

Fuel Consumption

I am, I confess, something of a fanatical keeper of records. I write a diary, I keep track of how much I earn and how much I spend, I take brief note of the day's weather (so that when I return to sunnier climes I won't forget what it was really like) — and I record everything I spend on my car. Once the spending is recorded, it seems silly not to analyse it; so I spend a little time each month or two working out things like fuel consumption, cost per mile, and total amount spent on the car.

The fuel consumption, apart from being a constant reminder of the price one pays for comfort, can be a useful indicator of things starting to go wrong. The cost per mile is nice, because it consistently confirms my belief that a cheap second-hand car is cheaper than a new one, despite what all the motoring reports try to tell me. And the total amount spent can be pretty overwhelming.

If you're interested in using this program to help you gather your own figures, you'll need to have the patience to keep a logbook. Each time you buy petrol you must record: the milometer reading; whether the tank was filled; the price of the petrol (pence per gallon or litre); and the actual cost. You should also keep a separate note of other costs: repairs, tax, MOT inspections, insurance, and so on. Figure 1 shows a sample page from my logbook.

Negative costs are also possible. Refunds for work-related driving, payments from insurance companies, proceeds of selling the car, should all be entered as negative costs. And speaking of selling the car, it's my opinion that the only meaningful depreciation is the difference between buying price and selling price; so I wouldn't recommend trying to allow for depreciation among the other costs.

The essence of a logbook as used by this program is the continuation figures. As well as giving results for a page of entries, the

program uses previous totals to produce a set of updated totals. So there must be a starting set of 'previous totals'. If you start the logbook when you acquire the car, that's easy. The total cost is the initial cost of the car, including its purchase, insurance, tax, and whatever else one pays when buying a car. All other totals (miles travelled, gallons of petrol, etc) are zero.

The more likely case, though, is that you want to start a logbook when you have had the car for some time. Things are a little more difficult in this case. You must estimate all the previous totals, getting them as accurate as possible. The major costs won't be difficult: you should remember how much you paid for the car, and for any major repairs carried out. The mileage shouldn't be

Date	Mileage	Comment	Petrol Cost		Other Costs
9 Dec 81	04707	Fill at 34.3	14.68		
15 Dec 81	04886	Fill at 33.5	13.03		
11 Jan 82	05054	Fill at 33.1	13.00 +84	} Lots of snow driving	
15 Jan 82	05209	Fill at 33.5	11.65		
16 Jan 82	05429	Fill at 158	14.42		
19 Jan 82	05662	Fill at 33.1	14.31		
26 Jan 82	05844	Fill at 32.2	12.50		MOT work ticket (including steering bits, 2 new front tyres)
3 Feb 82	06061	Fill at 32.2	14.03	TAX	70.00
12 Feb 82	06268	at 165	5.00		
12 Feb 82	06275	Fill at 32.3	9.71		
20 Feb 82	06484	Fill at 31.5	12.54		
25 Feb 82	06696	Fill at 31.1	12.52		
4 Mar 82	06835	Fill at 30.6	8.30 +10	½L oil	
5 Mar 82	07024	Fill at 30.7	8.60	Service (plugs, points, condenser, grease, oil)	25.00
9 Mar 82	07237	Fill at 138.9	11.24	New exhaust	33.81
17 Mar 82	07433	Fill at 150	13.80		
27 Mar 82	07641	at 34.2	10.00		
16 Apr 82	07803	Fill at 34.9	14.50	Now advanced, running very lean	
26 Apr 82	08042	Fill at 35.8	15.14	2 new rear tyres	39.90
22 May 82	08240	Fill at 34.0	13.70		
This page: miles: 3730 km: 6002 petrol: 161.43 gals, 23.1 mpg; 12.2 L/100 km Oil: 1L; 3730 m/L; 6002 km/L Running Cost: £244.41 Running Expense: 6.55 p/m; 4.07 p/km Total Cost: £539.24 Total Expense: 14.45 p/m; 8.78 p/km				Overall: miles: 23340 km: 37560 petrol: 767.5 gals, 24.07 mpg; 11.73 L/100 km Oil: 2½L; 9336 m/L; 15024 km/L Running Cost: £1331.15 Running Expense: 5.70 p/m; 3.54 p/km Total Cost: £2686.10 Total Expense: 11.51 p/m; 7.15 p/km	

Figure 1. A sample page from a logbook. The figures at the bottom were computed by this program.

too difficult: you should be able to estimate how far you travel each week (or month, or year) and multiply it by the time you've had the car.

But how do you estimate your running cost (how much you've spent on petrol and oil), and how many gallons you have used? If you could do that properly you wouldn't need this program. I can make two suggestions, and leave you to decide. First, do your best. Bear in mind how many miles you've travelled, and what you think your fuel consumption might be, and the average price of petrol over the time you've had the car, and you might do quite well. Or second, pretend that you've just bought the car. Assess your total costs to date, but pretend you've travelled no miles. In other words, make a fresh start.

