```
#include "FibLFSR.hpp"

// Constructor to create LFSR with the given initial seed
FibLFSR::FibLFSR(string seed) {
    // Checks for invalid input
    if(seed.size() != 16) {
        throw length_error("Parameter of size "+to_string(seed.size()));
    }
    if(notZeroOne(seed)){
        throw invalid_argument("Parameter contains !1 && !0");
    }
    // Converts to bitset for easy xors and shifts
    bitset<16> bits(seed);
    state = bits;
}

// Simulate one step and return the new bit as 0 or 1
int FibLFSR::step() {
    int feedback = 0;
    for(int i = 0; i < 4; i++){
        // Performs the 4 xors for the 4 taps
        feedback = XOR(state[TAPS[i]], feedback);
    }
    // Leftshifts, inserts result, returns result
    state <<= 1;
    if (feedback) state[0] = 1;
    return feedback;
}

// Simulate k steps and return a k-bit integer
int FibLFSR::generate(int k) {
    int total = 0;
    for(int i = 0; i < k; i++){
        // For every iteration, add one on success and shift bits left
        total *= 2;
        if(step()) total +=1;
    }
    return total;
}

// Getters:
string FibLFSR::getState(void) const {
    return state.to_string();
}
const int* FibLFSR::getTaps(void) const {
    return TAPS;
}

// Output operator overload
ostream& operator<<(ostream& out, const FibLFSR& lfsr){
    out << lfsr.getState();
    return out;
}

// Helpers:
int FibLFSR::XOR(int a, int b) {
    if ((a || b) && !(a && b)) return 1;
    else return 0;
}
bool FibLFSR::notZeroOne(const string seed) {
    for(char a : seed){
        if(a != '0' && a != '1') return 1;
    }
    return 0;
}
```