COMP10002 Assignment 1, 2017s2

Last updated: September 18, 2017

Submission Instructions

A complete submission is shown in the lecture recording starting at around the 10 minute mark of the class on 5 September.

Information and Resources

26. *Sample solution*: Linked here.

    And here is the additional input file and the four expected outputs on the four test runs done as part of the marking process:
    numb.txt
    newtest1-out.txt
    newtest2-out.txt
    newtest3-out.txt
    newtest4-out.txt
    Note that in the case of test 3 and test 4, the output is truncated by the `tail -30` command, to show just the last 30 lines.

    The output that you got mailed to you in the "username-testing.pdf" file shows what you generated for these four test cases, and in the left column, whether or not your output agreed with what was expected. A "-" means that for some reason you generated a line that shouldn't have been there or had an error in it, and a "+" indicates (mostly) what had been expected instead of it. Small differences in white-space are ignored during this comparison process.

25. *Marks*: Marks and feedback for on-time projects will be emailed shortly. Late projects will be sent in a couple of days.

    Overall mark distribution:

```
     0-  0 |1
     1-  1 |
     2-  2 |
     3-  3 |3
     4-  4 |333
     5-  5 |32
     6-  6 |32
     7-  7 |333331
     8-  8 |33332
     9-  9 |333333333
    10- 10 |33333333333
    11- 11 |333333333331
    12- 12 |3333333333333
    13- 13 |3333333333333333331
    14- 14 |333333333333333333
    15- 15 |33333333333333333333333333332

381 values; min=0.0; max=15.0; mean=11.6; median=12.5; sd=3.0
```

    with the "15" category including 14.5, and so on. Note that marks are *not* final until confirmed at the end of semester. An "academic honesty" evaluation is yet to be carried out.

In 2016 semester 2:

```
251 values; min=0.0; max=15.0; mean=11.9; median=12.5; sd=3.0
```

In 2015 semester 2:

```
219 values; min=2.0; max=15.0; mean=11.7; median=12.5; sd=3.1
```

24. *Late submissions*: On-time submissions have now closed. After that time, if you wish to make a late submission, you will need to make a submission to project `ass1.late` rather than to project `ass1`:

    ```
    submit comp10002 ass1.late myprogram.c
    ```

    If you do make a late submission at all, any on-time submission you have made (via project `ass1`) will *not* be marked. Note also, you will need to verify to the same assignment label:

    ```
    verify comp10002 ass1.late > my-receipt.txt
    ```

    Unless you have requested additional time for medical or personal reasons, *and* have had that request approved by me via an emailed response, late submissions will incur a penalty of two marks for each day or part day that they are late.

23. *Sir, SIR, SSIIRR!!, This Is Really Weird!*: My program works perfectly on my own computer. But when I submit to dimfox, I'm getting the wrong answers being showing. See, I attached screenshot to show my program is working correctly. PLEASE BE KIND TO ME, I WRKED REALLY HARD ON THIS, AND I'M V. DISPARATE. *Response:* Dear Victor, on dimefox variables are not initalized to zero when they are declared. So what you are seeing is a classic issue arising from (a) you (the programmer) not initializing variables before they are used, and (b) a "friendly" development computer that does set them to zero giving you a false sense of confidence, and then (c) porting your your program to an "honest" computer that doesn't initialize variables and would rather your incorrect program failed than worked by luck. Check right through your entire program making sure that everything is give a value before it is used, or before it is added to. Oh, and dare I say it, if you had tried making your first submission before 4:03pm this afternoon (did I beg? did I plead? did I warn? did I cajole? did you ignore me?), you might have been alerted to this issue earlier. Good luck...

22. *There Are Going To Be Tears!*: As of 10am on Sunday 17th , there have been around 300 students make one or more submissions, which is (only) 2/3 of the class. The next 24 hours are going to be fun (not)! (As of 10pm: 346 submissions). (As of 8am on the 18th: 394 submissions).

21. *Another small oops*: It seems that there is another instance where I didn't quite run the same program when generating the examples in the specification as when I was generating the example output. In the specification, the Stage 4 output lines don't have an `=` sign after the word `line`:

    ```
    S4: line 9, score = 0.668
    ```

    but in the sample output they do,

```
S4: line = 9, score = 0.668
```

If you are still working on the project, go with the latter, with the `=` sign included. If you have already submitted your final version, don't panic, we will **not** take marks off in regard to the detailed formatting of the Stage 4 output lines.

