Lab 3: Part 3. Incremental improvements.

**Due date:** Friday, October 16, 11:59pm.

# Lab Assignment

## Assignment Preparation

**READ THE INSTRUCTIONS FIRST.** Please read this document completely **before** you start your work. The submission instructions and the appendix contain a lot of important information about preparing, testing and submitting your programs. In this assignment, you are expected to test your programs using the facilities provided to you by the instructor. Failure to do so may result in errors in your code, and subsequent severe point deductions.

**Lab type.** Part 1 of the lab is a **pair programming assignment**. You will work in the same pairs as for the Lab 3 Part 1 assignment.

**Collaboration.** Students work in pairs, and it is considered cheating, if members of the team (pair) do not work together. Communication between pairs during lab time is allowed, but no direct sharing of code is allowed.

**Purpose.** The lab allows you to practice the use of conditional statements. It also facilitates learning the use of boolean expressions as conditions in `if` and `switch` statements.

**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. Also note the the Lab 2 requirement for the content of the header comment in each file you submit applies **to each assignment** (lab, programming assignment, homework) in this course.

**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.

For each program you have to write, you will be provided with instructor's executable and with a battery of tests. The programs you submit must pass all tests made available to you. You can check whether or not a program produces correct output by running the instructor's executable on the test case, then running yours, and comparing the outputs.

**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!** Feel free to test your programs on test cases you have invented on your own[1].

## The Task

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

For this lab, you will write and submit three four simple programs. Three programs will be revisions of the programs from Lab 2, and one program will revise a program from Lab 3 part 1.

## Program 1: Exam score computation (newscore.c)

You will modify the exam score computation program `score.c` from Lab 2 to properly handle incorrect inputs.

A **variable guard** is code that runs after a variable has been read from input, that verifies that the variable obtained the value that allows further program execution to proceed.

Your program will institute variable guard on both inputs, and will also provide some extra output at the end of the program. The following new requirements must be implemented.

**NCR1.** Variable guard for test score. The first input to your program (test score achieved) must be non-negative. If a negative score is entered your program shall output the following text:

```
Error: negative test score
```

and stop execution.

**NCR2.** Variable guard for Max test score. The second input to your program (maximum number of points in a test) must be positive, and must be less than or equal to 4000. If a negative max score is entered, your program shall output:

```
Error: max test score not positive
```

[1] In this lab, we provide you with a large collection of test cases. In some future labs, test case development will be a major part of the lab assignment, so it never hurts to start early

If the score entered is too large, your program shall output:

```
Error: max test score out of range
```

In both cases, the program then shall stop execution.

**NCR3.** Letter grade computation. Upon computing the percentage of the grade earned on the test, and upon outputting it, your program shall also compute and output the letter grade earned by the student on the test. The letter grade shall be computed using the following rubric:

| Score range | Letter grade |
|---|---|
| $[90\%, 100\%]$ | A |
| $[80\%, 90\%)$ | B |
| $[70\%, 80\%)$ | C |
| $[60\%, 70\%)$ | D |
| $[0\%, 60\%)$ | F |

Once your program computes the letter grade, it shall output it using the following text:

```
Your grade is <Grade>
```

replacing `<Grade>` with the letter grade computed.

**NCR4.** Extra credit. Your program will earn 30% of extra credit if you use `switch` statement to compute the letter grade.

**NCR5.** File name. Name your program `newscore.c`.

**Sample run.**

```
$ more score-test24
10
0
$ newscore < score-test24
What's the test score? Out of how many points?
Error: max test score not positive
$ more score-test28
4000
4001
$ newscore < score-test28
What's the test score? Out of how many points?
Error: max test score out of range
$ more score-test26
4000
4000
$ newscore < score-test26
What's the test score? Out of how many points?
You have earned 100.000000 percentage points
Your grade is A
```

## Program 2: Temperature Converter (converter.c)

You shall modify your temperature converter program from Lab 2 (`converter.c`) to guard the input and to determine whether the temperatures specified are below/above the freezing and boiling points of water.

