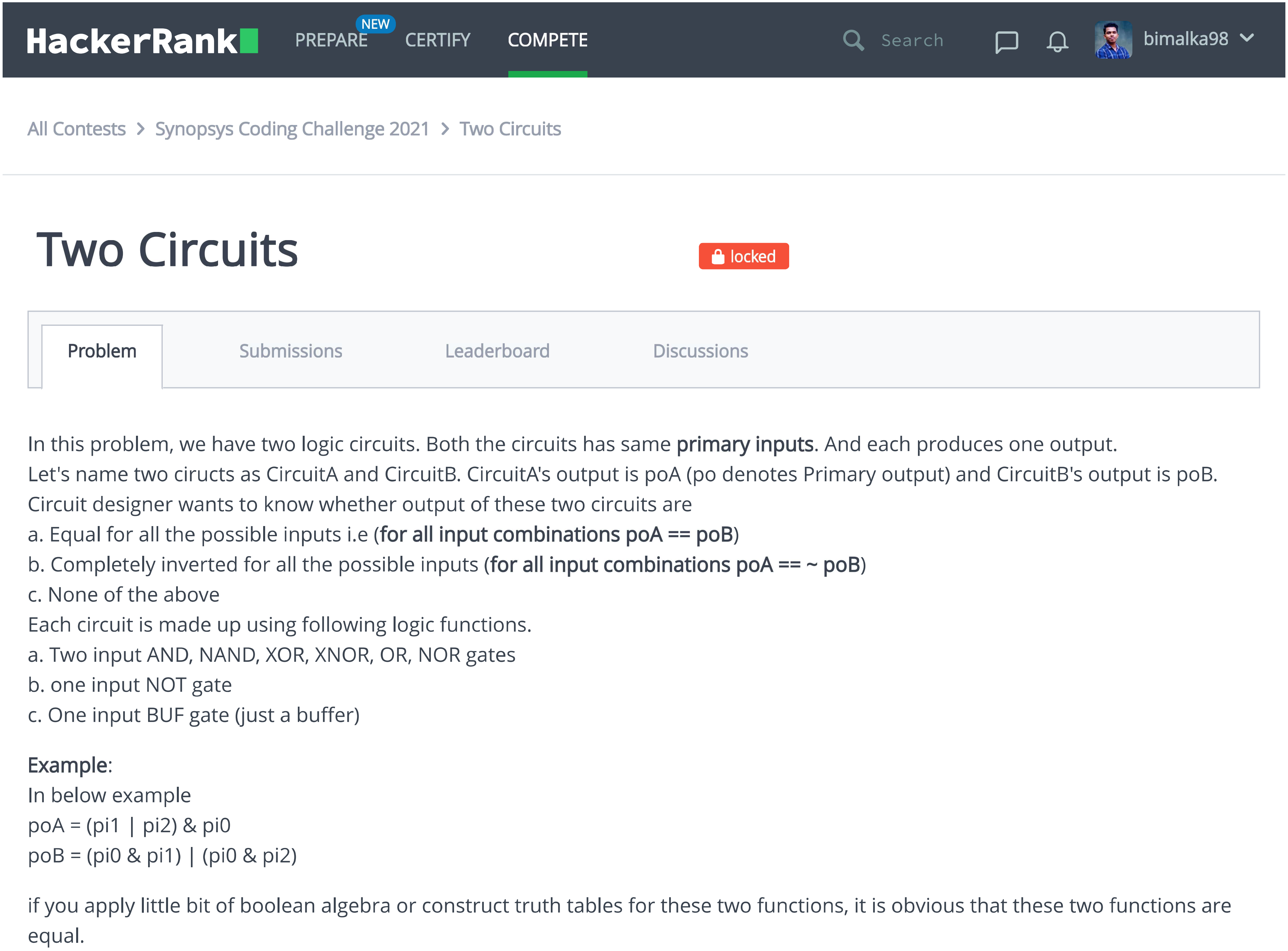
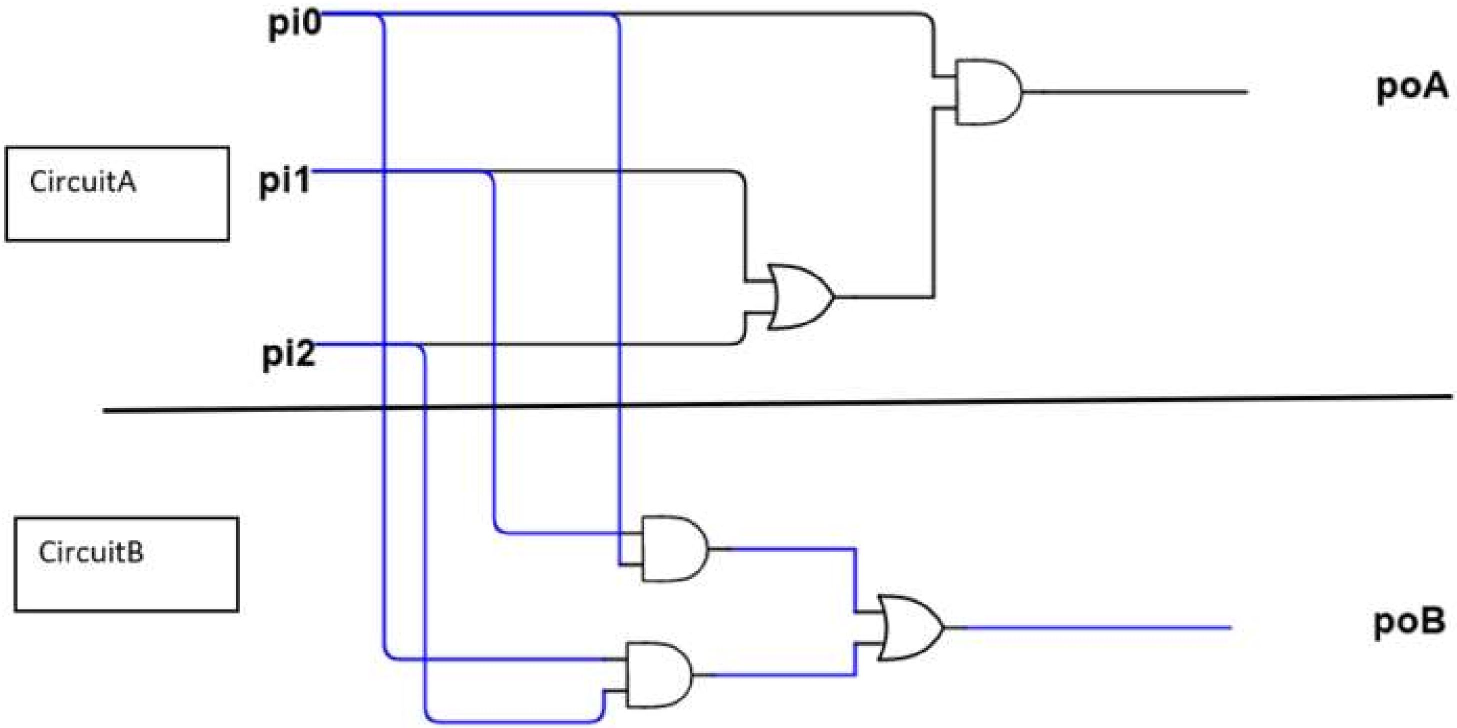


