Subversive Elements

With careful planning and a step-by-step approach, the time taken to de-bug a program can be dramatically reduced

As you become more skilled at writing programs, you will also tend to become more accomplished at 'de-bugging' them. The syntactical mistakes and errors in logic, which even the most experienced computer programmers can make, become less frequent and less problematic as your experience increases. Here are some hints to help you avoid programming errors and become more efficient at de-bugging your code.

The first place to begin is at the precise point where a program begins — in your head! If the concept of a program is badly thought out at the beginning, then it is sure to be infested with bugs when it is written.

It is a far better idea to begin writing a program by first stating the problem as clearly as possible to yourself or someone else. Then divide the problem into logically complete parts — Input, Output, Algorithms, Data Structures, Processes, etc. — and consider each of those parts as a separate problem. If necessary, break down each of these problems into its component problems, and so on, until the original problem is a structured collection of sub-problems, each of which is easy for you to program. A formal approach, such as using a pseudo-language or a flowchart, is essential in the design stage as a way of keeping track of, and preserving, the program structure as a whole. You must try to stay away from the keyboard until you can honestly say that you know how to program every part of the problem This is called the top-down approach to programming, and the method can dramatically cut your de-bugging time.

Splitting problems into solvable tasks will lead you to write programs that are really collections of subroutines or procedures linked by a skeleton main program. This makes finding bugs easier, and it enables you to build a library of bug-free subroutines for use in later programs. The alternative is called 're-inventing the wheel': every time you write a program that sorts data, for example, you re-solve the problem of how to write a sort routine, and probably rewrite the same old bugs, as well! It is much easier to write and debug it once, save it, and recall it whenever you need it thereafter.

As far as BASIC allows, always try to use appropriate variable names, even if they have to be abbreviated. NET=GROSS−TAX, for example, explains itself; and NT=GR−TX isn't a bad substitute; but N=G−T is extremely ambiguous, and gives no clue as to what variables are involved. It's good practice to keep a variable table, which shows you all the variables used in the program and what they're for. This can lead you to standardise your use of variables (such as, always using certain single letter variables as loop counters), and stops you using the same variable for different purposes. Similarly, it's good practice to store constant values in

Pest Control

```
100 GOTO 200:X$="THAT'S ALL FOLKS"
120 I=12:K=1984
140 FOR K=1 TO LT
160 PRINT"WHO NEEDS STRUCTURE ?:N$:NEXT
180 RESTORE
190 FOR L=1 TO I
200 INPUT'ENTER YOUR NAME":N$
220 INPUT'ENTER YOUR AGE";LT
240 GOSUB 100
260 PRINT IF YOU'RE";LT;"NOW"
280 PRINT'YOU WERE BORN IN";K-LT
300 YR$=K-LT
320 LY=INT(YR)/4*4
340 IF LY=YR THEN IF INT(LY/100)*100=LY THEN GOTO 370
380 PRINT YR'WAS A LEAP YEAR":GOTO 420
390 PRINT "YR WAS NOT A LEAP YEAR"
400 NEXT
420 PRINT X$
440 STEP
```

This statement will never be executed, as the GOTO command skips over it

These two lines are in the wrong order. Line 100 should have: GOTO 190

K is supposed to contain a constant, but this statement will eliminate it

Because the quotes are missing from here, the NEXT will not be executed

This should read: RETURN

Syntax Error: the colon ':' should be a semi-colon ';'

This will cause big trouble. It should probably read: GOSUB 140

The quotation marks are missing

This will result in some meaningless number, because K has been changed in value since line 120

Syntax Error: this should be YR=K-LT

The close bracket is misplaced, causing the calculation to fail. This should read: INT (YR/4)*4

There is no line 370!

This should be: " GOTO 420 means jumping out of the FOR...NEXT loop

This may need the name of the loop variable: i.e. NEXT L

X$ has not been initialised, so this statement will do nothing

Syntax Error: this should be a STOP