Linear cellular automata

A cellular automaton consists of a grid of cells. Each cell is in a certain state, and there are only a finite number of possible states. The content of the whole grid, called a generation, changes in one step according to some, usually simple rules.

The rules for computing a new generation satisfy four requirements

1. the rules use only the current generation to compute the next one.
2. the rules define one state value per cell
3. the rules are the same for each cell
4. the rules define the new state value based on the state values in a set of cells in fixed positions with respect to the cell the value is computed for; such a set is often referred to as the neighbourhood of the cell.

This exercise concerns linear cellular automata, i.e. automata with a one-dimensional array of cells as grid. The neighbourhood of a cell consists of the immediate neighbours in the array and the cell itself. Furthermore, there are only two different states, referred to as “occupied” and “empty” (≡ not occupied). In the book A new kind of science Stephen Wolfram, the creator of the Mathematica tool, shows that such simple automata can display very interesting behavior. This exercise will show the effect of some rules.

The following three types of automaton are considered

A An automaton of type A has the following fixed rule set

1. if a cell is currently occupied, it remains occupied only if exactly one neighbour is occupied
2. if a cell is currently empty, it remains empty only if both neighbours are empty

B An automaton of type B has the following fixed rule set

1. if a cell is currently occupied, it remains occupied only if the right neighbour is empty
2. if a cell is currently empty, it becomes occupied if exactly one neighbour is occupied

U For an automaton of type U (called a universal automaton) the rule set is defined by a table of the following form (where boolean values true and false are used to indicate the occupied and empty state, respectively)
where $b_0$, $b_1$, ..., and $b_7$ are boolean values. (NB Obviously the rule sets for automata of type A and B can also be defined in this way).

You are asked to develop a program that reads the description of an automaton (including initial configuration) from standard input, and then computes and displays a specified number of generations. The input is specified as follows (all items mentioned are separated by white space)

1. the letter A, B, or U indicating the type of automaton to be executed
2. a positive integer $L$ being the number of cells of the automaton
3. a positive integer $G$ being the number of generations to be computed and displayed (including the initial configuration)
4. the word `init_start`, followed by one or more positive integers, followed by the word `init_end`. The integers are the positions of the occupied cells in the initial configuration (the first cell in the grid has position 1). All other cells are empty in the initial configuration. When an integer is higher than the number of cells $L$ it should be ignored
5. in case of automaton type U (universal automaton) 8 0’s and 1’s representing the boolean values $b_0$, $b_1$, ..., and $b_7$ mentioned in the description of the rule set for a universal automaton above (0 representing false, 1 representing true).

The output should be a sequence of $G$ lines representing the successive generations. The first line is the initial generation. Each line consists of $L$ characters each representing the state of one cell. An empty cell is displayed as a space, an occupied cell as the character *. Each line is terminated with an end-of-line.

Input and corresponding output are illustrated in the following 3 examples

**Example 1**

**Input**

A 11 10
init_start 6 init_end

**Output**

2
Example 2
Input

B
61
20
init_start
20 40
init_end

Output

Example 3 Input
Your program should introduce methods for subtasks like printing a generation, computing the next generation, computing the new state of a cell, etc. For each of the methods introduced you should give a specification such that the method can be used without knowing its implementation (preferably you should give the precondition and postcondition of pure methods (procedures, result type \texttt{void}), and the precondition and result of functions).

Some hints follow regarding the design and implementation of your program:

- Represent the grid of cells as an array of booleans. Let boolean value \texttt{true} represent an occupied cell, and boolean value \texttt{false} an empty cell. This is a natural choice in the case of two values, allowing simple coding of the rules: the values in the cells can be directly used to form boolean expressions that can be used in guards. To avoid having to deal with head and tail borderline cases for the neighbourhood definition, add one extra cell to the beginning and one extra cell at the end of the array (dummy cells). These cells will not be displayed and are always empty (\texttt{false}, in our case), but their state will be used in the computation of the next state of the first and last actual cell.

- To compute the state values of the cells in the next generation the current state values of the cells are needed. Therefore, the newly computed state values should be stored in a separate array until they are all computed. Only then should the new state values be copied to the array representing the grid, thereby overwriting the old state values.

- Represent the rule set of a universal automaton as an array of booleans of length 8 storing the boolean values $b_0$, $b_1$, \ldots, and $b_7$ mentioned in the description. Computing the new state value of a cell can be done as follows: assume the left neighbour and the cell itself are occupied and the right neighbour is empty; the states of the neighbourhood of the cell can be summarized in bit sequence 110; this bit sequence is interpreted as the binary representation of the integer number 6; the new state value of the cell is given by the element with index 6 in the array representing the rule set; in the other cases the computation can be done in a similar way.