There is one more requirement if the program is to give you reasonable consumption figures in the short term. You should fill the tank once, noting the mileage, before starting the logbook proper. The program assumes that the car starts with a full tank, and calculates the consumption accordingly, so all purchases will be referred back to this initial full tank. When the program asks you for the last recorded mileage on the previous page, you should type the mileage at this pre-logbook fill.

This program gives you the option of saving data on tape from one run to the next. It's hardly necessary: there are only six numbers to be saved, and you will have written those in the logbook anyway. But it will serve as a useful example of a tape file; and if things are going to go wrong, you might as well discover it with a nice small file. Of course you don't have to use the file option if you don't want.

How to use the program

This is a menu-driven program. When you start it running, it presents a list of options and asks which you want next. It takes steps to ensure that you don't try an option for which it doesn't yet have the necessary information.

So long as you have started a logbook in the format described above, and can read instructions, the program is self-explanatory. If you intend to make use of the cassette file facility, I recommend that you locate the file directly after the program on the tape. The first time you record the data, and when you read it on sub-

sequent runs, the tape will be in the right position; and you will be reminded to reposition it before recording on subsequent runs. Then again, if you don't have a tape counter perhaps you should allow space for the data file at the beginning of the tape; then you are less likely to overwrite part of the program with data.

One feature of the program's use deserves special mention. The most tedious part of the program, entering details from the current page of the logbook, is also the most error-prone, so there are several checks on this. First, you are asked to check and confirm each entry after making it; do not overlook this important step. Second, every time you fill the tank you will be shown an approximate fuel consumption since the last fill. You shouldn't worry if this figure seems a little out — that could be due to an incompletely filled tank — but if it is grossly wrong you should check the whole entry with particular care. Third, if you enter a mileage which is higher than the last mileage on the page, you have obviously made a mistake, so the procedure will finish and pass you back to the main menu to try again.

```
These results are taken straight from
the previous totals.

Miles travelled: 19610
Kilometres travelled: 31558

Petrol consumed: 808.1 gals
                  24.3 mpg;  11.6 l/100km

Oil consumed: 1.5 litres
13073 miles/litre;  21035 km/litre

Running cost: £1086.74
Running expense: 5.5p/mile, 3.4p/km

Total cost: £2146.86
Total expense: 10.9p/mile, 6.8p/km

Press RETURN to get back to the menu. _
```

