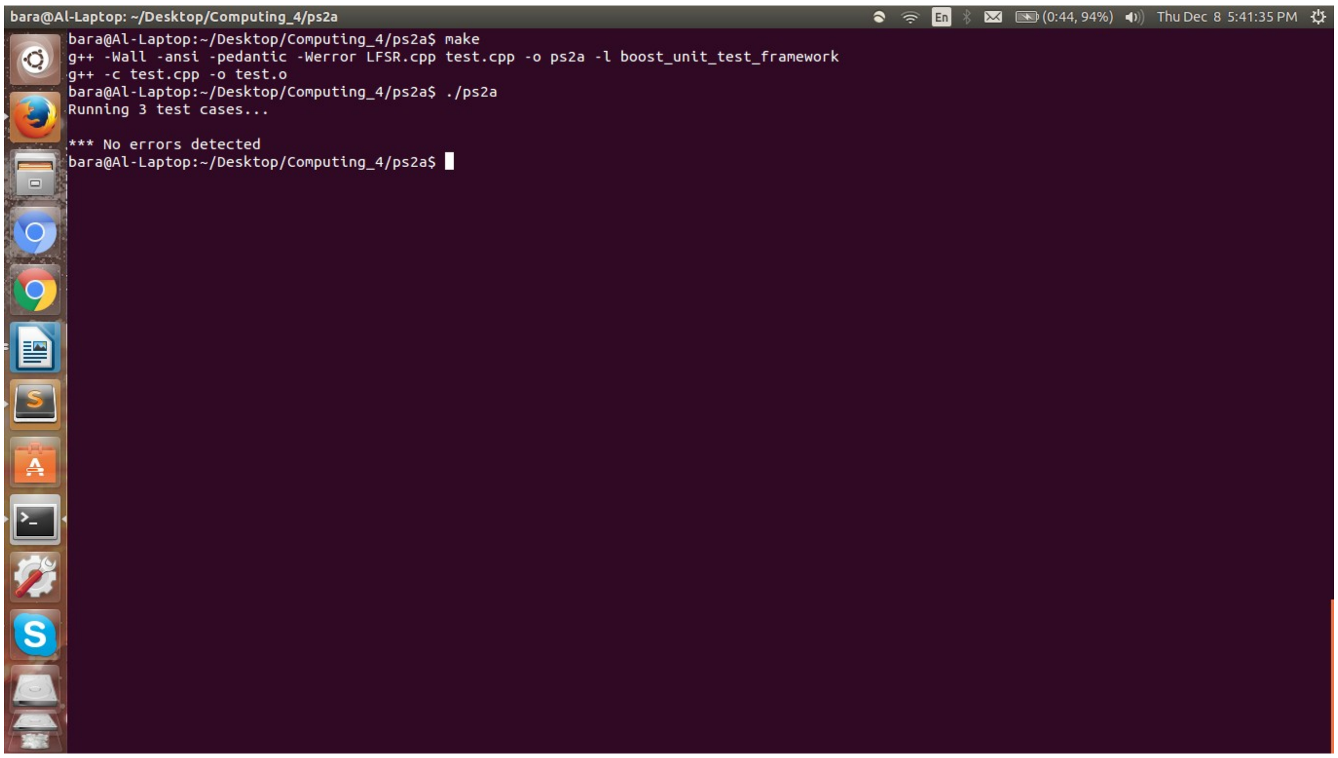
<u>PS2</u>: *Linear Feedback Shift Register and Image Encoding*

<u>PS2a</u>:

This assignment was to learn to register bits that shifts bits one position to the left and replaces the vacated bit by the exclusive or of the bit shifted off and the bit previously at a given tap position in the register. To test our LFSR is working, we needed to use software called boost.

I used this a class that accepts the seed and number of taps for the constructor. The step function would LFSR only once and would return the bit to test in my boost file. The generate function was essentially call the step function but would also give a condition.