Two Circuits

In this problem, we have two logic circuits. Both the circuits has same **primary inputs**. And each produces one output.
Let's name two ciructs as CircuitA and CircuitB. CircuitA's output is poA (po denotes Primary output) and CircuitB's output is poB.
Circuit designer wants to know whether output of these two circuits are
a. Equal for all the possible inputs i.e (**for all input combinations poA == poB**)
b. Completely inverted for all the possible inputs (**for all input combinations poA == ~ poB**)
c. None of the above
Each circuit is made up using following logic functions.
a. Two input AND, NAND, XOR, XNOR, OR, NOR gates
b. one input NOT gate
c. One input BUF gate (just a buffer)

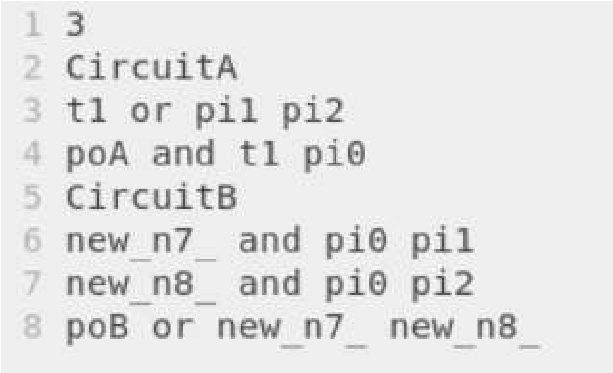
Example:
In below example
poA = (pi1 | pi2) & pi0
poB = (pi0 & pi1) | (pi0 & pi2)

if you apply little bit of boolean algebra or construct truth tables for these two functions, it is obvious that these two functions are equal.



