COS126: General Computer Science	Spring 1996
Problem Set # 9	
Handed out: April 24, 1996	*Due: May 1, 1996*

In this assignment you will write a simulator for digital circuits of the kind we have been discussing in lecture. Your simulator should work for both combinational and sequential circuits.

The simulator will read a file containing the description of a circuit using AND gates, OR gates and inverters (i.e. NOT gates). No other logic gates are allowed as building blocks for the circuit. Circuit inputs and outputs will be treated like gates, and when we say "gates" (versus "logic gates") we will include circuit inputs and outputs (more on the details of the description later). The simulator will work interactively. After reading the file describing the circuit, the simulator will prompt the user for input values for the circuit, calculate the outputs of the circuit, and display the output values.

The simulation algorithm:

The algorithm you will use for simulation is called *event-driven simulation.* This means that the simulator computes the output value of a logic gate only if an input value of the gate has changed (this change is the "event"). One way to do this is to keep a queue of gates that need to be processed. Each gate in the queue has one or more inputs that have changed. The algorithm removes a gate from the queue, looks up its input values, and computes its output value. If this computed output value is a change of output value, the algorithm determines which gates use this output as an input and adds those gates to the queue. If the computed output value is the same value as the old output value for the gate, no new gates are added to the queue. A simulation is begun by initializing all the outputs of all the gates as "unknown". Then the special gates representing the inputs to the circuit are put on the queue. Each new input value is considered when its input gate is removed from the queue; new gates that have inputs connected to this input gates are put on the queue. When the queue is empty, the simulation for the given set of input values is complete, and the values of the outputs of the circuit are known. Note that an output gate is put on the queue whenever a circuit output changes value. A circuit output can change value several times before it "settles" to a final value. This is part of the normal operation of many digital circuits. Processing an output gate does not add any new gates to the queue — an output gate is just a mechanism for representing the output of the circuit.

After the simulation of initial input values described in the paragraph above, the algorithm processes any change in input values requested by the user by using the current values for all the gate outputs as the values at the beginning of the new simulation. The input gates for the inputs whose values are changing are put on the queue, and the simulation is resumed. For circuits with memory, like flip-flops, it is imperative that the simulation *not re-initialize* gate outputs – that is where the values are being remembered.

The circuit description:

The circuit to be simulated will be described in a file of the following format:

Line 0 (the first line) of the file contains 3 integers: the number of inputs (m), the number of logic gates (AND gates, OR gates and inverters) (g), and the number of outputs (n).

The next $m + g + n$ lines will each contain the informaion for one input, logic gate, or output, as detailed below. Inputs, outputs, and logic gates will be identified by their line number in this file.

Lines 1 through m of the file each contain the information for an input. The line for an input contains a comma-separated list of logic gates and outputs (by line number) whose inputs are connected to this input.

Lines $m + 1$ through $m + g$ of the file each contain the information for one logic gate Each of these lines is separated into three fields, separated by one or more spaces:

field1 type of gate, one of `"A"` for AND gate `"O"` for OR gate or `"N"` for inverter (NOT gate);

field2 comma-separated list of inputs and logic gates (by line number) providing the inputs to this gate;

field3 comma-separated list of logic gates and outputs (by line number) whose inputs come from the output of this gate.

Lines $m + g + 1$ through $m + g + n$ of the file each contain the information for an output. The line for an output contains a comma-separated list of logic gates and inputs (by line number) whose outputs are connected to this circuit output.

Interacting with the user

The user provides the description of the circuit to be simulated by giving a command-line argument that is the name of the file containing the description, e.g.

```
%a.out myflipflop
```

Once the simulator has read the file, it should prompt the user for an value for each input, one at a time, *using the line number (1 through m) to identify the input.* Once the user has completed entering the input values, the simulator simulates the circuit. When the simulation is complete, the simulator reports the values of the circuit outputs on the screen, one value on each line *using an output number that is the line number of the output minus* $m + g$ to identify the output (so that output numbers are 1 through n). The simulator then asks the user if he/she wants to quit. If the user types “Y”, the program is terminated; if the user types “N”, the simulator prompts the user for a value for each input and continues as above. *Please do not change this protocal for quitting the simulation. The grader will rely on it to simplify testing your program.*

The assignment:

Write a circuit simulator with the behavior described above. You are free to design your own data structures. However, note that the consecutive numbering of gates (i.e. circuit inputs, logic gates, and circuit outputs) makes using an array to store the information for each gate a viable alternative. Gates can be uniquely identified by their line number in the file.

