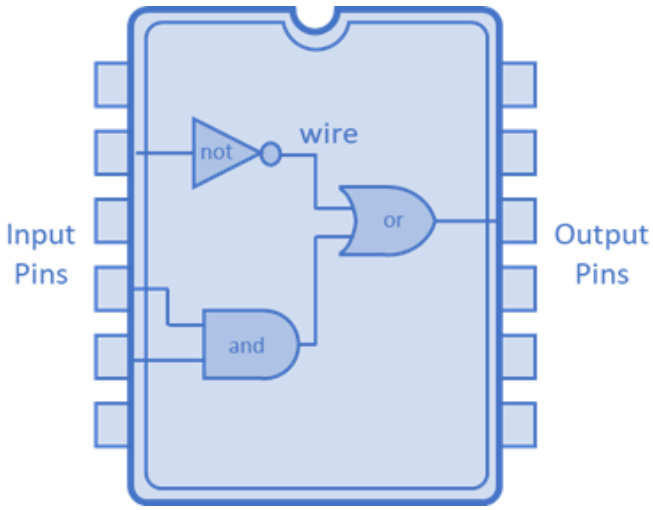
Circuit Sim

You are part of a team that designs electrical circuits. Your job is to provide your colleagues with a simulator that can be used to test the circuit design and verify if it is working correctly before the circuit will be produced in quantities.



A circuit is represented by a set of wires where each wire can have a binary value '0' or '1'. The wires are connected to each other with ports that can perform the bitwise operations 'and' (any number of inputs are combined by a bitwise 'and' that results in a single output), 'or' (any number of inputs are combined by a bitwise 'or' that results in a single output) and 'not' (one input that results in one output by a bitwise inversion of the input). Some wires are considered the input or output of the complete circuit and are connected to the external pins of the chip. There are no cycles in the circuit wiring.

Computation of Outputs

For the specification of the circuits, a semi-colon separated list is used that specifies the wire labels and the bitwise operations that are applied to compute them. The label of a wire is an identifier starting with a letter and containing only letters and digits. The external input/output wires are labeled with all-uppercase identifiers and the internal wiring of the circuit is labeled with all-lowercase identifiers. The operation is specified by its name, followed by a comma separated list of inputs between parentheses. This list of inputs contains exactly one input for a 'not' port but can have any number of inputs for the 'and'/'or' port (note, that an 'and' with no inputs will output a '1' and an 'or' with no inputs will output a '0').

As a simple example:

```
c = or(A, B);
d = and(A, B);
e = not(A);
f = not(B);
g = or(e, f);
h = and(e, f);
I = and(c, g);
J = or(d, h)
```

There are four wires labeled with an uppercase identifier 'A', 'B', 'I' and 'J'. The wires 'A' and 'B' are not the result of any operation and are therefore the input of the circuit, 'I' and 'J' are the result of an operation and are therefore the output of the circuit. If we want to know how this circuit behaves, we could work this out by computing the output for every input as shown in the following table (apparently, I and J are the 'xor' and 'xnor' bitwise operations on the inputs):

A	B	c	d	e	f	g	h	I	J
0	0	0	0	1	1	1	1	0	1
0	1	1	0	1	0	1	0	1	0
1	0	1	0	0	1	1	0	1	0
1	1	1	1	0	0	0	0	0	1

However, for our circuit simulator, we assume that the value on the input pins is provided as a binary string (a sequence of bits in lexicographical order of the input pins) and that the simulator produces the values on the output as a binary string (a sequence of bits in lexicographical order of the output pins). So, when taking the yellow row in the table above as an example, if "01" is provided as input (A = 0, B = 1), the output of the simulated circuit is "10" (I = 1, J = 0).

