ECE 4122/6122 Lab 1: Dynamic Programming, Simple Multithreading & File Input/Output

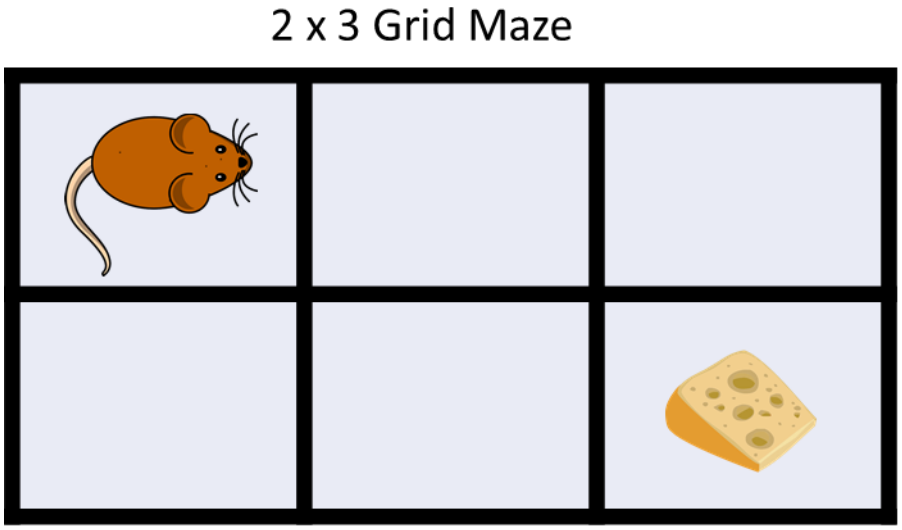
(100 pts)

Due: Monday September 26th, 2022 by 11:59 PM

Problem 1: Mouse in a Grid Maze (50 pts)

You are applying for your dream job and the interviewer asks you to solve the following problem using dynamic programming as efficiently (shortest run time) as possible:

A researcher is running an experiment where a mouse is placed at the top left position on a ***m x n*** grid maze. The mouse can only either move 1 position to right or one position down at a time in order to reach a piece of cheese at the lower right corner of the grid.



The researcher needs to know how many *unique* paths are possible for different sized grid mazes. You need to write an application that uses ***command line arguments*** to input the number of rows and the number of columns of the grid maze and output the total number of possible unique paths to a text file (NumberPaths.txt). The first command line argument is the number of rows and the second argument is the number of columns. The output file is overwritten each time the program is executed. You need to write a function contained in a separate implementation file (**numberGridPaths.cpp**) to calculate the total number of paths using the function prototype show below:

uint64_t numberGridPaths(unsigned int nRows, unsigned int nCols);

Sample Results:

m (rows)	n (columns)	Total Number of Paths
1	1	1
2	3	3
0	any valid number	0

any valid number	0	0
18	18	2,333,606,220

Turn-In Instructions:

Your application needs to be made up of two files for this problem:

1) main.cpp
2) numberGridPaths.cpp

Place the two files into a zip file called Lab1Prob1.zip and upload to the Canvas assignment.

Example Program Input\Output: *(m & n values will be restricted to not exceed 18 in either direction)*

Ex 1:

Lab1Prob1.out 2 3

NumberPaths.txt contains single line ➔ Total Number Paths: 3

Ex 2:

Lab1Prob1.out 2 (only one argument, invalid)

NumberPaths.txt contains single line ➔ Invalid Input!

Sample Invalid Values:

000, 01, abcd, also having only one argument or more than two is invalid.

Reference Material:

https://www.tutorialspoint.com/command-line-arguments-in-c-cplusplus
https://cplusplus.com/reference/fstream/ofstream/ofstream/
https://collegenote.net/curriculum/data-structures-and-algorithms/41/451/

Hint:

Draw a recursion tree to understand the recursive nature for a 2x3 grid maze. This problem is just a slight modification to the Fibonacci problem covered in class.

Problem 2: Su Doku (50 pts)

(www.projecteuler.net) Su Doku (Japanese meaning *number place*) is the name given to a popular puzzle concept. Its origin is unclear, but credit must be attributed to Leonhard Euler who invented a similar, and much more difficult, puzzle idea called Latin Squares. The objective of Su Doku puzzles, however, is to replace the blanks (or zeros) in a 9 by 9 grid in such that each row, column, and 3 by 3 box contains each of the digits 1 to 9. Below is an example of a typical starting puzzle grid and its solution grid.

0 0 3	0 2 0	6 0 0
9 0 0	3 0 5	0 0 1
0 0 1	8 0 6	4 0 0
0 0 8	1 0 2	9 0 0
7 0 0	0 0 0	0 0 8
0 0 6	7 0 8	2 0 0
0 0 2	6 0 9	5 0 0
8 0 0	2 0 3	0 0 9
0 0 5	0 1 0	3 0 0

4 8 3	9 2 1	6 5 7
9 6 7	3 4 5	8 2 1
2 5 1	8 7 6	4 9 3
5 4 8	1 3 2	9 7 6
7 2 9	5 6 4	1 3 8
1 3 6	7 9 8	2 4 5
3 7 2	6 8 9	5 1 4
8 1 4	2 5 3	7 6 9
6 9 5	4 1 7	3 8 2

Write a program that takes as a **command line argument** that is the path to an input file. There is a sample input file called *input_sudoku.txt* included in the assignment.

