Project 1: Breaking Zendian Ciphers

Introduction

The Zendians send encrypted messages to their spies and operatives by reading the letters of the messages over a radio. We are able to intercept the radio messages and, fortunately, the Zendians are poor cryptographers, so we are able to break the messages. However, they send so many encrypted messages that we need to be able to quickly decipher a message and determine whether it is interesting. In this project, you will be working with a sample of ten encrypted Zendian messages and a list of five *cribs*, words that are known to occur at the start of interesting messages. For each message, you will compute all possible decipherments; if you find one that starts with a crib, print the deciphered message to the console. If no decipherment of a particular message starts with a crib, then the message is not interesting.

The cipher used by the Zendians is called a *Caesar Cipher*. Many of you are already familiar with the Caesar cipher: it is one of the oldest and simplest encipherment algorithms. It is named after Julius Caesar and does in fact date to the Roman Empire.

The process for enciphering with the Caesar cipher is:

1. Write down the message to be enciphered. To keep things simple, we will only use uppercase letters and spaces for our messages, with no punctuation or lowercase letters. Spaces will be treated as word delimiters and will not be enciphered. The letters of a message are naturally identified with the integers 0 - 25.

2. Choose a *key* from the letters A - Z. The key is also identified with an integer in the range 0 - 25.

3. For each letter in the message, add the key modulo 26 to produce the cipher character.

Example: Encipher the message "ATTACK AT DAWN" with key "R".

Message	A	T	T	A	C	K		A	T		D	A	W	N
Message as Integers	0	19	19	0	2	10		0	19		3	0	22	13
Key	R	R	R	R	R	R		R	R		R	R	R	R
Key as Integer	17	17	17	17	17	17		17	17		17	17	17	17
Message + Key mod 26	17	10	10	17	19	1		17	10		20	17	13	4
Cipher	R	K	K	R	T	B		R	K		U	R	N	E

Exercise: How do you decrypt a Caesar-enciphered message? The following cipher message was Caesar-enciphered using key "D". Decipher it.

```
PHHWPHRQWKHEHQFKLQWKHVTXDUH
```

If you completed the exercise, you have seen that it is easy to decipher a message if you know the key. What if you do *not* know the key? Since there are only 26 possible keys, it is easy to try them all; for each putative decryption, you can check whether the message begins with a crib, and if it does, you can assume that the key is correct.

The Wikipedia article provides additional background and examples.

Assignment

Your assignment is to write a program that can decipher Caesar-enciphered messages with each of the 26 possible keys, and for each decipherment determine whether it begins with one of several cribs. If a decipherment is found to start with one of the cribs, then print the value of the key and the deciphered message; if not of the decipherments begin with a crib, indicate that the the message is not "interesting." To test your program, you are provided with 10 cipher messages and five cribs.

To begin, copy the the project header file to your working directory:

```
linux2[12]% cp /afs/umbc.edu/users/c/m/cmarron/pub/cmsc202/proj1.h .
linux2[13]% ls
proj1.h
```

Note: Do not omit the '.' at the end of the cp command.

Read the comments in the the interface file `proj1.h` carefully. It is essential that your implementation function as described in the comments and in accordance with any additional requirements described in this document.

The enciphered messages and cribs are defined in two arrays, `cipher` and `cribs`, in `proj1.h`. The arrays are two-dimensional arrays of characters, but can be thought of as one-dimensional arrays of C-strings. Remember that a C-string is an array of characters, so individual characters can be accessed using subscript notation.

Your program will be tested using a suite of tests written in the Google Test framework. A subset of the tests will be made available to you prior to submitting your project, but you are responsible for fully testing your program. In particular, it must work correctly with data other than that provided in `cipher` and `cribs`.

In addition to passing functional tests, your project submission must satisfy the following general requirements:

1. Your submission must include the following files: a main program (`proj1.cpp`), the header file (`proj1.h`), and a functioning `Makefile`.

2. The main program must contain `main()` and must implement the functions `Decipher()` and `SolveCipher()` following their prototypes and descriptions in the header file. It is not necessary to use additional helper functions with `main()`, but if additional functions are used, their prototypes must be added at the top of `proj1.cpp` (after all `#include`s but before `main()`) and their definitions must follow `main()`.

3. You may not change the header file `proj1.h`,

4. Running `make` with your `Makefile` must produce the executable file `proj1` (*not* `Proj1` or `proj1.out`, etc.).

Helpful Tips

There are a number of string manipulation functions defined in the `cstring` library that you may find useful:

- `strncpy(destination, source, max)` — copies the `source` string to the `destination` string, but will copy at most `max` characters. `max` would typically be set to the length of the `destination` array.

- `strncmp(string1, string2, max)` — compares the first `max` bytes of `string1` and `string2`; returns 0 if they are equal, -1 if `string1` < `string2`, and +1 if `string1` > `string2`.

- `strnlen(string, max)` — returns the length of `string`, up to `max` characters.

Recall that '`%`' is the C/C++ modulo operator. For example, to compute the sum of *x* and *y* modulo 15, you would use the expression `(x + y) % 15`.

Sample Program Build and Execution

```
linux3[10]% make
g++ -c proj1.cpp
g++ proj1.o -o proj1
linux3[11]% ./proj1
Message #1 was not interesting.
Message #2, Key H
   DROP THE PACKAGE BEHIND THE DUMPSTER
Message #3 was not interesting.
Message #4 was not interesting.
Message #5, Key N
   AGENT BRAVO WILL GATHER THE PACKAGE
Message #6 was not interesting.
Message #7, Key D
   ATTACK TO BEGIN AT NOON
Message #8, Key L
Message #9, Key V
   WEATHER CLOUDY MODERATE WINDS
Message #10 was not interesting.
```

Project Submission

To submit your project, you need to copy the required files to your submit directory on GL. If you followed the instructions in Lab 1, you will have a directory called `cs202proj` in your home directory, in which case you can copy the files from your working directory with the command

```
linux2[14]% cp proj1.cpp proj1.h Makefile ~/cs202proj/proj1
```

Once the files have been copied, you may test your submitted files in the submit directory:

```
linux2[15]% cd ~/cs202proj/proj1
```

then build and execute the program as shown above.