20. *Measuring speed*: Note that when you have lots of output coming to the terminal screen, the scrolling and video-update processes can become time-consuming, making it seem like the program is executing slowly. If you want to measure the speed of your program, send the output into a file:

    ```
    myass1 word nerd bird slurred < pg11.txt > temp.txt
    ```

    Your program shouldn't give an extended lag when you do this, the next prompt should appear within just one second (or if you have a really slow computer, maybe two seconds).

19. *What should Stage 1 output?*: When I ran my program in class on Sept 8th, I didn't have the right messages being generated for the "incorrect query" lines, I was missing the initial `S1:`. Your submitted programs should comply with the specification.

18. *Tricksy wicksy input*: Hey, will we ever get command-line input strings like

    ```
    ass1-soln "" < alice-eg.txt
    ```

    or

    ```
    ass1-soln "string with blanks" < alice-eg.txt
    ```

    that are not just single words? *Answer:* The latter of these two should be reported as a Stage 1 error, since the string you get via `argv` contains a non-alnum character. The former one won't appear in any testing that I do, but you are welcome to also report strings of length zero as Stage 1 errors if you wish.

17. *Compilation*: Dear Dr Moffat, (1) I have noticed that I cannot compile my program on dimefox without the flag "`-std=c99`" or higher. I was wondering if this is ok. (2) Is it ok to submit multiple files and how would I do this, as I have broken up my program into multiple header files. *Answer:* the compilation line I'll be using on `dimefox` when testing your programs is

    ```
    gcc -Wall -o ass1 filename.c -lm
    ```

    The `-lm` is to get the maths library.

    Your program should be written in such a way that it compiles cleanly (no warnings) on dimefox with this command. Note that also means that you should be submitting a *single* file.

16. *Example Program*: As an example of the standard of work that is expected (including the type and degree of commenting), here are:
    - The 2013 Assignment 1 specification
    - The skeleton program that was the starting point for the 2013 Assignment 1
    - A sample solution to the 2013 Assignment 1.

    Note that you are **not** required to use `struct`s in your Assignment 1 submission, but may do so if you wish to.

15. *Blank lines*: Sir, what about blank lines? What about if the whole input file is empty? *Answer*: If you look at the outputs linked below you'll see that blank lines are presemed to have line numbers, but don't generate any output. A completely empty input file will generate the Stage 1 output, but no Stage 2 or Stage 3 or Stage 4 output.

14. *Apostophes and etc*: Dear Sir, your output has the lines

    ```
    many miles I’ve fallen by this time?’ she said aloud. ’I must be getting
    S2: line = 2, bytes = 72, words = 15
    ```

    but it seems you are counting "`I’ve`" as two words rather than one. Shouldn't that be just one word? *Answer*: The definition of "word" we are working to makes it two words. (And, in any case, isn't "`I’ve`" short for "`I have`"?)

    But Sir, if we count "`I’ve`" as two words, and one of the query terms is "`v`", then won't that mean there is a match that has to be counted? *Answer*: Correct, and that is what the specification intends and that is what my program does.

13. *Value of argc*: Dear Sir, you say in the handout that `argc` of zero represents no query. But don't you mean `argc` of 1? *Answer*: Yes, I guess I do...

12. *Access Denied on dimefox*: Note that you won't have an account yet on dimefox if you have never logged in to a lab machine so far this semester, for example, if you have been using your own computer for all the workshops, or if you simply haven't attended any workshops. The symptoms of this will be an "access denied" message when you try and connect with scp/puttyscp or ssh/putty when you want to copy/submit your program. You will need to login to a lab machine, get you account initialized, and then wait a few hours for everything to percolate through the various processes involved with transferring those account details on to dimefox. Then you'll be in a position to ssh/scp and eventually submit.

    If you still have problems after you have taken this step, and are sure you are using the right password (and can log in to other University services using it), send me an email confirming that you have taken all of these steps and **still** can't get access to `dimefox`. *Don't leave this until the last minute. It is a problem that can't be fixed in a minute!*

11. *What Can Be Stored?*: Dear Sir, you say "You can only retain five lines and their scores at any given time, plus the current line that is being processed", do you mean that *exactly*, or do you really mean "You can only retain five data structures representing five lines (possibly, for example, in original as well as processed/parsed format) and their scores at any given time, plus the current line that is being processed?". *Answer*:Yes, good question, and I mean: you can retain at most five instances of the data structure(s) that represent a single line.

