Programming Fundamentals

Mini Project 2

Submitted To:

Dr. Omer Ahmed

Submitted By:

Manaal Waseem

FA18-BCE-074

Mini Project 2: Develop Conway’s Game of Life.

Project Description:

Conway’s Game of Life:

The Game of Life, also known simply as Life, is a cellular automaton devised by the British mathematician John Horton Conway in 1970.
The game is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves, or, for advanced players, by creating patterns with particular properties.

Rules:

The universe of the Game of Life is an infinite, two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, alive or dead, (or populated and unpopulated, respectively). Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:
1. Any live cell with fewer than two live neighbours dies, as if by underpopulation.
2. Any live cell with two or three live neighbours lives on to the next generation.
3. Any live cell with more than three live neighbours dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.
The initial pattern constitutes the seed of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed; births and deaths occur simultaneously, and the discrete moment at which this happens is sometimes called a tick. Each generation is a pure function of the preceding one. The rules continue to be applied repeatedly to create further generations.
Figure 1:

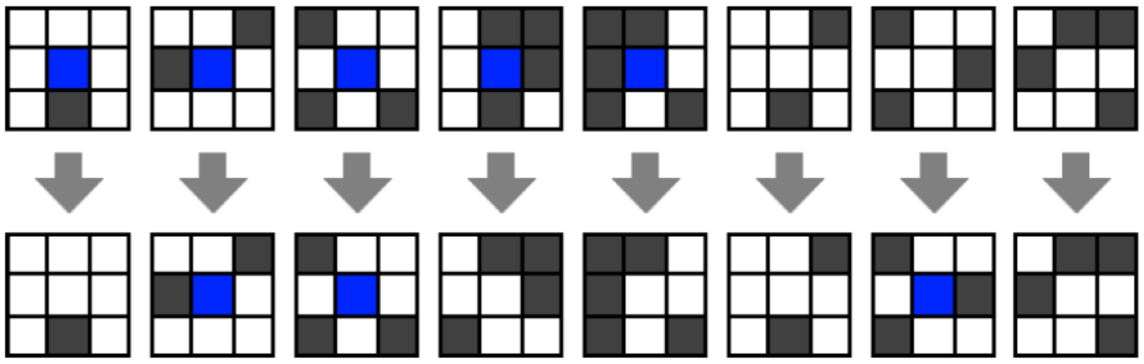


***Figure 1**: Conway's Game of Life. Depiction of Rules*

Task:

In this project you will make a simulation of the Conway's Game of Life. The rules are given in the section above and they are depicted in **Figure 1** for clarity.

Your task is to write a C program that fulfills the following requirements:

1. Declare a two-dimensional square grid (using arrays of type **'char'**) of variable size. By 'variable size' I mean that the user will have the option to select the size of the simulation grid. For example 9x9, 10x15, (m x n in general), at the start of the simulation.
2. A **'live'** cell will have a '*' as its contents while a dead cell will have a **'space'**.
3. For the initial state of the game the user should be given an option weather to enter the initial state himself (via keyboard) or to have it randomly generated.
4. The next states should be generated by either pressing a key (you may use the **getc()** function) or after a specific time interval.
5. You must make a function 'void **next_tick** (char * ptr_prev, char * ptr_next, int m, int n)' that computes the next generation based on the previous one provided as argument.
6. You should place the above function in a file named **'conway.c'** and its prototype in **'conway.h'**.
7. The simulation should run until the user presses **'q'**.

Project Design:

This project consists of following three files:

main.c

This file contains **`main()`**. In the **`main()`** function after declaration of variables firstly user is asked to enter the dimensions of the grid. Then two 2D char type arrays namely **`grid_pre`** and **`grid_next`** are declared with user-specified dimensions. Two char type pointers **`ptr_grid_pre`** and **`ptr_grid_next`** are declared and initialized.
The user is then presented with the option to configure the grid manually or the initial configuration is generated randomly.

Once the grid is configured it is printed using the function **`print_array`**. Then a while loop is executed to run the simulation until the user enter **`'q'`** from the keybord. Inside the while loop next generation is generated using the function **`next_tick`** and passing the pointers **`ptr_grid_pre`** and **`ptr_grid_next`** along with length and width of the grid to it. Then this generation is printed and swapped.

```c
#include <stdio.h>
#include <stdlib.h>
#include "conway.h"

int main()
{
    int length, width;
    char initial;

    printf("\nEnter the dimensions of borad.\nLength: ");
    scanf("%d",&length);
    printf("\nWidth: ");
    scanf("%d",&width);

    char grid_pre[length][width];
    char grid_next[length][width];

    char *ptr_grid_pre = grid_pre;
    char *ptr_grid_next = grid_pre;

    printf("\n\nDo you want to configure the initial board. If yes press Y.");
    initial = getch();

    if ((initial == 'Y')||(initial == 'y'))
        read_array(ptr_grid_pre, length, width);
    else
    {
        initialize_random(ptr_grid_pre, length, width);
        printf("\nBoard is initialized randomly\n");
    }

    print_array(ptr_grid_pre,length,width);

    while((getc(stdin)!='q'))
    {
        next_tick(ptr_grid_pre, ptr_grid_next, length, width);
        print_array(ptr_grid_next,length,width);
        swap(ptr_grid_pre, ptr_grid_next,length,width);
    }

    return 0;
}
```

