```
 1: CC= g++
 2: CFLAGS= -Wall -Werror -ansi -pedantic
 3: Boost= -lboost_unit_test_framework
 4: all:
 5:         make test
 6: ps1a: FibLFSR.cpp
 7:         $(CC) FibLFSR.cpp -c -o FibLFSR.o $(CFLAGS)
 8: test: test.cpp FibLFSR.o
 9:         $(CC) test.cpp FibLFSR.o $(CFLAGS) $(Boost) -o test
10:
11: clean:
12:         rm *.o *.out test
```

```
 1: // "Copyright 2020 <Greg Kaplowitz>"
 2: #include <SFML/Graphics.hpp>
 3: #include <string>
 4: #include <iostream>
 5: #include <sstream>
 6: #include <cmath>
 7: #include "FibLFSR.hpp"
 8: // using namespace sf;
 9: // using namespace std;
10:
11:   LFSR::LFSR(std::string Seed) {
12:   seed = Seed;
13:   }
14:   int LFSR::step() {
15:   char endbit = seed[15];
16:   char stepBit;
17:   if (((seed[0] == '1') ^ (seed[2] == '1')) ^
18:   ((seed[3] == '1') ^ (seed[5] == '1'))) {
19:   stepBit = '1';
20:   } else {
21:   stepBit = '0';
22:   }
23:   std::string new_string =  seed.substr(1);
24:   seed = new_string + stepBit;
25:   if (endbit == '1') {
26:   return 1;
27:   }
28:   return 0;
29:   }
30:   std::ostream& operator<< (std::ostream &out, const LFSR &lFSR) {
31:   out << lFSR.seed;
32:   return out;
33:   }
34:   int LFSR::generate(int k) {
35:   int x = 0;
36:   int temp;
37:   for (int i = 0; i < k; i++) {  // run step k times
38:   temp = step();
39:   if (temp == 1) {
40:   x += pow(2, i);
41:   }
42:   }
43:   // cout<<"flag"<<x<<endl;
44:   // cout<<x;
45:   return x;
46:   }
```

```
 1: // "Copyright 2020 <Greg Kaplowitz>"
 2: #ifndef FibLFSR_HPP
 3: #define FibLFSR_HPP
 4:
 5:
 6: #include <string>
 7: #include <iostream>
 8: #include <sstream>
 9: #include <cmath>
10:   // using namespace sf;
11:   using namespace std;
12:
13:   class LFSR {
14:   public:
15:   LFSR(std::string Seed);// constructor
16:   int step();// simulates one step
17:   int generate(int k);// generates an integer of k bits of the processes
18:   friend std::ostream& operator<< (std::ostream &out, const LFSR &lFSR);
19:   private:
20:   std::string seed;
21:   };
22: #endif
```

```
 1: // "Copyright 2020 <Greg Kaplowitz>"
 2: #define BOOST_TEST_DYN_LINK
 3: #define BOOST_TEST_MODULE Main
 4: #include <boost/test/unit_test.hpp>
 5: #include <iostream>
 6: #include <string>
 7:
 8: #include "FibLFSR.hpp"
 9: // using namespace std;
10: BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) {
11:   LFSR l("1011011000110110");
12:   BOOST_REQUIRE(l.step() == 0);
13:   BOOST_REQUIRE(l.step() == 0);
14:   BOOST_REQUIRE(l.step() == 0);
15:   BOOST_REQUIRE(l.step() == 0);
16:   BOOST_REQUIRE(l.step() == 1);
17:   BOOST_REQUIRE(l.step() == 1);
18:   BOOST_REQUIRE(l.step() == 0);
19:   BOOST_REQUIRE(l.step() == 0);
20:
21:   LFSR l2("1011011000110110");
22:   BOOST_REQUIRE(l2.generate(7) == 48);
23: }
24:
25: BOOST_AUTO_TEST_CASE(zeroTest) {
26:   LFSR la("101100");
27:   BOOST_REQUIRE(la.step() == 0);
28:   BOOST_REQUIRE(la.step() == 0);
29:   BOOST_REQUIRE(la.step() == 0);
30:   BOOST_REQUIRE(la.step() == 0);
31:   BOOST_REQUIRE(la.step() == 0);
32:   BOOST_REQUIRE(la.step() == 0);
33:   BOOST_REQUIRE(la.step() == 0);
34:   BOOST_REQUIRE(la.step() == 0);
35:
36:   LFSR l2a("101100");
37:   BOOST_REQUIRE(l2a.generate(8) == 0);
38: }
39:
40: BOOST_AUTO_TEST_CASE(twelveTest) {
41:   LFSR lb("10110110001101100011100011011");
42:   BOOST_REQUIRE(lb.step() == 0);
43:   BOOST_REQUIRE(lb.step() == 0);
44:   BOOST_REQUIRE(lb.step() == 0);
45:   BOOST_REQUIRE(lb.step() == 1);
46:   BOOST_REQUIRE(lb.step() == 1);
47:   BOOST_REQUIRE(lb.step() == 1);
48:   BOOST_REQUIRE(lb.step() == 0);
49:   BOOST_REQUIRE(lb.step() == 0);
50:
51:   LFSR l2b("10110110001101100011100011011");
52:   BOOST_REQUIRE(l2b.generate(8) == 56);
53: }
```