Victoria University of Wellington
School of Engineering and Computer Science

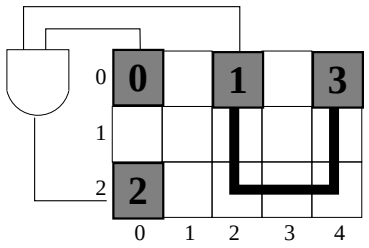
SWEN221: Software Development

Mid-term Test (worth 10% of overall mark)

Circuit Simulation

In this test you will be working on a program for simulating *electronic circuit boards*. The program reads in a file which represents a *circuit board*, and runs the circuit for a number of clock cycles. An example circuit file is given below on the left, along with a graphical visualisation on the right:

```
5,3
4 (0,0:T) (2,0:T) (0,2:F) (4,0:F)
1 3->1[4,2 2,2]
1 G(A:0,1,2)
```



The first line gives the width and height of the circuit board in that order. In this case, the board is `5` squares wide and `3` squares deep. The second line begins with the number of pins, followed by their positions on the board and initial value (where `T` is true, and `F` is false). Each pin is identified by its position in this list, starting from zero (e.g. pin 1 is `(2,0:T)`). The third line begins with the number of wires, followed by a string encoding each wire. The string `"3->1[4,2 2,2]"` defines a wire from pin 3 to pin 1, with joints at positions `4,2` and `2,2`.

The final line of a circuit file begins with the number of components followed by a string encoding each component. Initially, the valid component strings are:

- `G(A/O/X:a,b,c)` — indicates a *logic gate* with type `A`, `O` or `X` which connects pins `a` and `b` (i.e. the *input pins*) to pin `c` (i.e. the *output pin*). The gate type is one of: `A` (for AND gate), `O` (for OR gate), and `X` (for XOR gate).

Download

You can download the code provided for the circuits simulation from here:

`http://ecs.victoria.ac.nz/~djp/files/test-300413.jar`

You will find several Java source files, including a JUnit test file.

Submission

You should submit your solutions through the usual assignment submission system. Please make sure you submit to the correct session. The URL for submission is:

`http://ecs.victoria.ac.nz/cgi-bin/auth/submit?course=SWEN221`

Late submissions will get zero marks (unless you have arranged this with us, which will only be in exceptional circumstances).

PLEASE TURN OVER

1 Debugging the Parser (worth 10%)

Begin by importing the code provided into Eclipse and running the JUnit tests. At least the first five tests should fail, and helpful information regarding these errors is printed into the Eclipse console. There are two bugs in `Parser.java`, and your aim is to find and correct these mistakes. Upon completing this, you should find that tests `validFile_1`, ..., `validFile_5` now pass. **NOTE:** you should also find that many or all of the other tests now fail!

2 Logic Gates (worth 15%)

You should find that some or all of the tests `validFile_6`, ..., `validFile_12` currently fail. This is because the classes `OrGate` and `XorGate` are implemented incorrectly. Your aim is to implement the methods `OrGate.getOutput()` and `XorGate.getOutput()` correctly. Once finished, you should find that tests `validFile_6`, ..., `validFile_12` now pass.

3 Pin Positions (worth 10%)

This part concerns the following requirements for a circuit board:

1. Every pin should occupy a position within the dimensions of the circuit board.
2. No two pins should occupy the same square on the circuit board.

You should find that some or all of the tests `invalidFile_13`, ..., `invalidFile_16` currently fail. This is because `Parser.java` does not check that the above requirements are met. Your should extend `Parser.checkValidBoard()` to throw a `SyntaxError` when the above requirements are not met. Once finished, you should find that tests `invalidFile_13`, ..., `invalidFile_16` now pass.

HINT: we recommend checking each requirement using a separate method which you call from `Parser.checkValidBoard()`.

4 Pin Usage (worth 15%)

This part concerns the following requirements for a circuit board:

3. No pin should be contained more than once in any component.
4. No pin should be contained by more than one component.

You should find that some or all of the tests `invalidFile_17`, ..., `invalidFile_20` currently fail. This is because `Parser.java` does not check that the above requirements are met. Your should extend `Parser.checkValidBoard()` to throw a `SyntaxError` when the above requirements are not met. Once finished, you should find that tests `invalidFile_17`, ..., `invalidFile_20` now pass.

5 Valid Wiring (worth 10%)

This part concerns the following requirements for a circuit board:

5. The input pin for a wire may not be the input pin of a component.
6. The output pin for a wire may not be the output pin of a component.

