```c
/*
Name: Zhenjiang Tian
BlazerID: ztian
Project: Sort listings by host_name and price using qsort and output new files.
Compile: gcc -Wall -Wextra -O2 listing.c -o listing
Run: ./listing_sort listings.csv
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define LINESIZE 1024
#define MAX_ROWS 30000
struct listing {
        int id, host_id, minimum_nights, number_of_reviews, calculated_host_listings_count,availability_365;
        char *host_name, *neighbourhood_group, *neighbourhood, *room_type;
        float latitude, longitude, price;
};

struct listing getfields(char* line){
        struct listing item;

        item.id = atoi(strtok(line, ","));
        item.host_id = atoi(strtok(NULL, ","));
        item.host_name = strdup(strtok(NULL, ","));
        item.neighbourhood_group = strdup(strtok(NULL, ","));
        item.neighbourhood = strdup(strtok(NULL, ","));
        item.latitude = atof(strtok(NULL, ","));
        item.longitude = atof(strtok(NULL, ","));
        item.room_type = strdup(strtok(NULL, ","));
        item.price = atof(strtok(NULL, ","));
        item.minimum_nights = atoi(strtok(NULL, ","));
        item.number_of_reviews = atoi(strtok(NULL, ","));
        item.calculated_host_listings_count = atoi(strtok(NULL, ","));
        item.availability_365 = atoi(strtok(NULL, ","));

        return item;
}

static void free_listing(struct listing *p) {
    if (!p) return;
    free(p->host_name);
    free(p->neighbourhood_group);
    free(p->neighbourhood);
    free(p->room_type);
}


static int cmp_host_name(const void *a, const void *b) {
    const struct listing *x = *(const struct listing * const *)a;
    const struct listing *y = *(const struct listing * const *)b;
    const char *sx = x->host_name ? x->host_name : "";
    const char *sy = y->host_name ? y->host_name : "";
    return strcmp(sx, sy);
}

static int cmp_price(const void *a, const void *b) {
    const struct listing *x = *(const struct listing * const *)a;
    const struct listing *y = *(const struct listing * const *)b;
    if (x->price < y->price) return -1;
    if (x->price > y->price) return 1;
    const char *sx = x->host_name ? x->host_name : "";
    const char *sy = y->host_name ? y->host_name : "";
    return strcmp(sx, sy);
}

static void write_one(FILE *fp, const struct listing *p) {
    fprintf(fp,
        "%d,%d,%s,%s,%s,%.6f,%.6f,%s,%.2f,%d,%d,%d,%d\n",
        p->id,
        p->host_id,
        p->host_name ? p->host_name : "",
        p->neighbourhood_group ? p->neighbourhood_group : "",
        p->neighbourhood ? p->neighbourhood : "",
        p->latitude,
        p->longitude,
        p->room_type ? p->room_type : "",
        p->price,
        p->minimum_nights,
        p->number_of_reviews,
        p->calculated_host_listings_count,
        p->availability_365
    );
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "Usage: %s listings.csv\n", argv[0]);
        return 1;
    }

    const char *infile = argv[1];
    FILE *fptr = fopen(infile, "r");
    if (!fptr) {
        perror("fopen");
        return 1;
    }

    char line[LINESIZE];
        if (fgets(line, sizeof line, fptr) == NULL) {
        fprintf(stderr, "Empty file?\n");
        return 1;
        }
    struct listing *rows = malloc(sizeof(struct listing) * MAX_ROWS);
    if (!rows) {
                perror("malloc"); fclose(fptr);
                return 1;
        }

    int count = 0;
    while (fgets(line, sizeof line, fptr)) {
        if (line[0] == '\n' || line[0] == '\r' || line[0] == '\0') continue;
                if (count >= MAX_ROWS) {
                fprintf(stderr, "Too many rows; if could please increasing MAX_ROWS\
n");
                break;
        }
        rows[count++] = getfields(line);
    }
    fclose(fptr);

    struct listing **by_name = malloc(sizeof(*by_name) * count);
        struct listing **by_price = malloc(sizeof(*by_price) * count);

    if (!by_name || !by_price) {
        perror("malloc");
        free(by_name); free(by_price);
        for (int i=0;i<count;i++) {
                        free_listing(&rows[i]);
                }
        free(rows);
        return 1;
    }
    for (int i = 0; i < count; i++) {
        by_name[i]  = &rows[i];
        by_price[i] = &rows[i];
        }

    qsort(by_name,  count, sizeof(*by_name),  cmp_host_name);
        qsort(by_price, count, sizeof(*by_price), cmp_price);

    FILE *f_name = fopen("sorted_by_host_name.csv", "w");
    FILE *f_price = fopen("sorted_by_price.csv", "w");

    if (!f_name || !f_price) {
        perror("fopen output");
        if (f_name) {
                        fclose(f_name);
                }
        if (f_price) {
                        fclose(f_price);
                }
        for (int i=0;i<count;i++) {
                        free_listing(&rows[i]);
                }
        free(rows);
                free(by_name);
                free(by_price);
        return 1;
    }

        fputs("id,host_id,host_name,neighbourhood_group,neighbourhood,latitude,longitude,room_type,price,minimum_nights,number_of_reviews,calculated_host_listings_count,availability_365\n", f_name);
        fputs("id,host_id,host_name,neighbourhood_group,neighbourhood,latitude,longitude,room_type,price,minimum_nights,number_of_reviews,calculated_host_listings_count,availability_365\n", f_price);

    for (int i = 0; i < count; i++) {
                write_one(f_name,  by_name[i]);
        }
    for (int i = 0; i < count; i++) {
                write_one(f_price, by_price[i]);
        }

    fclose(f_name);
    fclose(f_price);

    for (int i = 0; i < count; i++) {
                free_listing(&rows[i]);
        }
        free(rows);
        free(by_name);
        free(by_price);

    return 0;
}
```