Diffchecker

- 131 Removals
+ 100 Additions

1.c

```
#include <stdio.h>

#define ROW 40
#define COLUMN 80

unsigned char matrix[ROW][COLUMN];
FILE* file;

void openFile()
{
        short x = 0;
        short y = 0;
        short z = 0;
        short bit;
        unsigned char line[400];

        fread(&line, sizeof(char), 400, file);

        //Iterate through the y-axis of the matrix.
        for(y = 0; y < ROW; y++)
        {
                //Iterate through the x-axis of the matrix.
                for(x = 0; x < COLUMN; x++)
                {
                        /*
                        This loop goes through each line element (8bit) and perform bit operation ~> shifts right and & with 128 to isolate each bit.
                        Afterwards, it assigns the value into the matrix.
                        */
                        matrix[y][x] = (line[z] & (128 >> bit));
                        bit++;
                        if(bit == 8)
                        {
                                bit = 0;
                                z++;
                        }
                }
        }
        fclose(file);
}

//This function check the matrix element for 1(alive) and replaces it with "O", otherwise it will be " ".
char* cellAlive(int y, int x)
{
        if(matrix[y][x])
                return "O";
        else
                return " ";
```

```
#include <stdio.h>

#define ROW 40
#define COLUMN 80

unsigned char matrix[ROW][COLUMN];
FILE* file;

void openFile() {
  short x = 0;
  short y = 0;
  short z = 0;
  short bit;
  unsigned char line[400];

  fread(&line, sizeof(char), 400, file);

  // Iterate through the rows of the matrix.
  for (y = 0; y < ROW; y++) {
    // Iterate through the columns of the matrix.
    for (x = 0; x < COLUMN; x++) {
      // Here we do some magic
      matrix[y][x] = (line[z] & (128 >> bit));
}

//This function prints the content of the matrix.
void printGrid()
{
        int x;
        int y;
        for(y = 0; y < ROW; y++)
        {
                for(x = 0; x < COLUMN; x++)
                {
                        printf("%s", cellAlive(y, x));
                }
                printf("\n");
        }
}

//This function checks the surrounding cells. Counter only increments if it is within the matrix and not the original position.
int cellCheck(int y, int x)
{
        int counter = 0;
        signed int horizontal;
        signed int vertical;

        for(vertical = -1; vertical <= 1; vertical++)
        {
                for(horizontal = -1; horizontal <= 1; horizontal++)
                {
                        if((horizontal || vertical) && (horizontal + x < COLUMN && horizontal + x >= 0) && (vertical + y < ROW && vertical + y >= 0))
                        {
                                        if(matrix[y + vertical][x + horizontal]) counter++;
                        }
                }
        }
        return counter;
}

/*
This function changes the matrix based on the rules established:
1. Any live cell with fewer than two neighbors is dead in the next generation.

```

```
      bit++;
      if (bit == 8) {
        bit = 0;
        z++;
      }
    }
  }
  fclose(file);
}

// This function check the matrix element for 1(alive) and replaces it with "X",
// otherwise it will be " ".
char* cellAlive(int i, int j) { return matrix[i][j] ? "X" : " "; }

// This function prints the content of the matrix.
void printGrid() {
  int x;
  int y;
  for (y = 0; y < ROW; y++) {
    for (x = 0; x < COLUMN; x++) {
      printf("%s", cellAlive(y, x));
    }
    printf("\n");
  }
}

// This function checks the surrounding cells. Counter only increments if it is
// within the matrix and not the original position.
int cellCheck(int y, int x) {
  int counter = 0;
  signed int horizontal;
  signed int vertical;

  for (vertical = -1; vertical <= 1; vertical++) {
    for (horizontal = -1; horizontal <= 1; horizontal++) {
      if ((vertical || horizontal) &&
          (x + horizontal < COLUMN && x + horizontal >= 0) &&
          (y + vertical < ROW && x + vertical >= 0)) {
        if (matrix[vertical + y][horizontal + x]) counter++;
      }
    }
  }
  return counter;
}

/**
 * We generate a generation here
 */
void generation(int turn) {
```

```
2. Any live cell with more than three neighbors is dead in the next generation.
3. Any live cell with two or three neighbors survives.
4. Any empty cell with exactly three neighbors becomes live in the next generation.
5. Any empty cell with a number of neighbors not equal to three remains empty.
*/
void generation(int turn)
{
        unsigned char tempMatrix[ROW][COLUMN];
        int currentTurn;
        int x;
        int y;
        int counter;
        
        for(currentTurn = 0; currentTurn < turn; currentTurn++)
        {
                for(y = 0; y < ROW; y++)
                {
                        for(x = 0; x < COLUMN; x++)
                        {
                                counter = cellCheck(y, x);
                                switch (counter)
                                {
                                        case 2:
                                                tempMatrix[y][x] = matrix[y][x];
                                                break;
                                        case 3:
                                                tempMatrix[y][x] = 1;
                                                break;
                                        default:
                                                tempMatrix[y][x] = 0;
                                }
                        }
                }
                for(y = 0; y < ROW; y++)
                {
                        for(x = 0; x < COLUMN; x++)
                        {
                                matrix[y][x] = tempMatrix[y][x];
                        }
                }
        }
}

int main(int argc, char* argv[])
{
        if(argc != 3)
        {
                printf("Please supply file and number of generations in that order.\n");
                return 1;
        }
        else
        {
```

```
  unsigned char tempMatrix[ROW][COLUMN];
  int currentTurn = 0;
  int x;
  int y;
  int counter;

  while (currentTurn < turn) {
    for (y = 0; y < ROW; y++) {
      for (x = 0; x < COLUMN; x++) {
        counter = cellCheck(y, x);
        switch (counter) {
          case 2:
            tempMatrix[y][x] = matrix[y][x];
            break;
          case 3:
            tempMatrix[y][x] = 1;
            break;
          default:
            tempMatrix[y][x] = 0;
        }
      }
    }
    for (y = 0; y < ROW; y++) {
      for (x = 0; x < COLUMN; x++) {
        matrix[y][x] = tempMatrix[y][x];
      }
    }
    currentTurn++;
  }
}

int main(int argc, char* argv[]) {
  if (argc == 3) {
    file = fopen(argv[1], "r");
    openFile();
    generation(atoi(argv[2]));
    printGrid();
    return 0;
  } else {

```

```
                file = fopen(argv[1], "r");
                openFile();
                generation(atoi(argv[2]));
                printGrid();
                return 0;
        }
}
```

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