## Worksheet for Lab #5 Ex3: Game of Life

http://www.comp.nus.edu.sg/~cs1010/labs/2017s1/lab5/2D\_arrays.html

## **Task Statement**

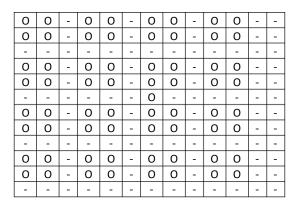
Implement the *Game of Life* in a  $20 \times 20$  community. Life cells are represented by 'O', while dead cells by '-'. The rules are as follows:

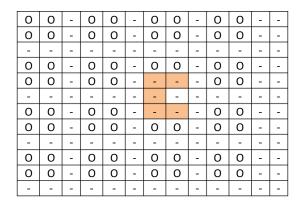
- A live cell will die of loneliness in the next generation if it has fewer than 2 neighbours in the current generation.
- A live cell will die of overcrowding in the next generation if it has more than 3 neighbours in the current generation.
- A live cell will remain as a live cell in the next generation if it has either 2 or 3 neighbours in the current generation.
- A dead cell will become alive in the next generation if it has exactly 3 neighbours in the current generation. All other dead cells will remain dead in the next generation.
- All births and deaths take place instantaneously.

In this worksheet, we will explore only one part of the task — to generate the next generation. As sentinels are used, we will see how sentinels can make our code neater.

## I. Without sentinels

Here, we use a  $20 \times 20$  character array **currentGen** to represent the current community, and a  $20 \times 20$  character array **nextGen** for the next community. We use **life14.in** here, but shift the community to the top-left. We show the top 12 rows and left-most 13 columns in the diagrams below.





**Current generation** 

Next generation: changed cells are highlighted.

The partial algorithm to generate the next generation is shown below. We will focus on the **countNeighbours()** function to return the number of neighbours of a cell at currentGen[r][c].

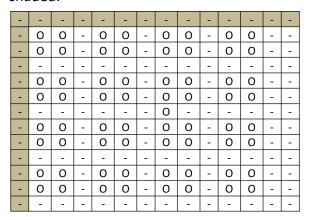
```
for (r = 0; r < 20; r++) { // for II: for (r = 1; r <= 20; r++)
  for (c = 0; c < 20; c++) { // for II; for (c = 1; c <= 20; c++)
     numNeighbours = countNeighbours(currentGen, r, c);
     ...
}
</pre>
```

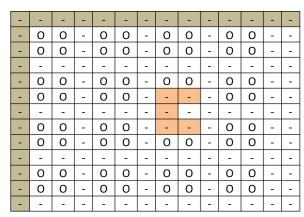
Write your algorithm for **countNeighbours()** below:

```
int countNeighbours(char arr[][20], int rPos, int cPos) {
```

## II. With sentinels

Now, with a boundary of sentinels (dead cells) around the community, we have a  $22 \times 22$  character array **currentGen** to represent the current community, and a  $22 \times 22$  character array **nextGen** for the next community. We use the same example as above. We show the top 13 rows and left-most 14 columns in the diagrams below. The boundary cells are shaded.





Current generation

Next generation: changed cells are highlighted.

Write your algorithm for countNeighbours() below. Compare it with the other version.