```
/*****************************************************************
 * Jared Dembrun
 * 10/8/2013
 *
 * The infamous computer science assignment, the Game of Life
 *****************************************************************/

#include <iostream>
#include <cstdlib>
#include <cstring>
using namespace std;

#define MAX_WIDTH 40

// function prototypes
void ParseCmdLine(int, char**);
void ShowUsage(char*);
int ReadGen0(bool[MAX_WIDTH][MAX_WIDTH]);
void DisplayBoard(bool[MAX_WIDTH][MAX_WIDTH], int, int);
void updateBoard(bool[MAX_WIDTH][MAX_WIDTH], int);


// command line parameters (default values)
int numGens = 10;
int genInt = 1;

/*******
 * MAIN
 *******/
int main(int argc, char **argv)
{
        // "*" => true, " " => false
        bool board[MAX_WIDTH][MAX_WIDTH];
        int len;

        // parse the command line, read gen0, and display the board
        ParseCmdLine(argc, argv);
        len = ReadGen0(board);
        DisplayBoard(board, 0, len);

        // compute new generations and display them by specified parameters
        for(int i = 1; i <= numGens; i += genInt)
        {
                updateBoard(board, len);
                DisplayBoard(board, i, len);
        }
        return 0;
}

// parses the command line for parameters
void ParseCmdLine(int argc, char **argv)
{
        if(argc == 1)
        {
                ShowUsage(argv[0]);
                exit(0);
        }

        for(int i = 1; i < argc; i++)
        {
                // default values
                if(!strcmp(argv[i], "-D"))
                        break;
                // help/usage
                if(!strcmp(argv[i], "-h"))
                {
                        ShowUsage(argv[0]);
```

```
                        exit(0);
                }
                // number of generations to produce
                else if(strcmp(argv[i], "-n") > 0)
                {
                        argv[i] += 2;
                        numGens = atoi(argv[i]);
                }
                // display generation interval
                else if(strcmp(argv[i], "-i") > 0)
                {
                        argv[i] += 2;
                        genInt = atoi(argv[i]);
                }
        }
}

// displays help/usage
void ShowUsage(char *filename)
{
        cout << "Usage: " << filename << " [-h] -D [-(ni)<val>]\n";
        cout << " e.g.: " << filename << " -D < gen0\n";
        cout << "  -D\t\tUse default values\n";
        cout << "  -n<val>\tSet the number of generations to produce to <val> (=10)\n";
        cout << "  -i<val>\tSet the generation display interval to <val> (=1)\n";
        cout << "  -h\t\tShow this screen\n";
}

// reads the initial generation
int ReadGen0(bool board[MAX_WIDTH][MAX_WIDTH])
{
        //erase all values contained by board
        for(int row = 0; row < MAX_WIDTH; row++)
                for(int col = 0; col < MAX_WIDTH; col++)
                        board[row][col] = false;

        char line[MAX_WIDTH];
        int i = 0;
        int len = 0;

        // we assume that the input contains a blank border as specified in class
        while(cin.getline(line, MAX_WIDTH))
        {
                // grab each character of each line and make the board
                for(int j = 0; j < strlen(line); j++)
                        board[i][j] = (line[j] == '*');
                i++;
                // note the board size
                len = strlen(line);
        }

        return len;
}

// displays the current board
void DisplayBoard(bool board[MAX_WIDTH][MAX_WIDTH], int gen, int len)
{
        cout << "Gen" << gen << ":\n";

        //display the current generation
        for(int row = 0; row < (len - 1); row++)
        {
                for(int col = 0; col < (len - 1); col++)
                {
                        if(!col && !row)//if col == 0 && row == 0
                                cout << "  ";
                        else if(!row)//if row == 0 && col != 0
```

```
                        cout << col << " ";
                else if(!col)//if col == 0 && row != 0
                        cout << row << " ";
                else//if col != 0 and row != 0
                        if(board[row][col])//if this cell is alive
                                cout << "* ";
                        else
                                cout << "  ";
        }
        cout << endl;
    }

    cout << endl;
}

void updateBoard(bool board[MAX_WIDTH][MAX_WIDTH], int len)
{
    bool tempBoard[MAX_WIDTH][MAX_WIDTH];

    //make a copy of board in tempBoard
    for(int row = 0; row < MAX_WIDTH; row++)
        for(int col = 0; col < MAX_WIDTH; col++)
            tempBoard[row][col] = board[row][col];

    for(int i = 0; i < genInt; i++)
        for(int row = 1; row < (len - 1); row++)
            for(int col = 1; col < (len - 1); col++)
            {
                int neighbors = 0;
                //beginning left of the cell and working clockwise, check all adjacent
squares for a true value and count the number of trues
                if(board[row][(col - 1)])
                    neighbors++;
                if(board[(row - 1)][(col - 1)])
                    neighbors++;
                if(board[(row - 1)][col])
                    neighbors++;
                if(board[(row - 1)][(col + 1)])
                    neighbors++;
                if(board[row][(col + 1)])
                    neighbors++;
                if(board[(row + 1)][(col + 1)])
                    neighbors++;
                if(board[(row + 1)][col])
                    neighbors++;
                if(board[(row + 1)][(col - 1)])
                    neighbors++;

                if(neighbors != 2)//if the cell is not stable
                    tempBoard[row][col] = (neighbors == 3);
            }
    //completely repopulate the game board using the temp board
    for(int row = 1; row < (len - 1); row++)
        for(int col = 1; col < (len - 1); col++)
            board[row][col] = tempBoard[row][col];
}
```