Diffchecker

- 198 Removals + 195 Additions

6_2.cpp

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

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

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

Q2: Why are the arrays passed to both play life functions not const?

    The arrays passed to play life are not const because they 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 functions 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 functions below.
    DO NOT MODIFY the declarations.
    You may create your own functions, but you must use these.
*/

/** 2d array functions, notice that for 2d arrays the second size
```

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

*/

#include <iostream>

using namespace std;

/// global constant for fixed rows and cols of the 2D array
int iterationsR =
0, iterationsC = 0, gameCounter = 0, gameCounterMax = 0, aliveCounters = 0, temp = 0;
const int rownumber = 10, columnnumber = 10, minimum = 0, maximum = 9;
const bool alive;
// Don't douch the declarations
```

```
    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.
*/
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);
```

```
void showGrid(const bool[][columnnumber]);
void startGrid(bool[][columnnumber]);
void initializeGrid(bool[][columnnumber]);
void getGridCopy(bool grid[][columnnumber]); //hard copies c
reated grid for game use
int returnNeighbor(const bool[][columnnumber], int r, int
 c);
```

```
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;
```

```
int main()
{

    // reads in the number of numIterate to run
    int numIterate;
    cin >> numIterate;
    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[rownumber][columnnumber];
    initializeGrid(arr);

    // prints the game we are playing
    cout << "Game Of Life rows=" << rownumber
 << " cols=" <<  columnnumber  <<
         " numIterate=" << numIterate << 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
}

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

```
    // prints the initial board
    showGrid(arr);

    // completes x numIterate
    for (int i = 0; i < numIterate; i++)
    {
        cout << endl;
        startGrid(arr);
        showGrid(arr);
    }

    return 0; /// return a ok
}

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

void getgrid(char line[], bool grid[][columnnumber])
{
        //Local Declarations
        char c;
        int iterationsC = 0;
        while (iterationsC < columnnumber)
    {
                c = line[iterationsC];
                cout << c;
                 if ((isNotNullChar(c)) && (c == '.') || (c
 != 32))
                 {
                            grid[iterationsR][iterationsC] = fa
lse;
                 }
                else if ((c == '*') && ((isNotNullChar(c)) |
| (c != 32)))
        {
                            grid[iterationsR][iterationsC] = tru
e;
                 }
                 iterationsC++;
        }
        iterationsR++;
        cout << endl;
        if (iterationsR >= 10)
    {

```

```
    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]) {

```

```
                getGridCopy(grid);
        }
}


void initializeGrid(bool arr[][columnnumber])
{
    char line[columnnumber + 1]; // line to take in r input
    // gets the first line
    cin.getline(line, columnnumber + 1);
    // sets values to true and false in the bool array
    // '*' is true and '.' is false
    for (const i = 0; i < rownumber; i++)
    {
        for (const j = 0; j < columnnumber; j++)
        {
            if (line[j] == '*')
            {
                arr[i][j] = 1;
            } else
            {
                arr[i][j] = 0;
            }
        }
        cin.getline(line, columnnumber + 1);
    }
}

bool isNotNullChar(char c)
{
    if (c == '\0')
    {
        return 0
    }
    else
    {
        return 1
    }
}

void getGridCopy(bool grid[][NUM_COLS])
{
        //Declarations
        iterationsR = 0, iterationsC = 0;

        while (iterationsR < NUM_ROWS) {
                while (iterationsC < NUM_COLS)
        {
                        gridCopy[iterationsR][iterationsC] =
grid[iterationsR][iterationsC];
                        iterationsC++;
                }
                iterationsR++; iterationsC = 0;
        }
}

```

```
    // 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
            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 tally 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
```

```
void startGrid(bool board[][columnnumber])

{
    bool tempBoard[rownumber][columnnumber];
    for (const i = 0; i < rownumber; i++)
    {
        for (const j = 0; j < columnnumber; j++)
        {
            tempBoard[i][j] = board[i][j];
        }
    }
    for (const i = 0; i < rownumber; i++)
    {
        for (const j = 0; j < columnnumber; j++)
        {
            int neighbors = returnNeighbor(tempBoard, i, j);

            bool alive = tempBoard[i][j];
            if (alive && (neighbors < 2 || neighbors > 3))
            {

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

int returnNeighbor(const bool board[][columnnumber], int r, int c)
{
    int neighbors = 0; // initializes starting neighbors

    // sets up the boundaries
    int a = c+1, b = r+1, d = c-1, e = r-1;
    bool ifcolumnmax = (a) <= maximum;
    bool ifrowmax = (b) <= maximum;
    bool ifcolmin = (d) >= minimum;
    bool ifrowmin = (e) >= minimum;
    if (ifcolumnmax && board[r][c + 1]) neighbors++;


    else if (ifcolmin && board[r][c - 1]) neighbors++;
    else if (ifrowmax && board[r + 1][c]) neighbors++;

    else if (ifrowmin && board[r - 1][c]) neighbors++;
    else if (ifrowmax && ifcolumnmax && board[r + 1][c + 1]) neighbors++;
    else if (ifrowmin && ifcolumnmax && board[r - 1][c + 1]) neighbors++;
    else if (ifrowmax && ifcolmin && board[r + 1][c - 1]) neighbors++;
    else (ifrowmin&& ifcolmin && board[r - 1][c - 1]) neighbors++;
```

```
185         && board[row - 1][col]) neighbors++;
186
187     if (rowLessMax && colLessMax
188         && board[row + 1][col + 1]) neighbors++;
189
190     if (rowGreaterMin && colLessMax
191         && board[row - 1][col + 1]) neighbors++;
192
193     if (rowLessMax && colGreaterMin
194         && board[row + 1][col - 1]) neighbors++;
195
196     if (rowGreaterMin&& colGreaterMin
197         && board[row - 1][col - 1]) neighbors++;
198
199     return neighbors;
200 }
```

```
194     return neighbors;
195 }
```