The new requirements are outlined below.

**NCR1.** Variable guard. The temperature in degrees Fahrenheit cannot be lower than $-459.67$ — the absolute zero. If the input value is lower than $-459.67$. your program shall output:

```
Error: temperature value below absolute zero
```

and stop program execution.

**NCR2.** Temperature checks. The freezing point of water is 0 degrees Celcius [2]. The boiling point of water at sea level is 100 degrees Celcius[3]. If the temperature you compute is at or below the freezing point, output

```
The water's frozen
```

If the temperature is between the freezing point and the boiling point of water, output

```
The water is liquid
```

If the temperature is equalt to or above the boiling point of water, output

```
Vapor! Vapour!
```

(footnote: note the spelling of both words.)

**NCR3.** File name. Keep the same file name for the program: `converter.c`.

**Sample Run.** Here is a sample output for the example discussed above.

```
 $ more converter-test33
31.9
$ converter < converter-test33
Temperature Conversion: Farenheight to Celcius

Enter temperature in degrees Fahrengeit:
The temperature in degrees Celcius: -0.055556
The water's frozen
$ more converter-test40
459.68

$ converter<converter-test40
```

[2]It is actually just above 0, but for our purposes, 0 is fine.

[3]Unlike the freezing point, the boiling point is altitude-dependent; water will boil at lower temperatures at higher altitudes due to lower atmospheric pressure.

```
Temperature Conversion: Farenheight to Celcius

Enter temperature in degrees Fahrengeit:
The temperature in degrees Celcius: 237.599991
Vapor! Vapour!
$ more converter-test35
-459.67
$ converter<converter-test35
Temperature Conversion: Farenheight to Celcius

Enter temperature in degrees Fahrengeit:
Error: temperature value below absolute zero
```

## Program 3: Simple Electoral College (simple-elections.c)

This part of Lab 3 asks you to produce a new version of the electoral college program. While time is still not right to simplify the program and make it shorter and less repetetive, you **can now** make the program a bit more **informative**.

The new functionality of your program is described below.

**EC0.** Name your program `simple-elections.c`.

**EC1.** Your program will now run in one of two "modes", verbose or silent. Informally, in verbose mode, your program will output election results for each state, as well as the overall election results. In silent mode, your program will behave similarly to its behavior in Lab 2: only the overall election results will be reported.

**EC2.** The input to your program now shall be a **sequence of 52 zeroes or ones**. The first number in the sequence is the **mode indicator**, while the remaining 51 numbers remain as before the results of elections in each of the states and DC provided in the alphabetical order of the state name.

**EC3.** If the **mode indicator** (the first input of the program) is equal to **0** then the program shall run in the silent mode.

If the **mode indicator** is equal to **1**, the program shall run in the verbose mode.

**EC4.** Your program shall start by initializing `votesMcCain` and `votesObama` as prescribed by requirement **ECF3** (see Lab 2). After that, your program shall read the **mode indicator**[4].

**EC5.** After that, your program shall proceed as before (see **ECF4.** from Lab 2), with the following changes to its behavior after each `stateDecision` is read:

**EC5-1.** Your program shall check if the **mode indicator** points to verbose or silent mode of running.

[4]Please note, that **mode indicator** is a new piece of data your program will work with, therefore, a new variable to store it shall be declared. In the future, it is left up to you to understand when new variables need to be declared in similar situations.

**EC5-2.** If the **mode indicator** points to silent mode, your program proceeds as before (see **ECF4.** from Lab 2).

**EC5-3.** If the **mode indicator** is set to verbose mode, your program shall perform the following actions:

- Check if the current state had been won by McCain or Obama.
- If Obama has carried the state[5], print

  ```
  <State> goes to Obama
  ```

  where `<State>` is the name of the current state (`Alabama, Alaska, Arizona`, etc. . . ).
- if McCain has carried the state, print

  ```
  <State> goes to McCain
  ```

After outputting the text as above, your program shall proceed reading the decision of the next state as prescribed by **ECF4** of Lab 2.