The kind of results the program produces

Program listing

```

100 REM Fuel consumption program, by Simon.
110 MODE6: totals=FALSE: current=FALSE
120 REPEAT
130   CLS: PRINT"Fuel consumption program, by Simon."
140   PRINT" 1. Read from a file the totals from"
150   PRINT"    the previous page of your logbook."
160   PRINT" 2. Type in the totals from the prev-"
170   PRINT"    ious page of your logbook."
180   PRINT" 3. Enter details of latest fuel"
190   PRINT"    purchases and other costs."
200   PRINT" 4. See the breakdown of figures for"
210   PRINT"    the previous totals."
220   PRINT" 5. See the results of this run."
230   PRINT" 6. Save the results of this run in a"
240   PRINT"    file."
250   PRINT" 7. Finish."
260   INPUT""Which would you like to do now? "choice
270   IF choice=1 THEN PROCfiletotals ELSE IF choice=2 THE
N PROCbooktotals ELSE IF choice=3 THEN PROCentries ELSE IF c
hoice=4 THEN PROColdresults ELSE IF choice=5 THEN PROCscreen
results ELSE IF choice=6 THEN PROCfileresults
280   UNTIL choice=7
290   CLS
300   END
310
1000 DEF PROCbooktotals
1010   CLS: totals=TRUE
1020   PRINT""Please enter the following items from"
1030   PRINT"the previous page of the logbook:--"
1040   PRINT"First, the mileage reading at the last"
1050   INPUT"petrol purchase: "miles1
1060   PRINT"and now the overall totals:--"
1070   INPUT""Miles travelled: "totmiles
1080   INPUT"Gallons of petrol: "totpetrol
1090   INPUT"Litres of oil: "totoil
1100   INPUT"Running cost: £"tottruncost
1110   INPUT"Overall cost: £"totcost
1120   PROCold
1130   ENDPROC
1140
1500 DEF PROCfiletotals
1510   CLS: totals=TRUE
1520   file=OPENIN"FUELDAT"
1530   INPUT#file,miles1,totmiles,totpetrol,totoil,totruncost
,totcost
1540   PRINT"Here's the data I've read from the""file. It s
hould correspond to data on""the previous page of your logb
ook.""If you don't think it's right, you'll""have to ente
r the totals yourself""(option 2)."
```

```

1550 PRINT"\"Mileage at the last purchase: ";miles1
1560 PRINT"\"Overall totals:-\"
1570 PRINT"\"Miles travelled: ";totmiles
1580 PRINT"\"Petrol consumed: ";FNchop(totpetrol,1);" gallons
"
1590 PRINT"\"Oil consumed: ";totoil;" litres\"
1600 PRINT"\"Running cost: £";totruncost
1610 PRINT"\"Overall cost: £";totcost
1620 CLOSE#file
1630 PROChold
1640 ENDPROC
1650
2000 DEF PROCentries
2010 CLS: IF totals THEN current=TRUE ELSE PRINT"\"You can't
do that until I've been given\"the previous totals (option
1 or 2).\": PROChold: ENDPROC
2020 INPUT"\"Turn to the current page of your log-\"book.
What is the milometer reading on\"the last entry? \"miles2:
IF miles2<10000 AND miles1>90000 THEN miles2=miles2+100000
2030 PRINT"\"Now enter the following details for\"each time
you bought petrol:-\"
2040 cumgals=0: runcost=0: fillgals=0: fillmiles=miles1
2050 REPEAT
2060 INPUT"\"Mileage: \"odometer: IF odometer<10000 AND mil
es1>90000 THEN mileage=odometer+100000 ELSE mileage=odometer

2070 INPUT"\"Filled? (Y or N) \"fill$: IF fill$="y" THEN fill
l$="Y"
2080 INPUT"\"Price (pence) per gallon or litre: \"price: pri
ce=price/100: IF price<1 THEN price=price*4.546
2090 INPUT"\"Cost (in pounds and pence): £\"cost: IF cost<>0
THEN thisgals=cost/price
2100 IF fill$="Y" THEN PRINT"\"Approx short-term consumptio
n: ";FNchop((mileage-fillmiles)/(fillgals+thisgals),2);"mpg.
"
2110 INPUT"\"Check that entry. Is it OK? (Y or N) \"ans$: a
ns$=LEFT$(ans$,1): IF ans$="y" THEN ans$="Y"
2120 IF ans$<>"Y" THEN PRINT"\"All right, I've cancelled i
t. Try again.\": mileage=0 ELSE cumgals=cumgals+thisgals: run
cost=runcost+cost: IF fill$="Y" THEN fillgals=0: fillmiles=m
ileage ELSE fillgals=fillgals+thisgals
2130 UNTIL mileage>=miles2
2140 IF mileage>miles2 THEN PRINT"\"Hmmm. That milometer rea
ding is higher\"than what you said was the last\"recorded
mileage on the page.\"\"Perhaps you should start option 3 aga
in.\": PROChold: ENDPROC
2150 INPUT"\"Fine. Now how many litres of oil were\"used o
n this page? \"oil
2160 IF oil>0 THEN INPUT"\"At what total cost? £\"oilcost: run
cost=runcost+oilcost
2170 othercost=0: PRINT"\"Last, please enter all other cost
s,\"finishing with zero.\"

```

```

2180 REPEAT INPUT" £"cost
2190 othercost=othercost+cost
2200 UNTIL cost=0
2210 PROChold
2220 ENDPROC
2230
2500 DEF PROColdresults
2510 CLS: IF NOT totals THEN PRINT""You can't do that until I've been given""the previous totals (option 1 or 2).": PROChold: ENDPROC
2520 IF totmiles=0 THEN PRINT""The previous total of miles driven is""zero. I'm not going to waste my time""working out a performance for that!": PROChold: ENDPROC
2530 PRINT""These results are taken straight from""the previous totals."";
2540 IF current THEN PRINT" They take no""account of the current entries you""have typed in via option 3."
2550 PROCoutput(totmiles,totpetrol,totoil,totruncost,totcost)
2560 PROChold
2570 ENDPROC
2580
3000 DEF PROCscreenresults
3010 CLS: IF NOT (totals AND current) THEN PRINT""You can't do that until I've been given""the previous totals (option 1 or 2) and""the current details (option 3).": PROChold: ENDPROC
3020 CLS: PRINT""Right. Here are the results.""First, for this page alone:-"
3030 PROCoutput(miles2-miles1,cumgals,oil,runcost,runcost+othercost)
3040 INPUT""Press RETURN for the remaining results."ans$: CLS
3050 PRINT"" . . . and now for the period since the""logbook was started:-"
3060 PROCoutput(totmiles+miles2-miles1,totpetrol+cumgals,totoil+oil,totruncost+runcost,totcost+runcost+othercost)
3070 PROChold
3080 ENDPROC
3090
3100 DEF PROCoutput(dist,pet,oil,rcost,tcost)
3110 PRINT""Miles travelled: ";dist
3120 PRINT"Kilometres travelled: ";FNchop(dist*1.60926,0)
3130 PRINT""Petrol consumed: ";FNchop(pet,1);" gals"
3140 PRINT FNchop(dist/pet,1);" mpg; ";FNchop(pet*282.49/dist,1);" l/100km"
3150 PRINT""Oil consumed: ";: IF oil=0 THEN PRINT"nil" ELSE PRINT;oil;" litres"FNchop(dist/oil,0);" miles/litre; ";FNchop(dist*1.609/oil,0);" km/litre"
3160 PRINT""Running cost: £";FNchop(rcost,2)'Running expense: ";FNchop(rcost*100/dist,1);"p/mile, ";FNchop(rcost*62.14/dist,1);"p/km"