Input Format

Follows the input format for Example given in the description.



```
3
CircuitA
t1 or pi1 pi2
poA and t1 pi0
CircuitB
new_n7_ and pi0 pi1
new_n8_ and pi0 pi2
poB or new_n7_ new_n8_
```

line 1 - **N** number of primary inputs, in this case we have 3 primary inputs. Primary inputs are named from **pi0** to **pi(n-1)**
line 2 - this will always be **CircuitA** to denote the start of CircuitA. After line 2 there will be **variable number of lines** to represent logic gates in the circuit.
line 3 - adds an **or** gate to circuit.

Follows the format of a operation.
output-variable logical-operation operand1 operand2

- Note that operand2 is not applicable for **buf**, **not** operations.
- Also note that if **~** is used before any input operand, it means the inversion.
- Valid logical operations are **or,nor,xor,xnor,and,nand,buf,not**
- operand1/operand2 are **either a primary input or output of a logic gate defined in an earlier line**

in this example, at line4 we have the logic gate driving **poA** which is the output of the **CircuitA**. After this line we start definition of **CircuitB**.
At the last line we have the logic gate driving **poB**, which denotes the completion of CircuitB.

Constraints

N <= 20
Number of **logic gates in a circuit < 300** (**Note**: in place inversion using **~** is not counted as a separate gate, in other words maximum 300 lines of operators will appear for a circuit)
length of any literal in input < 20 characters

Output Format

print single word
- **Identical** if circuits are identical
- **Inverse** if circuitB is ~circuitA
- **None** if none of the above

Sample Input 0

```
3
CircuitA
t1 or pi1 pi2
poA and t1 pi0
CircuitB
new_n7_ and pi0 pi1
new_n8_ and pi0 pi2
poB or new_n7_ new_n8_
```

Sample Output 0

```
Identical
```

Explanation 0

This is the case explained in description.

Sample Input 1

```
4
CircuitA
t1 or pi0 pi1
t2 or t1 pi2
t3 or t2 pi3
poA buf ~t3
CircuitB
new_n1 and ~pi0 ~pi1
new_n2 and new_n1 ~pi2
new_n3 and new_n2 ~pi3
poB not new_n3
```

Sample Output 1

```
Inverse
```

Explanation 1

poA = ~(pi0 | pi1 | pi2 | pi3)
poB = ~(~pi0 & ~pi1 ~pi2 & ~pi3)

if we apply De Morgan's law to poA, we could see poB is the inverted function of poA.

Sample Input 2

```
3
CircuitA
t1 xor pi0 pi1
poA not t1
CircuitB
some_temp_net or pi1 pi2
some_other_temp and pi1 pi0
poB or some_temp_net some_other_temp
```

Sample Output 2

```
None
```

Sample Input 3

```
5
CircuitA
t1 xor pi0 pi1
t2 not t1
t3 or pi4 t2
t4 or pi2 pi3
t5 and t3 t4
poA not t5
```

```
CircuitB
new_n7_ and pi0 ~pi1
new_n8_ and ~pi0 pi1
new_n9_ and ~new_n7_ ~new_n8_
new_n10_ and pi4 ~new_n9_
new_n11_ and ~pi4 new_n9_
new_n12_ and pi4 new_n9_
new_n13_ and ~new_n10_ ~new_n11_
new_n14_ and ~new_n12_ new_n13_
new_n15_ and pi2 ~pi3
new_n16_ and ~pi2 pi3
new_n17_ and pi2 pi3
new_n18_ and ~new_n15_ ~new_n16_
new_n19_ and ~new_n17_ new_n18_
poB or new_n14_ new_n19_
```