Question: write a (Java) simulator that accepts a file for the circuit definition and a string with the binary input pin values, and that produces the binary output pin values as a string; assuming the following circuit is provided, what is the associated output if the input pin values are "01101010"?

```
a0z = not(A0);
a1z = not(A1);
a01p = and(A0, a1z);
a01pn = not(a01p);
a01m = and(a0z, A1);
a01mn = not(a01m);
a01pmo = or(a01p,a01m);
a01pmon = not(a01pmo);
b0z = not(B0);
b1z = not(B1);
b01p = and(B0, b1z);
b01pn = not(b01p);
b01m = and(b0z, B1);
b01mn = not(b01m);
b01pmo = or(b01p,b01m);
b01pmon = not(b01pmo);
c0 = and();
c1 = and();
c0z = not(c0);
c1z = not(c1);
c01p = and(c0, c1z);
c01pn = not(c01p);
c01m = and(c0z, c1);
c01mn = not(c01m);
c01pmo = or(c01p,c01m);
c01pmon = not(c01pmo);
z01pab = and(a01p, b01p, c01mn);
z01pac = and(a01p, b01mn, c01p);
z01pbc = and(a01mn, b01p, c01p);
c2 = or(z01pab, z01pac, z01pbc);
z01mab = and(a01m, b01m, c01pn);
z01mac = and(a01m, b01pn, c01m);
z01mbc = and(a01pn, b01m, c01m);
c3 = or(z01mab, z01mac, z01mbc);
rbc01z = and(b01pmon, c01pmon);
rbc01p = or(b01p, c01p);
rbc01m = or(b01m, c01m);
rbc01pe = and(rbc01p, b01pmo, c01pmo);
rbc01me = and(rbc01m, b01pmo, c01pmo);
rbc01pez = or(rbc01pe, rbc01z);
rbc01mez = or(rbc01me, rbc01z);
rac01z = and(a01pmon, c01pmon);
rac01p = or(a01p, c01p);
rac01m = or(a01m, c01m);
rac01pe = and(rac01p, a01pmo, c01pmo);
rac01me = and(rac01m, a01pmo, c01pmo);
rac01pez = or(rac01pe, rac01z);
rac01mez = or(rac01me, rac01z);
rab01z = and(a01pmon, b01pmon);
rab01p = or(a01p, b01p);
rab01m = or(a01m, b01m);
rab01pe = and(rab01p, a01pmo, b01pmo);
rab01me = and(rab01m, a01pmo, b01pmo);
rab01pez = or(rab01pe, rab01z);
rab01mez = or(rab01me, rab01z);
r01p1 = and(a01p, rbc01pez);
r01p2 = and(b01p, rac01pez);
r01p3 = and(c01p, rab01pez);
R0 = or(r01p1, r01p2, r01p3);
r01m1 = and(a01m, rbc01mez);
r01m2 = and(b01m, rac01mez);
r01m3 = and(c01m, rab01mez);
R1 = or(r01m1, r01m2, r01m3);
a2z = not(A2);
a3z = not(A3);
a23p = and(A2, a3z);
a23pn = not(a23p);
a23m = and(a2z, A3);
a23mn = not(a23m);
a23pmo = or(a23p,a23m);
a23pmon = not(a23pmo);
b2z = not(B2);
b3z = not(B3);
b23p = and(B2, b3z);
b23pn = not(b23p);
b23m = and(b2z, B3);
b23mn = not(b23m);
b23pmo = or(b23p,b23m);
b23pmon = not(b23pmo);
c2z = not(c2);
c3z = not(c3);
c23p = and(c2, c3z);
c23pn = not(c23p);
```

```
c23m = and(c2z, c3);
c23mn = not(c23m);
c23pmo = or(c23p,c23m);
c23pmon = not(c23pmo);
z23pab = and(a23p, b23p, c23mn);
z23pac = and(a23p, b23mn, c23p);
z23pbc = and(a23mn, b23p, c23p);
Z2 = or(z23pab, z23pac, z23pbc);
z23mab = and(a23m, b23m, c23pn);
z23mac = and(a23m, b23pn, c23m);
z23mbc = and(a23pn, b23m, c23m);
Z3 = or(z23mab, z23mac, z23mbc);
rbc23z = and(b23pmon, c23pmon);
rbc23p = or(b23p, c23p);
rbc23m = or(b23m, c23m);
rbc23pe = and(rbc23p, b23pmo, c23pmo);
rbc23me = and(rbc23m, b23pmo, c23pmo);
rbc23pez = or(rbc23pe, rbc23z);
rbc23mez = or(rbc23me, rbc23z);
rac23z = and(a23pmon, c23pmon);
rac23p = or(a23p, c23p);
rac23m = or(a23m, c23m);
rac23pe = and(rac23p, a23pmo, c23pmo);
rac23me = and(rac23m, a23pmo, c23pmo);
rac23pez = or(rac23pe, rac23z);
rac23mez = or(rac23me, rac23z);
rab23z = and(a23pmon, b23pmon);
rab23p = or(a23p, b23p);
rab23m = or(a23m, b23m);
rab23pe = and(rab23p, a23pmo, b23pmo);
rab23me = and(rab23m, a23pmo, b23pmo);
rab23pez = or(rab23pe, rab23z);
rab23mez = or(rab23me, rab23z);
r23px = and(a23p, rbc23pez);
r23py = and(b23p, rac23pez);
r23pz = and(c23p, rab23pez);
R2 = or(r23px, r23py, r23pz);
r23mx = and(a23m, rbc23mez);
r23my = and(b23m, rac23mez);
r23mz = and(c23m, rab23mez);
R3 = or(r23mx, r23my, r23mz)
```

Note: in order to reduce complexity, the circuit definition is already ordered such that any operation that is defined only relies on pin-inputs and/or on the outputs of earlier defined wire values. This means that there is no need to apply graph algorithms (unless you can't hold yourself back of course).

Partial Input

The hardware engineers deem stability testing of the circuit one of the key features of the simulator. They wish to examine the stability of the output wrt. undetermined input values. This is to be simulated by using an 'X' as a value for the input pin, instead of a '0' or '1'. In the simple 'xor' / 'xnor' example given above: if the input "X1" was provided (A = X, B = 1), the output of the simulator must be "XX" (I = X, J = X) because both values of I and J are undetermined. In fact, for the 'xor' and 'xnor' example, this is the case if any of the inputs is undetermined, so the circuit can considered to be unstable w.r. t. changing inputs.

Question: adapt the simulator to implement behavior for undetermined input values; assuming the same circuit is provided as in the first part, what is the output if the input pin values are "01XX1101"?