CS-627, Cryptology: Assignments

Charles Abzug, Ph.D.
Department of Computer Science
James Madison University
Harrisonburg, VA 22807

Cell Phone: 443-956-9424,
E-mail: CharlesAbzug@ACM.org
Home Page: https://users.cs.jmu.edu/abzugcx/public/index.htm

Programming Project 1

PART *I*:

Purpose: (i) to Compile statistics on frequency of occurrence of individual letters and letter groups in a Text, and
(ii) to prepare a clean plaintext file from a crude source file for use as input to another program that will encrypt the text.

Summary of Program Behavior:

- **Queries the user to identify *which* file (Index Number) on which to operate.**
- **Continually informs the user of the progress of program execution.**
- **Takes a text file as input — whether ASCII or Unicode (either/or, doesn't have to handle both).**
- **Strips out non-alpha characters from the input text file, and outputs an all-lower-case alpha file for subsequent use as input to the PART *IIa* encryption program.**
- **Counts the frequency of occurrence of each individual letter of plaintext.**
- **Counts the number of instances of every possible digram, trigram, tetragram, and pentagram.**
- **Produces a screen display and an output file reporting the statistical findings, as well as the date-and-time stamps of the program progress milestones.**

Programming Project 1

PART *I*:

(1) Locate and compile text file groups from *at least* five distinct sources, such as:

- **High-literary-quality news reports or essays (*e.g.*, *New York Times*, especially the Sunday magazine section; *National Review*; *Wall Street Journal*)**
- **Classical works of distinguished authorship (*e.g.*, Shakespeare, Melville, Stevenson, Poe, Clemens, Dickens, Tennyson)**
- **Scientific writing (*e.g.*, *Scientific American*, *Science*, *Nature*, or a journal specific to some particular scientific discipline, such as *Journal of Physiology*, *Mathematical Gazette*, *Geographic and Global Issues Quarterly*)**
- **Computer Science (*e.g.*, *IEEE Computer*, *Communications of the ACM*, *Dr. Dobbs' Journal*)**
- **Other sources of distinctly different content**

(2) Each sample of text to contain at least 1,000 alphabetic characters.

(3) Save the original source file in whatever form you found it, *e.g.*, .pdf, .doc, or .mht (except for .htm or .html documents, which you should save in .mht format).

Programming Project 1

PART *I* (continued):

(4) NAMING CONVENTIONS for DATA files (index *n*):

TEXT-SOURCE (for each textual source): source-*n*.htm, source-*n*.html, source-*n*.mht, source-*n*.pdf, source-*n*.doc, or whatever file type may be appropriate

INPUT: crude-plaintext-*n*.txt — **Includes formatting (line feed, tabs, *etc.*), indentation, spacing, non-alpha chars**

$OUTPUT_1$: processed-plaintext-*n*.txt — **Lower-case alphabetic characters exclusively; upper case converted to lower case, and everything else stripped off.**

$OUTPUT_2$: statistics-*n*.txt — **Frequency of occurrence (%) for each letter**
Number of instances of each non-zero-occurrence digraph
Number of instances of each non-zero-occurrence trigraph

Programming Project 1

PART *I* (continued):

(5) NAMING CONVENTION for PROGRAM file:

crypto-project-part-I-*yourJMUusername*

For example, had I written the program: crypto-project-part-I-*abzugcx*

Programming Project 1

PART *I* (continued):

(6) Program Behavior (not necessarily in chronological order):

Queries the user for a two-decimal-digit Index Number (*n*).

Measures and informs user of the size (number of characters) of the input file.

Informs the user of current activity — updated every 5 seconds, and includes in each update message the current date and 24-hr-format time.

Informs the user of major program milestones, via time-stamped messages output to the display, such as:

(a) Program starting to run (date, and time in 24-hr format).

(b) Generating an output file consisting of All-Lower-Case-Alpha (date-time).

(c) Completing the generation of the All-Lower-Case-Alpha processed-plaintext file (date-time).

(d) Beginning to Count the number of occurrences of the individual letters and letter groups (date-time).

(e) Completed the counting of letter and letter-group occurrences (date-time).

(f) Outputting the processed-plaintext file (date-time).

(g) Adding milestone-achievement information to the statistics file (date-time).

(h) Printing plaintext statistics and program execution information (date-time).

(h) Program operation successfully completed; exiting at: Date-Time.