```

```

3170 PRINT "Total cost: £";FNchop(tcost,2)' "Total expense:
";FNchop(tcost*100/dist,1);"p/mile, ";FNchop(tcost*62.14/dis
t,1);"p/km"
3180 ENDPROC
3190
3500 DEF PROCfileresults
3510 CLS: IF NOT (totals AND current) THEN PRINT'"You can'
t do that until I've been given'"the previous totals (optio
n 1 or 2) and'"the current details (option 3).": PROChold:
ENDPROC
3520 PRINT'"Don't forget to position the tape'"correctly
if you're using cassettes."
3530 file=OPENOUT"FUELDAT"
3540 PRINT#file,odometer,totmiles+miles2-miles1,totpetrol+c
umgals,totoil+oil,totruncost+runcost,totcost+runcost+otherco
st
3550 CLOSE#file
3560 PRINT'"OK, the values have been saved."
3570 PROChold
3580 ENDPROC
3590
4500 DEF PROChold: LOCAL dummy$
4510 INPUT"Press RETURN to get back to the menu."dummy$
4520 ENDPROC
4530
5000 DEF FNchop(x,n): LOCAL factor,whole
5010 REM To chop x to a rounded number with n decimal place
s.
5020 factor=10^n: whole=INT(x*factor+0.5)
5040 =whole/factor

```

Comments on the program

1. Program structure and line numbering.

This is a well-structured program. The menu is presented in a repeat loop in the body of the program, and each of the menu options is dealt with by its own procedure. There are two additional program segments, PROChold and FNchop.

PROChold, which is called before control is returned to the menu from any procedure, is designed to allow the user time to read any messages displayed by the procedure before they are wiped out by the menu. It is written as a procedure to avoid repetition: one procedure and eleven calls are significantly easier to write (and to read) than eleven repetitions of the same piece of code, even if it is only one line.

FNchop is a function which will come in handy in many different programs. It's all very well messing about with @% (see comment 2 to the telephone costs program) if all numbers are to be output in the same format, but if we want some numbers to show two decimal places, some one, and some none, this function makes things a lot easier. It takes two arguments (another word for parameters), and returns the value of the first argument, chopped to a value whose number of decimal places is given by the second argument.

In fact the word 'chop' is a little misleading, as the function ensures that the returned value is correctly rounded, rather than truncated. The addition of 0.5 in line 5020 takes care of this. If you don't see how, try working a few examples with pencil and paper.

Now what about the line numbering? It's rather pretty: a different range of 500 numbers for each procedure. But what does it achieve? In my view, very little. It certainly helps make the program's structure a little clearer; but if the program is properly written that shouldn't be necessary. And, oh dear, the trouble it causes if you want to alter the program! You can't selectively renumber bits of the program, so you just can't use RENUMBER without completely destroying the scheme. If you know for a fact that the program will never require any alteration, perhaps you'll find this type of scheme justified; but who can ever know that about any program without being hopelessly static?

If it's any consolation, there are no GOTOs in the program. If I were you I'd type it in on AUTO numbering, and ignore the line numbers I've used here.

(Talk about learning from the mistakes of others!)

2. Going round the clock.

The IF statements in lines 2020 and 2060 serve only to deal with a car which 'goes round the clock', or passes from 99999 to 00000. Don't laugh. My present car went round the clock two years ago, and my previous car (in Australia) went round for the second time in the early '70s. Of course you should feel free to dispense with the statements if you don't go in for such reliable and long-lasting transport.

3. User-friendly error avoidance.

The message in line 2520 sounds friendly enough. What user would know that it really means 'If I proceed I'll be dividing by

zero, which isn't nice, so I'll stop here'? The fear which many people have of computers can be greatly diminished by personal-sounding comments like this. They're worth the little effort which goes into thinking them up.

4. Boolean state-checkers.

Sounds difficult, but it isn't. In a sense, this program can be seen as passing through various states, some of which cannot be entered until certain prerequisite states have been passed. For instance, the results cannot be calculated if either the previous totals or the current entries have not been input. Yet the program has good reason for displaying the same menu each time. So it must take responsibility for checking whether the prerequisites have been satisfied for a chosen menu option.

