Problem #1 (20 points): Calculate Election Results

To elect a president the US uses the electoral college. In this method, individual voters of a state vote for a candidate (we will assume only 2 candidates). For each state, the candidate with the **most individual** (popular) votes will win the **state's electoral** votes. Each state has some number of electoral votes. The winner is the candidate who has the most **electoral votes** (not necessarily total/popular votes).

Your task for this problem is to find the results of an election that follows this process.

Your program should determine the winner and output the following information.

1. The winner of the election (most electoral votes).
2. The total electoral votes for each candidate
3. The totals of the popular vote (raw individual votes) for each candidate.
4. Identify the winning candidate's best state (i.e. in which state they did the best, and what percent of the vote they received for that state)
5. Identify the losing candidate's best state and percentage they received for that state

Note: A few other countries use this technique, not just the US. Thus there could be any number of states…not just 50 or 51 (Washington D.C. is the 51 electoral "state").

You will be given two files: one that describes the electoral votes and one that has popular vote totals for each state.

We have provided a small test input with an election between "Trojan" and "Bruin" so that you can easily debug your code on a small problem. We also provide the data from the 2012 election. However, your code should work on ANY number of states no matter how many or how few. You cannot assume it is just 51 states at most.

- The **electoral college file** gives the number of states, the state names, and how many electoral votes each state has. It will follow the format:

```
Line 1: number of states (N), and how many state/vote pairs will follow
Line 2 to N+1: <State Name> <Electoral Votes>
```

An example for a simple 4 state system is provided below (as `elec_college1.txt`)

- The **voting data file** has the following format. It assumes the same number of states, N, as the electoral file.

```
Line 1: <Candidate 1 Name>
Line 2: <Candidate 2 Name>
Line 3 to N+2: <Candidate 1 votes> <Candidate 2 votes> <State Name>
```

An example is shown below as (`vote_totals1.txt`):

Example Electoral file (elec_college1.txt)	Example Voter File (vote_totals1.txt)

`4`
`BayArea 17`
`Westwood 14`
`SouthernCal 13`
`OtherPlace 12`
	`Trojan`
	`Bruin`
	`450 225 BayArea`
	`300 100 SouthernCal`
	`250 275 OtherPlace`
	`50 325 Westwood`

Here we see that Trojan had more votes in BayArea and SouthernCal, while Bruin had more in OtherPlace and Westwood. Thus Trojan gets the 17+13=30 electoral votes from BayArea and SouthernCal, while Bruin gets the 14+12=26 electoral votes from Westwood and OtherPlace. Thus, Trojan wins the election.

We will provide the filenames at the command line (electoral file first). **The correct output for this set of programs is**:

```
$./election elec_college1.txt vote_totals1.txt
```

```
Trojan defeated Bruin in the electoral college.

The electoral vote count was 30 votes to 26 votes.

The popular vote total was 1050 to 925

Trojan's best state was SouthernCal where they won 75 percent of the votes

Bruin's best state was Westwood where they won 86.6667 percent of the votes
```

Assumptions and Requirements:

- We will provide you with the following struct in the skeleton code:

```
// Stores a state's name and how many
// electoral votes that state has
struct StateElectoralInfo{
  string state;
  int elecVotes;
};
```

- Neither the candidate names, nor the state names will include spaces (all one word). This will help you parse the input files more simply.
- You may assume the file formats will be correct (i.e. we won't put stray characters or error-filled input values). However, you must check that the files we provide on the command actually exist and can be opened. If any file cannot, you should output one of the error messages below (based on which file could not be opened) and exit the program by returning 1.
  - `"Electoral college file not found."`
  - `"Voter totals file not found."`
- It would be best to read in the state names and each state's electoral votes into an array of these structs. That array **must be** dynamically allocated.
- **Tip 1: When reading data from the files, you should NOT need getline(). It can be done more simply.**
- **Tip 2: If you store the electoral college info in an array, then you should NOT need to store the voter totals for each state in an array. Just read in the voter data and process it immediately.**
- The states may/will be in **a different order** in the Electoral College file vs. the voting file. Therefore, you should implement the following function and then call it as needed:
  - **`int getStateElecVotes(StateElectoralInfo state_info[], int len, string stateName)`**
  - You will need (and must use) this function to find the number of electoral college votes of a given state (**`string StateName`**) in the **`state_info[].`** We will test this function without your **`main()`**, so you cannot skip this function by reproducing the functionality inside your **`main()`**.
- There will be **no draws** (i.e. equal votes for both candidates) in either the electoral or popular vote.
- You should delete any dynamically allocated memory before the program quits
- You should use C++ strings.

To summarize, complete the implementations of getStateElecVotes(…) and main(…).

We have provided a second set of test inputs which are data for the 2012 election. The results for that second set of files should be:

```
$ ./election elec_college2.txt vote_totals2.txt
```

```
Obama defeated Romney in the electoral college.

The electoral vote count was 332 votes to 206 votes.

The popular vote total was 66618909 to 61700716

Obama's best state was D.C. where they won 92.5876 percent of the votes

Romney's best state was Utah where they won 74.6262 percent of the votes
```

Submit your code at the following URL: http://bits.usc.edu/cs103/pm-sp17

Problem #2 (up to 4 points credit): `new_cubstr()`

In class and labs we have interacted with several C standard library functions for manipulating C-strings (**`strlen()`**, **`strcpy()`**, etc.). In this problem you will implement the following function:

`char* new_csubstr(const char* str, unsigned int pos, unsigned int len);`

Given a valid input C-string (**`str`**), you should return a pointer to a newly allocated C-string that contains the substring starting at **`str[pos]`** and ending at **`str[pos+len-1]`** (inclusive). However, if the requested length would push past the end of the input string **`str`**, then you should return a new string with contents from **`str[pos]`** to the end of **`str`**. You may assume that pos (the start position) is a valid (in-bounds) index in the str array. You cannot change the contents of the input (**`str`**) array.

A few examples (assume NULL terminated C-strings):

`char input1[] = "Hello world";  char input2[] = "USC";`

`new_csubstr(input1, 3, 4)` would return a pointer to the C-string "lo w" (with NULL termination).

`new_csubstr(input2, 0, 2)` would return a pointer to the C-string "US" (with NULL termination).

`new_csubstr(input2, 1, 8)` would return a pointer to just the C-string "SC" (with NULL termination). Since the length is beyond the end we just return the portion that exists (i.e. "SC").

The skeleton code contains the empty function (that you need to complete) and a **`main()`** with a few test calls to your function. Your implementation **may NOT use** any of the library functions (**`strlen(), strcpy()`**, etc.). Your implementation **must** return a pointer to a newly allocated C-string with proper NULL termination.

Submit your code at the following URL: http://bits.usc.edu/cs103/pm-sp17