The OTHERS
'DATA' statements in G-Pascal

by David Roth

One of the major drawbacks with G-Pascal is the awkwardness of setting up tables of numbers or characters. It can be done by tediously assigning data to arrays, e.g.

```
a[1] := 40 ;
a[2] := 20 ;
..... and so on
```

or

```
alpha[1] := "abc" ;
alpha[2] := "def" ;
```

But there is a better way. The following technique provides a useful way of setting up BASIC-like DATA statements in a G-Pascal program. It allows tables of strings or numbers to be more easily set up.

1. Numbers

The following program sets up the 'DATA' at the start of the program as 'comments'. Each data element must be 3 digits, i.e. 1 = 001.

```
(*
001002003004
095160160186186186186186186186
  186186186160105££££*)

varj : char;
z : array [80] of char;
(* read the data lines *)
procedure read data;
const start source = $4000;
var i, ptr : integer;
begin
for i := 0 to 79 do
   z[i] := 0 ; (* clr array *)
i := 0;
ptr := start source - 1;
repeat
ptr := ptr + 1
until memc[ptr] = "(";
ptr :=ptr + 3;
repeat
z[i] := (memc[ptr] - $30) * 100;
z[i] := z[i] + ((memc[ptr + 1] - $30) * 10);
z[i] := z[i] + memc[ptr + 2] - $30;
i := i+1; ptr := ptr + 3;
until (memc[ptr] = "     ; (* delimiter ? *)
end;
(* mainline - display the data *)
begin
read data;
(* display the data *)
forj := 0 to 74 do
writeln(z[j]);
end.
```

This technique takes advantage of the G-Pascal standard that source code starts at $4000. Therefore, if the program starts with a 'comment area', that area can be used to store data. The first 'data' line must start with the "(*". The first data element can from one to any numbers of blanks after the "(*", since the editor tokenises the blanks so that two blanks take up as much space as four blanks. The end of the data is marked by a row of pound signs (or whatever delimiter you prefer). Extra protection could be added to check that the index for the receiving array does not exceed the array bounds. No check is made that each element does not exceed 255. If numbers larger than 255 are required, then the receiving array must be defined as INTEGER.

2. Strings

The following program handles string data.

```
(*
Mary had a little lamb, its fleece
was white as snow.£££££)

var j, k : char;
z : array [80] of char;
(* read the data lines *)
procedure read data;
const start source = $4000;
var i, ptr : integer;
begin
for i := 0 to 79 do
   z[i] := " " ; (* clr array *)
i := 0;
ptr := start source - 1;
repeat
ptr := ptr + 1
until memc[ptr] = "(";
ptr :=ptr + 3;
repeat
z[i] := memc[ptr] ;
i := i+1; ptr := ptr + 1;
until (memc[ptr] = "     ;
end;
(* mainline - display the data *)
begin
read data;
(* display the data *)
forj := 0 to 79 do
write(z[j]);
writeln
end.
```

Notice that the use of "*)" as a delimiter for the 'DATA' rather than looking for the ending "*)" for the comment allows "*"s to be included in the 'DATA'.

© David Roth 1985

REVIEW

Commodore 64 Graphics with COMAL

Author: Len Lindsay
Publisher: Prentice Hall (Aust)
Price: R.R.P.$33.95

This book follows the excellent example of Len Lindsay's "COMAL Handbook". Each of the graphics control commands (including TURTLE graphics) built into COMAL is explained by a working example. The discussion of each command is an object lesson to the writers of computer manuals – a clear, low-jargon explanation, notes on its correct use and syntax, and a sample program. The sample programs are for the most part easy to follow and are good examples of sound structured programming style. They can also be readily incorporated as useful subroutines in your own COMAL programs. If you don't have the COMAL handbook, appendices are provided explaining COMAL structured programming, COMAL keywords and useful functions and procedures.

The book is, I think, pitched towards the 'practical' programmer or student, rather than the technical theorist. The book is basically a manual of graphics commands and does not attempt to give a full explanation of Commodore 64 graphics concepts. Each category of graphics command (TURTLE, GRAPHICS and SPRITES is introduced by a clear and simple explanation of the concepts used. If these explanations and the examples are followed through carefully, you will gain a good basic knowledge of graphics and a sound understanding of structured programming.

Some of the examples are a little complex – the sample program for SPRITECOLOR has too many ELIFs (ELSE IFs) when a CASE construct would have been simpler. But perhaps this is an implied invitation to the reader to improve the program as a learning exercise. But most of the sample programs are quite well thought out and present interesting ideas for the reader to build on. They could be improved if they were tied together by a common theme. If each sample program was a 'building block' in a bigger program then you would have a clearer idea of how the commands fit in together. For example, in a 'shoot 'em up' game, SPRITECOLLISION could be used to detect hits, PLOTTEXT to give the score, the TURTLE to draw a landscape, and so on. Having completed the examples in the book, you would then have a completed project to fiddle with, modify and learn from, rather than a disconnected set of examples, however excellent in themselves.

The book could also be improved by the inclusion of pictures or diagrams of the screen when the sample program is run. It is far easier to check a program from a picture than from a verbal description. And it seems strange that a book on graphics should have no pictures.

One drawback for users of the public domain COMAL version 0.14 is the limited memory available for user programs – 6 to 8K. I understand that the new COMAL version 2.0 for the 64 – now available from COMAL User Groups in the UK or USA – has over 100K available to the user, but it is expensive (over $100 for the cartridge). The graphics commands available do get around the memory limitation to some extent, since they condense a good deal of power into one simple statement (imagine the number of BASIC lines required to implement a TURTLE). If you are too lazy to type in the examples, you can send away for a companion disk. I don't recommend this, since you can learn far more by typing in and debugging the programs (and hopefully modifying them to try out your own ideas). The disk is also rather pricy, at $20 (US).

On the subject of price, I am very critical of Australian publishers' pricing policy for

Continued overleaf