17:         float latitude, longitude, price;
   18: };
   19:
   20: struct listing getfields(char* line){
   21:         struct listing item;
   22:
   23:         item.id = atoi(strtok(line, ","));
   24:         item.host_id = atoi(strtok(NULL, ","));
   25:         item.host_name = strdup(strtok(NULL, ","));
   26:         item.neighbourhood_group = strdup(strtok(NULL, ","));
   27:         item.neighbourhood = strdup(strtok(NULL, ","));
   28:         item.latitude = atof(strtok(NULL, ","));
   29:         item.longitude = atof(strtok(NULL, ","));
   30:         item.room_type = strdup(strtok(NULL, ","));
   31:         item.price = atof(strtok(NULL, ","));
   32:         item.minimum_nights = atoi(strtok(NULL, ","));
   33:         item.number_of_reviews = atoi(strtok(NULL, ","));
   34:         item.calculated_host_listings_count = atoi(strtok(NULL, ","));
   35:         item.availability_365 = atoi(strtok(NULL, ","));
   36:
   37:         return item;
   38: }
   39:
   40: static void free_listing(struct listing *p) {
   41:     if (!p) return;
   42:     free(p->host_name);
   43:     free(p->neighbourhood_group);
   44:     free(p->neighbourhood);
   45:     free(p->room_type);
   46: }
   47:
   48:
   49: static int cmp_host_name(const void *a, const void *b) {
   50:     const struct listing *x = *(const struct listing * const *)a;
   51:     const struct listing *y = *(const struct listing * const *)b;
   52:     const char *sx = x->host_name ? x->host_name : "";
   53:     const char *sy = y->host_name ? y->host_name : "";
   54:     return strcmp(sx, sy);
   55: }
   56:
   57: static int cmp_price(const void *a, const void *b) {
   58:     const struct listing *x = *(const struct listing * const *)a;
   59:     const struct listing *y = *(const struct listing * const *)b;
   60:     if (x->price < y->price) return -1;
   61:     if (x->price > y->price) return 1;
   62:     const char *sx = x->host_name ? x->host_name : "";
   63:     const char *sy = y->host_name ? y->host_name : "";
   64:     return strcmp(sx, sy);
   65: }
   66:
   67: static void write_one(FILE *fp, const struct listing *p) {
```

```
   68:     fprintf(fp,
   69:         "%d,%d,%s,%s,%s,%.6f,%.6f,%s,%.2f,%d,%d,%d,%d\n",
   70:         p->id,
   71:         p->host_id,
   72:         p->host_name ? p->host_name : "",
   73:         p->neighbourhood_group ? p->neighbourhood_group : "",
   74:         p->neighbourhood ? p->neighbourhood : "",
   75:         p->latitude,
   76:         p->longitude,
   77:         p->room_type ? p->room_type : "",
   78:         p->price,
   79:         p->minimum_nights,
   80:         p->number_of_reviews,
   81:         p->calculated_host_listings_count,
   82:         p->availability_365
   83:     );
   84: }
   85:
   86: int main(int argc, char *argv[]) {
   87:     if (argc < 2) {
   88:         fprintf(stderr, "Usage: %s listings.csv\n", argv[0]);
   89:         return 1;
   90:     }
   91:
   92:     const char *infile = argv[1];
   93:     FILE *fptr = fopen(infile, "r");
   94:     if (!fptr) {
   95:         perror("fopen");
   96:         return 1;
   97:     }
   98:
   99:     char line[LINESIZE];
  100:         if (fgets(line, sizeof line, fptr) == NULL) {
  101:         fprintf(stderr, "Empty file?\n");
  102:         return 1;
  103:         }
  104:     struct listing *rows = malloc(sizeof(struct listing) * MAX_ROWS);
  105:     if (!rows) {
  106:                 perror("malloc"); fclose(fptr);
  107:                 return 1;
  108:         }
  109:
  110:     int count = 0;
  111:     while (fgets(line, sizeof line, fptr)) {
  112:         if (line[0] == '\n' || line[0] == '\r' || line[0] == '\0') continue;
  113:                 if (count >= MAX_ROWS) {
  114:                 fprintf(stderr, "Too many rows; if could please increasing MAX_ROWS\
n");
  115:                 break;
  116:         }
  117:         rows[count++] = getfields(line);
  118:     }
  119:     fclose(fptr);
  120:
  121:     struct listing **by_name = malloc(sizeof(*by_name) * count);
  122:         struct listing **by_price = malloc(sizeof(*by_price) * count);
  123:
  124:     if (!by_name || !by_price) {
  125:         perror("malloc");
  126:         free(by_name); free(by_price);
  127:         for (int i=0;i<count;i++) {
  128:                         free_listing(&rows[i]);
  129:                 }
  130:         free(rows);
  131:         return 1;
  132:     }
  133:     for (int i = 0; i < count; i++) {
  134:         by_name[i]  = &rows[i];
```

```
  135:         by_price[i] = &rows[i];
  136:         }
  137:
  138:     qsort(by_name,  count, sizeof(*by_name),  cmp_host_name);
  139:         qsort(by_price, count, sizeof(*by_price), cmp_price);
  140:
  141:     FILE *f_name = fopen("sorted_by_host_name.csv", "w");
  142:     FILE *f_price = fopen("sorted_by_price.csv", "w");
  143:
  144:     if (!f_name || !f_price) {
  145:         perror("fopen output");
  146:         if (f_name) {
  147:                         fclose(f_name);
  148:                 }
  149:         if (f_price) {
  150:                         fclose(f_price);
  151:                 }
  152:         for (int i=0;i<count;i++) {
  153:                         free_listing(&rows[i]);
  154:                 }
  155:         free(rows);
  156:                 free(by_name);
  157:                 free(by_price);
  158:         return 1;
  159:     }
  160:
  161:         fputs("id,host_id,host_name,neighbourhood_group,neighbourhood,latitude,longi
tude,room_type,price,minimum_nights,number_of_reviews,calculated_host_listings_count,availa
bility_365\n", f_name);
  162:         fputs("id,host_id,host_name,neighbourhood_group,neighbourhood,latitude,longi
tude,room_type,price,minimum_nights,number_of_reviews,calculated_host_listings_count,availa
bility_365\n", f_price);
  163:
  164:     for (int i = 0; i < count; i++) {
  165:                 write_one(f_name,  by_name[i]);
  166:         }
  167:     for (int i = 0; i < count; i++) {
  168:                 write_one(f_price, by_price[i]);
  169:         }
  170:
  171:     fclose(f_name);
  172:     fclose(f_price);
  173:
  174:     for (int i = 0; i < count; i++) {
  175:                 free_listing(&rows[i]);
  176:         }
  177:         free(rows);
  178:         free(by_name);
  179:         free(by_price);
  180:
  181:     return 0;
  182: }
```