Anagrams

An anagram is a word spelled by rearranging the letters of another word. The "other word" may or may not be a real word; if it is not, we say the letters have been *jumbled* or *scrambled*. Note that some "other words" yield more than one anagram. The letters TPSO, for example, can be rearranged to spell OPTS, POST, POTS, SPOT, STOP, and TOPS.

The obvious way to extract an anagram from a jumbled string of letters is to arrange the letters in all possible ways, checking each arrangement against a lexicon (ie, a list of valid words). This brute-force approach can quickly get out of hand, however. The number of ways that three unique letters can be arranged is only 3 times 2 times 1, or 6; but the number of ways that 9 unique letters can be arranged is 362,880. So we need a better way.

The not-so-obvious way to extract an anagram from a jumbled string is to first convert the lexicon into a *dictionary* where each entry in the dictionary is a sorted word, and the "definition" is the original, unscrambled word. These are the Plain English type definitions we need to describe that:

```
A dictionary is a thing with some entries.
An entry is a thing with a sorted string and a string.
```

Our Plain English development system includes a lexicon of the 64,000 most commonly used English words in the form of a text file, one word per line, so we can use that to make our dictionary. These are the Plain English routines that do that:

```
To create a dictionary:
Allocate memory for the dictionary.
Put "z:\g4g articles\anagrams\lexicon.txt" into a path.
Read the path into a buffer.
Slap a rider on the buffer.
Loop.
Get a string from the buffer given the rider.
If the string is blank, break.
Trim the string.
Sort the string into a sorted string.
Add an entry to the dictionary using the sorted string and the string.
Repeat.

To add an entry to a dictionary given a sorted string and a string:
Allocate memory for the entry.
Put the string into the entry's string.
Put the sorted string into the entry's sorted string.
Append the entry to the dictionary's entries.
```

A random sample of entries from our “anagram dictionary” looks like this:

```
aeprrs parser
aeprrss parsers
aeprrss sparser
aeprrssstu superstars
aeprrsstu superstar
aeprrssy sprayers
aeprrstu raptures
aeprrsy prayers
aeprrsy sprayer
aeprrtu rapture
```

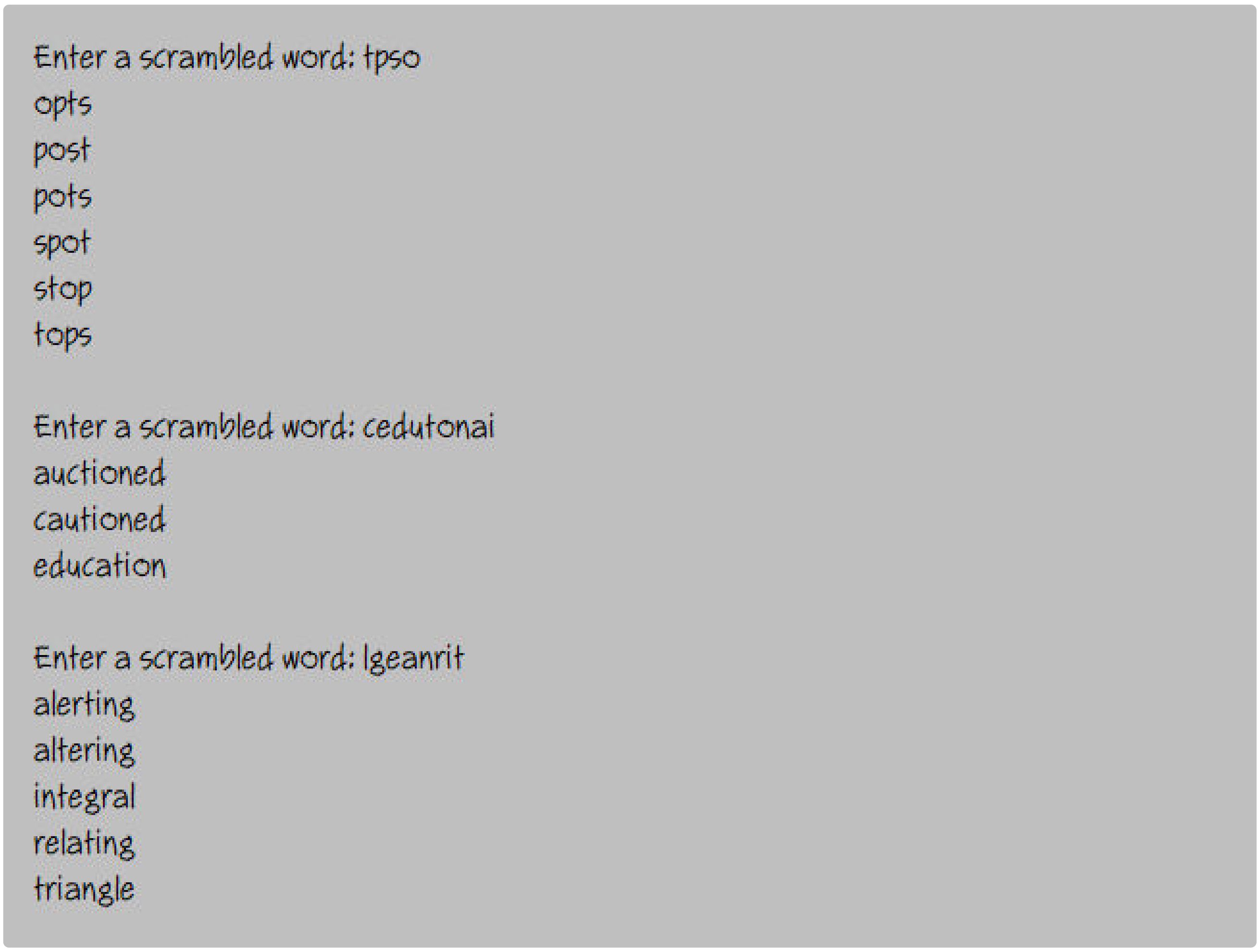
Note that some of the sorted words (like *aeprrss*) have more than one “definition” (like *parsers* and *sparser*).

Now we can find all the anagrams for a given string simply by sorting the input string and searching for matches in the dictionary. Here are the Plain English routines that do that:

```
To run:
Start up.
Create the dictionary.
Loop.
Write “Enter a scrambled word: ” on the console without advancing.
Read a string from the console. If the escape key is down, break.
Unscramble the string.
Repeat.
Write “Done...” on the console.
Destroy the dictionary.
Shut down.
```

```
To unscramble a string:
Trim the string.
Lowercase the string.
Sort the string into a sorted string.
Loop.
Get an entry from the dictionary’s entries.
If the entry is nil, break.
If the entry’s sorted string is not the sorted string, repeat.
Write the entry’s string on the console.
Repeat.
```

A typical run looks like this:



```
Enter a scrambled word: tpso
opts
post
pots
spot
stop
tops

Enter a scrambled word: cedutonai
auctioned
cautioned
education

Enter a scrambled word: lgeanrit
alerting
altering
integral
relating
triangle
```

It takes a second or two to create the dictionary, but that only has to be done once, and it can be saved for later use. The unscrambling part is aaeinnnossttu. Whoops! Sorry. I meant to say, “instantaneous.”