**EC6.** After the information about the decisions of all states is read from standard input, your program shall compute and output the electoral college votes for Obama and McCain as prescribed by requirement **ECF5, ECF6** of Lab 2. After that, your program shall determine the winner of the electoral college.

Remember, that the winner of the elcoral college is the candidate who won the majority of the electoral college vote. At present, the Electoral College admits three possible results:

- Barack Obama gains 270 or more elecotral college votes and wins the Electoral College outright (and gains more Electoral College votes than John McCain). In this case, your program shall output

  ```
  The next President of the United States is Barack Obama.
  ```

- John McCain gains 270 or more electoral college votes and wins the Electoral College outright (and gains more Electoral College votes than Barack Obama). In this case, you program shall output.

  ```
  The next President of the United States is John McCain.
  ```

- Both John McCain and Barack Obama gain 269 Electoral College votes[6]. According to the Constitution, Electoral College ties are broken in the House of Representatives. As the Democrats currently hold the majority there [7], it is likely, although not 100% that Barack Obama will become the next President. If a tie case is detected by your program, it shall output the following text:

  ```
  It’s an Electoral College tie! The next President is probably Barack Obama.
  ```

[5] Nebraska, our special case, has 5 electoral votes. Obama wins the state if he has the majority of the votes. Otherwise, McCain wins the state.

[6] A situation which was indeed quite possible. For this Obama needed to win all "Kerry states" except for New Hampshire, and add to them Iowa, New Mexico and Colorado.

[7] The exact procedure of voting is somewhat more complicated, but the Democratic party also holds the majority of state delegations as well.

## Program 4: Last digit revisited (newdigit.c)

You will revisit your `digit.c` program from Part 1 of Lab 3. You will modify the code of the program to use the `switch` statement rather than the collection of `if` statements to provide correct computation of the last digit of an input number.

Name your program `newdigit.c`.

The behavior of `newdigit` program shall be exactly the same as the behavior of the `digit` program from Lab 3 Part 1.

## Submission.

**Files to submit.** You shall submit five files: `newscore.c`, `converter.c`, `simple-elections.c`, `newdigit.c` and `team.txt`.

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

**Submission procedure.** As with Lab 1, you will be using `handin` program to submit your work. The procedure is as follows:

- `ssh` to `vogon` (`vogon.csc.calpoly.edu`).
- When prompted for `vogon password`, enter your CSL password – the one you use to log onto lab machines in 14-302.
- Upon login, change to your Lab 2 work directory[8].
- Execute the `handin` command. Please note the following:

```
> handin dekhtyar lab03-3 newscore.c converter.c simple-elections.c newdigit.c team.txt
```

**Other submission comments.** `handin` will now accept only the filename specified in the beginning of this section. You will receive an email, if you submitted wrong files. **Please, DO NOT submit binary files**.

`handin` archives **all** your submissions.

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

### Grading

The lab grade is formed as follows:

| | |
|---|---|
| `newscore.c` | 25% |
| `converter.c` | 25% |
| `simple-elections.c` | 30% |
| *newdigit.c* | 20% |

Any submitted program that does not compile earns 0 points.

[8]Lab 1 experience should teach you to create a new working directory for each assignment you do for this class inside your `cpe101` directory.

Any submitted program that compiles but fails at least one **public** (i.e., made available to you) test earns no more than 30% of its full score (and can possibly earn less).

Any submitted program that compiles and succeeds on **all** publically available tests earns at least 50% of its full score.

All programs will be checked for style conformance. Any style violation will be noted. The program will receieve a 10% penalty.

