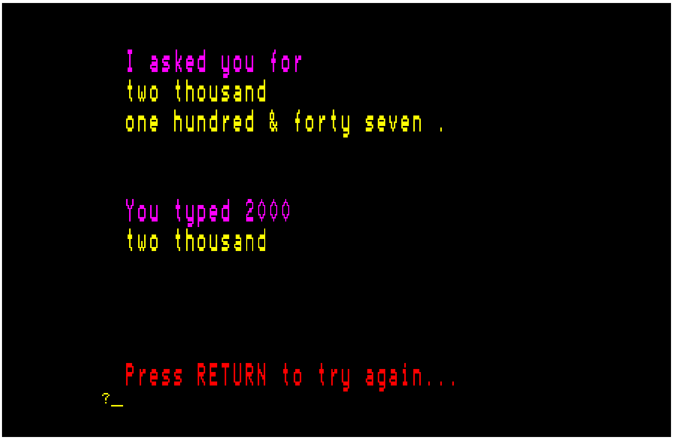
21

Readnum

Here is a program that will give your kids practice in reading and understanding numbers, for it asks in English for a digital equivalent. For example, 'Type seven thousand three hundred and fifty eight'. The user is then expected to type 7358. The program will handle any number of digits from one to nine, so that young children can start to handle problems like, 'Type three', while older children will have things like 'Type seventy seven million, three hundred and four thousand, nine hundred and twenty'.

If the input is incorrect, help is at hand, for the program will say, 'I asked you for eighty seven. You typed 800 - eight hundred'. In this way, the number values are constantly reinforced and results are guaranteed.



```
   I asked you for
   two thousand
   one hundred & forty seven .


   You typed 2000
   two thousand




   Press RETURN to try again...
 ?_
```

The program is written around the subroutine starting at line 380. It is recursive, which is to say that it constantly calls itself as a subroutine. This definition immediately brings to mind two questions: (a) how do you get out of it, and (b) what is the purpose?

The first is no great trouble. One simply counts the number of recursive calls and ensures that there are the same number of RETURN commands encountered. The purpose of a recursive routine is to save programming space when the same operations or sequence of operations are to be performed repeatedly on data.

When we look at a set of digits such as 12345 and translate it as 'twelvethousand, three hundred and forty five', we have scanned the set of digits with the eye, starting from the right, and broken it down into groups of three. Each group of three has a name - thousands, millions - and within the group the digits are again given names such as hundreds, tens and units. The highest pair of digits that can be expressed as a single word - twelve, fifteen, or twenty - is detected, and as a postscript we add the name of the group.

To do the job, a computer program must follow the same rules, and since the operations performed on each group are the same, the program becomes recursive.

'Readnum' was first developed when a parent expressed concern that her child could not read numbers. It was altered slightly for *26 Programs for Your Micro* (Newnes Technical Books, 1982), and is here altered again to take advantage of the colour and double-size print offered by the BBC computer. It will find a variety of uses in the educational field, but even if you never use the program, it is worth 'hand-running' simply in order to get a feel for what a recursive program is and how it operates.

The bulk of the program listing is either straightforward or, in the case of the procedures, described elsewhere in this book, so we can move straight to a consideration of the subroutine.

Let us take for our example the value given - 12345. (You can try other, larger, numbers for yourself.) On entry into the routine at line 380, N contains our value 12345. At line 400 we are routed to line 500, where S is given the value 12345 and N is reduced to 12. This is our new N and the subroutine calls itself again, so we start back at 380.

This time, we pass line 400 and at 410 are routed to line 620. The READ pointer is restored to the beginning of the data (at line 770) by the RESTORE statement of line 620, and then we are routed to line 690. Lines 640 and 650 cause 'twelve' to be written into A$, and then calls PROCS and PROCD cause the word 'twelve' to be printed on the screen. Going back to line 660, a RETURN statement is encountered. Unreeling itself, BASIC will find that its last GOSUB command was encountered at line 510, so command returns to the line following, at 520, which causes a printout of the word 'thousand'.

At line 540, N is given a new value by picking up the old value from the variable S and removing the thousands value. We are left with 345, and line 570 causes the process to repeat.

Again I recommend hand-running the program, using different numbers. The experience will teach you more about recursive subroutines than will a million words of text.

