Digital Design Verification

Weekly Task

Game of Life

Release: 1.0

Release Date: 26-Feb-2024

NUST Chip Design Centre (NCDC), Islamabad, Pakistan

Revision History

Revision Number	Revision Date	Revision By	Nature of Revision	Approved By
1.0	26/02/2024	Hira Sohail	Complete manual	Deputy PD

Contents

Objective

The purpose of this weekly task is to:

- To test all the concepts covered in the previous week.

Tools

- GCC
- GDB

Introduction

The Game of Life, invented by John Conway in 1970, is an example of a zero-player "game" known as a cellular automaton. The game consists of a two-dimensional world extending infinitely in all directions, divided into "cells." Each cell is either "dead" or "alive" at a given "generation." The game consists of a set of rules that describe how the cells evolve from generation to generation. These rules calculate the state of a cell in the next generation as a function of the states of its neighboring cells in the current generation. In a 2-D world, a cell's neighbors are those 8 cells vertically, horizontally, or diagonally adjacent to that cell. Conway's set of rules are summarized as:

1. A live cell with fewer than two live neighbors dies.

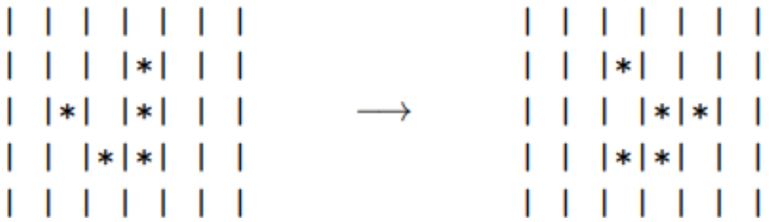
2. A live cell with more than three live neighbors also dies.

3. A live cell with exactly two or three live neighbors lives.

4. A dead cell with exactly three live neighbors becomes alive.

In this lab, you will be implementing Conway's Game of Life, with the minor restriction that our 2-D world is finite. The neighbors of a cell on the edge of the world that would be beyond the edge are assumed dead. You can read more about Conway's Game of Life on Wikipedia at http://en.wikipedia.org/wiki/Conway's_Game_of_Life.

Example (the so-called "glider"):



Legend: * = alive, = dead

TASK 1: Implementing Evolution

In this part, we will focus on implementing the rules needed to evolve the world from one generation to the next. To assist you, we provide several functions you should use to access and modify the current and next state of the world. These functions are

described in the header file lifegame.h and implemented in the file lifegame.c. Also, we have provided a skeleton describing what you need to do for this part in lab1a.c.

When compiling, you need to compile lab1a.c and lifegame.c together to generate a single executable file (let's call it lab1a.o) with all the code in it (otherwise, you'll get "undefined reference" errors). Start by examining the contents of lifegame.h and lab1a.c. You need to fill in a few lines in main () and complete the functions next generation (), get next state(x,y), and num neighbors(x,y). There is no need to modify the files lifegame.h or lifegame.c in this part.

(a) How will you initialize the world for the Game of Life? Write the appropriate code in the main () function.

(b) How will you output the final state of the world once all the evolutions are done? Write the appropriate function call in main ().

(c) The main () function calls next generation () for each generation to handle the evolution process. Your code should set each cell's state in the next generation according to the rules specified in the introduction of this lab. Once the states of all the cells have been set for the next generation, calling finalize evolution () will set the current world state to the next generation and reset the next generation state. Your code should make use of the get next state(x,y) function to compute the next state of each cell.

(d) Write the code for get next state(x,y), so the function returns the next state (ALIVE or DEAD) of the cell at the specified coordinates using the number of live neighbors (returned by the num neighbors(x,y) function) and the Game of Life rules.

(e) Fill in the function num neighbors(x,y), so it returns the number of live neighbors (cells vertically, horizontally, or diagonally adjacent) for the specified cell. Since our world is finite, adjacent cells which are beyond the edge of the world are presumed DEAD.

Now that you're done, compile and run the program. Feel free to change the definition of NUM GENERATIONS, but before moving to next task, make sure NUM GENERATIONS = 50 and save its screenshot for the report.

TASK 2: The World in a File

In the first part of this lab, the initial state of the world was hard-coded into lifegame.c and the final state of the world was output to the console. In this part, we will modify the code so that the initial state of the world can be read from a file, and the final state is output to a file. First, let's examine lifegame.c. Notice the functions you need to implement: initialize world from file(filename) and save world to file(filename).

(a) The first of these, initialize world from file(filename), reads the file specified by filename and initializes the world from the pattern specified in the file. Basically, the file is a matrix of characters, '*' to specify a live cell, and ' ' (space) to specify a dead cell. The ith character of the jth line (zero-indexed) represents the initial state of the cell located at (i,j). If the line doesn't contain enough characters or the file doesn't

contain enough lines, the unspecified cells are presumed dead. Fill in the function to read the file and initialize the world. Don't forget to reset all the next generation states to DEAD. Use appropriate error handling.

(b) The other function, save world to file(filename), saves the final state of the world to the file filename in the same format as used in the initialization function: the ith character of the jth line (zero-indexed) represents the state of the cell located at (i,j) in the world.

(c) Fill in the contents of lab1b.c using the code from Part A (lab1a.c) and modifying to call these functions. The name of the file to load will be specified in the first command line argument passed to the executable. If no file is specified, you should default to initializing the world to the hard-coded default "glider" pattern. Save the final output to the file "world.txt."

To help you test your code, we've provided a couple test files: glider.txt (should match your output from Part A) and sship.txt (output in sshipout.txt).

To finish, write a brief (1 para max.) lab report describing your experience completing this lab, what challenges you faced and how you addressed them, and what you learned from this lab. Turn in a zip file containing all your code (lab1a.c, lab1b.c, lifegame.h, and lifegame.c), and your lab report.

Submission:

Please submit the .c files of all the tasks along with the screenshots of outputs on LMS in a proper report.