National University of Singapore
School of Continuing and Lifelong Education
TIC1002: Introduction to Computing and Programming II
Semester II, 2019/2020

<u>Alien in Space! [10 marks]</u>

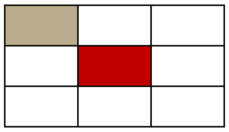
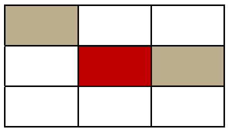
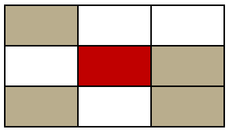
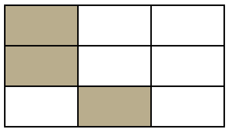
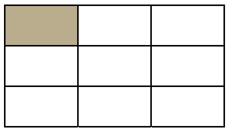
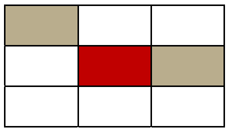
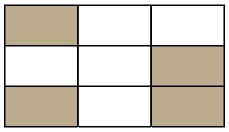
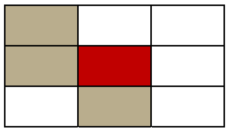
In this task, we are going to simulate how alien evolve on a planet! The alien planet is represented as an **<u>20 x 20, 2D character array</u>**. Each location (cell) represents whether there is a living alien lifeform on that spot: **'X'** represents a live alien, **'O'** means the location is empty.

Lifeform Evolution on Alien Planet

Each cell in the world can contain a live or a dead (i.e. empty) lifeform. To evolve from one generation **G** to the next **G+1** generation, each cell will interacts with its eight *neighbour cells*, which are the cells that are horizontally, vertically, or diagonally adjacent. The following rules[1] are used to determine the status of the cell **<u>in the next generation</u>**:

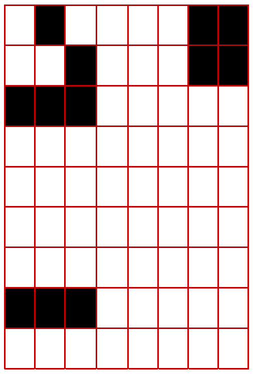
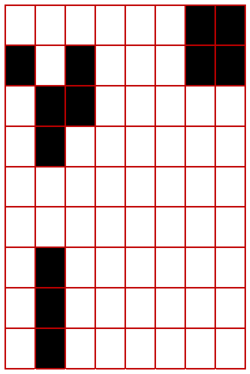
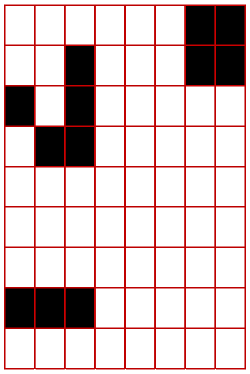
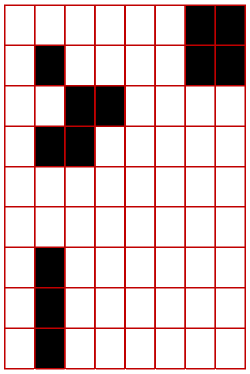
1. [**Under-population**] Any live cell with fewer than two live neighbours dies.
2. [**Survive**] Any live cell with two or three live neighbours lives on to the next generation.
3. [**Overpopulation**] Any live cell with more than three live neighbours dies.
4. [**Reproduction**] Any dead cell with exactly three live neighbours becomes a live cell.

For example, the middle cell (shaded red) can change according to the rules from one generation to the next (Shaded cell = alive, Non-shaded cell = dead (empty)):

	Rule 1	**Rule 2**	**Rule 3**	**Rule 4**
Generation G				
Generation G+1				

[1] This is essentially **Conway's Game of Life** ruleset

Note that the rules are applied to every single cell in the world. For example, below showed 3 rounds of evolutions of a 9 x 8 world:

Generation 0	**Generation 1**	**Generation 2**	**Generation 3**
			

Given function: `void init_alien_planet( const char filename[],`
`char alienPlanet[][20]);`

Function to open and read from a text file with `filename` to initialize the alien planet (the 2D array). The file contains **20 lines where each line contain 20 characters ('O' or 'X').** The information represent the "Generation 0" of the planet.

For example, `init_alien_planet( "planet_sample.txt", myPlanet);` should read from the sample file "`planet_sample.txt`" and initialize `myPlanet`, which should be declared as 20 x 20, 2D character array.

This function has already been implemented for you. Use it to review / learn C++ file stream so that you can implement the `save_alien_planet()` function below.

Deliverable: `void evolve_alien_planet(`
`char alienPlanet[][20], int nGeneration);`

Function evolve the planet according to the rules for `nGeneration` generations.

For example, `evolve_alien_planet( myPlanet, 3);` should evolve the planet for **3 rounds, (i.e. generation 0 → generation 3).**

Deliverable: `void save_alien_planet( const char filename[],`
`char alienPlanet[][20]););`

Function save (print out) the alien planet information into a file with `filename`. The output file format should be the same as the input file, i.e. 20 lines of 20 characters each.

Note: We will check this part by using the output file generated via `save_alien_planet`(), so make sure it is implemented **correctly with exact formatting**.

You are strongly encouraged to write additional helper functions, e.g. printing function for checking this part. The size 20 x 20 is chosen so that the planet readable on screen.

We have generated the first 3 generation of the planet for your reference. They are named " `"planet_sample_gen_1.txt"`, `"planet_sample_gen_2.txt"` and `"planet_sample_gen_3.txt"`

**~~~ End of Task ~~~**