**ECE260: Fundamentals of Computer Engineering – Lab #1  
Review and Introduction to C/C++**

**Name \_\_\_\_Le Quient Lewis II\_\_\_ Lab Partner(s) \_\_\_\_\_\_\_N/A\_\_\_\_\_\_\_\_**

**1. Introduction**: This lab reviews topics covered in CS 101 and examines student knowledge of C++ topics. In this lab you will complete an implementation of [Conway’s Game of Life](https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life#Rules) that is partially started by the instructor. Through completing a working implementation, you will review previous programming skills and learn some new skills.

**2. Background**:

The Game of Life  
As part of this lab you will implement a version of Conway’s Game of Life. This game simulates population dynamics through a simple 2-D grid and basic evolution rules. A typical game consists of a grid where each cell is a location for life to be ALIVE or DEAD. Each cell is assigned a position (X, Y) as shown in Table 1.

Table 1: Cell numbering for Game of Life

|  |  |  |  |
| --- | --- | --- | --- |
| (0,0) | (1,0) | (2,0) | (3,0) |
| (0,1) | (1,1) | (2,1) | (3,1) |
| (0,2) | (1,2) | (2,2) | (3,2) |

At each position in the grid, a marker is placed to indicate whether the cell is alive or dead. In this lab, a hash (#) indicates that a cell is alive and a period (.) indicates the cell is dead. Thus, in Table 2, cells (0,0), (1,1), and (0,2) are alive, while the other cells are dead.

Table 2: Initial grid for cell evolution

|  |  |  |  |
| --- | --- | --- | --- |
| # | . | . | # |
| . | # | . | # |
| # | . | . | . |

Each game in the simulation evolves using simple rules based upon the number of adjacent neighbors. Each cell, except those on the edges, has eight immediate neighbors. For each cell its future state in the next evolution is determined by the following rules:

1. Any living cell with fewer than two live neighbors **dies**.
2. Any living cell with more than three live neighbors **dies**.
3. Any living cell with two or three live neighbors **stays alive**.
4. Any dead cell with exactly three live neighbors **becomes alive**.

One consequence of these rules is that the grid cannot be updated iteratively. More specifically, the future values for each cell (0,0), (0,1), etc. must be determined before the values in those cells are over-written. In our C/C++ implementation we will use two grids: *currentGrid* and *futureGrid* to represent the current and future values of the grid.

C/C++ Implementation

In the “src” folder, open the “game\_of\_life.cpp” file with Notepad++ (right click file and select “Edit with Notepad++”). A main() function provides for the initialization and evolution of the Game following the general algorithm:

1. Initialize random *currentGrid*
2. Perform N times
   1. evolve(*currentGrid, futureGrid*)
   2. printGrid(*futureGrid*)
   3. *temp* = *currentGrid*
   4. *currentGrid* = *futureGrid*
   5. *futureGrid* = *temp*

Two helper functions are used in 2a and 2b to perform the Game, evolve which computes the future grid using the current grid, and printGrid which prints a grid to the console. Steps 2c and 2e are required to update the current and future grids to be ready for the next iteration. A final helper function countNeighbors is implemented to make evolve easier to compute.

Each grid in the simulation is represented by a 2-dimensional array of bool values. Any cell in the grid can be accessed using the convention *grid[x][y]* as long as x and y are within the bounds of the grid. A bool value of *true* indicates the cell is ALIVE, while a *false* value indicates the cell is DEAD. Use the predefined values of ALIVE and DEAD throughout your code in lieu of using ‘*true*’ and ‘*false*’.

**3. Procedure**

This lab is an assessment of your C/C++ knowledge and work habits – not a programming test for C++. Read this lab manual and the included source code to prepare for the steps below. Additionally, you may use any references (online or otherwise) that you find helpful. However, you must specifically cite where your information comes from. The website <http://www.cplusplus.com/doc/tutorial/> is a great C++ reference and is acceptable for use in this course.

If you haven’t already, open the “game\_of\_life.cpp” file using Notepad++, or any other programming environment. The code provided is incomplete and requires you to implement specific functionality.

There are both programming elements and free-form questions within the lab. They can be answered in any order. The specific programming tasks are described in Table 3. The free form questions are listed below in Table 4.