10. *Error in Printed Handout*: There was a small typo in the printed version of the handout that was ciculated in class on Friday 1 September that has now been corrected in the online version here: the very first example in the handout shows

    ```
    mac: ./ass1 < alice-eg.txt
    S1: No query specified, must provide at least one word
    mac: ./ass1 Lat 66 loNg 32 words < alice-eg.txt
    S1: query = lat 66 loNg 32 words
    ```

```
    S1: loNg: invalid character(s) in query
    mac:
```

and it should be showing

```
    mac: ./ass1 < alice-eg.txt
    S1: No query specified, must provide at least one word
    mac: ./ass1 lat 66 loNg 32 words < alice-eg.txt
    S1: query = lat 66 loNg 32 words                <--------
    S1: loNg: invalid character(s) in query
    mac:
```

with a lowercase "ell" in the second example commandline marked by the arrow.

9. *Marking Rubric*: The marking rubric is linked here. Lines that do not apply to your program will be removed during the marking process; your mark will then be the sum of the lines that remain, positives and negatives. Marks won't go below zero in each section, and won't go below zero overall either.

8. *Attribution for Re-Used Code*: It is ok to make use of code (for example, insertionsort, and/or getword(), and etc) from the book or from the lecture slides or from other published/public sources, but you should remember to add an attribution as a comment to each relevant function, saying where you got it from, what modifications you added to make it suit your purpose, and so on -- exactly as you would when quoting some other author when writing an essay.

   Of course, the expectation is that the assembly and "glueing together" of these bits to make a final program will all be your own work, and that the "quoted" bits will be a relatively small fraction of the "new" output you are being asked to generate. So it is **not** ok to take a whole solution from somewhere else, even if it appears on the web; and it is **not** ok to solicit or commission a solution by posting the specification to a forum or web site and asking for "assistance" or "guidance" or "suggestions". Just as it wouldn't be ok to submit something you found online in response to an assignment that involved writing an essay.

   And a reminder of what it says in the specification: we **will** be using similarity-checking software across all the submissions, and we **will** be referring cases of suspected academic misconduct for disciplinary hearings run by the School of Engineering, and in the past those hearings **have** resulted (including multiple times in **this** subject) in students being awarded penalties including final marks of **zero** for the subject, regardless of their other components of assessment.

7. *Debugging (more)*: If a C program encounters a run-time error and exits, there might still be pending output that has not been written. This is a particular problem in the `submit` environment, because it can look like the program is failing before it generates any output at all. If in doubt, add `fflush(stdout)` function calls after each of your your debugging `printf()`'s (or even, add it to the macro), to force all pending output to be written immediately. You'll then be able to get a much clearer idea of how far the program is getting before it fails.

6. *Debugging*: Try putting this at the top of your program:

```
    #define DEBUG 1
    #if DEBUG
    #define DUMP_DBL(x) printf("line %d: %s = %.5f\n", __LINE__, #x, x)
    #else
```

```
#define DUMP_DBL(x)
#endif
```

and then, later in your code, where you have a `double` variable (say) `score`, try

```
DUMP_DBL(score);
```

Then change `DEBUG` to `0` at the top of the program, compile it again, and then run it again. Get it?

Can then add `DUMP_INT` and `DUMP_STR`, and get the extra output turned on whenever you need it to understand what your program is doing. Then turn it all off again with one simple edit.

5. *Trouble with newline characters*: Text files that are created on a PC, or copied to a PC, edited and then saved again on the PC, may end up with PC-format two-character (`CR+LF`) newline sequence, see the Wiki page for details.

   If you have compiled your program on a PC, and it receives a `CR+LF` sequence, then `getchar()` will consume them both, and hand a single-character `'\n'` newline to your program. So in that sense, everything works as expected. Likewise, on a PC when you write a `'\n'` to `stdout`, a `CR+LF` pair will be placed in to the output file (or passed through the pipe to the next program in the chain).

   The problems arise when you copy your program and a PC-format test file to a Unix system and then try compiling and executing your program there. Now the `CR` characters get in the way and arrive via `getchar()` into your program as stand-alone `'\r'` characters.

