```c
#include <cs50.h>
#include <stdio.h>
#include <string.h>

// Max voters and candidates
#define MAX_VOTERS 100
#define MAX_CANDIDATES 9

// preferences[i][j] is jth preference for voter i
int preferences[MAX_VOTERS][MAX_CANDIDATES];

// Candidates have name, vote count, eliminated status
typedef struct
{
    string name;
    int votes;
bool eliminated;
}
candidate;

// Array of candidates
candidate candidates[MAX_CANDIDATES];

// Numbers of voters and candidates
int voter_count;
int candidate_count;

// Function prototypes
bool vote(int voter, int rank, string name);
void tabulate(void);
bool print_winner(void);
int find_min(void);
bool is_tie(int min);
void eliminate(int min);

int main(int argc, string argv[])
{
    // Check for invalid usage
    if (argc < 2)
    {
        printf("Usage: runoff [candidate ...]\n");
        return 1;
    }

    // Populate array of candidates
    candidate_count = argc - 1;
    if (candidate_count > MAX_CANDIDATES)
    {
        printf("Maximum number of candidates is %i\n", MAX_CANDIDATES);
        return 2;
    }
    for (int i = 0; i < candidate_count; i++)
    {
        candidates[i].name = argv[i + 1];
        candidates[i].votes = 0;
        candidates[i].eliminated = false;
    }

    voter_count = get_int("Number of voters: ");
    if (voter_count > MAX_VOTERS)
    {
        printf("Maximum number of voters is %i\n", MAX_VOTERS);
        return 3;
    }

    // Keep querying for votes
    for (int i = 0; i < voter_count; i++)
    {

        // Query for each rank
        for (int j = 0; j < candidate_count; j++)
        {
            string name = get_string("Rank %i: ", j + 1);

            // Record vote, unless it's invalid
            if (!vote(i, j, name))
            {
                printf("Invalid vote.\n");
                return 4;
            }
        }

        printf("\n");
    }

    // Keep holding runoffs until winner exists
    while (true)
    {
        // Calculate votes given remaining candidates
        tabulate();

        // Check if election has been won
        bool won = print_winner();
        if (won)
        {
            break;
        }

        // Eliminate last-place candidates
        int min = find_min();
        bool tie = is_tie(min);

        // If tie, everyone wins
        if (tie)
        {
            for (int i = 0; i < candidate_count; i++)
            {
                if (!candidates[i].eliminated)
                {
                    printf("%s\n", candidates[i].name);
                }
            }
            break;
        }

        // Eliminate anyone with minimum number of votes
        eliminate(min);

        // Reset vote counts back to zero
        for (int i = 0; i < candidate_count; i++)
        {
            candidates[i].votes = 0;
        }
    }
    return 0;
}

// Record preference if vote is valid
bool vote(int voter, int rank, string name)
{
    // TODO
    // Find the candidate and store the preference of this candidate
    for (int i = 0; i < candidate_count; i++ )
    {
        if (strcmp(candidates[i].name, name) == 0)
        {
            preferences[voter][rank] = i;
            return true;
        }
    }
    return false;
}

// Tabulate votes for non-eliminated candidates
void tabulate(void)
{
    // TODO
    // First clear votes for each candidate
    for (int i = 0; i < candidate_count ; i++)
    {
        candidates[i].votes = 0;
    }
    // Update votes of each candidate who has not been eliminated
    for (int i = 0; i < voter_count ; i++)
    {
        for (int j = 0; j < candidate_count ; j++ )
        {
            if (candidates[preferences[i][j]].eliminated == 0)
            {
                candidates[preferences[i][j]].votes++;
                break;
            }
        }
    }
    return;
}

// Print the winner of the election, if there is one
bool print_winner(void)
{
    // TODO
    // If any candidate has more than half of the vote,
    // their name should be printed and the function should
    // return true
    for (int i = 0; i < candidate_count; i++)
    {
        if (candidates[i].votes > candidate_count / 2)
        {
            printf("%s\n", candidates[i].name);
            return true;
        }
    }
    return false;
}

// Return the minimum number of votes any remaining candidate has
int find_min(void)
{
    // TODO
    // Return the minimum vote total for any candidate who is still
    // in the election.
    int min = 100000;
    for (int i = 0; i < candidate_count; i++)
    {
        if ((candidates[i].eliminated == 0) && (candidates[i].votes < min))
        {
            min = candidates[i].votes;
        }
    }
    return min;
}

// Return true if the election is tied between all candidates, false otherwise
bool is_tie(int min)
{
    // TODO
    // Return true if every candidate remaining in the election has
    // the same number of votes, and should return false otherwise.
    bool is_all_tie = true;
    for (int i = 0; i < candidate_count; i++)
    {
        if ((candidates[i].eliminated == 0) && (candidates[i].votes != min))
        {
            return false;
        }
    }
    return true;
}

// Eliminate the candidate (or candidates) in last place
void eliminate(int min)
{
    // TODO
    // Eliminate the candidate (or candidates) who have min number of votes.
    for (int i = 1; i < candidate_count; i++)
    {
        if (candidates[i].votes == min)
        {
            candidates[i].eliminated = 1;
        }
    }
    return;
}
```