This responsibility is assigned to the boolean (i.e. logical or true/false) variables called current and totals. In any procedure which uses the previous totals, a check is made on totals: if it is TRUE, the procedure can proceed, otherwise a suitable message is displayed and the procedure is aborted. Similar checks are made on current in procedures which require the current entries to have been input.

Notice that the action taken if one of these checks fails is an ENDPROC. Many programmers fall too easily into the use of GOTOs in such circumstances, despite the fact that ENDPROC is actually a lot more meaningful.

5. What parameters to pass.

The previous totals totmiles, totpetrol, totoil, tottruncost and totcost are never actually updated, although one might have thought that this was the entire purpose of the program. The reason is that the user must always be able to start any phase of the program again, and must be able to use correctly option 4 at any stage. Consider the careful user who wants to type the current entries twice and compare the results: (s)he obviously wants the current totals to be added to the same previous totals each time. But this wouldn't happen if the totals were actually updated in the course of the program.

Notice what is done instead. PROCscreenresults makes two calls to its subsidiary PROCoutput, one with the current figures and one with expressions combining the previous totals and the current figures. The new totals will be displayed, and if the

user wants to believe that the program has replaced the old totals with new ones, who are we to object? PROCfileresults uses a similar device when it writes results to the file.

6. Data files on cassette.

See how easy it is to store data on tape, or read it back into the program? All that is required is an OPENIN or OPENOUT command, which uses the filename and returns a numeric value called the channel number; an INPUT# or PRINT# which refers to that channel number; and a CLOSE# which also uses the channel number. Care must of course be taken to ensure that the tape is correctly positioned before the program saves data; otherwise valuable program might be destroyed.

Suggested amendments to the program

1. If you don't live in Britain, you will need to change currency and perhaps other units. The changes are fairly easy to make.
2. Even if you live in Britain you will eventually have to start thinking in kilometres and litres and the like. Perhaps as a first step you should change the program so that it gives these units first and the old imperial units as secondary. Whether you do this entirely is more likely to depend on the instruments in your car and your garage than on how forward-thinking you are.
3. The program takes great care to allow the user to change his/her mind about details of a petrol purchase, but makes no allowance for error with regard to oil. Perhaps you could write a user check there, too. The 'other costs' section isn't too bad — an erroneous cost can be wiped out by entering the negative of the same amount — but it would still look nicer with a user check and confirmation.

Pie Chart

The pie chart is one of the most common representations of a breakdown of figures — perhaps because so many people find it easy to digest. This program uses six colours in mode 2 to draw a pie chart from figures input by the user.

Unfortunately, there's so little to drawing a pie chart that the program hardly seemed to merit a place among the others in this book — until Gill Hersee suggested the touch that it needed: drawing comparative pie charts. If you're interested in seeing how changes in individual figures affect the overall picture, you just input the changes. A new pie chart will be drawn inside the original one, so that you can compare the two. You can follow these with a third, a fourth, and even a fifth pie chart — although by the time you get to the fifth the chart is pretty small.

