Computer Science 327
Project 1, Part c
C Programming Project
Cellular Automata and Sand Painting

Notes

Some parts of this project contain mathematical formulas, but nothing more than you have seen in Math 165. There may be parts of this project that may be ambiguous (not on purpose), but it is your job to also clarify program specifications given in a natural language. Please read the assignments early and asks questions in class. Questions that lead to further specification or changes in the project specification will be posted in Piazza or in the announcements section of the Canvas course.

Introduction

Cellular automata have long been used to model a variety of real and imaginary systems, and a Google search on "cellular automata" will yield a plethora web sites dedicated to different forms of cellular automata.

The project is divided into several parts. For submitting part c, we will ask that you create a git tag for your submission. More details are given below. Before writing any code, please review or work through Zybook chapter 7.9, Two-dimensional Arrays.

Project 1 part c

1. For this part of the project we will write functions that simulate a 2D cellular automata. To do this we will add a required flag to the structure, that determines if the CA is 1D or 2D. Make appropriate changes to the ca_data structure in ca.h. The prototypes are now:

   ```
   void displayCA( struct ca_data *ca );
   int set1DCACell( struct ca_data *ca, unsigned int x, unsigned char state);
   int set2DCACell( struct ca_data *ca, unsigned int x, unsigned int y, unsigned char state);
   void initCA( struct ca_data *ca, int state );
   ```

```
void step1DCA( struct ca_data *ca, unsigned char (*rule)(struct ca_data *, int x));
void step2DCA( struct ca_data *ca, unsigned char (*rule)(struct ca_data *, int x, int y));
struct ca_data *create1DCA( int w, unsigned char qstate);
struct ca_data *create2DCA( int w, int h, unsigned char qstate);
```

In the above prototypes functions involving a 1DCA, x is the index of the cell, and w is the width (number of cells) in the CA. For a 2DCA, x, y gives the coordinates of the cell and w, h give the width and height of the CA.

For grading purposes the ca_data structure must have these components with the names specified here.

> **cadata** is the pointer to the beginning of the CA array (either 1 or 2 dimensional)
>
> **dimension** is an unsigned char flag that contains 1 if the CA is 1 dimensions and if it is 2 dimensions.
>
> **wrap** is an unsigned char flag that contains 0 if there is no wrap and 1 if there is wrap.
>
> **width** is an int type that gives the width of the CA
>
> **height** is an int type that gives the height of the CA if it is two-dimensional

2. Implement the new versions of the prototypes in ca.c. Here are some notes and hints:

   You may put extra information about the CA into the ca structure. For example, it will be useful to store the width, and height (if it is a 2DCA) in this structure.

   The wrap property should also be stored in this structure, and thus the rule may use it to compute the next state. For a 2DCA the bottom wraps back to the top and the right wraps to the left. With this definition, you should not need to allocate buffer space around the CA, and depend on the rule to do the right thing. (If this is not clear, I will go over this in lecture to make sure everyone is doing the easy version of this.)

3. Write a main function in the file main.c. This program has the following command-line parameters.

   1) The first command-line parameter determines if it is a 1D or 2D CA. a 1 indicates 1 dimensional CA while a 2 indicates a 2-dimensional CA.
   2) The second command-line parameter is a file name (path) that gives the dimensions and initial state of a 2DCA. The format of this file is a text file that contains integer numbers separated by spaces and/or new line characters. The

first two numbers in the file are the number of rows followed by the number of columns. The rest of the data (rows * columns) number of entries is the initial state of the CA.

The program then displays the initial state of the CA in text with **one space** between each state. Each row must be separated by a new line character so that it is easy to read. For example:

```
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```

This represents a five row five column 2DCA all initialized to the state of 0.

The program then stops at waits for the user to press the return key. When they do, the program performs one step of the CA and displays the 2DCA again in its new state. This continues until the user enters any character, at which time the program terminates.

This main program should set the wrap flag to true, meaning that the game of life CA will wrap the edges.

Note also that since this rule only applies to 2DCA, your main program should indicate an error if the first parameter is a 1DCA.

The rule that this CA runs is fixed and is documented in this link:

https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

4. Make sure an updated make file works correctly, and that the executable file created is called “gol”

5. 10% points come from documentation and following the syllabus requirements for projects.

6. **Git submission: when you have completed this part we will pull your work from git using the tag named “partcsub” If you do not create a tag for your submission, your project may not be graded. The easiest way to create a tag is through the web interface.**