1. Write a Python/C++ program to generate 1M bytes of cryptographically secure random numbers.

```python
import secrets

with open("random.bin", "wb") as file:
    file.write(secrets.token_bytes(1048576))
```

2. Run the NISTSP800-22 statistical test on your 1M bytes of binary cryptographically secure random numbers and analyze the test results to identify any deviations from the expected statistical properties of random numbers.

```
$ ./assess 8388608
          G E N E R A T O R    S E L E C T I O N
          ---------------------------------------

    [0] Input File                 [1] Linear Congruential
    [2] Quadratic Congruential I   [3] Quadratic Congruential II
    [4] Cubic Congruential         [5] XOR
    [6] Modular Exponentiation     [7] Blum-Blum-Shub
    [8] Micali-Schnorr             [9] G Using SHA-1

   Enter Choice: 0


       User Prescribed Input File: random.bin

               S T A T I S T I C A L   T E S T S
               ---------------------------------

    [01] Frequency                       [02] Block Frequency
    [03] Cumulative Sums                 [04] Runs
    [05] Longest Run of Ones             [06] Rank
    [07] Discrete Fourier Transform      [08] Nonperiodic Template Matchings
    [09] Overlapping Template Matchings  [10] Universal Statistical
    [11] Approximate Entropy             [12] Random Excursions
    [13] Random Excursions Variant       [14] Serial
    [15] Linear Complexity

         INSTRUCTIONS
            Enter 0 if you DO NOT want to apply all of the
            statistical tests to each sequence and 1 if you DO.

   Enter Choice: 1

       P a r a m e t e r   A d j u s t m e n t s
       ----------------------------------------
    [1] Block Frequency Test - block length(M):         128
    [2] NonOverlapping Template Test - block length(m): 9
    [3] Overlapping Template Test - block length(m):    9
    [4] Approximate Entropy Test - block length(m):     10
    [5] Serial Test - block length(m):                  16
    [6] Linear Complexity Test - block length(M):       500
```

```
42
43    Select Test (0 to continue): 1
44
45    Enter Block Frequency Test block length: 65536
46
47          P a r a m e t e r   A d j u s t m e n t s
48          ----------------------------------------
49      [1] Block Frequency Test - block length(M):         65536
50      [2] NonOverlapping Template Test - block length(m): 9
51      [3] Overlapping Template Test - block length(m):    9
52      [4] Approximate Entropy Test - block length(m):     10
53      [5] Serial Test - block length(m):                  16
54      [6] Linear Complexity Test - block length(M):       500
55
56    Select Test (0 to continue): 0
57
58    How many bitstreams? 1
59
60    Input File Format:
61      [0] ASCII - A sequence of ASCII 0's and 1's
62      [1] Binary - Each byte in data file contains 8 bits of data
63
64    Select input mode:  1
65
66       Statistical Testing In Progress.........
67
68       Statistical Testing Complete!!!!!!!!!!!!
69
70  $ cat experiments/AlgorithmTesting/finalAnalysisReport.txt | grep 0/1
```

(a) The Frequency (Monobit) Test
The purpose of this test is to determine whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence.

(b) Frequency Test within a Block
The purpose of this test is to determine whether the frequency of ones in an M-bit block is approximately M/2, as would be expected under an assumption of randomness.

(c) The Runs Test
The purpose of the runs test is to determine whether the number of runs of ones and zeros of various lengths is as expected for a random sequence. In particular, this test determines whether the oscillation between such zeros and ones is too fast or too slow.

(d) Tests for the Longest-Run-of-Ones in a Block
The purpose of this test is to determine whether the length of the longest run of ones within the tested sequence is consistent with the length of the longest run of ones that would be expected in a random sequence.

(e) The Binary Matrix Rank Test
The purpose of this test is to check for linear dependence among fixed length substrings of the original sequence.

(f) The Discrete Fourier Transform (Spectral) Test
The purpose of this test is to detect periodic features (i.e., repetitive patterns that are near each other) in the tested sequence that would indicate a deviation from the assumption of randomness.

(g) The Non-overlapping Template Matching Test
The purpose of this test is to detect generators that produce too many occurrences of a given non-periodic (aperiodic) pattern.

(h) The Overlapping Template Matching Test
The focus of the Overlapping Template Matching test is the number of occurrences of pre-specified target strings. Both this test and the Non-overlapping Template Matching test of Section 2.7 use an m-bit window to search for a specific m-bit pattern.
The difference between this test and the test in Section 2.7 is that when the pattern is found, the window slides only one bit before resuming the search.

(i) Maurer's "Universal Statistical" Test
The purpose of the test is to detect whether or not the sequence can be significantly compressed without loss of information.

(j) The Linear Complexity Test
The purpose of this test is to determine whether or not the sequence is complex enough to be considered random.

(k) The Serial Test
The purpose of this test is to determine whether the number of occurrences of the 2^m m-bit overlapping patterns is approximately the same as would be expected for a random sequence.

(l) The Approximate Entropy Test
The purpose of the test is to compare the frequency of overlapping blocks of two consecutive/adjacent lengths (m and $m + 1$) against the expected result for a random sequence.

(m) The Cumulative Sums (Cusums) Test
The purpose of the test is to determine whether the cumulative sum of the partial sequences occurring in the tested sequence is too large or too small relative to the expected behavior of that cumulative sum for random sequences.

(n) The Random Excursions Test
The purpose of this test is to determine if the number of visits to a particular state within a cycle deviates from what one would expect for a random sequence.

(o) The Random Excursions Variant Test
Its purpose is to detect deviations from the expected behavior of a random sequence by examining the number of excursions to various states within the sequence.

3. Extra credit: Find out a non-cryptographically secure random number generator, such as random(), to demonstrate its lack of safety. Then, propose modifications to enhance its security to generate cryptographically secure random numbers that meet the highest standards of security and reliability.
   Non-secure:
   Uses Python's random module, which is not suitable for security-sensitive applications.

```python
import random

with open("badRandom.bin", "wb") as file:
    file.write(bytes(random.getrandbits(8) for _ in range(1048576)))
```

   Result:

```
$ cat experiments/AlgorithmTesting/finalAnalysisReport.txt | grep 0/1
 1   0   0   0   0   0   0   0   0   0   ----   0/1   NonOverlappingTemplate
```

   Modified:
   Mixes two sources of randomness resulting in a mixed-security byte array.

```python
import random
import os

random_bytes = bytearray(random.randint(0, 255) for _ in range(1048576))
random_bytes += bytearray(os.urandom(1048576))

with open("secureRandom.bin", "wb") as file:
    file.write(bytes(random_bytes))
```

   Result:

```
$ cat experiments/AlgorithmTesting/finalAnalysisReport.txt | grep 0/1
```