**3.1 Programming Tasks**

Table 3: Programming Tasks

|  |
| --- |
| 1) Read <http://www.cplusplus.com/doc/tutorial/dynamic/> Section "Operators new and new[]". In the .cpp file, find the “**TODO: Task #1**”. Use the function new to initialize *currentGrid[x]* *and futureGrid[x]* as arrays of bool objects with length *depth*. |
| 2) Read <http://www.cplusplus.com/doc/tutorial/dynamic/> Section "Operators delete and delete[]”. In the .cpp file, find the “**TODO: Task #2**”. Use the function delete[] to release the memory associated with *currentGrid* and *futureGrid*. |
| 3) In the .cpp file, find the “**TODO: Task #3**”. Implement the function printGrid(bool\*\* grid) using *cout* to print output to the terminal. Iterate through the grid and print out the state of each cell in the array using the characters ALIVE\_ICON (‘#’) and DEAD\_ICON (‘.’) to identify alive and dead cells. Print a tab character (‘\t’) between each grid cell and a new line (‘\n’) at the end of each row. Print out the grid as shown in Table 2 where each row appears on its own new line. |
| 4) In the .cpp file, find the “**TODO: Task #4**”. Complete the implementation of evolve(bool\*\* current, bool\*\* future) to update the *future grid* based upon the current state of the *current grid* and using the number of alive neighbors provided by the countNeighbors function. |

3.2 Free form responses to Lab Questions

Use the space below to electronically write your answers. You may modify the table to suit your answer length.

Table 4: Free-form responses for Lab #1

|  |
| --- |
| 1) In the .cpp file, find the “**TODO: Question #1**”. Examine the #include statements at the beginning of the .cpp file. What is the purpose of these statements and which libraries are being included in this project?  [These libraries allow us to access the tools we need such as datatypes, Booleans, functions, etc… The standard library, input/output stream library, time library, and string libraries are used in this project.] |
| 2) In the .cpp file, find the “**TODO: Question #2**”. Some variables are declared using the modifiers *const* and *static*. What is the purpose of those modifiers and how do they affect the ability to read/write to the variables?  [Const modifiers modify the variable to be interpreted as an immutable constant and the static static modifier makes the variable stay populated for as long as the program runs.] |
| 3) In the .cpp file, find the “**TODO: Question #3**”. The countNeighbors() function checks to see which neighbors are alive in the grid. Explain why the if-statement immediately below the TODO is required and what would happen if it weren’t there?  [The if statement is required to check if xPrime and yPrime are populated with valid nonnegative value and values less than the width. If it weren’t there, it would count the array of invalid indexes.] |
| 4) In the .cpp file, find the “**TODO: Question #4**”. Use the website <http://www.cplusplus.com/reference/> to look up the definitions of srand() and time(). Explain the operation of srand() and why it needs to have a unique seed to provide random results. Explain how time() provides that unique seed value.  [srand() is a function that returns a pseudo-random value on a millisecond cycle, but if the time is nullified it just returns a random value on every call.] |
| 5) In the .cpp file, find the “**TODO: Question #5**”. Read <http://www.cplusplus.com/doc/tutorial/dynamic> Section "Operators new and new[]". Immediately below the TODO, both currentGrid and futureGrid are defined as arrays. What type of object do they contain?  [Boolean pointer objects are inside the currentGrid and futureGrid arrays.] |
| 6) In the .cpp file, find the “**TODO: Question #6**”. Read <http://www.cplusplus.com/doc/tutorial/operators/> about the ternary operator (?). Explain how the ternary operator is used to initialize the current and future grids.  [The ternary operator is a one line if statement that runs logic and depending on the logic it will assign or return a value  if true left of the colon or the other side if false.] |
| 7) In the .cpp file, find the “**TODO: Question #7**”. A temporary pointer is used to update the *future grid* the *current grid* for the next iteration. Using your knowledge of pointers explain why the following code is incorrect and the temp pointer is required.  currentGrid = futureGrid;  futureGrid = currentGrid;  [If temp is not used it we cannot easily use the value we removed from current grid. The value that was once in currentGrid gets overwritten by the value in futureGrid.] |
| 8) In the .cpp file, find the “**TODO: Question #8**”. Read [http://www.cplusplus.com/doc/tutorial/dynamic/](http://www.cplusplus.com/doc/tutorial/dynamic/%20) Section "Operators delete and delete[]". Explain why delete[] is required and what is being deleted in the for-loop just below the TODO.  [Delete is used to remove the old grids to free up space because the values stored in that memory is no longer needed.] |

**4. Submission**

When you have finished your lab, demo your program for your instructor. Write your answers to the above questions electronically in this document. To submit your lab assignment, make sure your files have all been saved (*including this file*). In a Cygwin window type the commands:

cd h:  
cd ECE260  
cd Lab01\_Game\_of\_Life  
make submit

Enter your Marmoset username and password (which you should have received by email). Note that your password will not be echoed to the screen. Make sure that after you enter your username and password, you see a message indicating that the submission was successful. Log into [Marmoset](https://cs.ycp.edu/marmoset/login) via the web to check the files you submitted to ensure they are correct.

**DO NOT MANUALLY ZIP YOUR PROJECT AND SUBMIT IT TO MARMOSET. IT WILL NOT BE GRADED!!!  
YOU MUST USE THE make submit COMMAND.**