

CSCI 1120 (Low-Level Computing), Spring 2017

Homework X

Credit: Up to 30 extra-credit points.

1 General Instructions

Do as many (or few) of the following problems as you like. Notice that you can receive at most 30 extra-credit points, but be advised that any points you earn can only help your grade — that is, I will add them to your total points before dividing by the sum of the “perfect score” points on the required assignments.

I am also open to the possibility of giving extra credit for other work — other problems/programs, a report on something course-related, etc. If you have an idea for such a project, let’s negotiate (by e-mail).

NOTE that the usual rules for collaboration do not apply to this assignment. More in the following section.

2 Honor Code Statement

Please include with each part of the assignment the Honor Code pledge or just the word “pledged”, and the statement “This assignment is entirely my own work” (where “my own work” means “except for anything I got from the assignment itself, such as starter code, or from the course Web site or sample solutions to other assignments). For this assignment you should *not* work with or seek help from other students or from tutors, but you can consult other sources (other books, Web sites, etc.), as long as you identify them.

3 Programming Problems

Do as many of the following programming problems as you choose. 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 1120 hw X” or “LL hw X”). You can develop your programs on 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. (Up to 5 points) Write a C program that, given the name of a text file as a command-line argument, reads the contents of the file and produces a histogram of word lengths, where a “word” is one or more alphabetic characters. So for example given an input file containing the following text

```
Now is the time for all good persons to come to the aid of their party.
```

```
A really long word, though perhaps not the longest in English,  
is "antidisestablishmentarianism" (28 letters).
```

it would produce the following

```

1 *
2 *****
3 *****
4 *****
5 **
6 **
7 *****
8
9
10
11
12
13
14
15
16
17
18
19
>=20 *
```

(Notice that it groups all words of length at least 20 into a single output line — simpler to code and in my opinion reasonable.) To get maximum points, your program should do the following:

- Use arrays only when you need them (e.g, you probably need one for counters — how many words of length 1, how many of length 2, etc. — but probably not one for an input line).
 - Deal reasonably gracefully with input errors (no argument supplied, input file not found).
2. (Up to 5 points) Write a C program that evaluates polynomial $p(x)$ given the coefficients of p and one or more values of x . You can prompt for the coefficients or get them from command-line arguments; once you have them, repeatedly prompt for values of x until the user enters something non-numeric. A supposedly efficient way to evaluate a polynomial is with “Horner’s rule” (check the Wikipedia article if you’re not familiar with this approach), which can be implemented with a loop or recursion. A sample execution prompting for the coefficients:

```

% mypgm
degree of polynomial (highest power)?
3
coefficients (starting with highest power)?
2 4 3 5
p(x) = 2.000000(x**3) + 4.000000(x**2) + 3.000000(x**1) + 5.000000
x?
10
p(10.000000) = 2435.000000
x?
```

```

100
p(100.000000) = 2040305.000000
x?
invalid input

```

and one getting them from the command line:

```

% mypgm 2 3 4 5
p(x) = 5.000000(x**3) + 3.000000(x**2) + 4.000000(x**1) + 2.000000
x?
10
p(10.000000) = 5342.000000
x?
100
p(100.000000) = 5030402.000000
x?
invalid input

```

3. (Up to 22.5 points) Do Homework 8 from last semester's CSCI 1312¹. (For each part of the assignment, maximum extra-credit points are half the points shown in the assignment.)
4. (Up to 10 points) Write a C program that converts lengths from one unit to another — centimeters to inches, kilometers to miles, etc. To keep things simple, you can represent the different units with one- or two-character strings. The program should prompt repeatedly for an amount to convert and the two units, stopping when the user signals “end of file” (control-D on Linux). Sample execution:

```

% mypgm
enter amount and two units (control-D to end)
possible conversions:
    in to cm
    cm to in
    ft to m
    m to ft
    mi to km
    km to mi
1 in cm
1 in is 2.54 cm

enter amount and two units (control-D to end)
possible conversions:
    in to cm
    cm to in
    ft to m
    m to ft
    mi to km
    km to mi
2.54 cm in

```

¹http://www.cs.trinity.edu/~bmassing/Classes/CS1312_2016fall/Homeworks/HW08/Problems/homework.pdf