Sample Output 3

```
Identical
```

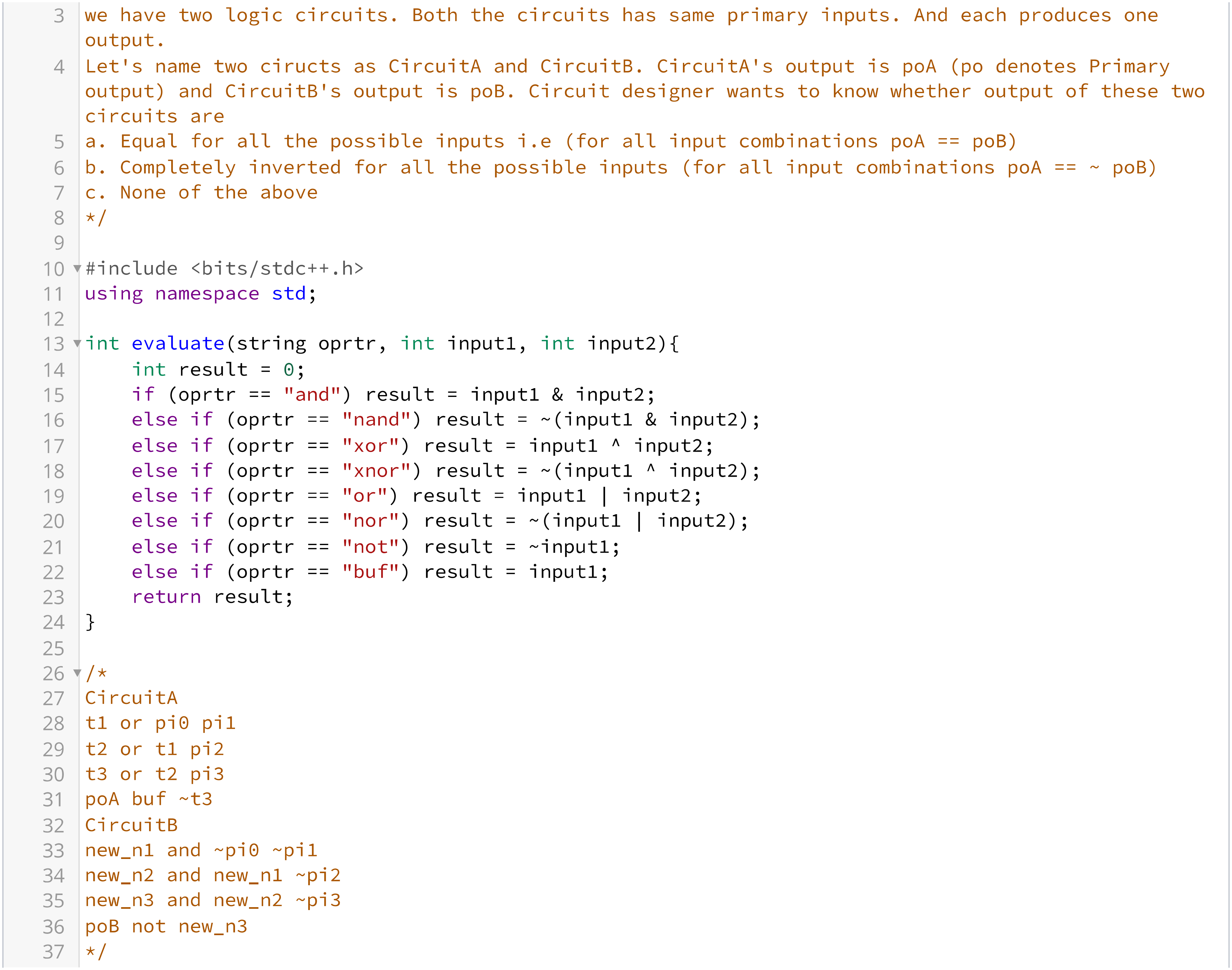
Submissions: 12
Max Score: 100
Difficulty: Hard