You should find the tests `invalidFile_21` and `invalidFile_22` currently fail because `Parser.java` does not check that the above requirements are met. You should extend `Parser.checkValidBoard()` to throw a `SyntaxError` when the above requirements are not met. Once finished, you should find that tests `invalidFile_21` and `invalidFile_22` now pass.

HINT: you should find `Component.isInputPin()` and `Component.isOutputPin()` helpful.

6 Logic Chips I (worth 5%)

This part concerns the following requirements for *logic chips*:

7. In a circuit file, a logic chip is described by a component string `I(x:(A/O/X:a,b,c),...)`. This denotes a logic chip with `x` gates, where each gate is described using the same notation as a single logic gate. For example, `I(2:(A:0,1,2),(O:3,4,5))` describes a logic chip made up of two gates: `(A:0,1,2)` and `(O:3,4,5)`.

8. A logic chip contains $n * 2$ logic gates, for some $n > 0$. Since every gate requires 3 pins, a logic chip requires $n * 2 * 3$ pins which are numbered consecutively from 0. For example, a chip with 2 gates has 6 pins numbered $0, 1, 2, 3, 4, 5$.

You should find that some or all of the tests `invalidFile_23` and `invalidFile_24` currently fail. To help you get started, the function `Parser.parseLogicChip()` has been provided. You should modify this function to check requirement (8) above. Once finished, you should find that tests `invalidFile_23` and `invalidFile_24` now pass.

HINT: you don't need to implement the class `LogicChip` yet!

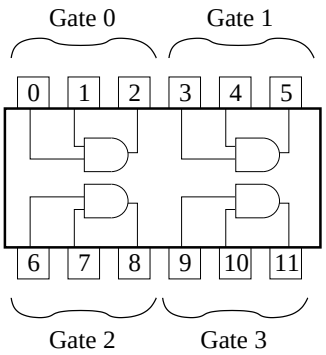
7 Logic Chips II (worth 15%)

The following additional requirements are given:

9. Each gate of a logic chip uses three consecutive pins of the logic chip, with gate 0 using pins 0,1,2, gate 1 using pins 3,4,5, and so on.

10. For each gate, the two lowest numbered pins are its inputs, and the highest its output. For example, for gate 0, the inputs are pins 0 and 1, whilst its output is pin 2.

To help you, the following graphical illustration of a logic chip is provided:



You should find that some or all of the tests `invalidFile_25`, ..., `invalidFile_29` currently fail. To fix this, create a class `LogicChip` which implements the `Component` interface. Once finished, you should find that tests `invalidFile_25`, ..., `invalidFile_29` now pass.

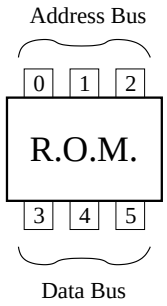
HINT: for this part you don't need to implement `LogicChip.clock()` — just leave it empty for now.

8 Logic Chips III (worth 10%)

You should find that some or all of the tests `validFile_30`, ..., `validFile_33` currently fail. This is because they require the method `LogicChip.clock()` to be implemented. You should study how `LogicGate.clock()` works, and then implement `LogicChip.clock()`. Once finished, you should find that tests `validFile_30`, ..., `validFile_33` now pass.

9 ROM Chips (worth 10%)

This part is concerned with developing a new *ROM chip* component. A ROM chip has 6 pins numbered consecutively from 0 where pins 0,1,2 form the *address bus*, whilst pins 3,4,5 form the *data bus*:



On every clock, the address bus is decoded into an integer *address* between 0 and 7 (following the usual binary representation where e.g. 101 = 5, 011=3, etc). Note, pin 0 holds the *least significant bit*. The *memory* stored at the given address is read, and the data is then written out onto the data bus.

An example circuit file containing a ROM Chip is given below:

```
5,3
6 (0,0:T) (2,0:T) (4,0:0) (0,2:F) (2,2:F) (4,2:F)
0
1 R(0,1,2,3,4,5:0,0,0,6,0,7,0,0)
```

This describes a ROM chip whose memory is stored as follows:

Address	Binary Value
0	0
1	0
2	0
3	110
4	0
5	111
6	0
7	0

In the given file, pin 0 is `true`, pin 1 is `true` and pin 2 is `false`. This decodes to address 3 (since pin 0 is the least significant bit), and the value stored at that address is 6 (or 110 in binary). Therefore, pin 3 should be set `false`, pin 4 should be set `true` and pin 5 should be set `true`.

HINT: two functions are given to help: `CircuitBoard.decode()` and `CircuitBoard.encode()`.

Finally, you should find that some or all of the tests `validFile_34` and `validFile_35` currently fail. Once you've implemented the ROM chip, you should find these tests now pass.