You will need three, not two, values for each gate output: 1, 0, or "unknown". All gate outputs are initialized to "unknown". However, after the first simulation for a circuit, if the user types "N" to the "quit?" prompt, so that the circuit will be re-simulated with new input values, *do not re-initialize all gate outputs to "unknown"*. The simulator cannot simulate sequential circuits if it does not hold gate outputs from one set of input values to the next.

Boolean logic is extended to include the value "unknown" as an input as follows:

inverter: if the input to an inverter is "unknown", the output is "unknown";

AND gate: if any input to an AND gate is "unknown," the output is "unknown" *unless* another of the inputs to the AND gate is a 0, in which case the output of the AND gate is 0;

OR gate: if any input to an OR gate is "unknown," the output is "unknown" *unless* another of the inputs to the OR gate is a 1, in which case the output of the OR gate is 1;

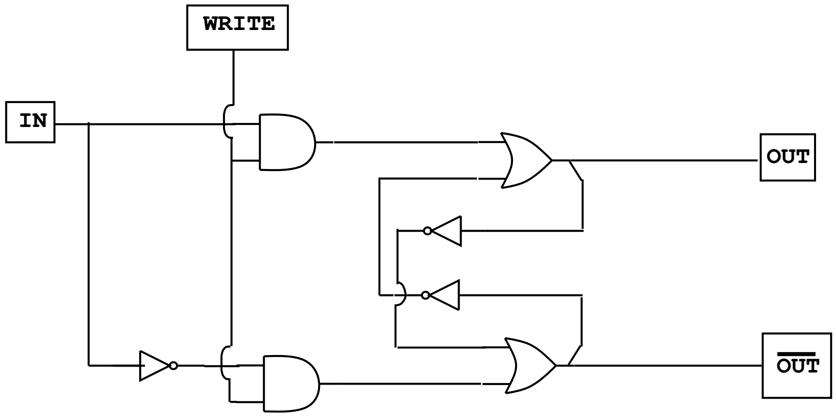
Some advice: keep your design modular and test modules individually. For example, write a routine to read the file and build the data structure for the circuit, and check this routine before using it. Similarly, implement the queue and test it before using it as part of the simulation algorithm.

Try your simulator on the files /u/cs126/data/circuit1 and /u/cs125/data/circuit2. See /u/cs126/data/README.circuit1 and /u/cs126/data/README.circuit2 for information on what inputs to use for the simulation.

Note that there are circuits whose outputs do not settle — they keep oscillating between 0 and 1. In this case, the queue is never empty and the simulator is in an infinate loop. One must be prepared to "control-C" out of such a loop. Find an example of such a circuit in /u/cs126/data/circuit3 (with information in /u/cs126/data/README.circuit3

An example:

Consider the flip-flop on page 4 of the notes for Lecture 20 and shown below.



Here is the contents of a file describing that circuit. (Note that there is not a unique order for listing the gates; I have just chosen one.)

```
2 7 2
3,4
3,5
A 1,2  6
N 1  5
A 2,4  8
O  3,9  7,10
N 6  8
O 5,7  9,11
N 8 6
6
8
```

Extra credit: The user interface of this simulator is particularly unpleasant because the user has to remember the number of each circuit input and each circuit output. In the data file, let us add a field to the line for each input and output that give a name for the input or output: a character string up to 20 characters long. Rewrite your simulator to prompt for input values using the input names and report output values using the output names. With this improvement, the file for the example above would contain

```
2 7 2
3,4 in
3,5 write
A 1,2  6
N 1  5
A 2,4  8
O  3,9  7,10
N 6  8
O 5,7  9,11
N 8 6
6 out
8 outbar
```

What to submit: You should submit your program as "simulator.c". Your electronic submission is due 10pm May 1; hardcopy is due in lecture May 2. If you do the extra credit, submit it as "bettersimulator.c".