There’s another kind of voting system known as a ranked-choice voting system. In a ranked-choice system, voters can vote for more than one candidate. Instead of just voting for their top choice, they can rank the candidates in order of preference. Here, each voter, in addition to specifying their first preference candidate, has also indicated their second and third choices.

 In an instant runoff election, voters can rank as many candidates as they wish. If any candidate has a majority (more than 50%) of the first preference votes, that candidate is declared the winner of the election.

If no candidate has more than 50% of the vote, then an “instant runoff” occurrs. The candidate who received the fewest number of votes is eliminated from the election, and anyone who originally chose that candidate as their first preference now has their second preference considered. Why do it this way? Effectively, this simulates what would have happened if the least popular candidate had not been in the election to begin with.

The process repeats: if no candidate has a majority of the votes, the last place candidate is eliminated, and anyone who voted for them will instead vote for their next preference (who hasn’t themselves already been eliminated). Once a candidate has a majority, that candidate is declared the winner.

Let’s open up runoff.c to take a look at what’s already there. We’re defining two constants: MAX\_CANDIDATES for the maximum number of candidates in the election, and MAX\_VOTERS for the maximum number of voters in the election.

Next up is a two-dimensional array preferences. The array preferences[i] will represent all of the preferences for voter number i, and the integer preferences[i][j] here will store the index of the candidate who is the jth preference for voter i.

Next up is a struct called candidate. Every candidate has a string field for their name, and int representing the number of votes they currently have, and a bool value called eliminated that indicates whether the candidate has been eliminated from the election. The array candidates will keep track of all of the candidates in the election.

The program also has two global variables: voter\_count and candidate\_count.

Now onto main. Notice that after determining the number of candidates and the number of voters, the main voting loop begins, giving every voter a chance to vote. As the voter enters their preferences, the vote function is called to keep track of all of the preferences. If at any point, the ballot is deemed to be invalid, the program exits.

Once all of the votes are in, another loop begins: this one’s going to keep looping through the runoff process of checking for a winner and eliminating the last place candidate until there is a winner.

The first call here is to a function called tabulate, which should look at all of the voters’ preferences and compute the current vote totals, by looking at each voter’s top choice candidate who hasn’t yet been eliminated. Next, the print\_winner function should print out the winner if there is one; if there is, the program is over. But otherwise, the program needs to determine the fewest number of votes anyone still in the election received (via a call to find\_min). If it turns out that everyone in the election is tied with the same number of votes (as determined by the is\_tie function), the election is declared a tie; otherwise, the last-place candidate (or candidates) is eliminated from the election via a call to the eliminate function.

If you look a bit further down in the file, you’ll see that these functions — vote, tabulate, print\_winner, find\_min, is\_tie, and eliminate — are all left up to you to complete!

Complete the implementation of runoff.c in such a way that it simulates a runoff election. You should complete the implementations of the vote, tabulate, print\_winner, find\_min, is\_tie, and eliminate functions, and you should not modify anything else in runoff.c (except you may include additional header files, if you’d like).

[**vote**](https://cs50.harvard.edu/x/2020/psets/3/runoff/#vote)

Complete the vote function.

* The function takes arguments voter, rank, and name. If name is a match for the name of a valid candidate, then you should update the global preferences array to indicate that the voter voter has that candidate as their rank preference (where 0 is the first preference, 1 is the second preference, etc.).
* If the preference is successfully recorded, the function should return true; the function should return false otherwise (if, for instance, name is not the name of one of the candidates).
* You may assume that no two candidates will have the same name.

[**tabulate**](https://cs50.harvard.edu/x/2020/psets/3/runoff/#tabulate)

Complete the tabulate function.

* The function should update the number of votes each candidate has at this stage in the runoff.
* Recall that at each stage in the runoff, every voter effectively votes for their top-preferred candidate who has not already been eliminated.

[**print\_winner**](https://cs50.harvard.edu/x/2020/psets/3/runoff/#print_winner)

Complete the print\_winner function.

* If any candidate has more than half of the vote, their name should be printed to stdout and the function should return true.
* If nobody has won the election yet, the function should return false.

[**find\_min**](https://cs50.harvard.edu/x/2020/psets/3/runoff/#find_min)

Complete the find\_min function.

* The function should return the minimum vote total for any candidate who is still in the election.

[**is\_tie**](https://cs50.harvard.edu/x/2020/psets/3/runoff/#is_tie)

Complete the is\_tie function.

* The function takes an argument min, which will be the minimum number of votes that anyone in the election currently has.
* The function should return true if every candidate remaining in the election has the same number of votes, and should return false otherwise.

[**eliminate**](https://cs50.harvard.edu/x/2020/psets/3/runoff/#eliminate)

Complete the eliminate function.

* The function takes an argument min, which will be the minimum number of votes that anyone in the election currently has.
* The function should eliminate the candidate (or candidates) who have min number of votes.