   The easiest way to defend against these confusions is to write your program so that it looks at every character that it reads, and if it ever sees a `CR` come through, it throws it away. That way, if you do accidentally get `CR` characters in your test files on the Unix server (or on your Mac) your program won't be disrupted by them. Here is a function that you should use to do this:

   ```
   int
   mygetchar() {
           int c;
           while ((c=getchar())=='\r') {
           }
           return c;
   }
   ```

   Then just call `mygetchar()` whenever you would ordinarily call `getchar()`, on both PC and Mac.

   Because most of you work on PCs (including in the labs), the test files that are provided have been created **with** the PC-style `CR+LF` newlines, and should work correctly when copied (use right-click->"Save as") to a PC. With `mygetchar()` they can also be used on a Mac, but won't interact sensibly using the standard `getchar()` function.

   To be consistent, the final post-submission testing will also be done using PC-style input files but will be executed **on a Unix machine**, meaning that **all** submitted programs will need to make use of `mygetchar()`.

You can use the "Preferences->Encodings" menu ("screwdrive/hammer Options->Encodings" in the PC version) in jEdit to select whether to use Unix (`LF`) or DOS/Windows (`CR+LF`) encodings in any test files that you create with jEdit. Note that this only applies to newly created files. jEdit will by default respect the formatting in any current files.

Note also that jEdit doesn't automatically add a newline after the last line of text files, you need to put it there explicitly yourself (just press enter one more time, so that jEdit thinks there is an empty line at the end of the file). Watch out for this problem if you are creating your own test files on Mac or PC. All the test files I supply will have a newline at the end of the last line of the file, including during the post-submission re-testing.

If in any doubt, use `od -a <file>` in a Unix shell to look at the byte-by-byte contents of a file, and check which format is being used, and whether there is a final newline character (or final `CR+LF` pair). You can do this on a PC by starting the MinGW shell and then using `cd` to reach the right directory. There is an `od` version available within the MinGW shell on the PCs in the labs. On a Mac, `Terminal` is a Unix shell.

4. *Tabs*: The default in jEdit is for tabs to be aligned every 8 character positions. Some of you have altered that to four (Preferences->Editing->Tab width), to reflect the layout that the programs in the book have. Then, on submission, the tabs have "appeared" in the output as being 8 again, which can make your program spill past the 80-character RH boundary. When I run the programs for marking, they'll **all** get formatted with tabs reflecting 4 character positions, not 8. But don't use any fewer than 4 in your jEdit (or other editor) settings.

3. *Magic numbers*: Here is a summary of the rules about magic numbers:
    - Where a number is totally self-defining, I'm happy for it to be used any number of times without a hash-define, provided the code is commented each time and/or explicitly sensible variable names are used. For example, in

      ```
      /* compute percentage */
      pcent = 100.0*count/totcount;
      ```

      I wouldn't expect 100 to have been hash-defined, since the comment explains the role of the 100, and it isn't going to change, ever, even if other percentages are calculated in the program using 100 too.
      This rule also allows 0 and 1, of course, unless they represent something other than the additive and multiplicative identities, in which case they should be hash-defined.
      This rule also allows `while (scanf("%lf%lf", &x, &y)==2)`, since the 2 is immediately obvious from the adjacent context (two variables to be read).
    - Where a constant is one that is a fact that is in no way ever going to be varied, then provided it only appears once in the program and is explained with a comment, then it need not be hash-defined. The example here is the `-32.0` in the temperature conversion computation, assuming that it is entirely within a function called `Cels2Fahr` or etc and that it isn't used in other places scattered through the program. Anything of this type that appears even twice should be hash-defined.
    - Where a factual constant is used more than once in a program, even if all occurrences are in a single function, it should be hash-defined.

   - Where a constant is one that is clearly an artifact of the problem description or the program that implements the solution (for example, numbers like `MAXINT`, or the number of variables), then they must be hash-defined, even if only used once in the program.

   Make sense?

2. *Test data*: Test data for Assignment 1, and some sample outputs:
   - alice-eg.txt
     (use right-click-saveas on the link to make a copy)
   - pg11.txt
     (use right-click-saveas on the link to make a copy)
   - output for "ass1 ali lat long < alice-eg.txt"
   - output for "ass1 f dow long way < alice-eg.txt"
   - output for "ass1 ali lat long < pg11.txt" (very long!)
   - output for "ass1 f dow long way < pg11.txt" (very long!)

1. *Specification*: The specification for the project was provided here on Friday 1 September.

Alistair Moffat, ammoffat@unimelb.edu.au