lab11

```
$ gcc life.c main.o
$ ./a.out < pat1.dat
Generation 30
. . O . . . O . . . . . O . . . O . . .
. . O O . . O . . . . . O . . O O . . .
. . . . . . O . . . . . O . . . . . . .
O O . . . . O O . . . O O . . . . O O .
. O . . O . . . . . . . . . O . . O . .
. . O . O . . . . . . . . . O . O . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . O . O . . . . . . . . . O . O . . .
. O . . O . . . . . . . . . O . . O . .
O O . . . . O O . . . O O . . . . O O .
. . . . . . O . . . . . O . . . . . . .
. . O O . . O . . . . . O . . O O . . .
. . O . . . O . . . . . O . . . O . . .
. . O . . . O . . . . . O . . . O . . .
. . O . . . O . . . . . O . . . O . . .
```

```
CPU time: 0.00667808 sec
score: 87
o. [Output] Program output is correct, good.
o. [Format] Program format can be improved
o. [Coding] life.c spelling errors: abd(1), ilfe(1)
o. [Efficiency] can still be improved.
```

life.c

```c
// EE231002 Lab11. Game of Life
// 109061158, 簡佳吟
// Date: 2020/12/14


#include "life.h"

// This function reads the initial pattern and store it to the next member
// of each cell.
// It also initializes the cell contents to ensure
// proper execution of the program
void readGrid(CELL grid[N][N])
{

    char ch;        // for reading each element
    int i, j;       // index for loop

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            grid[i][j].row = i;         // initialize the structure grid
            grid[i][j].col = j;
            grid[i][j].age = 0;
            grid[i][j].Nnbr = 0;
            grid[i][j].current = DEAD;

            scanf(" %c", &ch);              // read
            if (ch == '.') {
                grid[i][j].next = DEAD;     // dot represents DEAD
                grid[i][j].color = WHITE;   // it is white
            }
            else if (ch == 'O') {           // O represents LIVE
                grid[i][j].next = LIVE;
                grid[i][j].color = GREEN;   // it is green
            }
        }
    }
}
// This function checks for still ilfe pattern by comparing the cell members
// current and next.
// If a still pattern is found, it returns 1 otherwise it returns 0.
// Before returning, the cell status should also be updated,
// that is, the next state is copied the current state
int stillLife(CELL grid[N][N])
{

    int i, j;                   // index for loop
    int notsame = 0;            // for checking whether the current and the next
                                // is not same
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            if (grid[i][j].current != grid[i][j].next) {
                notsame++;                              // check the state
```

This line has more than 80 characters

```
            }
            grid[i][j].current = grid[i][j].next;       // copy the next state
                                                        // to the current state
        }
    }
    if (notsame != 0) return 0;         // if there exists different elements
                                        // return 0
    else return 1;                      // otherwise return 1
}
```

Need a blank line here.

```
// This function determines the status of each cell according to the rules
// given above.
// Other structure members, such as age abd color, should also be updated.
void nextGen(CELL grid[N][N])
{
    int i, j;               // index for loop
    int m, n;               // index for loop
    int r[3];               // array for recording row
    int c[3];               // array for recording column


    for (i = 0; i < N; i++) {
        if (i == 0) {                   // initialize r array
            r[0] = N - 1;
            r[1] = 0;
            r[2] = 1;
        }
        else if (i == N - 1) {
            r[0] = N - 2;
            r[1] = N - 1;
            r[2] = 0;
        }
        else  {
            for (m = 0; m < 3; m++) {
                r[m] = i - 1 + m;
            }
        }
        for (j = 0; j < N; j++) {
            grid[i][j].Nnbr = 0;    // reset grid[i][j].Nnbr
            if (j == 0) {           // initialize c array
                c[0] = N - 1;
                c[1] = 0;
                c[2] = 1;
            }
            else if (j == N - 1) {
                c[0] = N - 2;
                c[1] = N - 1;
                c[2] = 0;
            }
            else {
                for (m = 0; m < 3; m++) {
                    c[m] = j - 1 + m;
                }

            }
            for (m = 0; m < 3; m++) {
                for (n = 0; n < 3; n++) {
                    if (grid[r[m]][c[n]].current == LIVE) {
                        grid[i][j].Nnbr++;      // record the number of neighbor
                    }
                }
            }
            if (grid[i][j].current == LIVE) {
                grid[i][j].Nnbr--;              // discard itself
            }
            // the condition of DEAD cell turning to LIVE
            if (grid[i][j].current == DEAD) {
                if (grid[i][j].Nnbr == 3) {
                    grid[i][j].age++;
                    grid[i][j].next = LIVE;
                    grid[i][j].color = GREEN;

                }
            }

            else if (grid[i][j].current == LIVE){
            else if (grid[i][j].current == LIVE) {
                // the condition of LIVE cell still LIVE
                if (grid[i][j].Nnbr == 2 || grid[i][j].Nnbr == 3) {
                    grid[i][j].age++;
                    grid[i][j].next = LIVE;
                        switch (grid[i][j].age) {   // change its color
                    switch (grid[i][j].age) {   // change its color
                            case 1: grid[i][j].color = GREEN; break;
                        case 1: grid[i][j].color = GREEN; break;
                            case 2: grid[i][j].color = YELLOW; break;
                        case 2: grid[i][j].color = YELLOW; break;
                            default: grid[i][j].color = RED;
                        default: grid[i][j].color = RED;
                        }
                    }
                }
                // the condition of LIVE turning to DEAD cell
                if (grid[i][j].Nnbr < 2 || grid[i][j].Nnbr > 3) {
                    grid[i][j].next = DEAD;
                    grid[i][j].age = 0;
                    grid[i][j].color = WHITE;
                }
            }
        }
    }
}

```