the example we have given, is FS a misprint for F or FS or F4? Or something entirely different? If you were to show the complete listing to another competent programmer, he should be able to identify the faults and make the corrections. He would use two criteria in making his decisions: the context in which the program line appeared, and his own experience.

Strangely enough, this technique has been more widely applied to correcting English text than to checking program code. A spelling checker package, for example, will work through a text and highlight any words that don't match the entries in its dictionary of perhaps 50,000 words, held on disk. Most of these packages have the facility to learn new words (such as the spelling of company or proper names) and add these to their dictionaries. The more sophisticated will even suggest the correct spelling if a close match is detected. Experimental word processors have also been developed that can apply the same processes to grammar and writing style — pointing out such things as incorrect punctuation, repetition of words within a paragraph, mixed metaphors, and inapplicable adjectives. Again, these work by examining the context of any phrase, and by reference to a library of previously used examples.

More effort, however, has been put into the development of systems that will *create* programs, rather than *correct* existing ones. In 1981, a software product was announced that set off one of the fiercest battles ever waged within the microcomputer industry. Cleverly named The Last One, it purported to be a program that could write any other program you might want, and hence was the last program you would ever need to buy. This proved to be an unjustified claim, but The Last One was a very useful aid in the development of certain types of program — mainly business applications. There are now several such products on the market for business microcomputers, and a few for home computers, and these are collectively called 'program generators'.

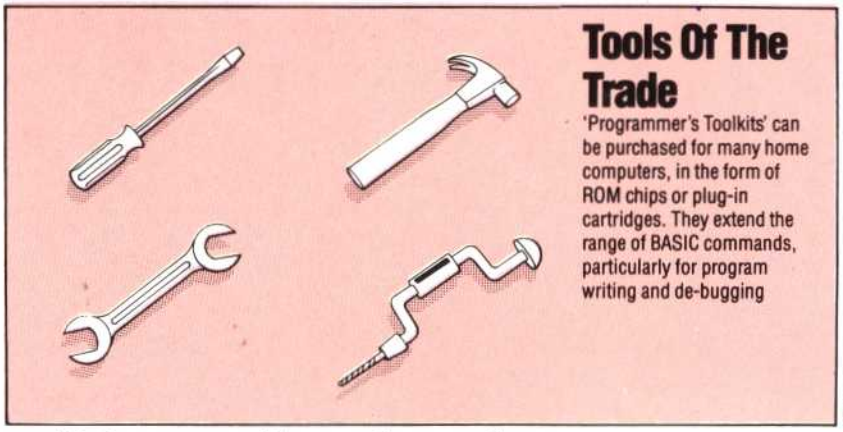
Let's now look at the basic concept behind a program that can write programs. Consider this trivial example:

```
10 PRINT "WHAT DO YOU WANT THE PROGRAM TO
   DISPLAY ON THE SCREEN?"
20 INPUT A$
30 PRINT "THE PROGRAM IS:"
40 PRINT "10 PRINT ";CHR$(34);A$;CHR$(34)
```

If you answer HELLO to the question, the program (which should run on most home computers) should print out the line:

```
THE PROGRAM IS
10 PRINT "HELLO"
```

If you apply the same technique to the input, calculation and output phases of the application you have in mind, then you could write yourself a very simple program generator. If all the questions that it asks are plainly worded, it should be possible for someone with no previous experience to develop a simple program using your generator.



Tools Of The Trade

'Programmer's Toolkits' can be purchased for many home computers, in the form of ROM chips or plug-in cartridges. They extend the range of BASIC commands, particularly for program writing and de-bugging

KEVIN JONES

Commercially produced program generators use the same techniques. Most business applications consist of a combination of five distinct processes: input of data, output to screen or printer, storage in a data file, retrieval, and calculation. The generator will have standard and very flexible subroutines for each of these. By asking you to specify the exact structure of the data you will be using, the calculations that go with that data, and the layouts you require on the screen and printer, the generator will change the values of certain variables in the subroutines, and string them together to create the program.

Although program generators are becoming more sophisticated, they are unlikely to replace human programmers in the immediate future because they suffer from the following limitations. First, the technique described is all very well for transaction-based business applications such as accounting or stock control, but generally these program generators can't be applied to writing word processor or games programs. Secondly, because the program generator has to make use of these standard flexible subroutines, the resulting listing won't be nearly as efficient (either in terms of speed or memory used) as it would have been if it had been purpose-written by a programmer. Thirdly, programs produced by generators generally aren't as user-friendly as the systems currently being produced by human programmers. For example, they seldom make good use of the graphics facilities offered by the latest machines.

Finally, the program generators that are now available can only really replace the final stage in programming — the writing of the code. The user still has to put the work into thinking out the exact form of the data, input and output that is needed. Generally, the earlier stages of programming are the most difficult, and require specific skills distinct from those of programming. Most large companies employ specialists called 'systems analysts' to specify the programs they need, and these specifications are then turned into code by programmers. Program generators have yet to acquire all the skills required to create a computer program.