Rate This Challenge:
☆ ☆ ☆ ☆ ☆

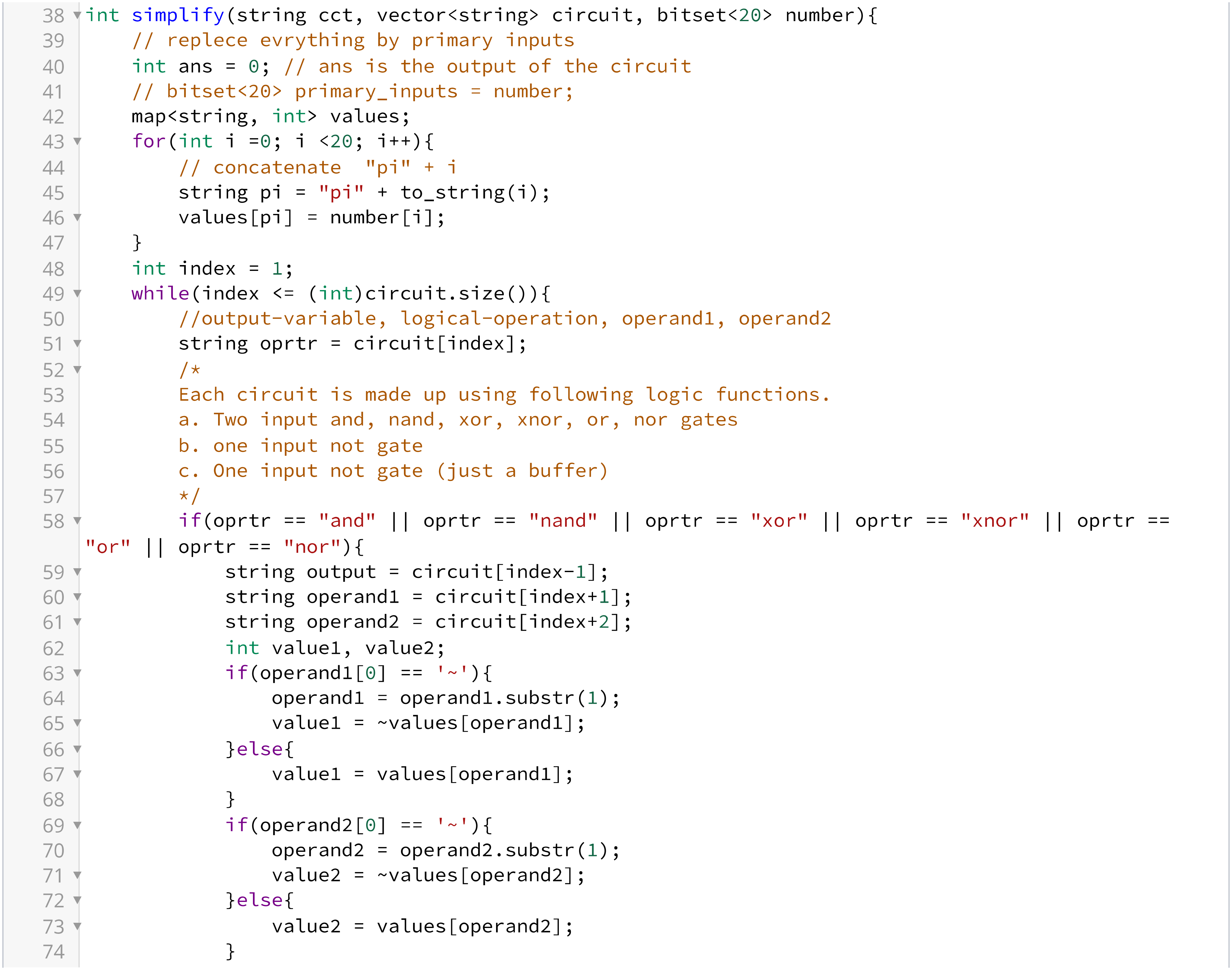
More

C++

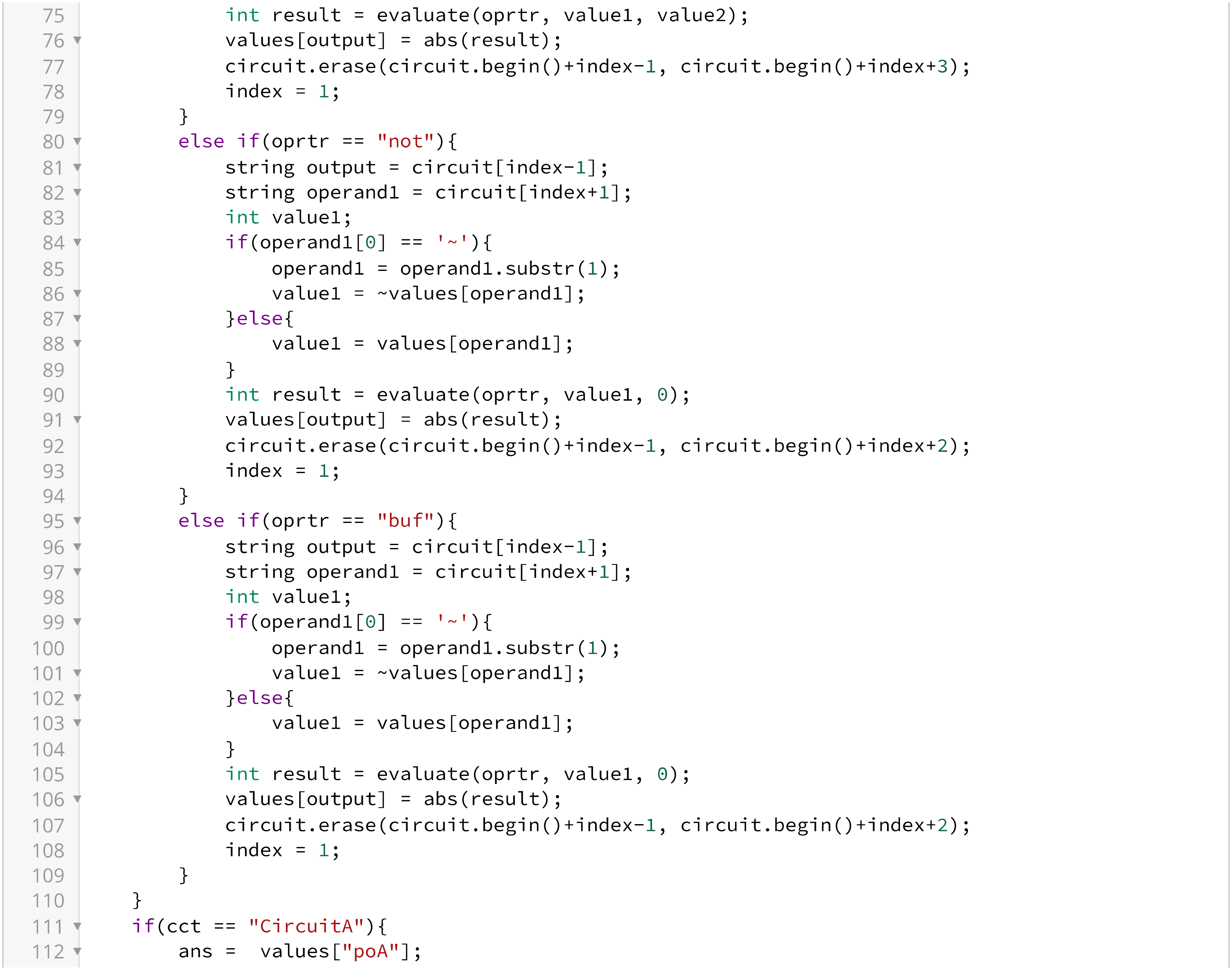
```
/*
Problem Definition:
```



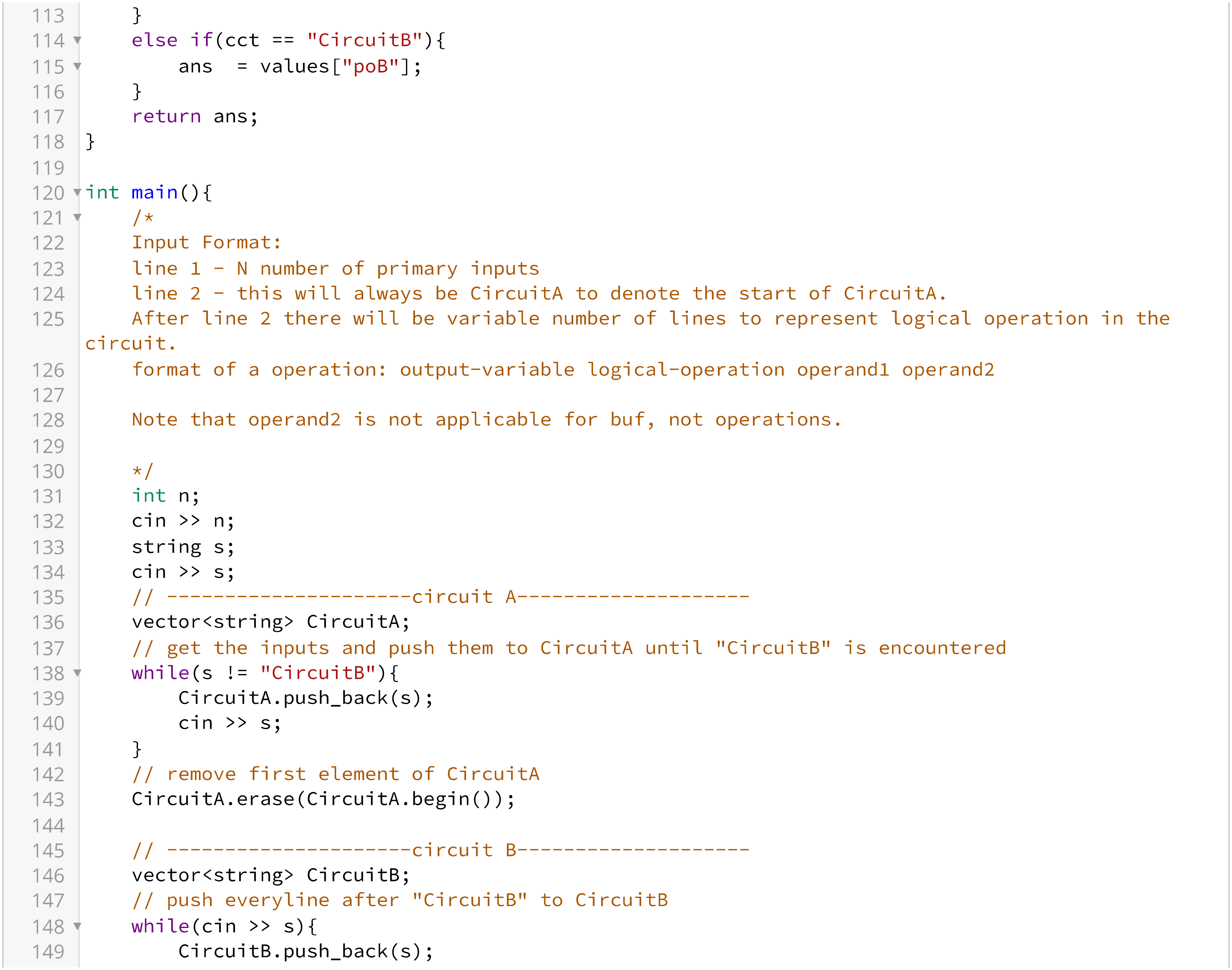
```
we have two logic circuits. Both the circuits has same primary inputs. And each produces one output.
Let's name two ciructs as CircuitA and CircuitB. CircuitA's output is poA (po denotes Primary output) and CircuitB's output is poB. Circuit designer wants to know whether output of these two circuits are
a. Equal for all the possible inputs i.e (for all input combinations poA == poB)
b. Completely inverted for all the possible inputs (for all input combinations poA == ~ poB)
c. None of the above
*/

#include <bits/stdc++.h>
using namespace std;

int evaluate(string oprtr, int input1, int input2){
    int result = 0;
    if (oprtr == "and") result = input1 & input2;
    else if (oprtr == "nand") result = ~(input1 & input2);
    else if (oprtr == "xor") result = input1 ^ input2;
    else if (oprtr == "xnor") result = ~(input1 ^ input2);
    else if (oprtr == "or") result = input1 | input2;
    else if (oprtr == "nor") result = ~(input1 | input2);
    else if (oprtr == "not") result = ~input1;
    else if (oprtr == "buf") result = input1;
    return result;
}

/*
CircuitA
t1 or pi0 pi1
t2 or t1 pi2
t3 or t2 pi3
poA buf ~t3
CircuitB
new_n1 and ~pi0 ~pi1
new_n2 and new_n1 ~pi2
new_n3 and new_n2 ~pi3
poB not new_n3
*/
```



```
int simplify(string cct, vector<string> circuit, bitset<20> number){
    // replece evrything by primary inputs
    int ans = 0; // ans is the output of the circuit
    // bitset<20> primary_inputs = number;
    map<string, int> values;
    for(int i =0; i <20; i++){
        // concatenate  "pi" + i
        string pi = "pi" + to_string(i);
        values[pi] = number[i];
    }
    int index = 1;