Code Listing 1: main.c

conway.h

This file contains the declarations for user defined functions which are as follows:

```
void read_array(char *arr,int row,int col);
void print_array(char *arr,int m,int n);
void print_int_array(int *arr,int m,int n);
void initialize_random(char *arr, int row, int col);
void count_neighbours(char * arr, int * neighbour, int row, int
col);
void next_tick(char * ptr_prev, char *ptr_next, int    m, int n);
void swap(char * ptr_prev, char *ptr_next, int m, int n);
```

```
void read_array(char *arr,int row,int col);
void print_array(char *arr,int m,int n);
void print_int_array(int *arr,int m,int n);
void initialize_random(char *arr, int row, int col);
void count_neighbours(char * arr, int * neighbour, int row, int
col);
void next_tick(char * ptr_prev, char *ptr_next, int m, int n);
void swap(char * ptr_prev, char *ptr_next, int m, int n);
```

Code Listing 2: conway.h

conway.c

This file contains implementations for user defined functions. Following are all the user defined functions used in this project and their descriptions:

void read_array(char *arr, int row, int col):

This function reads user inputs to configure the grid manually. The pointer to the grid is passed as an argument along with the size of grid.

void print_array(char *arr, int m, int n):

This function prints the grid whose pointer is passed as an argument along with its size.

void print_int_array(int *arr, int m, int n):

This function prints an integer type array whose pointer is passed as an argument along with its size. This function is developed to facilitate development by printing intermediate results like the neighbour array.

void initialize_random(char *arr, int row, int col):

This function populates the grid randomly. The pointer to the grid is passed as an argument along with the size of grid. This function generates a random number between 0 and 5. If the generated number is greater than 2 the function assign a **`'*'`** to the cell (meaning a live cell) otherwise a **`'space'`** is assigned to the cell (meaning a dead cell)

void count_neighbours(char * arr, int * neighbour, int row, int col):

This function traverses the grid for each cell to calculate its neighbours according to the criteria given in the project description. The pointer to the grid is passed as an argument along with the size of grid. The calculated number of neighbours for each cell is stored in the respective cell of an integer type array whose pointer is also passed as an argument. Note that the size of the neighbour array is same as that of the grid.

void next_tick(char * ptr_prev, char *ptr_next, int m, int n):

This function is a must function that computes the next generation based on the previous one provided as argument.

In this function first of all a 2D integer array called neighbour of the same size as grid is declared. **`ptr_neighbour`**, a pointer variable, is also declared and initialized with the address of neighbour array. Then **`count_neighbours`** function is called that populates the neighbour array. Once the neighbour array is populated next generation is generated according to the given rules.

void swap(char * ptr_prev, char *ptr_next, int m, int n):