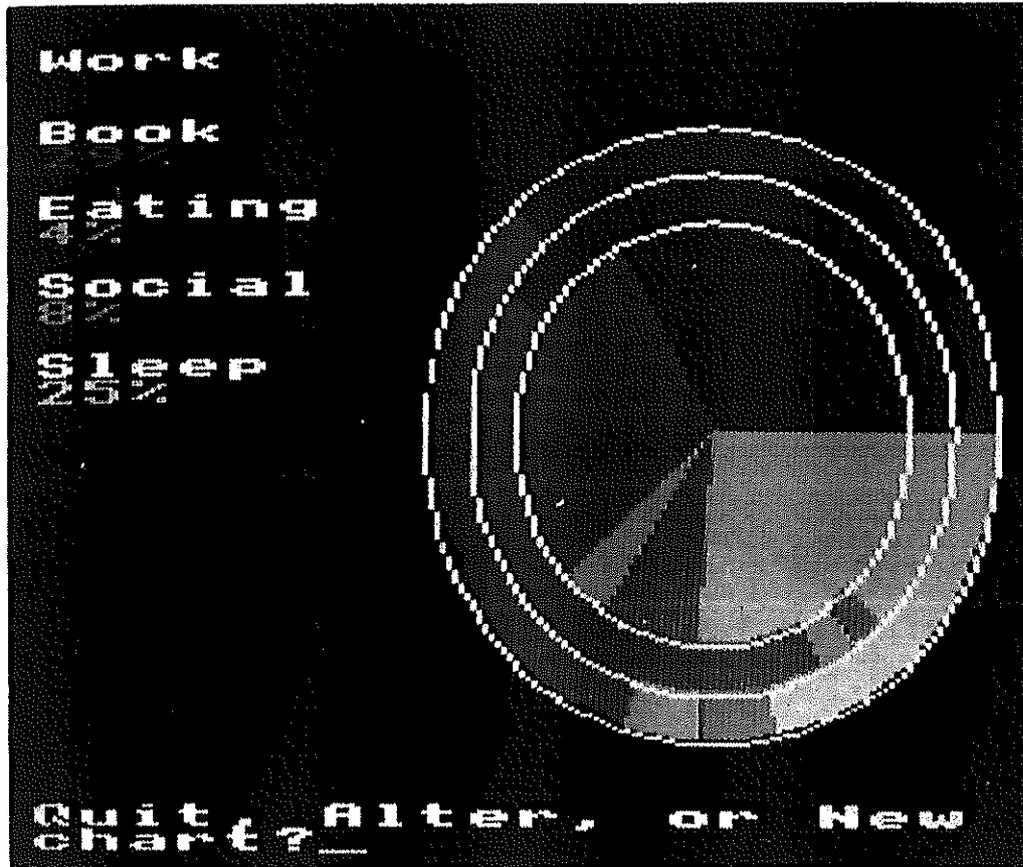
How to use the program

The program starts by asking for the names and associated values of the items you want to chart. There is a limit of ten items, so that they will all fit together on the screen when the chart is drawn in mode 2. For the same reason, names will be limited to seven characters when the chart is drawn; you should bear this in mind when entering them.

When you have entered the items the program switches to mode 2 and draws the chart. As each sector is drawn, the name of the item appears in a text window at the left of the screen, along with its percentage of the total. You will soon appreciate that the sectors are drawn anticlockwise, starting from the positive x-axis; but to help you sort out which item corresponds to which sector of the pie, the percentage is shown in the same colour as the sector.

You will now be asked whether you want to 'Quit, Alter, or

New chart?'. A reply of Q will stop the program; N will start it again; and A will present each of the values in turn and ask whether you wish to change it. If you don't want to alter a particular value, a RETURN is the easiest way of saying 'No, let's go on to the next one'. When all of the items have been listed for alteration, a new chart will be superimposed on the first one, but with a smaller radius, so that you can compare the two.



A pie chart revised a couple of times for comparison

Program listing

```

100 REM Pie Chart, by Simon.
110 max=10: DIM value(max),name$(max)
120 xcen=896: ycen=512: pies=0: degree=PI/180
130 REPEAT
140   IF pies=0 THEN MODE6: PROCinput: MODE2: VDU28,0,31,7
,0 ELSE PROCalter
150   PROCdraw: VDU26,31,0,30
160   IF pies=5 THEN PRINT"Quit or New chart?"; ELSE PRINT
"Quit, Alter, or New chart?";
170   a$=GET$
180   IF pies=5 THEN IF INSTR("Aa",a$) THEN a$="N"

```

```

190 IF INSTR("Nn",a$) THEN pies=0 ELSE IF INSTR("Aa",a$)
THEN PRINTTAB(0,30)SPC(35);: VDU28,0,31,7,0
200 UNTIL INSTR("NnAa",a$)=0
210 END
220
230 DEF PROCinput: LOCAL I,A
240 VDU19,0,4,0,0,0
250 PRINTTAB(4,1)"Pie Chart Program, by Simon."
260 PRINT"Please input the name and value of each"
270 PRINT"item you want charted. To finish, enter"
280 PRINT"an asterisk instead of a name."
290 VDU28,0,24,39,7: REM A text window.
300 radius=364: I=1: number=0
310 REPEAT
320 PRINT" Item ";I: INPUT"Name? "name$(I)
330 IF name$(I)="*" THEN number=I-1: I=max ELSE INPUT"Va
lue? "value(I)
340 I=I+1
350 UNTIL I>max
360 IF number=0 THEN number=max: PRINT""I hope that's all
- I haven't allowed""for any more.""""Press a key to see
the chart.": A=GET
370 ENDPROC
380
390 DEF PROCdraw
400 CLS: PROCcircle: pies=pies+1
410 total=0: theta=0: colour=0: MOVExcen+radius,ycen
420 FOR I=1 TO number: total=total+value(I): NEXT
430 FOR I=1 TO number: PROCsegment(I): NEXT
440 MOVExcen,ycen: PLOT85,xcen+radius,ycen
450 ENDPROC
460
470 DEF PROCcircle: LOCAL phi
480 GCOL0,7: MOVExcen+radius+8,ycen
490 FOR phi=0 TO 2*PI STEP degree*3
500 DRAWxcen+(radius+8)*COS(phi),ycen+(radius+8)*SIN(phi
)
510 NEXT
520 ENDPROC
530
540 DEF PROCsegment(J): LOCAL angle,phi,fraction
550 fraction=value(J)/total: angle=2*PI*fraction
560 PRINT'LEFT$(name$(J),7): colour=colour+1
570 IF colour=7 THEN colour=1: IF J=number THEN colour=4
580 GCOL0,colour: COLOURcolour
590 PRINT;INT(fraction*100);"%
600 FOR phi=theta+degree TO theta+angle STEP degree*3
610 MOVExcen,ycen
620 PLOT85,xcen+radius*COS(phi),ycen+radius*SIN(phi)
630 NEXT: theta=theta+angle: COLOUR7
640 ENDPROC
650

