STEGEDIT – Specification

Preface

stegedit is an editor for steganographic text, and is intended to help and assist in writing texts with a given number of words per sentence. While this task could be accomplished in any editor by manually counting and replacing words and phrases, stegedit has some special features to specifically be useful in those cases, while it is not a general-purpose editor like gedit.

There are basically two types of editing a sentence. The first one is by declaring certain words as ommitable using a special character or by offering alternatives in a given situation, with the alternatives having different numbers of words (e.g. “my sister” or “Veronique”). The second one is using special commands in front of the input (like `-ddtherefore` to delete every occurrence of the word therefore) or using the command alone (as in `-ix` to output the sentence with index numbering on every third word).

Call and usage

The editor should be a command-line program which does not use any kind of TUI (like ncurses, as in GNU nano), but plain I/O to the terminal. The first argument to be expected is the file to be steganographically disguised in the text file created with the editor. Second argument is the text file where the created text should be outputted.

The program should rearrange the bytes of the file to disguise so that there are two integers containing the high and low nibble (as if the byte was represented as two hex digits and those digits are treaded as individual integers). To every integer, a constant (say, 5; must be easily editable in source code) shall be added.

Now, the user should get the following prompt (in this example, with requiring 11 words):

```
(..)  11:
```

The 18 indicates that a sentence of 18 words is required, and the dots in brackets is a placeholder for the number of entered words which will be displayed as soon as a sentence has been entered. Next to the colon, a space should be placed. After this, the user should be prompted to enter her sentence. Say, the user inputs the following:

```
(..)  11: This is a sample sentence to demonstrate the
capabilities of the steganographic text editor.

(14)  11:
```

What now happened is the editor reporting to the user that despite 11 words had been requested, the sentence contains 14 words. Now the user has to do something in order to get a sentence with 11 words with a reasonable similar meaning. One option would be removing the phrase `the capabilities of`, which could be done using the -d-operator:

```
(14)  11: -d[dem],3

(..)  15:
```

Now, the editor has removed three words beginning at the first occurrence of “dem”, which happened to be “demonstrate”, and therefore removed the words *demonstrate*, *the*, and *capabilities*. This made a sentence consisting of 11 words and therefore, the next sentence (containing 15 words) was requested.

Several commands are intended to exist to make editing flexible and easy. Here is a list of all:

($ is placeholder for a string, % for an integer and & for an index, the n-th word of the sentence)

```
-a$   Append the following text to the current sentence

-d$   Deletes the given word at first occurrence

-d&$  Deletes the given word at its n-th occurrence

-dd$  Deletes every occurrence of the word

-d&,% Deletes % words starting at the index &

-i&$  Inserts the string $ after the &-th word

-ix   Print sentence with index numbering in brackets after every
      third word.

-ixx  As in -ix with indexing for every word

-i%   As in -ix with every %-th word

-l$   Puts the string $ to the left of the current sentence (to
      the beginning)

-r$   As in -l$, but to the right (the end of the sentence)

-o$   Declares the string $ as ommitable

-o&   Declares the &-th word as ommitable

-o!   Declares all suggested ommitable words as actually ommitable

-p&$  Replaces the &-th word with the string $
```

It shall always be made sure that the punctuation mark at the sentence’ end (test with `isgraph(c)`) is moved to the actual end, if, for instance, the command `-r$` is used to append text. Also, using the bracket operator, an index should be replaceable with a text and the index will be the occurrence of the index. This is demonstrated in the example with the three removed words.

Another feature is the usage of ommitable words. If, for example, a sentence like this:

```
(..) 13: In this ~short example, I want to demonstrate ~all the
capabilities of this ~steganographic editor.
```

Notice that all the words starting with a tilde are ommitable: If they were removed, the overall meaning of the sentence would not change fundamentally. Now, the editor, which requested 13 words, got 15, but since some words are ommitable, it can decide to randomly remove *short* and *all*, for example, to get the desired 13 words.

Also, there should be a text file read at the beginning and every occurrence of a word from the text file within the entered sentence should be marked with a degree-sign `°` just like ommitable words are marked with the tilde `~`. These are words that are most likely ommitable and using the command `-o!` the user can confirm that the words are, indeed, ommitable. The other o-commands can also be used to declare single words as ommitable.

At last, there should be the possibility to enter options if a given phrase can be expressed with different numbers of words, so the editor can choose the steganographically appropriate one. Let's look at an example:

```
(..) 8: *"I've","I have" decided to ask *"my sister","Veronique"
that later.
```

Here, the editor can choose several options, the correct one would be to use either "I have" and "Veronique" or "I've" and "my sister" to get eight as the total number of words. There should be at least eight possible phrases, all separated by comma. The asterisk indicates the possibility for the editor to choose different phrases.

Library and licensing requirements

The program must be licensed according to the GNU GPLv3 free software license with the option to choose a later one (that has yet to be released). It is allowed to use any function of the C standard library, especially all the string functions, type checks etc.

The usage of goto-statements is prohibited for the entire source-code. There shall be no usage of inline assembly, C++ dependence or usage of compiler-custom functions which are not part of C11. The program is meant to run under Debian 9, FreeBSD 11 and OpenBSD 6.3, but should be compatible with any standard-conforming environment.

It is also allowed to use the string_tools.c-library which I made myself for the ongoing SDTP project (which this editor is likewise part of) as well as any C11-compliant, portable library compatible with the GNU GPLv3 license. POSIX functions can also be used, but please ask me if you use unusual functions. Try to avoid it.

There should be reasonable commenting and naming of variables, functions etc.