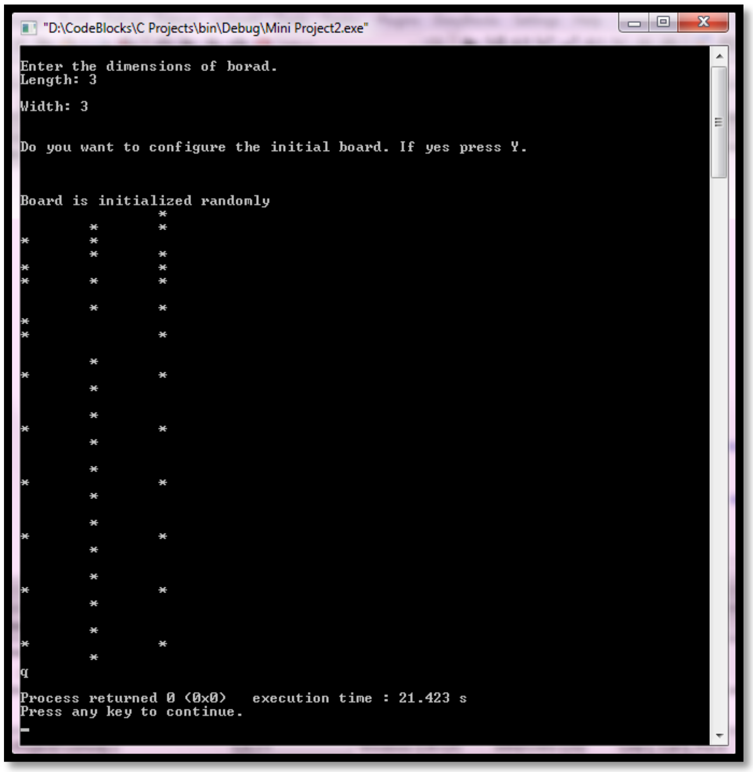
This function swaps the values in both grids.

```c
void read_array(char *arr,int row,int col)
{
    int cell_alive, flag=0;

    while(!flag)
    {
        printf("\nHow many cells you want to be alive?");
        scanf("%d",&cell_alive);
        if (cell_alive < (row*col))
            flag = 1;
    }

     int positions_alive[cell_alive][2];

    for(int i=0; i<cell_alive;i++)
    {
        printf("\n\nEnter the row coordinate for cell no %d between 0-%d: ", i+1, row-1);
        scanf("%d", &positions_alive[i][0]);

        printf("\nEnter the column coordinate for cell no %d between 0-%d: ", i+1, col-1);
        scanf("%d", &positions_alive[i][1]);
    }

    for(int i=0; i<row;i++)
        for(int j=0;j<col;j++)
            *((arr+i*col)+j) = ' ';

    for(int i=0; i<cell_alive;i++)
            *((arr+positions_alive[i][0]*col)+positions_alive[i][1])='*';

}

void print_array(char *arr,int row,int col)
{
    for(int i=0; i<row;i++)
    {
        for(int j=0;j<col;j++)
            printf("%c\t",*((arr+i*col)+j));
        printf("\n");
    }
}

void print_int_array(int *arr,int row,int col)
{
    for(int i=0; i<row;i++)
    {
        for(int j=0;j<col;j++)
            printf("%d\t",*((arr+i*col)+j));
        printf("\n");
    }
}

void initialize_random(char *arr, int row, int col)
{
    int value;

    for(int i=0; i<row;i++)
    {
        for(int j=0;j<col;j++)
        {
            value = rand()% 5;  //generate a random no. between 0 and 5
            if (value > 2)
                *((arr+i*col)+j) = '*';
            else
                *((arr+i*col)+j) = ' ';
        }
        printf("\n");
    }
}

void count_neighbours(char * arr, int * neighbour, int row, int col)
{
    for (int i=0; i<row; i++)
        for(int j=0; j<col; j++)
    {
        int counter = 0;

        for (int r=i-1; r<=i+1; r++)
        {
            if ((r <0) || (r > row))
                continue;
            for(int c=j-1; c<=j+1; c++)
            {
                if ((c <0) || (c >= col))
                    continue;
                if ((r == i) && (c == j))
                    continue;
                if (*((arr+r*col)+c) == '*')
                    counter++;
            }
        *((neighbour+i*col)+j) = counter;
        }
    }
}

void next_tick(char * ptr_prev, char *ptr_next, int row, int col)
```

```c
{
    int neighbour[row][col];
    int *ptr_neighbour = neighbour;

    count_neighbours(ptr_prev, ptr_neighbour, row, col);
   // print_int_array(ptr_neighbour, row, col);

    for (int i=0; i<row; i++)
            for(int j=0; j<col; j++)
            {
                if (*((ptr_prev+i*col)+j) == '*')
                    {
                        if ((neighbour[i][j] <2) || (neighbour[i][j] > 3))
                            *((ptr_next+i*col)+j) = ' ';
                        else
                            *((ptr_next+i*col)+j) = '*';
                    }
              else
              {
                  if ( (*((ptr_prev+i*col)+j) == ' ') && (neighbour[i][j] == 3))
                        *((ptr_next+i*col)+j) = '*';
                  else
                        *((ptr_next+i*col)+j) = ' ';
              }
            }
}

void swap(char * ptr_prev, char *ptr_next, int row, int col)
{
    char temp;

    for (int i=0; i<row; i++)
            for(int j=0; j<col; j++)
            {
                temp = *((ptr_next+i*col)+j);
                *((ptr_next+i*col)+j) = *((ptr_prev+i*col)+j);
                *((ptr_prev+i*col)+j) = temp;
            }

}
```

Code Listing 3: conway.c

<u>Output</u>:



"D:\CodeBlocks\C Projects\bin\Debug\Mini Project2.exe"

```
Enter the dimensions of borad.
Length: 3

Width: 3


Do you want to configure the initial board. If yes press Y.



Board is initialized randomly
                *
        *       *
*       *
        *       *
*               *
*       *       *

        *       *
*
*               *

        *
*               *
        *

        *
*               *
        *

        *
*               *
        *

        *
*               *
        *

        *
*               *
        *

        *
*               *
        *
q

Process returned 0 (0x0)   execution time : 21.423 s
Press any key to continue.
_
```

THE END
