



- 131 Removals + 100 Additions

1

```

1 #include <stdio.h>
2
3 #define ROW 40
4 #define COLUMN 80
5
6 unsigned char matrix[ROW][COLUMN];
7 FILE* file;
8
9 void openFile()
10 {
11     short x = 0;
12     short y = 0;
13     short z = 0;
14     short bit;
15     unsigned char line[400];
16
17     fread(&line, sizeof(char), 400, file);
18
19     //Iterate through the y-axis of the matrix.
20     for(y = 0; y < ROW; y++)
21     {
22         //Iterate through the x-axis of the matrix.
23         for(x = 0; x < COLUMN; x++)
24         {
25             /*
26              * This loop goes through each line element (8bit) and perform bit operation ~> shifts right and &
27              * with 128 to isolate each bit.
28              * Afterwards, it assigns the value into the matrix.
29              */
30             matrix[y][x] = (line[z] & (128 >> bit));
31             bit++;
32             if(bit == 8)
33             {
34                 bit = 0;
35                 z++;
36             }
37         }
38     }
39     fclose(file);
40
41     //This function check the matrix element for 1(alive) and replaces it with "0", otherwise it will be " ".
42     char* cellAlive(int y, int x)
43     {
44         if(matrix[y][x])
45             return "0";
46         else
47             return " ";
48     }
49 }

```

```

1 #include <stdio.h>
2
3 #define ROW 40
4 #define COLUMN 80
5
6 unsigned char matrix[ROW][COLUMN];
7 FILE* file;
8
9 void openFile() {
10     short x = 0;
11     short y = 0;
12     short z = 0;
13     short bit;
14     unsigned char line[400];
15
16     fread(&line, sizeof(char), 400, file);
17
18     // Iterate through the rows of the matrix.
19     for (y = 0; y < ROW; y++) {
20         // Iterate through the columns of the matrix.
21         for (x = 0; x < COLUMN; x++) {
22             // Here we do some magic
23             matrix[y][x] = (line[z] & (128 >> bit));
24
25             bit++;
26             if(bit == 8)
27             {
28                 bit = 0;
29                 z++;
30             }
31         }
32     }
33     fclose(file);
34
35     //This function check the matrix element for 1(alive) and replaces it with "0", otherwise it will be " ".
36     char* cellAlive(int y, int x)
37     {
38         if(matrix[y][x])
39             return "0";
40         else
41             return " ";
42     }
43 }

```

```

48 }
49
50 //This function prints the content of the matrix.
51 void printGrid()
52 {
53     int x;
54     int y;
55     for(y = 0; y < ROW; y++)
56     {
57         for(x = 0; x < COLUMN; x++)
58         {
59             printf("%s", cellAlive(y, x));
60         }
61         printf("\n");
62     }
63 }
64
65 //This
66 function checks the surrounding cells. Counter only increments if it
67 is within the matrix and not the original position.
68 int cellCheck(int y, int x)
69 {
70     int counter = 0;
71     signed int horizontal;
72     signed int vertical;
73     for(vertical = -1; vertical <= 1; vertical++)
74     {
75         for(horizontal = -1; horizontal <= 1; horizontal++)
76         {
77             if((horizontal || vertical) && (horizontal + x < COLUMN && horizontal + x >= 0) && (vertical + y < ROW && vertical + y >= 0))
78             {
79                 if(matrix[y + vertical][x + horizontal]) counter++;
80             }
81         }
82     }
83     return counter;
84 }
85
86 /*
87 This function changes the matrix based on the rules established:
88 1. Any live cell with fewer than two neighbors is dead in the next generation.

