ASSIGNMENT 4:

Cellular Automata

Due 1:15 p.m. Monday, October 7

PLEASE NOTE THAT THIS IS AN INDIVIDUAL ASSIGNMENT!

A one-dimensional *cellular automata* takes in a string, which in our case, consists of the characters '0' and '1', and changes it according to some predetermined rules. The rules consider three characters, which are a character at position *k* and its two neighbors, and determine what the character at the corresponding position *k* will be in the new string.

For example, if the character at position *k* in the string is '0' and its neighbors are '0' and '1', then the pattern is '001'. We look up '001' in the table below (for the case of Pattern 142), '001' corresponds to '1' which means that in the new string, '1' will be at position *k*.

Rules for Pattern 142

Value	Pattern in current string	Position *k* in new string	Contribution to pattern number is 0 if replaced by '0' and 1 if replaced by '1'
1	'000'	'0'	1 * 0
2	'001'	'1'	2 * 1
4	'010'	'1'	4 * 1
8	'011'	'1'	8 * 1
16	'100'	'0'	16 * 0
32	'101'	'0'	32 * 0
64	'110'	'0'	64 * 0
128	'111'	'1'	128 * 1
			= 142

To calculate the patterns which will have the central character '1', work out the values required to sum to the pattern number. For example, 32 = 32 so only pattern 32 which is '101' changes the central position to a '1'. All the others have a '0 ' in the next line. Since 23 = 16 + 4 + 2 + 1, patterns in the current string, '100', '010', '001' and '000', all lead to a '1' in the next line and the rest have a '0'.

For pattern 142, and starting string
`01001100`
the new strings created will be

`11011000` (generation = 1)
`10010001` (generation = 2)
`00110011` (generation = 3)
…

Note that the first position of the string is next to the last position in the string.

For this assignment you are to write a function, `cellular_automata()`, that takes three inputs:
- a non-empty string, whose length never exceeds 50,
- a pattern number which is an integer between 0 and 255 that represents a set of rules, and
- a positive integer *n*, which is the number of generations.

The function should return a string which is the result of applying the rules generated by the pattern to the string *n* times. Using this function, write a C program (**automata.c**) which will implement a simple cellular automaton.

Input to your program should consist of
a positive integer *m*, which represents the number of test cases,
followed by *m* lines, each containing
a non-empty string which represents the initial input to an automaton,
a pattern number which represents a set of rules, and
a positive integer which represents the number of generations.

Sample Input

```
3
010101010 17 2
010101010 249 3
00010000 125 10
```

Output should consist of a set of new strings, each of which is the result of applying the rules generated by the pattern to each input string.

Sample Output

```
111111100
010010101
00011111
```

Directions

1. Create a text file **`automata.in`** and enter input values for your program, for example, those given in the sample input.

2. Your program should not prompt the user for input. The data will come from your input file. Use **`scanf`**`()` to read your input.

3. **Do not start coding right away!** Think first about a strategy and commit your algorithm on a piece of paper. Use Stepwise Refinement strategy as you start coding.

4. Compile your program with:

   ```
   $ gcc -o automata automata.c
   ```

5. Run your program with:

   ```
   $ ./automata < automata.in
   ```

   where **`automata.in`** is your input file to the program.

6. When completed, upload your source **`automata.c`** and data file **`automata.in`** as Assignment 4 to the course website by **1:15 p.m. Monday, October 7**. Don't forget to fully document your source!

7. Your program will be graded based on the following criteria:
   a. Correctness – produces the correct result that is consistent with I/O specifications.
   b. Design – employs a good modular design, function prototypes.
   c. Efficiency – contains no redundant coding, efficient use of memory.
   d. Style – uses meaningful names for identifiers, readable code, documentation.

For more information about cellular automata see http://mathworld.wolfram.com/ElementaryCellularAutomaton.html.