EECS2031 Winter 2018

Software Tools

Assignment 2 (15%): C Programming

Due Date: 11:59pm on Fri, Mar 2, 2018

Objective

In this assignment, you will be writing four C programs. The first program (**triangle.c**) draws triangle patterns of a given number of rows. The second program (**caesar.c**) implements a Caesar cipher, used to encrypt text messages. The third program (**anagram.c**) is a tool that tests whether two words are anagrams. The fourth program (**rw.c**) is a tool for generating a random walk across a 2D array.

Important Notes:

- You must use the *submit* command to electronically submit your solution by the due date.
- All programs are to be written using *C* and compiled by *gcc*.
- Your programs should be tested on the *EECS labs* before being submitted. Failure to test your programs on *EECS labs* will result in a mark of 0 being assigned.
- To get full marks, your code must be well-documented.

What To Submit

When you have completed the assignment, move or copy your four C programs in a directory (e.g., assignment2), and use the following command to electronically submit your files within that directory:

```
% submit 2031M a2 triangle.c caesar.c anagram.c rw.c
```

You can also submit the files individually after you complete each part of the assignment– simply execute the *submit* command and give the filename that you wish to submit. You may submit your solutions as many times as you wish prior to the submission deadline. Make sure you name your files **exactly** as stated (including lower/upper case letters). Failure to do so will result in a mark of 0 being assigned. You may check the status of your submission using the command:

```
% submit -l 2031M a2
```

A. Triangle Patterns (25%)

Write a C program **"triangle.c"** that uses the "*" character to draw a triangle of a given number of rows. The program first prompts the user to enter the number of rows in the triangle. Your program may assume that the input is a valid integer from 1 to 20 (inclusive). Here are some sample outputs from the execution of the program. The output of your program should match the sample output.

```
$ ./triangle
Enter the number of rows in the triangle: 1
*

$ ./triangle
Enter the number of rows in the triangle: 2
 *
***

$ ./triangle
Enter the number of rows in the triangle: 3
  *
 * *
*****

$ ./triangle
Enter the number of rows in the triangle: 10
         *
        * *
       *   *
      *     *
     *       *
    *         *
   *           *
  *             *
 *               *
*******************
```

Example run

Hint: You may find it helpful to draw the required output on a piece of graph paper before writing your program.

B. Caesar's Cipher (25%)

Write a C program **"caesar.c"** that encrypts a message using one of the oldest known encryption techniques, called Caesar cipher, attributed to Julius Caesar. It involves replacing each letter in a message with another letter that is a fixed number of positions later in the alphabet (shift). If the replacement would go past the letter *Z*, the cipher "wraps around" to the beginning of the alphabet. For example, if each letter is replaced by the letter two positions after it (shift by 2), then *A* would be replaced by *C*, *Y* would be replaced by *A*, and *Z* would be replaced by *B*. The user will enter the message to be encrypted and the shift amount as a valid integer from 1 to 25 (inclusive). Here's an example of the desired output:

```
$ ./caesar
Enter message to be encrypted: Hello World
Enter shift amount (1-25): 3
Encrypted message: Khoor Zruog


$ ./caesar
Enter message to be encrypted: Khoor Zruog
Enter shift amount (1-25): 23
Encrypted message: Hello World
```

Example run

Notice that the program can also be used to decrypt a message if the user knows the original key by providing the encrypted message and using as shift amount 26 minus the original key (see example). You may assume that:

- The message does not exceed 80 characters.
- Characters other than letters should be left unchanged.
- Lower-case letters remain lower-case and upper-case letters remain upper-case.

Hint: You may wish to use operations on characters to handle the "wrap around" problem and to calculate the encrypted version of a lower-case or upper-case letter.

C. Anagrams (25%)

Write a C program **"anagram.c"** that tests whether two words are anagrams (permutations of the same letters). Your program should ignore any characters that aren't letters and should treat upper-case letters as lower-case letters. You may wish to use functions from *<ctype.h>*, such as *isalpha* and *tolower*. Here's an example of the desired output:

```
$ ./anagram
Enter first word: Hello
Enter second word: eolHl
The words are anagrams.

$ ./anagram
Enter first word: Hello
Enter second word: olhle
The words are anagrams.

$ ./anagram
Enter first word: Hello
Enter second word: World
The words are not anagrams.

$ ./anagram
Enter first word: Hello
Enter second word: ello
The words are not anagrams.
```

Example run

Hint: You may wish to use an array of 26 integers to keep track of how many times each letter (a-z) has been seen. For example, after the world "Hello" has been read, the array should contain the values:

0 0 0 0 1 0 0 1 0 0 0 2 0 0 1 0 0 0 0 0 0 0 0 0 0 0

reflecting the fact that Hello contains one *e*, one *h*, two *l*, and one *o*.

D. Random Walk (25%)

Write a C program **"rw.c"** that generates a random walk across a 10x10 array. Initially, the array will contain only dot characters ('.'). The program must randomly "walk" from element to element, always going up, down, left, or right by one step. The elements visited by the program will be labeled with the letters *A* through *Z*, in the order visited. Here's an example of the desired output:

```
$ ./rw
A B . . . . . . . .
. C D E F . . . . .
. . . . G P Q . . .
. . . . H O R . . .
. . . . I N S T . .
. . . . J M . U V W
. . . . K L . . . X
. . . . . . . . . Y
. . . . . . . . . Z
. . . . . . . . . .
```

Example run

Hint: Use the *srand* and *rand* functions to generate random numbers [0, 1, 2, and 3] that will indicate the direction of the next move of the walk. Before performing a move, check that:

- It won't go outside the array.
- It doesn't take the walk to an element that has already a letter assigned (blocked element).

If either condition is violated, try moving in another direction. If all four directions are blocked, the program must terminate. Here's an example of premature termination (M is blocked on all four sides):

```
$ ./rw
A L K . . . . . . .
B M J . . . . . . .
C H I . . . . . . .
D G . . . . . . . .
E F . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
```

Example run