The simplest way to hold an election, though, is via a method commonly known as the “plurality vote” (also known as “first-past-the-post” or “winner take all”). In the plurality vote, every voter gets to vote for one candidate. At the end of the election, whichever candidate has the greatest number of votes is declared the winner of the election.

Let’s now take a look at plurality.c and read through the distribution code that’s been provided to you.

The line #define MAX 9 is some syntax used here to mean that MAX is a constant (equal to 9) that can be used throughout the program. Here, it represents the maximum number of candidates an election can have.

The file then defines a struct called a candidate. Each candidate has two fields: a string called name representing the candidate’s name, and an int called votes representing the number of votes the candidate has. Next, the file defines a global array of candidates, where each element is itself a candidate.

Now, take a look at the main function itself. See if you can find where the program sets a global variable candidate\_count representing the number of candidates in the election, copies command-line arguments into the array candidates, and asks the user to type in the number of voters. Then, the program lets every voter type in a vote (see how?), calling the vote function on each candidate voted for. Finally, main makes a call to the print\_winner function to print out the winner (or winners) of the election.

If you look further down in the file, though, you’ll notice that the vote and print\_winner functions have been left blank. This part is up to you to complete!

Complete the implementation of plurality.c in such a way that the program simulates a plurality vote election.

* Complete the vote function.
  + vote takes a single argument, a string called name, representing the name of the candidate who was voted for.
  + If name matches one of the names of the candidates in the election, then update that candidate’s vote total to account for the new vote. The vote function in this case should return true to indicate a successful ballot.
  + If name does not match the name of any of the candidates in the election, no vote totals should change, and the vote function should return false to indicate an invalid ballot.
  + You may assume that no two candidates will have the same name.
* Complete the print\_winner function.
  + The function should print out the name of the candidate who received the most votes in the election, and then print a newline.
  + It is possible that the election could end in a tie if multiple candidates each have the maximum number of votes. In that case, you should output the names of each of the winning candidates, each on a separate line.

You should not modify anything else in plurality.c other than the implementations of the vote and print\_winner functions (and the inclusion of additional header files, if you’d like)