Computer Science 327
Project 1, Part b
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. For this part of the project we will work with one-dimensional cellular automata. https://mathworld.wolfram.com/ElementaryCellularAutomaton.html Note that the examples shown in this link are elementary with only 0 and 1 for states. In general cellular automata can have many states.

The project is divided into several parts. For submitting part b, we will ask that you create a git tag for your submission. More details are given below.

Project 1 part b

1. For part b of the project we will write functions that simulate cellular automata. To begin, we will create a header file and encapsulate the cellular automata inside structures. You may also put other information in this file as needed and appropriate. The header file you are to create is named "**ca.h**". You will add to this file as you do subsequent parts, but to begin with, add a structure to store a 1DCA. This structure should contain enough information to completely represent a 1DCA. At a minimum it should certainly contain the array of cells and the number of cells, but you may store other information in the structure. **The structure must have a structure tag named ca_data**. Be sure not to accidently create a variable for the structure in the header file.

2. Convert the functions **display1DCA**, **set1DCACell**, and **init1DCA** from Part a to utilize this new structure. The signatures (return type and parameter types) are as follows:

   void display1DCA(struct ca_data *)
   int set1DCACell(struct ca_data *, unsigned int, unsigned char)
   void init1DCA(struct ca_data *, int)

   Add prototypes for these functions to ca.h.

   Note that init1DCA now takes a second parameter that is the value that all the cells are initialized to.

   All of these functions must reside in the file named "ca.c"

3. Create a function named **create1DCA** that takes two parameters. The first parameters is of type unsigned int and specifies the number of cells in the 1DCA. The second parameters is an unsigned char that specifies the initial state of all the cells. The return value is a pointer to a ca_data structure initialized as such. The signature is

   struct ca_data * create1DCA(int, unsigned char)

   You are required to use malloc to create the structure and the arrays inside the structure. **The array size must match the requested size of the 1DCA**.

   Add this function to ca.c and be sure to add the function prototype to your ca.h file.

4. Create a function named **stepCA** in ca.c that takes a single parameter that is that is a pointer to a ca_data strcuture, and a pointer to a function that defines the rule for the 1DCA. The signature for this function that is placed in ca.h is

   void stepCA(struct ca_data *, unsigned char (*)(struct ca_data *, int), int)

   This function was/will be discussed in lecture in detail, but a brief description is given here. The function stepCA performs one step of the 1DCA given by the first parameter. The rule for how to perform the step is given in by the second parameter which is a function pointer that points to a function that takes two parameters (a pointer to a ca_data structure, and integer type) The function computes the rule that updates the ca. For example,

```
unsigned char aRule( struct ca_data *, int index)
{
        // compute the new value at cell index and return that value
        // return computed value
}
```

   The final parameter is a flag that indicate how to handle the edge cases. A value of false indicates that cells with indices outside of cellular automata are given a value of the quiescent state, the state the cells were initialized to. If the flag is true, the indices are wrapped around to the other edge of the cellular automata.

5. Write a main function and place it in the file main.c. This main function allows for the following command line parameters in this order:

    1) The number of cells in the 1DCA. This is an integer input that must be between 10 and 100. Please use constants in your program to define these parameters so that they can be easily changed.
    2) The number of states possible for each cell. The states are labeled 0 through the number of states – 1.
    3) A flag to determine if the simulation wraps or not at the edge. (Discussed in class). Valid entries for this command line parameter are "wrap" and "nowrap".
    4) An integer that gives the initial state of each cell. This number is between 0 and the number of states possible - 1, or negative 1 which will initialize each cell to a random number between 0 and the number of possible states - 1. (Each cell is a different random number.)
    5) An integer that gives the number of steps to simulate the 1DCA.

    Error checking should be done on these command line parameters and an error message written to stdout if there is a problem.

    If there are no problems with the command line input parameters, the main program initializes a 1DCA appropriately and simulates the specified number of steps. Output of the simulation are text lines for each step of the simulation. For this main, you are to write a rule that implements "rule 110" and outputs the results of that CA.
    https://mathworld.wolfram.com/Rule110.html
    Note that this 1DCA has only two states 0 and 1, where 0 is white and 1 is black. (For your output you can output 0 and 1 appropriately.

    A sample output for a 10 cell 1DCA (with a very simple rule) with three states would look like

    0110111211
    0000020000
    0000000000
    0000000000

    You should of course utilize the data structures and functions defined previously in this part when writing your main function. (very little credit will be given if you do not do this)

6. Create a makefile for this project so that when a user types make in the top directory for the repository, the executable program named "odca" is created.

7. 10% points come from documentation and following the syllabus requirements for projects.

8. **Git submission: when you have completed this part we will pull your work from git using the tag named "partbsub" If you do not create a tag for your submission, your project may not be graded. (Our will pull this tag name.) The easiest way to create a tag is through the web interface.**