Project 2 Report:

1. My solution uses a recursive approach to compute the fault lists of each gate. Beginning from the output gates, the recursive function calls itself until it reaches PIs. It takes the fault lists of the inputs and then computes the faults of the corresponding output gates.
To avoid re-calculating the fault-lists at gates that are traversed more than once, I used a map data structure as hash table to store the fault-list as key and set its value to true if the gate is traversed. This reduces the exponential running time to quadratic.
I defined all the gate functionalities to simulate the fault lists.
- For AND and NAND since the controlling input is 0, the same function was used to compute the outputs.
- Similarly for OR and NOR the same function is used to compute the fault-list.
- For NOT, BUFFER, FANOUT gates same function was used to simulate fault-lists.
- For XOR and XNOR I had to spend some time to correctly get the functionality.
- I had to use multiple map data structures to keep the input value and the fault-list of each gate and pass them to simulate and generate a fault-list for each gate.

2. I verified my code using the test benches provided.
- target2.bench, target.bench, hw.bench, nandtest.bench, xor.bench. These test benches initially to check if the fault-lists at each gate is getting calculated accurately.
- c17.bench, c432.bench, c5315.bench to check for efficiency of the algorithm.
- Since the project 1 code was working, I used the same approach with changes to call a different function to compute fault-lists.
- Initially instead of check the results for all the input test cases, I used only one test case to check if the results were as expected.
- I tried clocking the speed of the algorithm for the three test benches. For the one with 5315 inputs, the output was computed in less than 2sec.

3. Troubles encountered:
- One main trouble I encountered was I was unable to us the set data-structure which would have made union, intersection, subtraction operations so much easier. That was because the operator '<' had to be overloaded to support vector<Faults>. I was unsure of how to do that.
- To overcome this I coded the intersection and union operations inside the gate definitions by passing simple vectors as inputs.
- To deduce fault-lists for XOR and XNOR gates was initially difficult. I had to spend time to determine how they worked.
- I referred the hw2 problem to get through it.

- The code seems to be working correctly and efficiently without errors.