Lab 7: Arrays and Pointers
Part 1

**Due date:** Thursday, November 13, 11:59pm.

# Lab Assignment

## Assignment Preparation

**Lab duration.** Because we do not have a class on November 11, this lab will last until November 13 (Thursday). The lab consists of two parts. **Both parts are due November 13**.

**Lab type.** This is an **individual lab**. Each student will submit his/her set of deliverables.

**Collaboration.** Students are allowed to consult their peers[1] in completing the lab. Any other collaboration activities will violate the *non-collaboration* agreement. No direct sharing of code is allowed.

**Purpose.** You will implement a number of programs which use one- and two-dimensional arrays and employ functions with out parameters.

**Programming Style.** All submitted C programs must adhere to the programming style described in detail at

http://users.csc.calpoly.edu/∼cstaley/General/CStyle.htm

When graded, the programs will be checked for style. Any stylistic violations are subject to a 10% penalty. Significant stylistic violations, epsecially those that make grading harder, may yield stricter penalties.

[1] A peer for the purpose of CPE 101 is defined as *"student taking the same section of CPE 101"*.

**Testing and Submissions.** Any submission that does not compile using the

```
gcc -ansi -Wall -Werror -lm
```

compiler settings will receive an automatic score of 0.

**Program Outputs** must co-incide. Any deviation in the output is subject to penalties (e.g., use of different words in the output).

**Please, make sure you test all your programs prior to submission!**

## The Task

**Note:** Please consult the instructor if any of the tasks are unclear.

For this part of the lab, you will write and submit the following programs:

1. `elections.c`: the next installation of the electoral college computation program. The new version will use arrays and functions with out parameters.
2. `sortGame.c`: a simple array sorting game which illustrates the use of arrays as function parameters.

## Program 1: `elections.c/elections.h`

Create a new version of the electoral college evaluation program, incorporating the use of functions with out parameters and arrays.

You will modify two files: `elections.h` and `elections.c` file. As in Homework 3, your `elections.c` file shall consist only of a single `main()` function. All other functions and all defined constants shall be located in `elections.h`.

**Use of arrays.** Your program shall store all state information in arrays. In particular, your `main()` function will declare an `int stateECV[51]` array (remember, it will contain 51 elements indexed from 0 to 50). The elements of this array will contain the Electoral College votes for each individual state, sorted in alphabetical order of the state's name. Thus, `states[0]` will store the EC votes of Alabama, ..., `states[50]` will store the EC votes of Wyoming.

The `states` array needs to be initialized manually. You are allowed to initialize it in the following manner:

```
int stateECV[50] = {AL,AK,AZ,AR, ..., WI WY};
```

(this assumes that `AL, AK, ...WY` are `#define`d as appropriate EC vote constants. Use the names of your constants in place of mine in your code).

In addition, you shall also declare an array variable `int states[51]` in your `main()` function. This array will store state decisions as they are read from the input.

**Input.** The input to your program shall be exactly as in Homework 3.

- The first input parameter is a mode indicator. Value of 1 means verbose mode, Value of 0 means silent mode. All other values are invalid. The mode indicator is followed by a line break.

- Next, the input will consist of 51 lines. Each line will contain information about the election results in a single state. The format of each line is the same:

  ```
  ST x
  ```

  Here `S` and `T` are two **uppercase** characters, jointly representing the US Postal code for a specific state or Washington, DC (e.g., `AL, CA, WV, UT, RI`). `x` is a state result, which is a 0 if the state is won by John McCain and 1 if the sate is won by Barack Obama.

**Modes.** As before, your program shall operate in two modes: silent and verbose. The meaning of each mode is the same as in prior assignments: in verbose mode, the program shall output information about the winner of each state. In silent mode, only the total Electoral College vote tally is printed.

**Structural changes.** Your program will use two functions, defined in `elections.h`:

```
int getStateIndex(char letter1, char letter2)
void printMessage(char letter1, char letter2, int decision)
```

**void printMessage(char letter1, char letter2, int decision)** . This function does not change from Homewok 3.

**int getStateIndex(char letter1, char letter2)** . This function effectively replaces the `getStateECVote()` from Homework 3. Note that now, all EC votes are stored in the `stateECV[]` array. The key problem with obtaining the necessary information from this array is that one needs to know the array index for each state.