Your program must include the following:

- Using std::thread to create a multi-threaded program to solve a set of sudoku puzzles.
- Create a class called **SudokuGrid** that is used to hold a single puzzle with a constant 9 x 9 array of **unsigned char** elements.
- Create the following member variables for **SudokuGrid**:
  - std::string m_strGridName;
  - unsigned char gridElement[9][9];
- Create the following member functions for **SudokuGrid**
  - friend fstream& operator>>(fstream& os, const **SudokuGrid** & gridIn);
    - reads a single SudokuGrid object from a fstream file.
  - friend fstream& operator<<(fstream& os, const **SudokuGrid** & gridIn);
    - writes the SudokuGrid object to a file in the same format that is used in reading in the object
  - solve();
- Your **main()** function needs to dynamically determine the maximum number of threads (numThreads) that can run concurrently. Your **main()** function should then spawn (numThreads-1) threads calling the function **solveSudokuPuzzles()**.
- Use global variables
  - std::mutex outFileMutex;
  - std::mutex inFileMutex;

    - std::fstream outFile;
        - In the main() function, open the output file **Lab2Prob2.txt** using the command line argument. Use the same format for the output file as in the input file.
    - std::fstream inFile;
        - In the main() function open the file using the command line argument.
- In the function **solveSudokuPuzzles()** use a std::mutex(s) to protect the global variables ***inFile*** and ***outFile.***
    - The function needs to have a do-while loop to continue to read in and solve puzzles and then write out the solution until the end of the file is reached.
    - once the end of the file is reached the function should return.
- After all threads are finished close both the global fstream variables.
- You are free to add any other member functions you think are needed.

You are free to use online solutions for solving the Sudoku problems. Here is one such example: https://www.tutorialspoint.com/sudoku-solver-in-cplusplus. I have not tested this solution so it is up to you to find and/or implement code that works. Make sure you add a comment referencing any online solutions you use.

Turn-In Instructions:

Place all the files used in your solution into a zip file called Lab1Prob2.zip and upload to the Canvas assignment.

Grading Rubric

If a student's program runs correctly and produces the desired output, the student has the potential to get a 100 on his or her homework; however, TA's will **randomly** look through this set of "perfect-output" programs to look for other elements of meeting the lab requirements. The table below shows typical deductions that could occur.

⇨ ***Execution Time is important for this lab.***

AUTOMATIC GRADING POINT DEDUCTIONS PER PROBLEM:

Element	Percentage Deduction	Details
Files named incorrectly	10%	Per problem.
Execution Time	Up to 8%	0 pts deducted < 10x shortest time pts deducted = (2/10) (your time/shortest time) - 2 (rounded up to whole point value, with 4 pts maximum deduction for each problem)
Does Not Compile	30%	Code does not compile on PACE-ICE!
Does Not Match Output	10%-90%	The code compiles but does not produce the correct outputs.
Clear Self-Documenting Coding Styles	10%-25%	This can include incorrect indentation, using unclear variable names, unclear/missing comments, or compiling with warnings. (See Appendix A)

LATE POLICY

Element	Percentage Deduction	Details
Late Deduction Function	score – 0.5 * H	H = number of hours (ceiling function) passed deadline

***You are free to post solution times on pizza so that other students can gauge their run times.

Appendix A: Coding Standards

Indentation:

When using *if/for/while* statements, make sure you indent 4 spaces for the content inside those. Also make sure that you use spaces to make the code more readable.
For example:

```
for (int i; i < 10; i++)
{
    j = j + i;
}
```

If you have nested statements, you should use multiple indentions. Each { should be on its own line (like the *for* loop) If you have *else* or *else if* statements after your *if* statement, they should be on their own line.

```
for (int i; i < 10; i++)
{
   if (i < 5)
   {
       counter++;
       k -= i;
   }
   else
   {
       k +=1;
   }
   j += i;
}
```

Camel Case:

This naming convention has the first letter of the variable be lower case, and the first letter in each new word be capitalized (e.g. firstSecondThird). This applies for functions and member functions as well! The main exception to this is class names, where the first letter should also be capitalized.

Variable and Function Names:

Your variable and function names should be clear about what that variable or function is. Do not use one letter variables, but use abbreviations when it is appropriate (for example: “imag" instead of “imaginary”). The more descriptive your variable and function names are, the more readable your code will be. This is the idea behind self-documenting code.

File Headers:
Every file should have the following header at the top

```
/*
Author: your name
Class: ECE4122 or ECE6122 (section)
Last Date Modified: date

Description:

What is the purpose of this file?

*/
```

Code Comments:

1. Every function must have a comment section describing the purpose of the function, the input and output parameters, the return value (if any).
2. Every class must have a comment section to describe the purpose of the class.
3. Comments need to be placed inside of functions/loops to assist in the understanding of the flow of the code.