    while(index <= (int)circuit.size()){
        //output-variable, logical-operation, operand1, operand2
        string oprtr = circuit[index];
        /*
        Each circuit is made up using following logic functions.
        a. Two input and, nand, xor, xnor, or, nor gates
        b. one input not gate
        c. One input not gate (just a buffer)
        */
        if(oprtr == "and" || oprtr == "nand" || oprtr == "xor" || oprtr == "xnor" || oprtr == "or" || oprtr == "nor"){
            string output = circuit[index-1];
            string operand1 = circuit[index+1];
            string operand2 = circuit[index+2];
            int value1, value2;
            if(operand1[0] == '~'){
                operand1 = operand1.substr(1);
                value1 = ~values[operand1];
            }else{
                value1 = values[operand1];
            }
            if(operand2[0] == '~'){
                operand2 = operand2.substr(1);
                value2 = ~values[operand2];
            }else{
                value2 = values[operand2];
            }
```



```cpp
            int result = evaluate(oprtr, value1, value2);
            values[output] = abs(result);
            circuit.erase(circuit.begin()+index-1, circuit.begin()+index+3);
            index = 1;
        }
        else if(oprtr == "not"){
            string output = circuit[index-1];
            string operand1 = circuit[index+1];
            int value1;
            if(operand1[0] == '~'){
                operand1 = operand1.substr(1);
                value1 = ~values[operand1];
            }else{
                value1 = values[operand1];
            }
            int result = evaluate(oprtr, value1, 0);
            values[output] = abs(result);
            circuit.erase(circuit.begin()+index-1, circuit.begin()+index+2);
            index = 1;
        }