# Appendix A. Testing

For each program, you have access to the following:

- Instructor's executable;
- A full set of public tests (in a `.tar.gz` archive);
- Two `.csh` scripts for testing instructor's executable and your program.

All this is available from the Lab 3 Tests and Data page at the following URL:

http://users.csc.calpoly.edu/∼dekhtyar/101-Fall2009/labs/lab3.html

Please, follow the instructions from the previous lab assignments on how to download and use this information.

## Appendix B. Electoral College

| No. | State | Abbr. | Population | Electoral College Votes |
|---|---|---|---|---|
| 1. | Alabama | AL | 4,500,752 | **9** |
| 2. | Alaska | AK | 648,818 | **3** |
| 3. | Arizona | AZ | 5,580,811 | **10** |
| 4. | Arkansas | AR | 2,725,714 | **6** |
| 5. | California | CA | 35,484,453 | **55** |
| 6. | Colorado | CO | 4,550,688 | **9** |
| 7. | Connecticut | CT | 3,483,372 | **7** |
| 8. | Delaware | DE | 817,491 | **3** |
| 9. | District of Columbia | DC | 563,384 | **3** |
| 10. | Florida | FL | 17,019,068 | **27** |
| 11. | Georgia | GA | 8,684,715 | **15** |
| 12. | Hawaii | HI | 1,257,608 | **4** |
| 13. | Idaho | ID | 1,366,332 | **4** |
| 14. | Illinois | IL | 12,653,544 | **21** |
| 15. | Indiana | IN | 6,195,643 | **11** |
| 16. | Iowa | IA | 2,944,062 | **7** |
| 17. | Kansas | KS | 2,723,507 | **6** |
| 18. | Kentucky | KY | 4,117,827 | **8** |
| 19. | Louisiana | LA | 4,496,334 | **9** |
| 20. | Maine | ME | 1,305,728 | **4** |
| 21. | Maryland | MD | 5,508,909 | **10** |
| 22. | Massachusetts | MA | 6,433,422 | **12** |
| 23. | Michigan | MI | 10,079,985 | **17** |
| 24. | Minnesota | MN | 5,059,375 | **10** |
| 25. | Mississippi | MS | 2,881,281 | **6** |
| 26. | Missouri | MO | 5,704,484 | **11** |
| 27. | Montana | MT | 917,621 | **3** |
| 28. | Nebraska | NE | 1,739,291 | **5** |
| 29. | Nevada | NV | 2,241,154 | **5** |
| 30. | New Hampshire | NH | 1,287,687 | **4** |
| 31. | New Jersey | NJ | 8,638,396 | **15** |
| 32. | New Mexico | NM | 1,874,614 | **5** |
| 33. | New York | NY | 19,190,115 | **31** |
| 34. | North Carolina | NC | 8,407,248 | **15** |
| 35. | North Dakota | ND | 633,837 | **3** |
| 36. | Ohio | OH | 11,435,798 | **20** |
| 37. | Oklahoma | OK | 3,511,532 | **7** |
| 38. | Oregon | OR | 3,559,596 | **7** |
| 39. | Pennsylvania | PA | 12,365,455 | **21** |
| 40. | Rhode Island | RI | 1,076,164 | **4** |
| 41. | South Carolina | SC | 4,147,152 | **8** |
| 42. | South Dakota | SD | 764,309 | **3** |
| 43. | Tennessee | TN | 5,841,748 | **11** |
| 44. | Texas | TX | 22,118,509 | **34** |
| 45. | Utah | UT | 2,351,467 | **5** |
| 46. | Vermont | VT | 619,107 | **3** |
| 47. | Virginia | VA | 7,386,330 | **13** |
| 48. | Washington | WA | 6,131,445 | **11** |
| 49. | West Virginia | WV | 1,810,354 | **5** |
| 50. | Wisconsin | WI | 5,472,299 | **10** |
| 51. | Wyoming | WY | 501,242 | **3** |