CE/CS 1337 – PROJECT 2 – Don't Cross the Streams

Pseudocode Due: 2/20 by 11:59 PM (No late submission)

Core Implementation Due: 2/27 by 11:59 PM (No late submission)

Final Submission Due: 3/6 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)**

Objective:

- Allocate memory dynamically.
- Utilize pointers to directly manipulate memory.

Problem: You have been hired by EA to work on a new game to coincide with the release of the new Ghostbusters movie, Ghostbusters Afterlife. Your project requires that you build a framework to determine a point in three-dimensional space where up to four lines may intersect. These lines will represent the proton pack streams from the Ghostbusters. As all good Ghostbusters know, you can't cross the streams, so this point is very important. Your framework will contain algorithms to perform the Gauss-Jordan Elimination method of solving a linear system of equations which will determine the point in which the streams would cross.

Pseudocode: Your pseudocode should describe the following items

- Main.cpp
    - Detail the step-by-step logic of the main function
    - List other functions you plan to create
        - Determine the parameters
        - Determine the 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
    - Specific input is not necessary
    - Describe what you are testing
    - Example: multiplying row 2 with a positive integer

Zybooks Information:

- Your source file will be named `main.cpp`
  - There is no template
- 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 not be checked

Core Implementation Phase:

- Read the sample file
- Create the matrix
  - Can be non-dynamic for this phase
- Perform scalar multiplication on a given row
  - Must use pointer arithmetic to edit matrix
- No swapping rows
- No scalar multiplication and addition
- No checks for solved matrix
- No invalid input

Details

- The coefficients of the equations will be held in a dynamically allocated array
  - You may create a dynamic single dimension array
    - Standard multi-dimensional arrays in C++ are stored as single dimension arrays in memory
    - This is very similar to how a multi-line file is stored (think back to Project 1)
- The number of rows will be dependent on the number of players (2-4 players)
- The number of columns will always be 4
  - The coefficients for x will be in column 1, y in column 2 and z in column 3
  - The constant on the right-hand side of the equal sign will be in column 4
- The size of the array indicates the total number of spots needed to hold all the data
  - For example, if there are 2 equations, the array will need 8 spots
- The linear equations will be read from a file
- Pptions available to the tester are:
  - Switch 2 rows
  - Multiply a row by a non-zero number (positive or negative)
    - This may be a floating point number or an integer
  - Add a scalar multiple of one row to another row
- If the matrix is in reduced echelon form (http://mathworld.wolfram.com/EchelonForm.html), do not allow the tester to continue row operations.
- If a row has been solved (all numbers on the row are 0 or only one of the variable columns contains a 1), the row cannot be multiplied by a non-zero scalar (row option 2 above).

- If the user tries to perform an illegal operation, post an appropriate error message and loop back to the menu
- All interaction with the array must be done using pointers and pointer arithmetic. That means no bracket notation in this project except to create the array and no offset notation. (-20 points if bracket notation used, -10 points if offset notation used)
    - **EXTRA CREDIT** – Instead of using pointers and pointer arithmetic, implement this program using smart pointers. **(+5 POINTS)**
- All numbers will be displayed to 2 decimal places
- **You are not expected to write a program that solves the matrix. We will leave it to someone else at EA to develop the AI that utilizes your framework and solves the matrix automatically. We are simply providing the functions that allow changes to be made to the array.**

Input and User Interface:

- Develop an easy to use menu system for the tester.
    - 1 – Switch two rows
        - Prompt the user for the number of the rows to switch
    - 2 – Multiply row by non-zero number
        - Prompt for row
        - Prompt for number to multiply
    - 3 – Add scalar multiple of one row to another row
        - Prompt for the following in the exact order listed
            - Row to use for multiplication
            - Multiplier
            - Row to be modified by adding scalar multiple
    - 4 – Quit
        - Exits the program
- All input will be entered from the keyboard and will be of the valid data type.
    - You do not have to worry about the tester entering the wrong data type
- You are expected to verify input is within range
    - If a number out of range is entered, loop until a valid number is entered

Equation Input:

- Equations will be read from a file
    - Prompt the user for the filename at the start of the program
- The file will contain equations for each player
    - The file will have two, three or four equations
- Each line in the file will end with a newline (except the last line which may or may not have a newline)
- Each equation will consist of up to 3 variables and a constant.
- The variables will always be x, y and z
    - There will not be multiples of the same variable in an equation
- An equation may not contain all of the variables
    - If a variable is missing from the equation, the coefficient is assumed to be zero

- The variables may have a coefficient
    - The coefficients can be floating point numbers
- If a variable does not have an explicit coefficient, it is assumed to be 1
- There will be no spaces in the equations
- All variables will be to the left of the equal sign
- The variables can be in any order
    - Examples
        - `3.5x+2.95y+.3z=-2`
        - `z+5y-3x=10`

Output:

- After every row operation, display the modified matrix to the console.
- If the matrix is in reduced echelon form, display the variables and their known values as illustrated by the matrix and end the program
    - Check for this before allowing a new command from the user
- Output format
    - Variables will be displayed in alphabetical order
    - `<variable><space>=<space><value>`
    - Remember that in a 2x4 matrix only 2 of the variables can be solved through Gauss-Jordan Elimination. You are not required to calculate the third variable in this situation.
    - In a 4x4 matrix, all variables must be solved, and the last row must be all zeroes to be in reduced echelon form