`int getStateIndex()` takes as input the two letters defining the US Postal Code for the state (which are read from input), and outputs back the index of the state in the `statesECV[]` array. Remember, that states are indexed in alphabetical order, so, `getStateIndex('A','L')` shall return 0, `getStateIndex('A','K')` shall return 1, and so on; `getStateIndex('W','Y')` shall return 50.

**NOTE:** You are allowed to use "magic numbers" (i.e., integer values not `#define`d in the code) to represent return values in the body of this function. This will NOT be considered a violation of the course code guidelines.

**Main function.** Your `int main()` function shall work as follows. After the variable initializationsm your program shall read in the input: the mode indicator, and then, using a loop, the election results for the 50 states as reported. Unlike previous implementations, rather than computing the electoral vote counts on the fly, your program shall populate the `states[]` array with the state decision information for each state. After the first loop of the program is complete, your program shall use a second loop, to go over the values stored in the `states[]` and `stateECV[]` arrays and compute the total vote and state counts for Barack Obama and John McCain. Once these are computed, your program shall output the total results.

**Output.** Your program shall produce exactly the same output as in Homework 3. To test your program we will use the test cases submitted by you as part of Homework 3.

## Program 2: Sorting Game: sortGame.c

An array sorting game as played as follows. The user is shown a list of numbers. On each step, the user can select two neighboring numbers and make them trade places. The game is over and the user wins when the list of numbers becomes sorted in ascending order.

Your job is to write a program that playes the array sorting game. Name the program `sortGame.c`. The program will use a few user-defined functions, which shall be put into a header file called `sortGame.h`.

Your program shall do the following.

**SG1.** The sort game will be played using a list of 5 numbers. Your program shall use an array to store the list of numbers.

**SG2.** At the beginning of the program, the program reads a list of 5 numbers from the standard input.

**SG3.** On each step of the game, the program displays the current list (see below for the output format), and prompts the user for the action.

**SG4.** A legal move in the game is an exchange of list positions by two consecutive numbers on the list. User input for the move is a pair of numbers, representing the two neighboring positions (e.g., `1 2` or `4 3`).

Consider the list `3 7 5 4 6`. A legal move, for example, is an exchange of values between the second and the third element of the list. This move will lead to the following new list: `3 5 7 4 6`.

Note that in your game implementation, list positions start at 0. Thus, given the list `3 7 5 4 6`, the move `2 3` actually shall result in the new list being `3 7 4 5 6`.

**SG5.** After each move is recorded, your program shall do the following:

1. Exchange the numbers in the positions specified.
2. Check if the new list of numbers is ordered.
3. If it is not, prompt for next move.
4. If the new list of numbers is ordered, proceed to **SG6.**

**SG6.** The game ends, when the list of numbers becomes ordered. In this case, the program shall output the congraulatory message, print out the final list, and specify how many turns it took to complete the task.

**SG7.** Upon completion of the game, the program shall ask if the user would like to play another game. If the user answers affirmatively, the program shall start a new array sort game using **the same array** read from standard input at the beginning of the program.

### Instructions

The `sortGame.h` header file shall contain the following three functions:

- `int isSorted(int list[])`. This function takes as input an array of numbers representing the current state of the game and outputs 1 if the array is sorted in ascending order, and 0 otherwise. Note, that the array is passed as an out parameter, but this function *shall have no side effects*.

- `int trade(int list[], int i, int j)`. This function takes as input an array of numbers representing the current state of the game and two array indexes. It checks if the two indexes (`i` and `j`) are next to each other, and if they are, performs the array contents switch: i.e., `list[i]` is assigned the contents of `list[j]` and vice versa. The function **changes** the contents of `list`, i.e., it produces a side effect. This function also returns a value, which helps calling program determine if any changes were made:

| `int trade()` return value | meaning |
|---|---|
| -2 | one of `i` or `j` is outside the array index range. no change to `list` |
| -1 | `i` and `j` are not adjacent. no change to `list` |
| 1 | `list` is modified by switching the contents of cells `i` and `j` |

- `void printArray(int list[])`. This function takes as input an array of numbers representing the current state of the game and prints it to standard output. The array is printed as two lines. The first line starts with the word `"Index"` and contains the sequence of array indexes. The second line starts with the word `"List"` and contains the array contents. For example, an array `list=` {`2 13 11 3 20`} will be printed by the `printArray()` function as follows:

```
Index: 0   1   2   3   4
List : 2   13  11  3   20
```

  This function may assume that all array contents are going to be integers smaller than 100 (i.e., each array value can take up one or two spaces). The first digit of each array element must be lined up directly under the array index. Your code is responsible for proper formatting.