Programming Project 1

PART *I* (continued):

(7) Specification for Content and Format of Statistical Output File:

- **(a) Name of INPUT file (*i.e.,* "crude-plaintext-*n*.txt").**
- **(b) Name of OUTPUT processed-plaintext file (*i.e.,* "processed-plaintext-*n*.txt")**
- **(c) Total number of characters in INPUT file.**
- **(d) Number of Alphabetic characters in both the INPUT and the OUTPUT files.**
- **(e) Letters in alphabetic order, with number and percentage occurrence of each (occurrences aligned on units digit, percentages calculated to precision of 0.01%, aligned on decimal point).**
- **(f) Digrams actually found, sorted into alphabetic order, with number of occurrences of each, aligned on units digit.**
- **(g) For each digram, show all trigrams found having the same starting characters and the number of occurrences of each. The trigrams should be indented so that they are shown to the right of the digram information, in the lines immediately below the digram.**
- **(h) Similarly for tetragrams and pentagrams.**
- **(i) A separate listing of all digrams and the number of occurrences of each, sorted in decreasing order of number of occurrences; same for trigrams, tetragrams, and pentagrams.**
- **(j) All of the milestones shown to the user during program execution.**

Programming Project 1

PART *I* **(continued):**

Sample Output:

```
  INPUT file: crude-plaintext-99.txt.
OUTPUT file: processed-plaintext-99.txt

Total number of characters in INPUT file:  14,267
Number of Alphabetic characters present:  12,348

Occurrences in the plaintext of individual letters:
          a   1,017   8.23%
          b     185   1.49%
          .
          .
          .
```

(continued)

Programming Project 1

PART *I* **(continued):**

Sample Output **(continued):**

Occurrences in the plaintext of letter groups (alphabetic order):

```
ab   47
        abd  12
        abn   3
        abo  21
                abor   9
                        abori    3
                                   aborig    3
                        abort    5
an    8
        ann    6
.
.
.
```

(continued)

Programming Project 1

PART *I* (continued):

Sample Output (continued):

.

.

.

30 Feb 2020, 0631 hrs: Program starting to run.

30 Feb 2020, 0642 hrs: Generating the processed-plaintext output file.

FURTHER INSTRUCTION:

You must also provide a narrative file (either *.doc, .htm, or .pdf*) describing the details of your approach. Your narrative should give an overview of how you tackled the problem, as well as a table listing all defined constants, all named variables, and all functions and classes that you originated. If the name of any of these items is less than fully evocative of its purpose, then add a brief explanation.

File name: narrative-crypto-project-part-I-yourJMUusername.*pdf*

Programming Project 1

PART *II*: Vigenère Encryption and Decryption

Purpose: (1) To encrypt a plaintext file using the Vigenère technique, based upon an encryption key supplied by the user.

(2) To decrypt a ciphertext file that had originally been encrypted using the Vigenère technique, based upon a decryption key supplied by the user.

PART *IIa*: Vigenère ENcryption

Program Behavior:

(1) Queries the user for the two-decimal-digit Index Number (*n*) associated with the plaintext message input: an ASCII text file composed ENTIRELY of lower-case Alpha characters (*i.e.,* one of the output-files from PART *I*).

(2) Checks the user-specified file to see whether or not it meets the specified input conditions. If not, outputs to the user an appropriately-worded message, and then aborts.

Programming Project 1

PART *IIa*: Vigenère Encryption (continued)

(3) Queries the user for a key word or key phrase, which MAY NOT include numerals, symbols or non-printing characters, BUT MAY include spaces and tabs, and may be of mixed case. If the user-supplied key word or key phrase does not meet the specification, outputs to the user an appropriately-worded error message, and then aborts.

(4) Computes the EFFECTIVE Key-Word or Key-Phrase by converting the input to all-lower-case and stripping off the second and subsequent appearances of any character present more than once in the input word or phrase.

(5) Outputs the ciphertext in the form of an all-upper-case-alpha ASCII file (plus spaces and/or tabs, and carriage-returns and line-feeds or newline characters).

Programming Project 1

PART *IIa*: Vigenère Encryption (continued)

(6) NAMING CONVENTIONS for PROGRAM files:

(a) You will have EITHER one program that can do BOTH encryption & decryption, OR two programs, ONE for encryption and ONE for decryption.

(b_1) Your choice: ONE program: crypto-project-part-II-*yourJMUusername*

For example, had I written one program: crypto-project-part-II-*abzugcx*

(b_2) Alternative:
TWO programs: crypto-project-part-II*A*-yourJMUusername (for ENcryption)
and crypto-project-part-II*B*-yourJMUusername (for DEcryption)

For example, had I written *two* programs: crypto-project-part-II*A*-*abzugcx*
and crypto-project-part-II*B*-*abzugcx*

Programming Project 1

PART *IIa*: Vigenère Encryption (continued)

(7) NAMING CONVENTIONS for DATA files (index *n*):

PLAINTEXT SOURCE (input file): processed-plaintext-*n*.txt (output from pt *I*)
Lower-Case Alphabetic characters exclusively; upper case has been converted by the PART *I* program to lower case, and everything else has been stripped off.

$OUTPUT_1$: ciphertext-*n*.txt
Upper-Case alphabetic characters
FIVE-character groups
THREE-character space separating each pair of groups
Length of each line not to exceed 80 characters
No partial blocks.

