EECS 2031 Section A
Software Tools
Assignment 2 (100 points), Version 1

Instructor: Ilir Dema
Release Date: Oct 17, 2022

Due: Nov 4, 2022, midnight

All your lab submissions must be compilable/executable on the department machines. It is then crucial that should you choose to work on your own machine, you are responsible for testing your project before submitting it for grading.

Check the Amendments section of this document regularly for changes, fixes, and clarifications.

Ask questions on the course forum on discord.

1 Policies

- Your (submitted or un-submitted) solution to this assignment (which is not revealed to the public) remains the property of the EECS department. Do not distribute or share your code in any public media (e.g., a non-private Github repository) in any way, shape, or form before you get the permission from your instructors.

- You are required to **work on your own for this lab**. No group partners are allowed.
- When you submit your solution, you claim that it is solely your work. Therefore, it is considered as an violation of academic integrity if you copy or share any parts of your code or documentation.
- When assessing your submission, the instructor and TA may examine your doc/code, and suspicious submissions will be reported to the department/faculty if necessary. We do not tolerate academic dishonesty, so please obey this policy strictly.

- You are entirely responsible for making your submission in time.

- You may submit multiple times prior to the deadline: only the last submission before the deadline will be graded.
- Practice submitting your work early even before it is in its final form.
- No excuses will be accepted for failing to submit shortly before the deadline.
- Back up your work periodically, so as to minimize the damage should any sort of computer failures occur. You can use a **private** Github repository for your labs/projects.
- The deadline is strict with no excuses.
- Emailing your solutions to the instruction or TAs will not be acceptable.

Amendments

So far, so good

2 Problem

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.

3 What to submit

When you have completed the assignment, move or copy your four C programs in a directory (e.g., a2), zip the whole folder, and submit the zip file on eclass. Make sure you name your files exactly as stated (including lower/upper case letters).

3.1 `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

3.2 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.

3.3 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

3.4 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