CSCI 1312 (Introduction to Programming for Engineering), Fall 2017

Homework 6

Credit: 40 points.

1 Reading

Be sure you have read, or at least skimmed, the assigned readings from chapter 7 and appendix H.

2 Honor Code Statement

Please include with each part of the assignment the Honor Code pledge or just the word "pledged", plus one or more of the following about collaboration and help (as many as apply).[1] Text *in italics* is explanatory or something for you to fill in. For written assignments, it should go right after your name and the assignment number; for programming assignments, it should go in comments at the start of your program(s).

- This assignment is entirely my own work. *(Here, "entirely my own work" means that it's your own work except for anything you got from the assignment itself — some programming assignments include "starter code", for example — or from the course Web site. In particular, for programming assignments you can copy freely from anything on the "sample programs page".)*
- I worked with *names of other students* on this assignment.
- I got help with this assignment from *source of help — ACM tutoring, another student in the course, the instructor, etc. (Here, "help" means significant help, beyond a little assistance with tools or compiler errors.)*
- I got help from *outside source — a book other than the textbook (give title and author), a Web site (give its URL), etc.. (Here too, you only need to mention significant help — you don't need to tell me that you looked up an error message on the Web, but if you found an algorithm or a code sketch, tell me about that.)*
- I provided help to *names of students* on this assignment. *(And here too, you only need to tell me about significant help.)*

3 Programming Problems

Do the following programming problems. You will end up with at least one code file per problem. Submit your program source (and any other needed files) by sending mail to `bmassing@cs.trinity.edu` with each file as an attachment. Please use a subject line that mentions the course and the assignment (e.g., "csci 1312 hw 6" or "CS1 hw 6"). You can develop your programs on

[1] Credit where credit is due: I based the wording of this list on a posting to a SIGCSE mailing list. SIGCSE is the ACM's Special Interest Group on CS Education.

any system that provides the needed functionality, but I will test them on one of the department's Linux machines, so you should probably make sure they work in that environment before turning them in.

1. (20 points) Write a C program that finds and outputs "words" in a text file. The program should get the names of the input and output files from its command-line arguments (*not* by prompting the user as most of our programs have done) and should print appropriate error messages if fewer than two command-line arguments are given or if it cannot open the specified files.

   The input file can contain any characters — alphabetic, numeric, punctuation, etc. —- but the output file should contain only the "words" (sequences of one or more alphabetic characters), each on a separate line, in lowercase. The program should also print (to standard output) the number of alphabetic characters and the number of total characters in the input file. So for example if file `words-in.txt` contains the following:

   ```
   hello world
   HELLO AGAIN!
   and some numbers 1234
   words, words, words, words.
   ```

   calling the program with the command

   ```
   a.out words-in.txt words-out.txt
   ```

   should print the line

   ```
   54 alphabetic characters, 75 characters in all
   ```

   and produce a file `words-out.txt` containing the following:

   ```
   hello
   world
   hello
   again
   and
   some
   numbers
   words
   words
   words
   words
   ```

   Since the logic for this program is a little tricky, I recommend that you start by writing a simpler but somewhat similar program that just converts alphabetic characters to lowercase, copies newline characters unchanged, and converts all other characters to spaces. If you can't figure out the full problem, you can turn in this simpler program for part credit.

   Hints:

   - You will probably find the library functions `isalpha` and `tolower` useful.

- You may find sample program simplefile-char.c[2] helpful.
- I *strongly* recommend that you read the input one character at a time rather than trying to read it into an array (which as of when this problem is being assigned we don't know about anyway!): This problem can be solved without arrays, and in fact I think they just complicate matters in some ways.
- If you're having trouble figuring how to detect "words" in the input, you might think in terms of having a variable that represents whether the preceding character was alphabetic.

2. (20 points) In class a while back we wrote a program that estimates the value of π using numerical integration. The program we wrote in class performed the integration by summing areas from left to right, but one could also sum areas from right to left, which might change the result. For this problem your mission is to write a C program that explores how the accuracy of the estimate varies with the number of "slices" and whether the two methods (left-to-right and right-to-left) produce different results.

   More specifically, your program should take as input a file containing different values for this number of slices and produce a file containing, for each input value, the input value and two output values, one for left-to-right and one for right-to-left, with each the absolute value of the difference between the estimated value of π and the best-available value of π as produced by `acos(-1.0)`.

   You will then use `gnuplot` to generate a plot showing the resulting data in graphical form.

   The program should get the names of the input and output files from its command-line arguments (*not* by prompting the user as most of our programs have done) and should print appropriate error messages if fewer than two command-line arguments are given or if it cannot open the specified files. It should also print appropriate error messages if the input file contains anything other than positive integers. (Probably it should also check that the input values are in ascending order, but you don't need to do that.)

   So for example if file `numint-in.txt` contains the following:

   ```
   10000
   20000
   30000
   40000
   ```

   calling the program with the command

   ```
   a.out numint-in.txt numint-out.txt
   ```

   should produce a file `numint-out.txt` containing the following:

   ```
   10000 8.3334095180020995031e-10 8.3333073774838339887e-10
   20000 2.0833246239249092469e-10 2.0834134417668792594e-10
   30000 9.2581942112701653969e-11 9.2588603450849404908e-11
   40000 5.2116977400373798446e-11 5.2073012568598642247e-11
   ```

   For this problem I think you will get best results if you "print" (to the output file) the two computed values with `%g` rather than `%f`, and in particular it may be useful to say you want

[2] `http://www.cs.trinity.edu/~bmassing/Classes/CS1312_2017fall/SamplePrograms/Programs/simplefile-char.c`

more than the default number of significant figures – I used `%22.20g`. I think you should also use more values than I did in the example, and you could consider having each value be twice the preceding one rather than a fixed amount more. Try different choices until you get a set (or sets) of input that produces output that seems meaningful.

To generate a plot, put the needed `gnuplot` commands in a file, such as numint.plotin[3]. The command `gnuplot numint.plotin` will then generate a plot `numint.png`, which you can view with the command `display numint.png`.

Turn in your program and, for at least one set of input values (you might want to try more than one):

- The input file (the one your program uses as input).
- The file of `gnuplot` commands used to create a plot.

(You don't need to turn in your program's output file, or the plot, since with the above I can recreate them.)

Hints:

- You should probably start by coping relevant parts of sample program pi-numint.c[4].
- You may find sample program `simplefile-fmt.c` simplefile-fmt.c[5] helpful.

[3] `http://www.cs.trinity.edu/~bmassing/Classes/CS1312_2017fall/Homeworks/HW06/Problems/numint.plotin`

[4] `http://www.cs.trinity.edu/~bmassing/Classes/CS1312_2017fall/SamplePrograms/Programs/pi-numint.c`

[5] `http://www.cs.trinity.edu/~bmassing/Classes/CS1312_2017fall/SamplePrograms/Programs/simplefile-fmt.c`