```

```

660 DEF PROCalter: LOCAL I,ans$
670 CLS: radius=radius-60
680 FOR I=1 TO number
690   PRINT 'LEFT$(name$(I),7)';value(I)' "Alter?": ans$=GET
$
700   IF INSTR("Yy",ans$) THEN INPUT "New val?"value(I)
710 NEXT
720 ENDPROC

```

Comments on the program

1. Text windows.

In PROCinput the program prints a heading and some instructions, and then sets a text window to exclude them. This means that any subsequent text (*viz.* the names and values input by the user) will scroll within the window, leaving the heading and instructions where they are.

A second type of text window is used by the main program, in lines 140 and 190. This window restricts text to the leftmost eight character positions, thus ensuring that the chart will not be spoiled by names, percentages, or anything else that might be written.

Notice that neither of these windows is explicitly removed. Instead, the program takes advantage of the fact that a MODE command restores all windows to the full screen, which it then clears.

2. Text size in modes 2 and 5.

The program must use mode 2 to have the benefit of colour, but this means that it must accept the double-width mode 2 characters. Notice what a limitation this is. There would be no hope of actually writing the item names on the chart, nor even of writing a code number there. It is left to the user to make the correct associations between the items and the sectors of pie. Even here the program is hindered: the item names and percentages are displayed in a text window at the left of the screen, but their number and length are drastically limited by the character size.

3. Radial drawing and resolution.

A common way of drawing circles is to increase x in constant steps, find the corresponding y , and plot them. But this program clearly needs to increase the angle in constant steps, so that it can

apportion the whole circle according to the fraction required by each item. No problem, once we realise that a line of length r which makes an angle of θ with the x -axis goes to the point $x = r \cos \theta$, $y = r \sin \theta$. We simply invent the angle, start it at 0, and keep adding to it in constant steps (three degrees in this case) until it reaches 2π , the full extent of the circle.

One problem with such radial drawing is that it really shows up the relatively poor resolution of mode 2. Seen from well back, the circle looks quite reasonable, but close up, it becomes extremely ragged. That said, it might be encouraging to know that drawing the charts in steps of three degrees is pushing the resolution to its limits. You can get a somewhat faster chart with only slightly worse resolution by plotting it in greater angular steps.

4. Solid shapes with triangles.

Notice how the program uses triangles of colour to make the segments of a circle. To fill in each triangle, it moves to the centre from the last point plotted on the circumference, and then uses PLOT85 to fill the triangle between those two points and the required new point. It isn't difficult to envisage a circle as comprising lots of thin triangles emanating from the centre, and this program makes practical use of that image.

When the fractional representation of each item is calculated, and is then turned into a whole number of degrees for plotting, a little bit of it, the bit after the decimal point, will be lost in the process. (This is an instance of 'rounding error', once greatly feared in scientific computing circles.) By the time the whole circle has been plotted, the net effect of these missing bits might be enough to leave a very small gap at the end of the chart. The easiest way of dealing with this is to explicitly fill in the triangle between the finishing point and the starting point, whether it needs it or not. This is taken care of in line 440.

5. Watching for adjacent colours.

Cycling round the colours from 1 (red) to 6 (cyan), there is no risk of plotting two adjacent colours — unless the colour of the last sector is the same as the colour of the first. So whenever the colour is changed back to 1, the program checks whether it is about to plot the last sector, and uses a different colour if it is. I have chosen colour 4, but it could have been anything other than 1, the one we're avoiding, or 5, the most recently plotted colour.

6. Forcing the choice.

The main loop in lines 130 to 200 is designed to repeat when asked to alter the present chart or draw a new chart. To limit the number of charts to 5, it gives the user a different prompt when that many have been plotted — but a different prompt isn't enough. A sly user could still try typing 'A' when prompted 'Quit or New chart?'. So the program checks for such a response, turning it to 'N' if it finds it.

Why bother? Why not let the user continue with smaller and smaller charts, at the expense of his or her eyesight? Because the charts don't keep getting smaller and smaller. As each altered chart is plotted, the radius is decreased by 60. After six charts it would become negative: the pies would start growing again, and would be plotted the other way round — not really conducive to serious comparison. By all means take out the forced decision for the purpose of investigation, but I'd be inclined to leave it in the final version of the program.