I learned about what LFSR was. One big thing I learned is boost. I learned how create test cases having different conditions to test out the step and generate functions. Though, I currently can't find any use for it in my own projects.



```
bara@Al-Laptop:~/Desktop/Computing_4/ps2a$ make
g++ -Wall -ansi -pedantic -Werror LFSR.cpp test.cpp -o ps2a -l boost_unit_test_framework
g++ -c test.cpp -o test.o
bara@Al-Laptop:~/Desktop/Computing_4/ps2a$ ./ps2a
Running 3 test cases...

*** No errors detected
bara@Al-Laptop:~/Desktop/Computing_4/ps2a$ 
```

```
cc = g++
#all

all : LFSR test

LFSR : LFSR.o test.o
	$(cc) -Wall -ansi -pedantic -Werror LFSR.cpp test.cpp -o ps2a -l boost_unit_test_framework

LFSR.o : LFSR.hpp
	$(cc) -c LFSR.cpp -o LFSR.o

test : test.cpp LFSR.hpp
	$(cc) -c test.cpp -o test.o


clean:
		rm *.o ps2a

run:
		./ps2a

debug: cc += -g
debug: ps2a

```

```cpp

#include "LFSR.hpp"

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>

BOOST_AUTO_TEST_CASE(fiveBitsTapAtTwo) {
  //steps through the function 8 times and checks to see the return bit is 1 or 0.
  LFSR l("00111", 2);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 0);

  //Doing the math, it turns out to 198
  LFSR l2("00111", 2);
  BOOST_REQUIRE(l2.generate(8) == 198);

}

BOOST_AUTO_TEST_CASE(Test1){
          // I stepped 5 times by doing the xor to the 2nd tap.
  LFSR l3("0110110", 2);
  BOOST_REQUIRE(l3.step() == 1);
  BOOST_REQUIRE(l3.step() == 0);
  BOOST_REQUIRE(l3.step() == 1);
  BOOST_REQUIRE(l3.step() == 1);
  BOOST_REQUIRE(l3.step() == 1);

}
BOOST_AUTO_TEST_CASE(Test2){
  //goes  through step 3 times and by doing it out by hand, the answer is 7
  //It steps 3 times.
  LFSR l4("0110001", 3);
  BOOST_REQUIRE(l4.generate(4) == 7);
}


```

```


#include <iostream>
#include <string>
#include <cmath>


class LFSR {

public:
  LFSR(std::string seed_, int tap_);
  ˜LFSR();
  int step();
  int generate(int k);

  friend std::ostream& operator<< (std::ostream &out, LFSR &lfsr);

private:
  std::string seed;
  int tap;
};

```

```cpp
/*
Name: ALbara Mehene
Date: 9/25/2016
Computing IV

*/


#include <iostream>
#include <string>
#include <cmath>
#include "LFSR.hpp"

//constructor
LFSR::LFSR(std::string seed_, int tap_){

        seed = seed_;
        tap = tap_;

}


int LFSR::step(){
        int bit;
        int size;

        size = seed.length();//stored the amount of elements

        bit =  seed.at(0) ^ seed [size - tap - 1]; //Took the total elements and subtracted by the tap and by 1


        seed.erase(0, 1); // erased the front element

        if(bit == 1){//condition if its 1, it would return the chracter 1
                seed.push_back('1');
        }
        else{//returns 0 if its anything else
                seed.push_back('0');
        }
        //returns bit to test the test.cpp
        return bit;
}

int LFSR::generate(int k){
        int temp = 0;

        //Condition to test the generate function in test.cpp
        for(int i = k - 1; i >= 0; i--){
                if(step() == 1){
                        temp += pow(2,i);
                }
        }
        return temp;

}
//prints out the string if I were to use the a main. It was not required in this assigment
std::ostream& operator<< (std::ostream &out, LFSR &lfsr){
        out << lfsr.seed;
        return out;
}



LFSR::~LFSR(){

}




```