CS/CE 1337 – PROJECT 3 – NFT Generator 2.0

Pseudocode Due: 3/11 by 11:59 PM (No late submission)

Core Implementation Due: 3/26 by 11:59 PM (No late submission)

Final Submission Due: 4/3 by 11:59 PM

KEY ITEMS: Key items are marked in red. Failure to include or complete key items will incur additional deductions as noted beside the item.

Submission and Grading:

- The pseudocode will be submitted in eLearning as a Word or PDF document and is not accepted late.
- All project source code will be submitted in zyLabs.
    - Projects submitted after the due date are subject to the late penalties described in the syllabus.
- Programs must compile using gcc 7.3.0 or higher with the following flags enabled
    - -Wall
    - -Wextra
    - -Wuninitialized
    - -pedantic-errors
    - -Wconversion
- **Type your name and netID in the comments at the top of all files submitted. (-5 points)**

Objectives:

- Use structures containing pointers to create a dynamic data structure.
- Use pointers to modify the dynamic data structure.
- Implement a recursive function.

Problem: Over the past few months, the popularity of NFTs has exploded. NFTs can be found in many popular pastimes including sports and video games. Since NFTs are all the rage these days, you have decided to take that idea and build an NFT generator that will produce ASCII art.

Pseudocode: Your pseudocode should describe the following items

- Creating the grid
- Printing the grid
- Deleting the grid
    - Each individual node needs to be deleted
- For each function
    - Determine the parameters and return type
    - Detail the step-by-step logic that the function will perform
- A list of at least 10 test cases you will check during testing (other than the examples below)
    - Specific input is not necessary
    - Describe what you are testing
    - Example: drawing a line down
    - Example: Attempting to move out of bounds to the left

Zybooks Information:

- Your source file will be named `main.cpp`
  - There is no template for this assignment
- Core implementation has unlimited submissions
  - This will help you make sure the basic actions of your program are working properly in the Zybooks environment
- Final submission is limited to 12 submissions
- White space will be checked to ensure numerals are written in the correct place

Core Implementation Phase:

- Create the grid
- Read sample commands file
- Produce sample output
- No invalid input
- No boundary checks
- Display / write to file can be non-recursive

Details:

- The user's canvas will be a grid of node structures (50 rows of 50 nodes)
- **Node structure**
  - Variable to hold character for drawing
  - 4 node pointers
    - Up, down, left and right
- The nodes will be connected to other nodes through the pointers inside them.
- The canvas will be constructed and connected without the help of any other data structures or arrays (-10 points if arrays/data structures used to help connect nodes)
- There will be a distinct border for the canvas indicated by null pointers
  - For example, a node on the top border of the canvas will have a null up pointer
  - There will not be any wraparound effect.
- Input will consist of a series of commands read from a file
- Each command in the file will indicate the digital pen status as well as movement direction and distance
  - Optionally, a command may also include a bold status to include a different character when drawing
- The digital pen status will determine if the characters are written to the file as the pen is moved
  - If the pen is "down" and moved, write the given number of characters in the file in the given direction
  - If the pen is "up", just move the file pointer to the proper position
- **The movements 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.
- All "drawing" is to be written to the canvas in memory

    - Do not use a 2-dimensional array to hold the drawing
      (-20 points for holding/manipulating the drawing in a 2D array or using random file access techniques)
- The default character to store in the grid will be an asterisk (*)
    - If the bold status is on, the character will be a hashtag (#)
- A single recursive function will be used to display the canvas (either to console or output file)
  (-10 points if done iteratively)
- The pen will begin at row 1, column 1 at the start of the program.

Input: All input will be read from a file. Prompt the user for the name of the input file. Each command will be on a separate line in the file and each line will have a new line character at the end of the line (except for the last line which may or may not have a newline character).

Each valid command will have the following format (note there are no spaces in the command):

```
<status>,<direction>,<distance>,<bold - optional>,<print - optional>
```

- Each element will be a single character
- Status
    - 1 – pen is up
    - 2 – pen is down
- Direction
    - N – north/up
    - E – east/right
    - S – south/down
    - W – west/left
- Distance – a positive integer value
- Bold - a capital B
    - This element is only valid if the pen is down
    - This element is optional and can be omitted from a valid command
- Print – a capital P
    - Print the current status of the drawing to the screen
    - This element is optional and can be omitted from a valid command

Examples of valid commands

- 1,N,5
- 2,E,2
- 2,S,3,B
- 1,W,4,4
- 2,N,10,B,4

It is possible for the file to have invalid commands. An invalid command is any command that does not adhere to the format listed above. It is also possible for a command to move the pen out of bounds of the grid. These commands should also be treated as an invalid.

If an invalid command is identified, do not execute it. Move to the next command in the file.

Output: If a print option is included with the command in the input file, display the current state of the canvas to the console window using the recursive function you created. Make sure that each row of the canvas is displayed on a separate line in the console window. After the canvas has been displayed to the console, send two blank lines to the console window as a buffer for the next print option in the file.

At the end of the program, write the contents of the canvas to **`paint.txt`** using the recursive function.