Assignment 6: Conway's Game of Life

Course 'Imperative Programming' (IPC031)

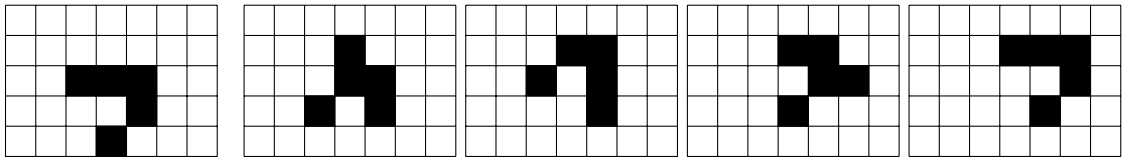
Make sure to start working on the assignment **before the lab** session,
otherwise you will be too late for the deadline

1 Background

In this assignment you work with text files and one-dimensional and two-dimensional arrays. You develop an implementation for a theoretically interesting problem known as the "Game of Life", introduced by John Conway in 1970. Conway's problem illustrates the concept of a cellular automaton invented by John von Neuman, in the 1940s. The "Game of Life" is played in a two-dimensional grid of cells, called the universe. It is played by creating an initial configuration and observing how this configuration evolves step by step. Each cell of the universe is in one of two possible states: dead or alive. Every cell interacts with its eight neighbors, which are the cells that are horizontally, vertically, or diagonally adjacent to it. In each step, a transition occurs that determines the state of the cells in the next configuration of the universe, on the basis of their state in the current configuration. The transition rules are amazingly simple:

1. A live cell with fewer than two live neighbors dies, as if caused by under-population.
2. A live cell with two or three live neighbors lives on to the next generation.
3. A live cell with more than three live neighbors dies, as if by overcrowding.
4. A dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.
   Otherwise the cell remains dead.

Despite their simplicity, these rules can lead to surprisingly complex configurations of live cells. Here is a small example of a frequently occurring pattern (the glider pattern):



initial configuration | after one step | after two steps | after three steps | after four steps

Before going any further, use desktop testing and reproduce the transitions in the diagrams (check for at least one complete transition).

On Brightspace the file "`assignment-06-mandatory-files.zip`" contains a number of text files with "Game of Life" configurations. The above sequence is found in the files "`glider0.txt`" up to "`glider4.txt`". The other patterns are "`10_cell_row.txt`", "`pulsar.txt`", and "`Gosper_glider_gun.txt`". The text files are formatted as follows: there is a fixed number of text lines (`NO_OF_ROWS`). Each line has the same number of cell characters (`NO_OF_COLUMNS`). Each cell character is either `'*'` indicating a live cell, or `'.'`, indicating a dead cell. Additionally, there is a "`main.cpp`" that you extend in this assignment, and "`main_test.cpp`" that contains the unit tests.
The universe in Conway's Game of Life is unbounded. This is a luxury that we can not afford. In this assignment, a bounded universe is represented by means of a two-dimensional array that contains `NO_OF_ROWS` rows of `NO_OF_COLUMNS` `Cell` values:

```
enum  Cell {Dead=0, Live};     // a cell is either Dead or Live (we rely on the fact that Dead has a zero value)
const char DEAD         = '.'; // the representation of a dead cell (both on file and screen)
const char LIVE         = '*'; // the representation of a live cell (both on file and screen)
const int NO_OF_ROWS    = 40;  // number of rows   (height) of the universe (both on file and screen)
const int NO_OF_COLUMNS = 60;  // number of columns (width) of the universe (both on file and screen)
```

Any cell 'outside' of the bounded universe is considered to have value `Dead`. Hence, all cells at the edges of the bounded universe have at least three `Dead` neighbor cells 'outside' of the bounded universe (five of them for corner cells, and three of them for other edge cells).

2 Learning objectives

After doing this assignment you are able to:

- Work with text files for input and output purposes;
- Work with arrays.

3 Assignment

Part 1: Get cell in bounded universe

Design and implement a function that returns the cell value of a bounded universe at some given coordinate, if the coordinate is a valid value. If the coordinate is not a valid value (it is 'outside' of the bounds of the bounded universe), the function returns the cell value `Dead`.

```
Cell cell_at (Cell universe [NO_OF_ROWS][NO_OF_COLUMNS], int row, int column)
```

Part 2: Setting the scene

Design and implement a function that displays the universe on the console and a function that reads a "Game of Life" configuration from a file (*e.g.*, "`glider0.txt`").

```
void show_universe (Cell universe [NO_OF_ROWS][NO_OF_COLUMNS])
bool read_universe_file (string filename, Cell universe [NO_OF_ROWS][NO_OF_COLUMNS])
```

The function `show_universe (universe)` displays the rows of `universe` one by one, each on a new line. Cells of value `Dead` must be displayed as the character `DEAD`, and cells of value `Live` as the character `LIVE`. The function `read_universe_file (filename, universe)` attempts to open the text file with file name `filename` and, if successful, attempt to read the content of that text file and fill the `universe` bounded array representation, line by line in the rows of `universe`. The `bool` result is true if and only if no error has occurred. Test your implementation by reading a universe configuration file from "`assignment-06-mandatory-files.zip`" and showing the universe on the console.

Part 3: The next generation

Design and implement a function that computes the next configuration of the universe according to the "Game of Life" rules.

```
void next_generation (Cell now [NO_OF_ROWS][NO_OF_COLUMNS], Cell next [NO_OF_ROWS][NO_OF_COLUMNS])
```

The array `now` is the current configuration of the universe, the array `next` contains the next configuration of the universe. Use `cell_at` (part 1) to retrieve the correct (neighbor) cell values in `now`. Test your implementation by reading a universe configuration file from "`assignment-06-mandatory-files.zip`", computing one transition step, and showing the resulting universe on the console with your `main` function.

4 Products

As product-to-deliver you only need to upload to Brightspace "`main.cpp`" and "`main_test.cpp`".

Deadline

Lab assignment: Friday, October 13, 2023, 23:59h