ERRATA: "Printing Common Words", *Communications of the ACM 30*, 7 (July 1987), 594–599. (Also appears as *Printing Common Words*, Tech. Report 86-18, Dept. of Computer Science, The Univ. of Arizona, Tucson, May 1986.)

Several readers found an error in the common words program presented in the "Literate Programming" column and others have suggested improvements.

The error, pointed out by Michael Shook and others, is in allocating `list` in `printwords`: it's potentially too small. The original intention was to allocate N entries in `list`, where N is the number of words in the input. However, only `total` entries are allocated, and `total` is the number of *unique* words in the input. If `total` is less than the maximum word frequency, `list` is indexed erroneously. This situation probably occurs infrequently since `total` is usually much larger than the maximum frequency for most "normal" inputs. The input `hello hello` demonstrates the problem and causes `common` to fail.

The error can be fixed by making `total` count the number of words in the input, which can be done by moving the statement `total++` from `addword` into the loop that calls `addword` in `main`. A better solution, however, is to eliminate `total` and make `list` just large enough to accommodate the largest frequency of occurrence, which can be done in `printwords` by making a pass over the hash table to compute the largest frequency. This version of `printwords` is

```
printwords(k)
int k;
{
  int i, max;
  struct word *wp, **list, *q;

  max = 0;
  for (i = 0; i <= HASHSIZE; i++)
    for (wp = hashtable[i]; wp; wp = wp->next)
      if (wp->count > max)
        max = wp->count;
  list = (struct word **) alloc(max + 1, sizeof wp);
  for (i = 0; i <= HASHSIZE; i++)
    for (wp = hashtable[i]; wp; wp = q) {
      q = wp->next;
      wp->next = list[wp->count];
      list[wp->count] = wp;
      }
  for (i = max; i >= 0 && k > 0; i--)
    if ((wp = list[i]) && k-- > 0)
      for ( ; wp; wp = wp->next)
        printf("%d %s\n", wp->count, wp->word);
}
```

Hans Boehm of Rice University noted that using the sum of the character codes as a hash function is a poor choice. By the definition of "word" given in the

program, there are only 52 distinct character codes. So, for example, all words of length five get hashed into a range of only $5 * 52 = 260$ hash codes and words of length ten get hashed into a range of $10 * 52 = 520$. Thus, most of the hash table is empty and collisions are likely, which explains in part the large number of calls to `strcmp`. While I knew about the potentially poor performance of the hash function, I didn't change it because `common` seemed to perform adequately.

I measured the lengths of the hash chains using the test file described in the paper as input. In the following table, the right-hand column is the chain length and the left column is the number of chains of that length.

1	16
3	15
5	14
12	13
13	12
21	11
22	10
37	9
33	8
58	7
55	6
83	5
94	4
148	3
176	2
335	1
3000	0

The 3000 empty slots and long chains confirm Boehm's predictions.

Boehm suggested shifting the sum left one bit after each addition, e.g.,

```
h = 0;
s = buf;
for (len = 0; *s; len++)
    h = (h<<1) + *s++;
```

Using this hash function gives a better distribution for the test input, but there are still many empty slots:

2	10
5	9
8	8
18	7
45	6
98	5
189	4
213	3
455	2
535	1
2432	0

Finally, Joe Warren of Rice suggested mapping the character codes into random numbers and summing the random numbers. The hash function is

```
h = 0;
s = buf;
for (len = 0; *s; len++)
    h += scatter[*s++];
```

where `scatter` is initialized with the first 128 values returned by the C library function `random`. Boehm tested this function with a 4K table on a dictionary and found only 13 empty slots. This is very close to the expected value, which Boehm computed as 10.4 for the given dictionary, a 4K hash table, and assuming a uniform distribution of hash values. Using this function on the test input for `common` gave the following distribution.

1	7
3	6
18	5
62	4
242	3
769	2
1522	1
1479	0

This version of `common` (including `printwords` above) runs 8 to 9 percent faster than the published version.