Diffchecker

- 195 Removals + 108 Additions

```
/**
Name: Phoenix Bishea
Eid: pnb338

Q1: Why are the arrays passed to both print life functions c
onst?

    The arrays passed to print life are not being directly m
odified.
    Since the value never changes it can be declared const.

Q2: Why are the arrays passed to both play life functions no
t const?

    The arrays passed to play life are not const because the
y are modified.
    Play life changes the cells in the array every iteration
depending
    on the rules of the game of life.

Q3: If we did change the arrays passed to play life function
s to make them const,
    what else would have to happen to make play life work?

    Under these circumstances an additional array would have
to be created and returned
    from play life to main to make play life work. The array
passed to play life could not
    be modified.
*/

#include <iostream>

using namespace std;

/// global constant for fixed rows and cols of the 2D array
const int NUM_ROWS = 10;
const int NUM_COLS = 10;
const int MIN = 0;
const int MAX = 9;

/** function declarations
    you will need to write the definitions of these function
s below.
    DO NOT MODIFY the declarations.
    You may create your own functions, but you must use thes
e.
*/

/** 2d array functions, notice that for 2d arrays the second
size
    needs to be fixed, or more precisely it needs to be a co
nst.
    if you typed a literal, like 5, it would be a literal co
nstant,
    which also works.
```

```
/**
Name: Phoenix Bishea
Eid: pnb338

*/

#include <iostream>

using namespace std;

const int numRow = 10;
const int numCol = 10;


//initialize arrays

void initializeArray(bool[][10]);
void printArray(const bool[][10]);

void playGameOfLife(bool[][10]);
int numNeighbors(const bool[][10], int row, int col);
```

```
*/
void printLife2DArray(const bool[][NUM_COLS]);
void playLife2DArray(bool[][NUM_COLS]);

 /** Initializes the bool array created in main to the input
array.
     Note: '*' is true and '.' is false.
 */
void initializeLife2DArray(bool[][NUM_COLS]);
int getNeighbors(const bool[][NUM_COLS], int row, int col);


int main() {

    /// read in the number of iterations to run

    /** make a 2d bool array with the number of rows and col
s

    Some input examples and explanation of game of life
    Look at http://en.wikipedia.org/wiki/Conway%27s_Game_of_
Life#Examples_of_patterns
    */

    /** print out what game we are playing */
    // cout << "Game Of Life rows=" << /* nrows */ << " cols
=" <<  ncols  <<
    //       " iterations=" << /* number of iterations */ <<
endl;

    /**
        start your code
    */

    // reads in the number of iterations to run
    int iterations;
    cin >> iterations;
    cin.ignore(); // consumes the newline character at the e
nd of the first line

    // initializes a 2d bool array with the number of rows a
nd cols
    bool arr[NUM_ROWS][NUM_COLS];
    initializeLife2DArray(arr);

    // prints the game we are playing
    cout << "Game Of Life rows=" << NUM_ROWS << " cols="
 <<  NUM_COLS  <<
          " iterations=" << iterations << endl;

    // prints the initial board
    printLife2DArray(arr);

    // completes x iterations
    for (int i = 0; i < iterations; i++) {
        cout << endl;
        playLife2DArray(arr);
        printLife2DArray(arr);
    }

    return 0; /// return a ok
```

```


int main() {
    int iters;
    int nothing;
    cin >> iters;

    cin.ignore();
    bool arr[numRow][numCol];
    initializeArray(arr);
```

```
    // prints the game we are playing
    cout << "Rows =" << numRow << " Columns =" <<  numCol
 << " Number of iters =" << iters << endl;
    printArray(arr);
    for (int i = 0; i < iters; i++) {
```

```
        cout << endl;
        playGameOfLife(arr);
        printArray(arr);
    }
    return nothing; /// return bool
```

```
}

void initializeLife2DArray(bool arr[][NUM_COLS]) {

    char line[NUM_COLS + 1]; // line to take in row input

    // gets the first line
    cin.getline(line, NUM_COLS + 1);

    // sets values to true and false in the bool array
    // '*' is true and '.' is false
    for (size_t i = 0; i < NUM_ROWS; i++) {
        for (size_t j = 0; j < NUM_COLS; j++) {
            if (line[j] == '*') {
                arr[i][j] = 1;
            } else {
                arr[i][j] = 0;
            }
        }
        cin.getline(line, NUM_COLS + 1);
    }
}

void printLife2DArray(const bool arr[][NUM_COLS]) {

    // converts the bool array into the board of life
    // '*' is true and '.' is false
    for (size_t i = 0; i < NUM_ROWS; i++) {
        for (size_t j = 0; j < NUM_COLS; j++) {
            if (arr[i][j]) {
                cout << '*';
            } else {
