CS/CE 1337 – PROJECT 1 – Mario Paint

Pseudocode Due: Section 001 (M/W) – 9/1 by 11:59 PM
Section 002 (T/R) – 9/2 by 11:59 PM

Project Due: Both sections - 9/14 by 11:59 PM

Submission and Grading:

- All project deliverables are to be submitted in eLearning.
- The pseudocode should be submitted as a Word or PDF document and is not accepted late.
- Submit the program as a .cpp file only.
- Projects submitted after the due date are subject to the late penalties described in the syllabus.
- Programs must compile using gcc 7.0.1 or higher with the following flags enabled
    - -Wall
    - -Wextra
    - -Wuninitialized
    - -pedantic-errors
    - -Wconversion
- Each submitted program will be graded with the rubric provided in eLearning as well as a set of test cases. These test cases will be posted in eLearning after the due date. Each student is responsible for developing sample test cases to ensure the program works as expected.
- **Type your name and netID in the comments at the top of all files submitted.**

Objective: Use advanced file I/O techniques to manipulate files directly.

Problem: Mario Paint was a great innovation in its time allowing people with a home console to create art and music easily. One of the features of Mario Paint was that it let users create their own 8-bit stamps to use within the game. With this idea, you are going to create a program that will allow users to create black and white pixel art. Since all of the output will be file-based, we can't easily add color to the drawings.

Details:

- The user's canvas will be a grid (50 characters x 50 characters)
    - The newline character on each line is not considered to be part of the canvas
    - Remember that a newline character in Windows is treated as 2 bytes
- Input will consist of a series of commands read from a file
- Commands
    - 1 – Pen up
    - 2 – Pen down
    - 3, D, # – Move pen in given direction the given number of characters
        - Example – 3, N, 10 would move the pen up (north) 10 characters
            - If the pen is down, the space where the pen resides is not filled in
            - Only the 10 spaces above the current position would be filled
        - Directions
            - N – north / up

            - S – south / down
            - E – east / right
            - W – west / left
    - 4 – Display output file in console
    - B – bold on
        - Use # to draw
    - b – bold off
        - Use * to draw
- No command will be given that makes the pen go outside of the canvas space
- All “drawing” is to be done directly in the file
    - Do not use a 2-dimensional array to hold the drawing and output it to the file at the end of the program.
- If the pen is “down” and moved, write the given number of characters in the file in the given direction
- The movement commands do not include the current spot of the pen
    - For example if the pen is down, the command 3, E, 2 would draw in the two spaces to the right of the current pen location.
- In the case of intersecting lines, bold will always take precedence.
- By default, bold is off when the program begins.
- The pen will begin at row 1, column 1 at the start of the program.

Input: All input will be read from a file named *commands.txt*. All commands in the file will be valid and of the proper format. Each command will be on a separate line.

Output: Each input file will produce an output file named *paint.txt*. If a print command (4) is listed in the input file, display the current state of the entire file to the console window, making sure that each line of the output file is displayed on a separate line in the console window. After the file has been displayed to the console, send two blank lines to the console window as a buffer for the next print command in the file.

After the entire input file has been processed, display the completed artwork to the console before the program exits.