$OUTPUT_2$: enciphering-info-*n*.txt
Original user-supplied raw encryption key.
Effective encryption key following processing.

Programming Project 1

PART *IIb*: Vigenère DEcryption

(1) Queries the user for the two-decimal-digit Index Number (*n*) associated with the ciphertext message input: an ASCII text file composed ENTIRELY of five-character blocks of upper-case Alpha characters, plus spaces and newline or carriage-return/linefeed characters.

(2) Checks the user-specified file to see whether or not it meets the specified input conditions. If not, outputs to the user an appropriately-worded message, and then aborts.

(3) Queries the user for a key word or key phrase, which MAY NOT include numerals, symbols or non-printing characters, BUT MAY include spaces and tabs, and may be of mixed case. If the user-supplied key word or key phrase does not meet the specification, outputs to the user an appropriately-worded error message, and then aborts.

(4) Computes the EFFECTIVE Key-Word or Key-Phrase by converting the input to all-lower-case and stripping off the second and subsequent appearances of any character present more than once in the input word or phrase.

Programming Project 1

PART *IIb*: Vigenère Decryption (continued)

(5) Outputs the *plain*text in the form of an ASCII file.

(7) NAMING CONVENTIONS for DATA files (index *n*):

CIPHERTEXT SOURCE (input file): ciphertext-*n*.txt (output from pt *II*)
Upper-case alphabetic characters, plus spaces and linefeeds.

$OUTPUT_1$: deciphered-text-*n*.txt Lower-Case alphabetic characters:
FIVE-character groups
THREE-character space separating each pair of groups
Length of each line not to exceed 80 characters
No partial blocks.

$OUTPUT_2$: deciphering-info-*n*.txt Original user-supplied raw decryption key.
Effective decryption key following processing.

Programming Project 1

PART *IIb*: Vigenère Decryption (continued)

FURTHER INSTRUCTION:

You must also provide a narrative file (either *.doc, .htm, or .pdf*) describing the details of your approach. Your narrative should give an overview of how you tackled the problem, as well as a table listing all defined constants, all named variables, and all functions and classes that you originated. If the name of any of these items is less than fully evocative of its purpose, then add a brief explanation.

File name: narrative-crypto-project-part-II-yourJMUusername.*pdf*

Programming Project 1

PART *III*: Cryptanalysis of a Vigenère Encipherment

Purpose: To cryptanalyze a Vigenère-encrypted ciphertext, recovering the effective key originally used to encrypt the plaintext (*i.e.*, the original key with duplicate letters stripped off), as well as the plaintext.

Program Behavior:

(1) Calculates the overall Index Of Coincidence (*IOC*) of a Vigenère–encrypted ciphertext.

(2) Determines the effective length of the Vigenère encryption key; two suggested approaches:

- **(a) Use the mathematical formula derived by William Friedman to calculate a best estimate of keylength, and then try out several nearby values looking for best fit. (The "educated guess" method)**
- **(b) Try all possible values of effective keylength from 2 through 26, calculating for each possible keylength value the IOC for each monoalphabetic substitution, and looking for the keylength value that gives the best fit. (The "brute force" method)**

Programming Project 1

PART *III*: Cryptanalysis of a Vigenère Encipherment (continued)

(3) Outputs two files:

$OUTPUT_1$: cryptanalytic-progression-*n*.txt shows the steps taken and each successive calculation performed, ***e.g.*:**

Trying keylength 4.
IOC = 4.73, 4.94, 4.47, 4.69
Trying keylength 5.
IOC = 4.99, 5.23, 5.14, 4.83, 5.08
.
.
.
FINAL KEYLENGTH = 10
EFFECTIVE KEYWORD = hieronymus

$OUTPUT_2$: recovered-plaintext-*n*.txt shows the Vigenère decryption of the cryptanalyzed ciphertext using the effective keyword

Programming Project 1

PART *III*: Cryptanalysis of a Vigenère Encipherment (continued):

(4) NAMING CONVENTION for PROGRAM file:

crypto-project-part-III-*yourJMUusername*

For example, had I written the program: crypto-project-part-III-*abzugcx*

FURTHER INSTRUCTION:

You must also provide a narrative file (either *.doc, .htm, or .pdf*) describing the details of your approach. Your narrative should give an overview of how you tackled the problem, as well as a table listing all defined constants, all named variables, and all functions and classes that you originated. If the name of any of these items is less than fully evocative of its purpose, then add a brief explanation.

File name: narrative-crypto-project-part-III-yourJMUusername.*pdf*

Programming Project 1

PART *IV* : Rivest-Shamir-Adelman (*RSA*) encryption and decryption

Purpose: (1) To generate a pair of prime numbers, *p* and *q*;
(2) To make use of the two primes to generate a pair of *RSA* keys, one private and the other public, and finally
(3) To use the two keys to encrypt and decrypt a plaintext message.

