Sean Evans
CS 6058
Data / Security and Privacy
Spring 2018
Homework 1

Problem 1

```
// Solution
// Key Length = 4
// Message  = FOOTBALL

#include <algorithm>
#include <iostream>
#include <stdlib.h>
#include <string>

int main( int argc, const char* argv[] )
{
    const std::string cipher_text = "JSSXFEPP";

    std::cout << "cipher text = '" << cipher_text << "'" << std::endl;

    for ( unsigned char key = 0 ; key < 26 ; ++key ) {
        std::string plain_text = cipher_text;

        std::cout << "trying key = '" << ( int )key << "'" << std::endl;

        std::transform(
            cipher_text.begin(),
            cipher_text.end(),
            plain_text.begin(),
            [&]( auto const & cipher ) {
                return cipher - key;
            }
        );

        std::cout << "plaintext = '" << plain_text << "'" << std::endl;

        std::cout << std::endl;
    }

    return EXIT_SUCCESS;
}
```

Problem 2

```
// Solution
// The cipher text is 'ZCMLNYGMWTSDAMCAMCCEWM'.
// The key space for a message space of 50 is 50! (50 factorial).

#include <algorithm>
#include <iostream>
#include <stdlib.h>
#include <string>

int main( int argc, const char* argv[] )
{
    static const char* cipher = "EXAUNDKBMVORQCSFHYGWZLJITP";

    std::cout << "substitution cipher is '" << cipher << "'" << std::endl;

    static const std::string plain_text = "universityofcincinnati";

    std::cout << "plain text is '" << plain_text << "'" << std::endl;

    std::string cipher_text = plain_text;

    std::transform(
        plain_text.begin(),
        plain_text.end(),
        cipher_text.begin(),
        [&]( auto const& v ){ return cipher[v - 'a']; }
    );

    std::cout << "cipher text is '" << cipher_text << "'" << std::endl;

    return EXIT_SUCCESS;
}
```

Problem 3

```
// Solution
// The ciphertext is 'famsuevmtimq'
// The key space for a Vigenere Cipher a 4 character string is 26 ^ 4 = 456976

#include <algorithm>
#include <iostream>
#include <stdlib.h>
#include <string>

template <size_t s> class Print;

int main( int argc, const char *argv[] )
{
    static const std::string key = "cats";

    static const std::string plain_text = "datasecurity";

    std::cout << "key is '" << key << "'" << std::endl;

    std::cout << "plain text is '" << plain_text << "'" << std::endl;

    std::string cipher_text = plain_text;

    size_t idx = 0;

    for ( size_t idx = 0 ; idx < plain_text.size() ; ++idx ) {
        cipher_text[idx] =
        ( ( plain_text[idx] + ( key[idx % key.size()] - 'a' ) - 'a' ) % 26 ) + 'a';
    }

    std::cout << "cipher text is '" << cipher_text << "'" << std::endl;

    return EXIT_SUCCESS;
}
```

Problem 4

```
// Solution
// IC = 0.071

#include <algorithm>
#include <iostream>
#include <iterator>
#include <numeric>
#include <stdlib.h>
#include <vector>

static const std::vector<unsigned char> histogram = {
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 5, 5, 5, 5, 5, 5, 5, 10, 10, 10, 10, 10, 0, 0, 0
};

int main( int argc, const char* argv[] )
{
    std::cout << "Computing Index of Coincidence" << std::endl;

    std::cout << "Histogram = ";

    std::copy(
        histogram.begin(),
        histogram.end() - 1,
        std::ostream_iterator<size_t>( std::cout, ", " ) );

    std::copy(
        histogram.end() - 1,
        histogram.end(),
        std::ostream_iterator<size_t>( std::cout ) );

    std::cout << std::endl;

    const auto N = std::accumulate(
        histogram.begin(), histogram.end(), ( size_t ) 0 );

    std::cout << "N = " << N << std::endl;

    const auto IC = std::accumulate(
        histogram.begin(), histogram.end(), ( double ) 0.0,
        [&]( const auto & current_sum, const double & current_value ) {
            const auto p = ( current_value / N );
            const auto pp = p * p;
            return current_sum + pp;
        }
    );

    std::cout << "IC = " << IC << std::endl;

    return EXIT_SUCCESS;
}
```

Problem 5

```
// Solution
// key length = gcd(4, 12) = 4

#include <iostream>
#include <stdlib.h>

template<class T> T gcd( T x, T y ) {
    while ( y != 0 ) {
        T t = y;
        y = x % y;
        x = t;
    }

    return x;
}

int main( int argc, const char* argv[] )
{
    std::cout << "key length = " << gcd( 4, 12 ) << std::endl;

    return EXIT_SUCCESS;
}
```

Problem 6

In order to find the key of a Vigenere Cipher using a known index of coincidence (IC), the key length must first be found. This is accomplished by trying key lengths and calculating the IC of subsequences of the cipher text until a value of IC is found that most closely matches the assumed known IC. These subsequences are generated by the following pattern:

$c_1 \quad c_{1+j} \quad c_{1+2j} \quad c_{1+3j} \ldots$

Where *c* is the cipher text, and *j* is the guess of the key length.

Once the best matching IC is found, this key length is used to generate further subsequences of cipher text. These subsequences are generated using the following pattern:

$c_i \quad c_{i+t} \quad c_{i+2t} \quad c_{i+3t} \ldots$

Where *c* is the cipher text, *i* is the starting index of the cipher text subsequence, and *t* is the key length found previously. The number of cipher text subsequences generated in this manner is equal to the key length. These subsequences are attacked to find the best matching IC, as before, to find the part of the key that corresponds to the starting offset, listed as *i* in the example above.