        else if(oprtr == "buf"){
            string output = circuit[index-1];
            string operand1 = circuit[index+1];
            int value1;
            if(operand1[0] == '~'){
                operand1 = operand1.substr(1);
                value1 = ~values[operand1];
            }else{
                value1 = values[operand1];
            }
            int result = evaluate(oprtr, value1, 0);
            values[output] = abs(result);
            circuit.erase(circuit.begin()+index-1, circuit.begin()+index+2);
            index = 1;
        }
    }
    if(cct == "CircuitA"){
        ans =  values["poA"];
```



```cpp
    }
    else if(cct == "CircuitB"){
        ans  = values["poB"];
    }
    return ans;
}

int main(){
    /*
    Input Format:
    line 1 - N number of primary inputs
    line 2 - this will always be CircuitA to denote the start of CircuitA.
    After line 2 there will be variable number of lines to represent logical operation in the circuit.
    format of a operation: output-variable logical-operation operand1 operand2

    Note that operand2 is not applicable for buf, not operations.

    */
    int n;
    cin >> n;
    string s;
    cin >> s;
    // ---------------------circuit A--------------------
    vector<string> CircuitA;
    // get the inputs and push them to CircuitA until "CircuitB" is encountered
    while(s != "CircuitB"){
        CircuitA.push_back(s);
        cin >> s;
    }
    // remove first element of CircuitA
    CircuitA.erase(CircuitA.begin());

    // ---------------------circuit B--------------------
    vector<string> CircuitB;
    // push everyline after "CircuitB" to CircuitB
    while(cin >> s){
        CircuitB.push_back(s);
    }

    // print the elements of CircuitA
    // cout << "CircuitA: ";
    // for(auto i : CircuitA) cout << i << endl;