### Other notes.

- In the instructor's implementation, the `"Play Again?  (Y/N)"` prompt accepts both lower-case and upper-case letters 'Y','y','N' and 'n'.

- Only legal moves count. In the example below, notice that two of the moves attempted are deemed invalid. They are not counted towards the number of moves (the game ends in 4 legal moves).

- As noted above, test cases will use only numbers from 0 to 99 as array element values.

- Instructor's executable **does not** check if the initial array is sorted. For example, if initial input is `1 2 3 4 5`, the game will commence, asking the user to make a move. If the user makes an illegal move, the game ends in 0 moves. If the user makes a legal move, the array will no longer be sorted and the game continues. The second output session illustrates this.
- No test case will include two different array elements with the same value.

### Testing the output

The output of your `sortGame` program will be checked using `diff` against the output of the instructor's program (available, together with the test cases from the course web page).

**Please, make sure that the outputs match exactly!** Failure to do so will result in deductions.

### Sample output

Here is a sample output of the program.

```
$ sortGame
1 3 17 8 11
ASG: Array Sort Game
Index: 0   1   2   3   4
List : 1   3   17  8   11

Your move:>>2 3
Index: 0   1   2   3   4
List : 1   3   8   17  11

Your move:>>4 3
Index: 0   1   2   3   4
List : 1   3   8   11  17

Congratulations! You won in 2 moves

Play again? (Y/N)y
ASG: Array Sort Game
Index: 0   1   2   3   4
List : 1   3   17  8   11

Your move:>>4 3
Index: 0   1   2   3   4
List : 1   3   17  11  8

Your move:>>2 3
Index: 0   1   2   3   4
List : 1   3   11  17  8

Your move:>>3 1
Invalid Move! Try again!
Index: 0   1   2   3   4
List : 1   3   11  17  8

Your move:>>3 4
Index: 0   1   2   3   4
List : 1   3   11  8   17

Your move:>>8 5
Invalid Move! Try again!
Index: 0   1   2   3   4
List : 1   3   11  8   17
```

```
Your move:>>3 2
Index: 0   1   2   3   4
List : 1   3   8   11  17

Congratulations! You won in 4 moves

Play again? (Y/N)N
Bye!

$ sortGame-alex
1 2 3 4 5
ASG: Array Sort Game
Index: 0   1   2   3   4
List : 1   2   3   4   5

Your move:>>7 8
Invalid Move! Try again!
Index: 0   1   2   3   4
List : 1   2   3   4   5

Congratulations! You won in 0 moves

Play again? (Y/N)Y
ASG: Array Sort Game
Index: 0   1   2   3   4
List : 1   2   3   4   5

Your move:>>0 1
Index: 0   1   2   3   4
List : 2   1   3   4   5

Your move:>>1 0
Index: 0   1   2   3   4
List : 1   2   3   4   5

Congratulations! You won in 2 moves

Play again? (Y/N)n
Bye!
```

## Submission.

**Files to submit.** For this part of the lab you shall submit four files: `elections.c`, `elections.h`, `sortGame.c` and `sortGame.h`.

No other files can be submitted. In fact, if you submit *other* file, or submit one of the three files about with an *incorrect filename*, you will receive an email informing you about a submission error, and asking you to resubmit.

- `ssh` to `vogon` (`vogon.csc.calpoly.edu`).
- Students from **Section 009** shall execute the following submission command:

  ```
  > handin dekhtyar-grader lab07-09  <your files go here>
  ```

- Students from **Section 011** shall execute the command:

  ```
  > handin dekhtyar-grader lab07-11 <your files go here>
  ```

`handin` is set to stop accepting submissions 24 hours after the due time.