

"shortint," and 32767 replaced by 127, aborts at the statement  $i := i + 1$ , with an "out of range" run-time error. If the language modeled in the program verifier does not know all that, and cannot take account of it, that verifier will be relatively useless because many programming errors result from such language features.

Howard E. Tompkins  
19 Congress Terrace  
Milford, MA 01757

### AN ERROR IN ERROR DETECTION?

I am writing to comment about a very common, but unfortunately widely repeated, error in the basic conception of coding theory, as recently found in Neal R. Wagner and Paul S. Putter's article, "Error Detecting Decimal Digits," (*Communications*, Jan. 1989, pp. 106-110). The erroneous statement (near the beginning of the article) is: "... the theory only works over a field."

In fact, error correcting schemes work for *any* integer greater than 2 by decomposing that integer into a product of prime products for different primes and then working separately over each prime. In the case treated by Wagner and Putter, we have the integer 10, which must be treated as 5 times 2. The coding schemes are to be worked out separately for 5 and for 2, and the resulting codes combined.

An example should suffice. We have the decimal number 5632078149 (say), to which a check digit is to be appended. The first step is to take each digit in the above decimal number and divide it by 5, collecting the quotients in one place and the remainders in another. In this case we obtain the base-2 number 1100011001 and the base-5 number 0132023144. (Thus, for example, 5, the *first* digit in 5632078149, divided by 5, gives quotient 1, the *first* digit in 1100011001, and remainder 0, the *first* digit in 0132023144. In general, the *K*th digit of the original number, divided by 5, gives the *K*th digit of each of the new numbers from the calculated quotient and remainder.)

Now apply whatever theory exists over a field of order 2 to obtain check digits for 1100011001. Suppose these check digits are 101. Now apply theory over a field of order 5, to obtain check digits for 0132023144. Suppose these check digits are 324. Now combine the two check digit sequences, 101 and 324, by the reverse of the process outlined above; that is, in this case,

$$101 \times 5 + 324 = 829$$

is the resulting check digit sequence, and, in the general case, the check digit sequence is  $5b + q$  where  $b$  is the computed binary check digit sequence and  $q$  is the computed base-5 check digit sequence.

It should be obvious that if there is a transposition of two unequal digits, there will be a corresponding transposition in at least one of the two numbers obtained in this way. Thus, if the operator transposes the 5 and the 6, obtaining 6532078149, the resulting base-5 number will be different (1032023144), and specifically different

by a transposition. The resulting check digit sequence  $q'$  will be different from  $q$ , and therefore  $5b + q'$  will be different from  $5b + q$ .

On the other hand, if the operator transposes the 4 and the 9, obtaining 5632078194, then the resulting base-5 number will be the same, but the base-2 number will be different (1100011010), and again, different because of a transposition. If the resulting check bit sequence  $b'$  is different from  $b$ , then  $5b' + q$  will be different from  $5b + q$ . This theory is immediately extensible to the case which we need  $x$  check bits and  $y$  check digits modulo 5, where  $x \neq y$ ; these may be combined into a single decimal check integer from 0 through  $2^x 5^y - 1$ .

W.D. Maurer  
The George Washington University  
School of Engineering and Applied Science  
Washington, D.C. 20052

The article, "Error Detecting Decimal Digits," by N.R. Wagner and P.S. Putter (*Communications*, Jan. 1989, pp. 106-110) brought to mind the following scheme for generating a series of self-checking decimal numbers. We have used this method at Rhode Island Hospital for many years; I have not seen it described in the literature. The algorithm is ...

### Use only those numbers which are multiples of thirteen.

So, the series would begin with 0, 13, 26, 39, ..., and include such sequences as 2405, 2418, 2431, 2444, ..., 239343, 239356, 239369, etc.

As simple as this algorithm is, it can detect all single digit substitution errors, all adjacent and jump transpositions, and most (greater than 90 percent) of the other common kinds of errors: twins, jump twins, and dropped or added digits. Its ability to detect single-digit substitution errors follows from the fact that any such error alters the value of the number by an integer multiple of an integer power of ten, the prime factors of which cannot include 13. Similar considerations can validate its other error-detecting properties.

In addition to being trivially easy to generate and to check in any programming language or environment, a consecutive sequence of these self-checking numbers has *uniformly distributed terminal digits*, an important consideration when the associated records are physically arranged according to terminal digit(s).

A disadvantage of the scheme is that it cannot be used to append numeric check digits onto an existing set of numbers.

One useful generalization should be pointed out: the remainder mod 13 did not necessarily have to be defined as zero; *any* set of numbers having *identical* remainders mod 13 will have error-detecting properties. So there are actually *thirteen different interleaved sets* of self-checking numbers. An organization could use the 0-remainder set for employee numbers, the 1-

