Plurality

Implement a program that runs a plurality election, per the below.

```
$ ./plurality Alice Bob Charlie
Number of voters: 4
Vote: Alice
Vote: Bob
Vote: Charlie
Vote: Alice
Alice
```

Background

Elections come in all shapes and sizes. In the UK, the Prime Minister is officially appointed by the monarch, who generally chooses the leader of the political party that wins the most seats in the House of Commons. The United States uses a multi-step Electoral College process where citizens vote on how each state should allocate Electors who then elect the President.

Perhaps 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.

Getting Started

Here's how to download this problem's "distribution code" (i.e., starter code) into your own CS50 IDE. Log into CS50 IDE and then, in a terminal window, execute each of the below.

- Execute `cd ~` (or simply `cd` with no arguments) to ensure that you're in your home directory).
- Execute `mkdir pset3` to make (i.e., create) a directory called `pset3`.
- Execute `cd pset3` to change into (i.e., open) that directory.
- Execute `mkdir plurality` to make (i.e., create) a directory called `plurality` in your `pset3` directory.
- Execute `cd plurality` to change into (i.e., open) that directory.
- Execute `wget https://cdn.cs50.net/2020/fall/psets/3/plurality/plurality.c` to download this problem's distribution code.
- Execute `ls`. You should see this problem's distribution code, in a file called `plurality.c`.

Understanding

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!

Specification

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).

Usage

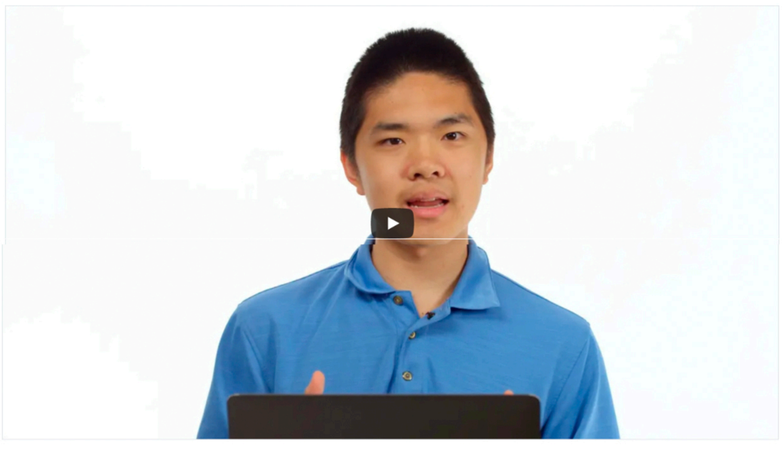
Your program should behave per the examples below.

```
$ ./plurality Alice Bob
Number of voters: 3
Vote: Alice
Vote: Bob
Vote: Alice
Alice
```

```
$ ./plurality Alice Bob
Number of voters: 3
Vote: Alice
Vote: Charlie
Invalid vote.
Vote: Alice
Alice
```

```
$ ./plurality Alice Bob Charlie
Number of voters: 5
Vote: Alice
Vote: Charlie
Vote: Bob
Vote: Bob
Vote: Alice
Alice
Bob
```

Walkthrough



Testing

Be sure to test your code to make sure it handles...

- An election with any number of candidate (up to the `MAX` of `9`)
- Voting for a candidate by name
- Invalid votes for candidates who are not on the ballot
- Printing the winner of the election if there is only one
- Printing the winner of the election if there are multiple winners

Execute the below to evaluate the correctness of your code using `check50`. But be sure to compile and test it yourself as well!

```
check50 cs50/problems/2021/x/plurality
```

Execute the below to evaluate the style of your code using `style50`.

```
style50 plurality.c
```

How to Submit

Execute the below, logging in with your GitHub username and password when prompted. For security, you'll see asterisks (`*`) instead of the actual characters in your password.

```
submit50 cs50/problems/2021/x/plurality
```