```c
#include <cs50.h>
#include <stdio.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;
    }
```

```
43
44      // Populate array of candidates
45      candidate_count = argc - 1;
46      if (candidate_count > MAX_CANDIDATES)
47      {
48          printf("Maximum number of candidates is %i\n", MAX_CANDIDATES);
49          return 2;
50      }
51      for (int i = 0; i < candidate_count; i++)
52      {
53          candidates[i].name = argv[i + 1];
54          candidates[i].votes = 0;
55          candidates[i].eliminated = false;
56      }
57
58      /**
59      // Display the candidates array
60      for (int i = 0; i < candidate_count; i++){
61          printf("%s\t", candidates[i].name);
62      }
63      printf("\n");
64
65      for (int i = 0; i < candidate_count; i++)
66      {
67          printf("%i\t", candidates[i].votes);
68      }
69      printf("\n");
70      */
71
72      voter_count = get_int("Number of voters: ");
73      if (voter_count > MAX_VOTERS)
74      {
75          printf("Maximum number of voters is %i\n", MAX_VOTERS);
76          return 3;
77      }
78
79      // Keep querying for votes
80      for (int i = 0; i < voter_count; i++)
81      {
82
83          // Query for each rank
84          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");
    }

    /**
    // Print preferences array to the screen
    printf("Printed 2D array: \n");
    for (int i = 0; i < voter_count; i++)
    {
        for (int j = 0; j < candidate_count; j++)
        {
            printf("%i\t", preferences[i][j]);
        }
        printf("\n");
    }
    */

    /**
    tablulate();
    // Display the candidates array
    for (int i = 0; i < candidate_count; i++){
        printf("%s\t", candidates[i].name);
    }
    printf("\n");

    for (int i = 0; i < candidate_count; i++)
    {
        printf("%i\t", candidates[i].votes);
    }
    printf("\n");
    */

```

```c
    // 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;
}
```

```
169 // Record preference if vote is valid
170 bool vote(int voter, int rank, string name)
171 {
172     // Look through the array candidates for the name
173         // If name is found
174             // Take candidate's index and put it in the 2D preferences array in the voter, rank location
175             // Return success/true
176     // Candidate not found, return false
177     return false;
178 }
179
180 // Tabulate votes for non-eliminated candidates
181 void tabulate(void)
182 {
183     // For each voter
184         // For each candidate
185             // Look at candidate preference
186             // If candidate is not eleminated
187                 // Update the vote count in candidates array by 1
188                 // break
189                 //break;
190
191     // Update votes in condidates array, if not eleminated
192     return;
193 }
194
195 // Print the winner of the election, if there is one
196 bool print_winner(void)
197 {
198     // TODO
199     return false;
200 }
201
202 // Return the minimum number of votes any remaining candidate has
203 int find_min(void)
204 {
205     // TODO
206     return 0;
207 }
208
209 // Return true if the election is tied between all candidates, false otherwise
210 bool is_tie(int min)
```

```
{
    // TODO
    return false;
}

// Eliminate the candidate (or candidates) in last place
void eliminate(int min)
{
    // TODO
    return;
}
```