Variables

D%	Digits in the problem
PR%	Number of current problem
N%	Value chosen for current problem
N1	Copy of above
A$	General print string
TR%	User's tries
V	Numeric value of user's input
Q$	Dummy
N$	User's input as string
B	Billions held over
M	Millions held over
S	Thousands held over
H	Hundreds held over

```
 10 REM - Readnum
 20 MODE7
 30 PROCDBL(5,5,131,"READING NUMBERS")
 40 PRINTTAB(6);CHR$131;"_______________"
 50 PRINT"How many digits should the highest"
 60 PRINT"number have? (6 digits = 999999 maximum,";
 70 PRINT"but the program will handle any number"
 80 PRINT"of digits from 1 to 9.)"
 90 REPEAT:INPUT'"Your choice",D%:UNTIL D%>=1 AND D%<=9
100 REPEAT:INPUT"How many problems",PROB%:UNTIL PROB%>=1
110
120 REM - Game loop
130
140 TR%=0:FOR PR%=1 TO PROB%
150 CLS:N=RND(10^D%)-1:N1=N
160 PRINT'CHR$133;"Problem ";PR%
170 PROCDBL(0,3,131,"Type the number -"):PRINT:A$=""
180 TR%=TR%+1:IF D%=1 AND N=0 PROCS("nought"):GOTO200
190 GOSUB380
```

```
200 PROCS("."):PRINTTAB(12,15);:PROCBOX(D%,4)
210 IF V<>N1 GOTO250
220 PROCP:PROCDBL(13,18,129,CHR$136+A$)
230 PROCWARBLE:PROCDBL(12,22,130,"Press RETURN...")
240 INPUT Q$:GOTO310
250 CLS:PROCDBL(0,1,133,"I asked you for")
260 N=N1:GOSUB380:PROCS(".")
270 PROCDBL(0,11,133,"You typed "+N$)
280 N=V:GOSUB380
290 PROCDBL(0,22,129,"Press RETURN to try again...")
300 INPUT Q$:CLS:N=N1:GOTO160
310 NEXT PR%
320 CLS:PROCDBL(5,7,131,"The End")
330 PRINT'''"You did the"''PR%-1;" problems in ";TR%;" tries."
340 END
350
360 REM - Print numbers in English
370
380 IF N>1E9 GOTO420
390 IF N>1E6 GOTO440
400 IF N>1E3 GOTO500
410 IF N>99 GOTO560 ELSE GOTO620
420 B=N:N=INT(N/1E9):GOSUB380:PROCS("billion")
430 N=B-INT(B/1E9)*1E9:IF N=0 RETURN ELSE PRINT:PRINT
440 M=N:N=INT(N/1E6)
450 GOSUB380
460 PROCS("million")
470 N=M-INT(M/1E6)*1E6
480 IF N=0 RETURN
490 PRINT:PRINT
500 S=N:N=INT(N/1E3)
510 GOSUB380
520 PROCS("thousand")
530 PRINT:PRINT
540 N=S-INT(S/1000)*1000
550 IF N=0 RETURN
560 H=N:N=INT(N/100)
570 GOSUB620
580 PROCS("hundred")
590 N=H-INT(H/100)*100
600 IF N=0 RETURN
610 PROCS(" & ")
620 RESTORE
630 IF N>15 GOTO690
640 FOR D=1 TO N+1
650 READ A$:NEXT D
660 A$=A$+" ":PROCS(A$):IF N<21 RETURN
670 N=INT(N/10)*10
680 IF N=0 RETURN
690 IF N>19 GOTO730
700 FOR D=1 TO N-9:READ A$:NEXT D
710 IF N=18 A$="eighteen":GOTO660
720 A$=A$+"teen":GOTO660
730 FOR D=1 TO INT(N/10)+15:READ A$:NEXT
```

```
 740 A$=A$+" ":PROCS(A$):N=N-INT(N/10)*10
 750 IF N=0 RETURN ELSE GOTO620
 760
 770 DATA no,one,two,three,four,five,six,seven,eight,nine
 780 DATA ten,eleven,twelve,thirteen,fourteen,fifteen
 790 DATA twenty,thirty,forty,fifty,sixty
 800 DATA seventy,eighty,ninety
 810
 820 DEFPROCDBL(X%,Y%,C%,X$)
 830 PRINTTAB(X%,Y%);CHR$141;CHR$C%;X$
 840 PRINTTAB(X%,Y%+1);CHR$141;CHR$C%;X$:ENDPROC
 850
 860 DEFPROCS(X$)
 870 PROCD(POS,VPOS)
 880 PRINTCHR$(11);
 890 ENDPROC
 900
 910 DEFPROCD(X%,Y%)
 920 IF X%>3 GOTO960
 930 PRINTTAB(0,Y%);CHR$141;CHR$131
 940 PRINTTAB(0,Y%+1);CHR$141;CHR$131
 950 X%=X%+2
 960 PRINTTAB(X%,Y%);X$
 970 PRINTTAB(X%,Y%+1);X$;
 980 ENDPROC
 990
1000 DEFPROCBOX(L%,C%)
1010 V%=VPOS:W%=POS
1020 PRINTTAB(W%,V%-1);CHR$(C%+144);"7";
1030 FOR I%=0 TO L%+1:PRINT"£";:NEXT:PRINT"k"
1040 PRINTTAB(W%,V%+1);CHR$(C%+144);"u";
1045 FOR I%=0 TO L%+1:PRINT"p";:NEXT:PRINT"z"
1050 PRINTTAB(W%,V%);CHR$(C%+144);"5"
1060 PRINTTAB(W%+L%+3,V%);CHR$(C%+144);"j"
1070 PRINTTAB(W%+2,V%);CHR$135;
1080 FORZ%=1TOD%:PRINT".";:NEXT
1090 PRINTTAB(W%+3,V%);:N$=""
1100 G$=GET$:IF ASC(G$)=13 THEN 1130
1110 IF LEN(N$)>=D% THEN N$=LEFT$(N$,D%-1):PRINTCHR$8;
1120 PRINTG$;:N$=N$+G$:GOTO1100
1130 V=VAL(N$):ENDPROC
1140
1150 DEFPROCWARBLE
1160 FORS%=1TO20:SOUND1,-12,30,1
1170 SOUND1,-12,100,1:NEXT:ENDPROC
1180
1190 DEFPROCP
1200 ON RND(6) GOTO1210,1220,1230,1240,1250,1260
1210 A$="Right!":ENDPROC
1220 A$="Hooray!":ENDPROC
1230 A$="Clever old you!":ENDPROC
1240 A$="Great!":ENDPROC
1250 A$="Smashing!":ENDPROC
1260 A$="How about that!":ENDPROC
```