2.54 cm is 1 in

enter amount and two units (control-D to end)

possible conversions:

in to cm

cm to in

ft to m

m to ft

mi to km

km to mi

10 ki mi

unknown conversion

enter amount and two units (control-D to end)

possible conversions:

in to cm

cm to in

ft to m

m to ft

mi to km

km to mi

10 km mi

10 km is 6.21371 mi

enter amount and two units (control-D to end)

possible conversions:

in to cm

cm to in

ft to m

m to ft

mi to km

km to mi

10 km miles

invalid input

enter amount and two units (control-D to end)

possible conversions:

in to cm

cm to in

ft to m

m to ft

mi to km

km to mi

1 ft m

1 ft is 0.3048 m

enter amount and two units (control-D to end)

possible conversions:

```

        in to cm
        cm to in
        ft to m
        m to ft
        mi to km
        km to mi
1 m ft
1 m is 3.28084 ft

```

To get maximum points, your program should do the following:

- Use an array of **structs** to represent a table of allowed conversions. Here is the **struct** definition I used along with an example of defining a possible element of the required array.

```

typedef struct {
    char *from_unit;
    char *to_unit;
    double factor;
} conversion_t;

conversion_t factor = { "in", "cm", 2.54 };

```

(You can use a different **struct** if you think of one that seems better to you.)

- Use this array to generate the prompt (the part beginning `possible conversions` in the sample output), and also (of course?) to figure out from user input what conversion is to be done and how to do it.
- Generate all the conversion factors from a single basic one (1 inch is 2.54 centimeters) (e.g., if you know that 1 inch is 2.54 centimeters, then 1 centimeter is $1/2.54$ inches).
- Deal reasonably gracefully with invalid input.

You will get some points for any program that works more or less as shown by the sample output, including one that just prompts once, does the requested conversion, and exits.

5. (Up to 10 points) Write a C program that performs encryption using the so-called Vigenère cipher, as described in [the Wikipedia article for it](#)². The program should take two command line arguments:
 - `--encrypt` or `--decrypt` (sort of self-explanatory)
 - the keyword to use for the cipher (lower-case alphabetic only)

It should get the text to encrypt (or decrypt) from standard input and write the result to standard output. As in the homework problem dealing with a simpler form of encryption, lower-case letters should encrypt as lower-case, upper-case letters as upper-case, and everything else as itself. So for example the following text

Hello!

Now is the time for all good persons to come to the aid of their country.

².

```
ABCD 1234 !@#$
```

```
Goodbye!
```

encrypted with keyword `abcd` gives

```
Hfnoo!
```

```
Qqz it vke ukpe gqu amn jopf sesurnt vr cpoh tp vke bkg og vkejt fovpwrz.
```

```
CECE 1234 !@#$
```

```
Iroedbe!
```

(If this sounds interesting to you, feel free to ask for hints on how to proceed. I won't include them here for reasons of time.)

6. (Up to 15 points) In most of the programs we wrote in class and for homework we made some attempt to “validate” user input (e.g., check that inputs are numeric when they're supposed to be, positive when they're supposed to be, etc.). Doing this for many variables is apt to produce a lot of uninterestingly-repetitive code. Also, if the input was not valid we just bailed out of the program rather than trying again. Propose and implement one or more functions that would address one or both of these possible shortcomings, and submit it/them with a short program that could be used to test it/them. Be sure to include comments that describe the function's parameters and behavior (does it exit the program on error or prompt again or what). You might like to have functions for working with input from standard input and also functions that work with command-line arguments.
7. (Up to 15 points) In class I said that getting “a line” of character data (a sequence of characters read from a file or standard input ending with the end-of-line character) was surprisingly difficult and error-prone in C. Propose and implement a function or functions that gets a full line of character data in a way that does not limit the length of the input data but also does not risk overflowing an array, and submit it/them with a short program that could be used to test it/them. (You will almost surely need `malloc` to make this work. You can use library functions, but only ones that are in the standard C library — so you can't just use the GNU extension `getline`.)
8. (Up to 20 points) Write a C program to solve a problem that seems interesting to you. How much credit you can get depends on difficulty – solving a relatively easy problem is worth fewer points than solving a more difficult one, and programs that are well-structured will get more points than those that are less so (e.g., good use of functions will get you more points, as will writing C-idiomatic code). Include comments in your program that explain what problem it solves and what input it needs from the user (command-line arguments, input files, input from `stdin`). If you have an idea for this problem but aren't sure how much credit you could get, ask me by e-mail.