CS256 — Exercise 7

February 15, 2016

Due: Monday, February 22, 2016 before midnight (60 points)

1 Preparing the Project

1. Go to https://codebank.xyz and you should see a project in the CS256 group named `CS256-EX7`. Fork this project for your own user.
2. Next, clone the repository so you have a local copy:

   ```
   $ git clone https://codebank.xyz/username/CS256-EX7.git
   ```

3. The repository should have two C++ files: `Conway.cpp`, `Conway.h` along with several test input files.

2 The `Conway` class

In this exercise, we will implement a cellular automaton known as Conway's Game of Life. It works as follows: there is a 2D grid of living and dead cells. The game iterates by looking at each cell and its neighbors and then either bringing the cell to life or killing it. Over time, this can lead to some interesting results.

We will implement the game with a class called `Conway` that has the following:

1. A 2D array of `bool`s called `board` that will store our cells. When a cell is `true` we consider it alive and when it is `false` we consider it dead.
2. Two `int` variables named `rows` and `cols` that are the number of rows and columns in `board` respectively.

You must also implement the following functions:

1. A constructor that takes a `std::string` as a file name that will read the content of the file and construct our game of life. The files will have the following format: they start with two integers that indicate the number of rows and columns. They then have that many integers that are all either 0 or 1 indicating if the cells are alive or dead.

   For example, this would be a possible input file:

   ```
   5 5
   0 0 0 0 0
   0 0 1 0 0
   0 0 1 0 0
   0 0 1 0 0
   0 0 0 0 0
   ```

2. A copy constructor.
3. A destructor that properly deallocates the memory for our game board.

4. An overloaded assignment operator for copying.
5. A function `bool alive(int row, int col) const;` that will return whether or not the given cell is alive.
6. A function `void flip(int row, int col);` that will flip the given cell (i.e., if it's alive, kill it and if it's dead, bring it to life).
7. A function `std::string str() const;` that generates a `std::string` representation of our game board. This function should make a 2D grid of `.` and `X` where a `.` is a dead cell and a `X` is a living cell.
8. A function `void step();` that will play one iteration of the game.
9. A function `void play(int n);` that will play `n` iterations of the game. Print the initial board using the `str()` function and the board after each iteration.
10. You may add additional functions if they would be helpful for implementing the above.

The game is played in iterations. On each iteration, you must check every cell in the board. For each of those cells, apply the following rules:

1. Any living cell with fewer than two living neighbors dies.
2. Any living cell with two or three living neighbors lives on.
3. Any living cell with more than three living neighbors dies.
4. Any dead cell with exactly three living neighbors comes to life.

We will define "neighbor" as the (up to) 8 cells around the cell. For example, if the cell is the `X` below, then each cell labeled `N` is its neighbor:

```
. . . . .
. N N N .
. N X N .
. N N N .
. . . . .
```

Be careful! When a cell is on an edge, we won't have eight neighbors. Make sure you do proper bounds-checking in your program. For example, in a corner we have only three neighbors:

```
X N . . .
N N . . .
. . . . .
```

On a side we have only five neighbors:

```
. N X N . . .
. N N N . . .
. . . . . . .
```

Note: be careful when updating the board for each iteration. Don't change the existing board because that would affect the other cells you have not yet handled for the current iteration. You should modify a copy of the board, then replace the old board with the copy.

We will start with an initial pattern, called the *seed*, that is stored in the input files. Several patterns were provided in the original repository.

In addition to the `Conway` class, provide a `main` function either in a separate `cpp` file or at the bottom of `Conway.cpp` that uses command line arguments to play the game. The first argument passed should be a file name corresponding to the input file to use and the second argument should be a positive integer corresponding to the number of iterations to play.

3 Submission

Once you have completed the program, you can use `git add`, `git commit`, and `git push` to push the changes to `https://codebank.xyz`. You can make as many commits and push as many times as desired until the deadline.