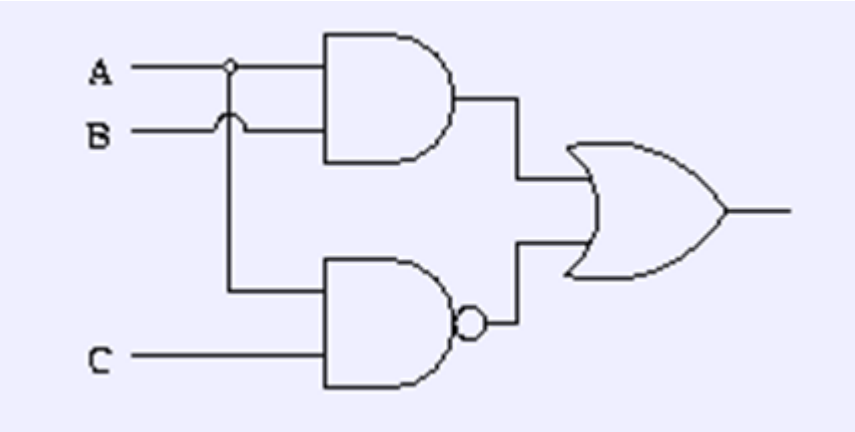
A simple logic simulator that provides a mechanism to run and simulate logic circuits then find the desired outputs for different designs.

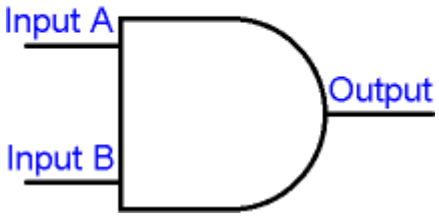


A class called "Node" that has name and value data members to represent each node in the circuit such as node A, B in the above circuit.

Class "Node" should have the following Specifications:

- Default and non-default Constructor.
- Provide setters and getters.
- Overload the operator ostream << to print the node information.
- Create methods called AND() / OR() / XOR() to perform the logical operation between two nodes objects.

A class Called "Gate" which has two input nodes and one output node data members.



Class Gate supports the following methods:

- Default and non-default Constructors.
- Provide getters and setters for its nodes.
- Implement the gate types AND / OR / NAND / NOR / XOR / XNOR / NOT.

- function simulateGate() returns the logic value of the gate according to its type.

A class called "Simulator" which accpets all types of gates then calculate circuits' outputs, It has the following specifications:

- array of pointers to Gate and array of pointers to Node.
- Method postGate() accepts a pointer to a created gate to insert it into the array.
- Method postNode() accepts a pointer to a created node to insert it into the array.
- Method FindNode() accepts a string node's name to find it in the container and return its address.
- Method startSimulate() start the simulation for each gate by looping over the gates container

A class called "GateGenerator" that generates the nodes and all sepcified gates from the input then **post** them to the **simulator**.

"GateGenerator" class should have the following specifications:

- Function parseInput() reads the inputs and parses each keyword ("SIM"/"OUT",..) to the appropriate logic.
- Function createNode() creates a node and return its address.
- Function createGate() creates a (AND,OR,...) gates according to the input and return its address, it is advisable to use a factory function to have better practice on polymorphic objects.