Programming Assignment #1

CptS 425/580—Network Security
Assigned: 18 February 2009
Due: 4 March 2009, 11:59pm

Objective:

Implement the Caesar cipher and develop tools that may be used to attack this cipher using a *ciphertext only* attack.

Software Design & Implementation Tasks

1) Cipher Implementation – `caesarCipher` (30 pts)

Implement an extended version of the Caesar cipher that uses all of the characters from ' ' (Space, ASCII 32) to '~' (tilde, ASCII 126). Any character (printable or not) that falls outside of the ASCII range of 32-126 should be left unchanged during both encryption and decryption (this will ensure that printable files will remain printable). Key values range from 0 to 94.

Example with key=1

```
Plaintext:       Hello {~World~}!
Ciphertext:      Ifmmp!| Xpsme ~"
Decrypted text:  Hello {~World~}!
```

Minimum Implementation Requirements: Command-line arguments shall be used to indicate an input file (-i *input_file*), an output file (-o *output_file*), and an integer key value (-k *key_value*). An optional parameter (-d), when present, indicates that the input file is a ciphertext file that is to be decrypted. Input files of up to 100 Kb in size shall be accommodated.

2) Character Frequency Generator – `charFreqGen` (15 pts)

Design and implement a program that will take as input a text file and output a list of character frequency pairs, given the text within the file. The frequencies generated shall be case insenstive. This program may ignore all characters outside of the ASCII 32-126 range.

Minimum Implementation Requirements: Command-line arguments shall be used to indicate the file (-i *input_file*) from which the character frequencies are to be extracted as well as the output file (-o *output_file*) to which these frequencies are to be stored. The top character frequencies shall be displayed to the user in a two-column format (in sorted order, highest first). By default, five character-frequency pairs shall be displayed, however this value may be overridden with a command-line parameter (-t *top_count*). Input file sizes of up to 100 Kb shall be accommodated.

3) Ciphertext Attack Tool – `caesarAttack` (20 pts)

Design and implement a program that uses character frequency statistics generated by your `charFreqGen` program to compute the correlation value, ø(i), for all possible key values (see Bishop's text, page 101). Try successive keys, ranked according to their ø(i) values, until the ciphertext is successfully decrypted. The program may prompt the user to determine if a file was successfully decrypted.

Minimum Implementation Requirements: Command-line arguments shall be used to indicate an input ciphertext file (-c *ciphertext_file*) and an input character frequency file (-f *frequency_file*). The top 5 ø(i) values shall be displayed to the screen (in sorted order, highest first) and all ø(i) shall be written to a file (-p *phi_file*) in a two-column format. The successfully decrypted text shall also be written to a file (-o *output_file*).

Written Deliverable (35 pts)

1) Cyptographic Checksum of your Source Code (5 pts)

Compute an md5 or SHA1 checksum of a tar file (or similar archive) of your source code and executable image. Include the name of this file, it's size, and it's checksum in your write-up.

2) Abbreviated Software Design Document (10 pts)

Briefly describe the data structures and algorithms used to implement the three required tools. (1-2 page limit)

3) Software Test Document (10 pts)

Describe your software test plan and methodology that you used to verify that you implemented the three required software tools correctly. List any know deficiencies. (1-2 page limit)

4) Lessons Learned / Project Evaluation (10 pts)

Discuss the lessons learned from this project. Are there improvements that should be made in a future release? During testing, did your real-world input data files generate similar character frequencies? Where you able to always predict the correct key on your first try? Why/Why not. Other possible questions to address: Was this assignment as easy as you anticipated? Did implementing this cipher inspire you to become a cryptanalyst? Etc.

5) Time Logs (Optional, no points)

List the time spent during software design, implementation, testing. Also list the time spent writing up your project.

Automated Software Deliverables & Testing (0 pts)

1) Source code tarball (0 pts, 3 penalty pts)

Email Dr. McKinnon and the TA a gzip'd tar file containing the source code for the three programs specified within this assignment. The email message shall conform to these requirements:

subject line:	`CptS425: Program#1 from` *`LAST_NAME`*
tarball file name:	`cpts425p1`*`LAST_NAME`*`.tar.gz`

2) Makefile / Building your programs (0 pts, 3 penalty pts)

Executing the command: `tar -xzf cpts425p1`*`LAST_NAME`*`.tar.gz` shall extract your source code and a Makefile in to the current working directory. Executing `make` shall build/compile the three programs specified within this assignment. If your programs do not need to be compiled (e.g., you are using an interpreted scripting language), then your Makefile's 'all:' target may be empty.

3) Automated test script (0 pts, 5 penalty pts)

An automated test script will be used to grade your three programs. An example test script and input files will be provided at http://www.tricity.wsu.edu/~mckinnon/cpts425/prog1/. Programs that do not compile and run using the example test script on an EECS machine will incur a penalty.

Additional Details

- You may implement this assignment in the programming language of your choice.
- One week before the assignment is due, an example set of test files will be provided. You will not receive full credit if your tools cannot correctly process these files. These files may not be sufficient to fully test your tools.
- Program grading will occur on departmental machines (Sloan 353 for Pullman students, West 151 for Tri-Cities students) unless prior approval is granted by the instructor.