                cout << '.';
            }
        }
        cout << endl;
    }
}


void playLife2DArray(bool board[][NUM_COLS]) {


    // creates a temporary board and sets it equal to the game board
    bool tempBoard[NUM_ROWS][NUM_COLS];
    for (size_t i = 0; i < NUM_ROWS; i++) {
        for (size_t j = 0; j < NUM_COLS; j++) {
            tempBoard[i][j] = board[i][j];
        }
    }

    // updates cells using the game of life rules
    for (size_t i = 0; i < NUM_ROWS; i++) {
        for (size_t j = 0; j < NUM_COLS; j++) {

            // gets a cell's neighbors and status
            int neighbors = getNeighbors(tempBoard, i, j);
            bool alive = tempBoard[i][j];

            // changes the cell's condition depending on its neighbors
```

```
}

void initializeArray(bool arr[][numCol]) {
    char letter[numCol + 1]; //take in row input
    cin.getline(letter, numCol + 1); //return first letter
    // '<' is true and '>' is false
    for (size_t i = 0; i < numRow; i++) {
        for (size_t j = 0; j < numCol; j++) {
            if (letter[j] == '>') {
                arr[i][j] = 1;
            } else {
                arr[i][j] = 0;
            }
        }
        cin.getline(letter, numCol + 1);
    }
}

void printArray(const bool arr[][numCol]) {
    for (size_t i = 0; i < numRow; i++) {
        for (size_t j = 0; j < numCol; j++) {
            if (arr[i][j] && j < numCol) {
                cout << '>';
            } else {
                cout << '<';
            }
        }
        cout << endl;
    }
}

int numNeighbors(const bool board[][numCol], int row, int col) {
    int numNeighbors = 0; // initializes starting numNeighbors

    int sizeMin = 0;
    int sizeMax = 9;
```

```
            if (alive && (neighbors < 2 || neighbors > 3)) {
                board[i][j] = 0;
            } else if (!alive && (neighbors == 3)) {
                board[i][j] = 1;
            }
        }
    }
}

int getNeighbors(const bool board[][NUM_COLS], int row, int
 col) {

    int neighbors = 0; // initializes starting neighbors

    // sets up the boundaries
    bool colLessMax = (col + 1) <= MAX;
    bool rowLessMax = (row + 1) <= MAX;
    bool colGreaterMin = (col - 1) >= MIN;
    bool rowGreaterMin = (row - 1) >= MIN;



    // if the operation is within bounds, complete it and ta
lly living
neighbors
    if (colLessMax
        && board[row][col + 1]) neighbors++;
 // increment if a living neighbor is found

    if (colGreaterMin
        && board[row][col - 1]) neighbors++;

    if (rowLessMax
        && board[row + 1][col]) neighbors++;

    if (rowGreaterMin
        && board[row - 1][col]) neighbors++;

    if (rowLessMax && colLessMax
        && board[row + 1][col + 1]) neighbors++;

    if (rowGreaterMin && colLessMax
        && board[row - 1][col + 1]) neighbors++;



    if (rowLessMax && colGreaterMin
        && board[row + 1][col - 1]) neighbors++;

    if (rowGreaterMin&& colGreaterMin
        && board[row - 1][col - 1]) neighbors++;

    return neighbors;
```

```
    // sets up the boundaries
    bool ifminCol = (col + 1) <= sizeMax;
    bool ifminRow = (row + 1) <= sizeMax;
    bool ifmaxCol = (col - 1) >= sizeMin;
    bool ifmaxRow = (row - 1) >= sizeMin;
    // if the operation is within bounds, complete it and ta
lly living
numNeighbors

    if (ifminCol && board[row][col + 1]) numNeighbors++;
 // increment if a living neighbor is found
    if (ifmaxCol
        && board[row][col - 1]) numNeighbors++;



    if (rowLessMax
        && board[row + 1][col]) numNeighbors++;

    if (ifmaxRow
        && board[row - 1][col]) numNeighbors++;

    if (rowLessMax && ifminCol
        && board[row + 1][col + 1]) numNeighbors++;

    if (ifmaxRow && ifminCol
        && board[row - 1][col + 1]) numNeighbors++;

    if (rowLessMax && ifmaxCol
        && board[row + 1][col - 1]) numNeighbors++;

    if (ifmaxRow&& ifmaxCol
        && board[row - 1][col - 1]) numNeighbors++;

    return numNeighbors;
}

void playGameOfLife(bool board[][numCol]) {
    //makes a temp board
    bool temp[numRow][numCol];
    for (size_t i = 0; i < numRow; i++) {
        for (size_t j = 0; j < numCol; j++) {
            temp[i][j] = board[i][j];
        }
    }
    // updates each cell in the array
```

```
200 }
```

```
    for (size_t i = 0; i < numRow; i++) {
112         for (size_t j = 0; j < numCol; j++) {
113             int numNeighbors = numNeighbors(temp, i, j);
114             bool isAlive = temp[i][j];
                if (isAlive && (numNeighbors < 2 || numNeighbors
115 > 3)) {
116                 board[i][j] = 0;
117             } else if (!isAlive && (numNeighbors == 3)) {
118                 board[i][j] = 1;
119             }
120         }
121     }
122 }
```