    // // print the elements of CircuitB
    // cout << "CircuitB: ";
    // for(auto i : CircuitB) cout << i << endl;
    // ---------------------simplify--------------------
    // get the simplified circuit
    // int out1 = abs(simplify("CircuitA", CircuitA, bitset<20>(0)));
    // int out2 = abs(simplify("CircuitB", CircuitB, bitset<20>(0)));
    // // print the simplified circuit
    // cout << "CircuitA: " << out1 << endl;
    // cout << "CircuitB: " << out2 << endl;
    /*
    check whether output of these two circuits are
    a. Equal for all the possible inputs i.e (for all input combinations poA == poB)
    b. Completely inverted for all the possible inputs (for all input combinations poA == ~ poB)
    c. None of the above
    */
    // check whether output of these two circuits are equal for all the possible inputs
    bool identical = true;
    bool inverted = true;
    for(int i = n/4; i < 3*n/4; i++){
        bitset<20> number(i);
        int out1 = abs(simplify("CircuitA", CircuitA, number));
        int out2 = abs(simplify("CircuitB", CircuitB, number));
        if(out1 != out2){
            identical = false;
            break;
        }
    }
    // check whether output of these two circuits are completely inverted for all the possible inputs
    for(int i = n/4; i < 3*n/4; i++){
        bitset<20> number(i);
        int out1 = abs(simplify("CircuitA", CircuitA, number));
        int out2 = abs(simplify("CircuitB", CircuitB, number));
        if(out1 != ~out2){
            inverted = false;
            break;
        }
    }
    if(identical){
        cout << "Identical" << endl;
    }
    if(inverted){
        cout << "Inverted" << endl;
    }
    if(!identical && !inverted){
        cout << "None" << endl;
    }
}
```

Line: 1 Col: 1

Upload Code as File ☐ Test against custom input Run Code Submit Code