```

```

24     bit++;
25     if (bit == 8) {
26         bit = 0;
27         z++;
28     }
29 }
30 }
31 fclose(file);
32 }
33
34 // This function check the matrix element for 1(alive) and replaces it with "X",
35 // otherwise it will be " ".
36 char* cellAlive(int i, int j) { return matrix[i][j] ? "X" : " "; }
37
38 // This function prints the content of the matrix.
39 void printGrid() {
40     int x;
41     int y;
42     for (y = 0; y < ROW; y++) {
43         for (x = 0; x < COLUMN; x++) {
44             printf("%s", cellAlive(y, x));
45         }
46         printf("\n");
47     }
48 }
49
50 // This
51 function checks the surrounding cells. Counter only increments if it
52 is within the matrix and not the original position.
53 int cellCheck(int y, int x) {
54     int counter = 0;
55     signed int horizontal;
56     signed int vertical;
57     for (vertical = -1; vertical <= 1; vertical++) {
58         for (horizontal = -1; horizontal <= 1; horizontal++) {
59             if ((vertical || horizontal) &&
60                 (x + horizontal < COLUMN && x + horizontal >= 0) &&
61                 (y + vertical < ROW && y + vertical >= 0)) {
62                 if (matrix[vertical + y][horizontal + x]) counter++;
63             }
64         }
65     }
66     return counter;
67 }
68
69 /**
70  * We generate a generation here
71  */
72 void generation(int turn) {

```

```

2. Any live cell with more than three neighbors is dead in the next generation.
89 3. Any live cell with two or three neighbors survives.
90 4. Any empty cell with exactly three neighbors becomes live in the next generation.
91 5. Any empty cell with a number of neighbors not equal to three remains empty.
92 */
93 void generation(int turn)
94 {
95     unsigned char tempMatrix[ROW][COLUMN];
96     int currentTurn;
97     int x;
98     int y;
99     int counter;
100
101     for(currentTurn = 0; currentTurn < turn; currentTurn++)
102     {
103         for(y = 0; y < ROW; y++)
104         {
105             for(x = 0; x < COLUMN; x++)
106             {
107                 counter = cellCheck(y, x);
108                 switch (counter)
109                 {
110                     case 2:
111                         tempMatrix[y][x] = matrix[y][x];
112                         break;
113                     case 3:
114                         tempMatrix[y][x] = 1;
115                         break;
116                     default:
117                         tempMatrix[y][x] = 0;
118                 }
119             }
120         }
121         for(y = 0; y < ROW; y++)
122         {
123             for(x = 0; x < COLUMN; x++)
124             {
125                 matrix[y][x] = tempMatrix[y][x];
126             }
127         }
128     }
129 }
130
131 int main(int argc, char* argv[])
132 {
133     if(argc != 3)
134     {
135         printf("Please supply file and number of generations in that order.\n");
136         return 1;
137     }
138     else
139     {

```

```

73     unsigned char tempMatrix[ROW][COLUMN];
74
75     int currentTurn = 0;
76
77     int x;
78     int y;
79     int counter;
80
81     while (currentTurn < turn) {
82         for (y = 0; y < ROW; y++) {
83             for (x = 0; x < COLUMN; x++) {
84                 counter = cellCheck(y, x);
85                 switch (counter) {
86                     case 2:
87
88                         tempMatrix[y][x] = matrix[y][x];
89                         break;
90                     case 3:
91                         tempMatrix[y][x] = 1;
92                         break;
93                     default:
94                         tempMatrix[y][x] = 0;
95                 }
96             }
97         }
98         for (y = 0; y < ROW; y++) {
99             for (x = 0; x < COLUMN; x++) {
100                 matrix[y][x] = tempMatrix[y][x];
101             }
102         }
103         currentTurn++;
104     }
105
106     int main(int argc, char* argv[]) {
107         if (argc == 3) {
108             file = fopen(argv[1], "r");
109             openFile();
110
111             generation(atoi(argv[2]));
112             printGrid();
113             return 0;
114         } else {

```

```
140     file = fopen(argv[1], "r");
141     openFile();
142     generation(atoi(argv[2]));
143     printGrid();
144     return 0;
145 }
146 }
```

```
    printf("Please supply file and number of generations in
that order.\n");
113     return 1;
114 }
115 }
```