PART *IVa*: Generation of an *RSA* Key Pair

Program Behavior:

(1) Generates a random number of 32-bit length: *p*.

(2) Tests the number for primality:
(a) "Sieve of Eratosthenes"
(b) Fermat Test.

NOTE: There may be other primality-testing techniques that are practicable for 32-bit primes, but for this exercise you should use a technique that is

Programming Project 1

PART *IVa*: Generation of an *RSA* Key Pair (continued):

readily extensible for use with 512-bit primes. Thus, for example, with a 32-bit randomly-generated number, it may be practical to test for primality by dividing the number by all known primes up to 16 bits in length. However, since this technique cannot be extended to 512-bit numbers, you should not use it.

(3) Repeats steps 1 and 2 to produce a second 32-bit prime: q.

(4) Defines the first member of an *RSA* key pair, a public key, as the pair of numbers consisting of:

(a) The 4th Fermat number: $2^{16} + 1$

(b) $n = p \bullet q$

(5) Applies the Extended Euclidian Algorithm to the 4th Fermat number to obtain its multiplicative inverse modulo $(p - 1) \bullet (q - 1)$

Programming Project 1

PART *IVa*: Generation of an *RSA* Key Pair (continued):

(6) Defines the second member of the *RSA* key pair, the private key, as the pair of numbers consisting of:

(a) The multiplicative inverse of the 4^{th} Fermat number, obtained in the previous step.

(b) $n = p \bullet q$

(7) Data Output Files: each an ASCII file consisting of two lines: exponent in first line, modulus in second line.

RSA-private-key-*yourJMUusername*

RSA-public-key-*yourJMUusername*

(8) FILE NAMING CONVENTIONS for PROGRAM file:

crypto-project-part-IVa-*yourJMUusername*

For example, had I written the program: crypto-project-part-IVa-*abzugcx*

Programming Project 1

PART *IVb*: *RSA* Encryption and Decryption

Program Behavior: ENCRYPTION MODE

(1) Queries the user for the two-decimal-digit Index Number (n) associated with the plaintext message input: an ASCII text file composed ENTIRELY of lower-case Alpha characters (i.e., one of the output-files from PART I).

(2) Checks the user-specified file to see whether or not it meets the specified input conditions (see PART *I*). If not, outputs to the user an appropriately-worded message, and then aborts.

(3) Pads the plaintext file to a multiple of the 32-bit block size to be used for *RSA* encryption, using one of the padding algorithms described in Fergusson and Schneier's *Practical Cryptography.*

(4) Queries the user for the file-identifier of a file holding one key of a matched *RSA* key pair.

Programming Project 1

PART *IVb*: *RSA* Encryption and Decryption: ENCRYPTION MODE (continued)

(5) Takes each block of the padded plaintext file, and encrypts it using the specified key. Do NOT use a built-in exponentiation function to accomplish the encryption. Instead, utilize the algorithm shown in class, successively squaring the 32-bit plaintext block and then immediately reducing the product by the modulus *n*. Once you have produced all of the squares up to the highest degree necessary, then multiply the individual values together one pair at a time and do a modulo *n* reduction each time.

(6) Deposit the ciphertext blocks in the output file. Note that the output file should be a binary or hex file, since there is no guarantee that the process of exponentiation will result in printable ASCII characters.

(7) NAMING CONVENTION FOR DATA OUTPUT FILE:

RSA-encrypted-ciphertext-*n*.bin

Programming Project 1

PART *IVb*: *RSA* Encryption and Decryption

Program Behavior: DECRYPTION MODE

(1) Queries the user for the two-decimal-digit Index Number (*n*) associated with the *RSA*-encrypted ciphertext file.

(2) Queries the user for the file-identifier of a file holding one key of a matched *RSA* key pair. This file must contain the companion key to the one that had been used for the encryption.

(3) Decrypts the file using the identical mathematical procedure that had been used to encrypt it, except for the difference in key.

(4) Removes the padding that had augmented the plaintext file immediately prior to encryption.

Programming Project 1

PART *IVb*: *RSA* Encryption and Decryption: DECRYPTION MODE (continued)

(5) Writes the decrypted and pad-free plaintext into an ASCII text file.

(6) NAMING CONVENTION FOR DATA OUTPUT FILE: RSA-decrypted-*n*.txt

FURTHER INSTRUCTION:

You must also provide a narrative file (either *.doc, .htm, or .pdf*) describing the details of your approach. Your narrative should give an overview of how you tackled the problem, as well as a table listing all defined constants, all named variables, and all functions and classes that you originated. If the name of any of these items is less than fully evocative of its purpose, then add a brief explanation.

File name: narrative-crypto-project-part-IV-yourJMUusername.*pdf*

END