7. RETURN as a response.

The wording of line 690 implies that a response of 'Yes' or 'No' is required. A quick look at the code, though, shows that while the positive response is indeed 'Y', any other key will be taken as a negative response. I find that I generally have a finger resting on the RETURN key, so I use that key to reply 'No'. Many programs which seem to require a Y/N reply only test for one of them. If you write programs in this way, you should make sure that the answer which requires special action is the one which is tested; otherwise a sloppily struck key will make the program perform the special action, which is hardly a desirable state of affairs. In this program, for instance, it is quite reasonable to ask you to tap a particular key if you wish to alter an entry; but it would be most unreasonable to specify a particular key, say N, as meaning 'don't alter', and then to take any other key as a request to alter.

Suggested amendments to the program

1. The program presently calculates the total from the items entered. Many pie charts, though, deal with several main items and a predefined total, and put aside a section for 'Other' items. This program could easily do this if modified to accept a total

before asking for the items and their values. You could then subtract the sum of the values from the given total to establish the size of 'Other'.

Well, I had to leave something for you to do!

Section 2

Educational

Phases of the Moon

I don't know whether it is true of Britain as a whole; people keep trying to persuade me that it isn't. But it is certainly true of Devon that one doesn't see the moon any more than about three nights a month. The rest of the time it is thoroughly obscured by cloud. (Shades of hankering for Australia!)

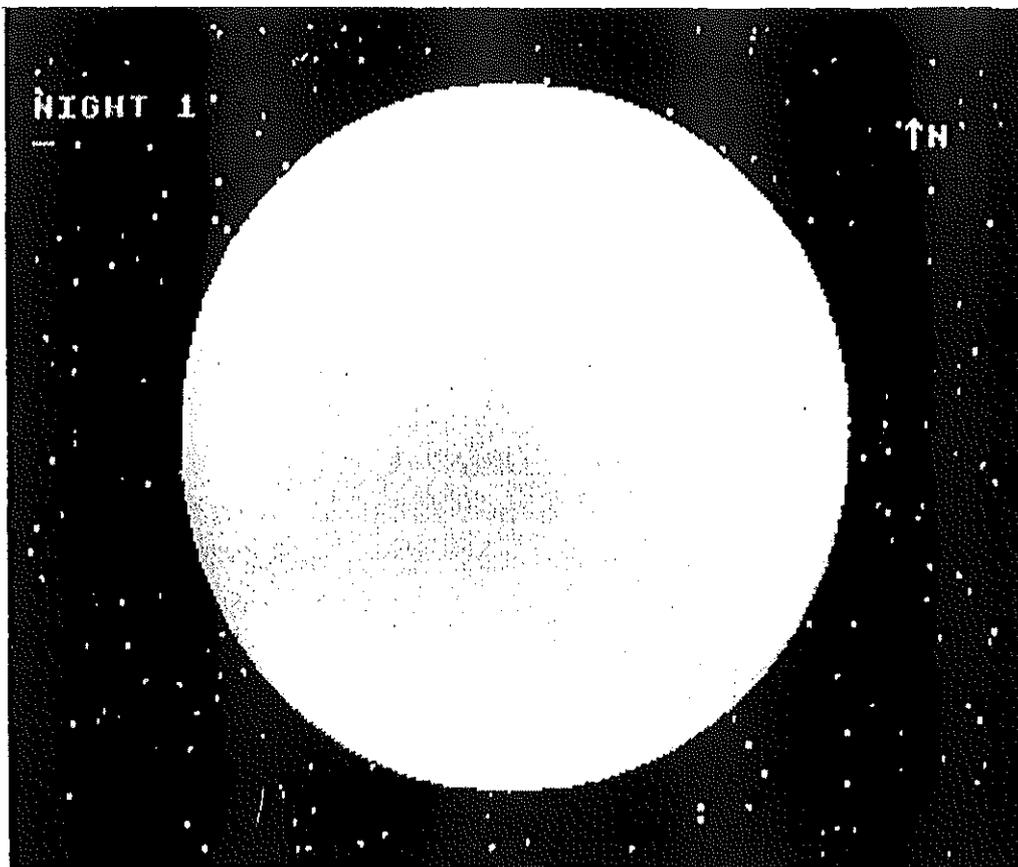
Consequently, or so it seems to me, the people here know very little about the moon's phases. This program can help put that right. What's more, it can show in about ten minutes what it would take twenty-nine clear nights to see in nature.

Not, mind you, that I intend the computer to supplant nature. I am reminded of a drawing by the superb Australian cartoonist Leunig, in which a man proudly shows his son a magnificent sunrise on television while an equally magnificent sunrise goes unnoticed at the window behind them. No, my intention is rather to show the user an aspect of nature which is too seldom seen here — perhaps even whetting his or her appetite for a place where the real thing can be relied on to show itself now and then.

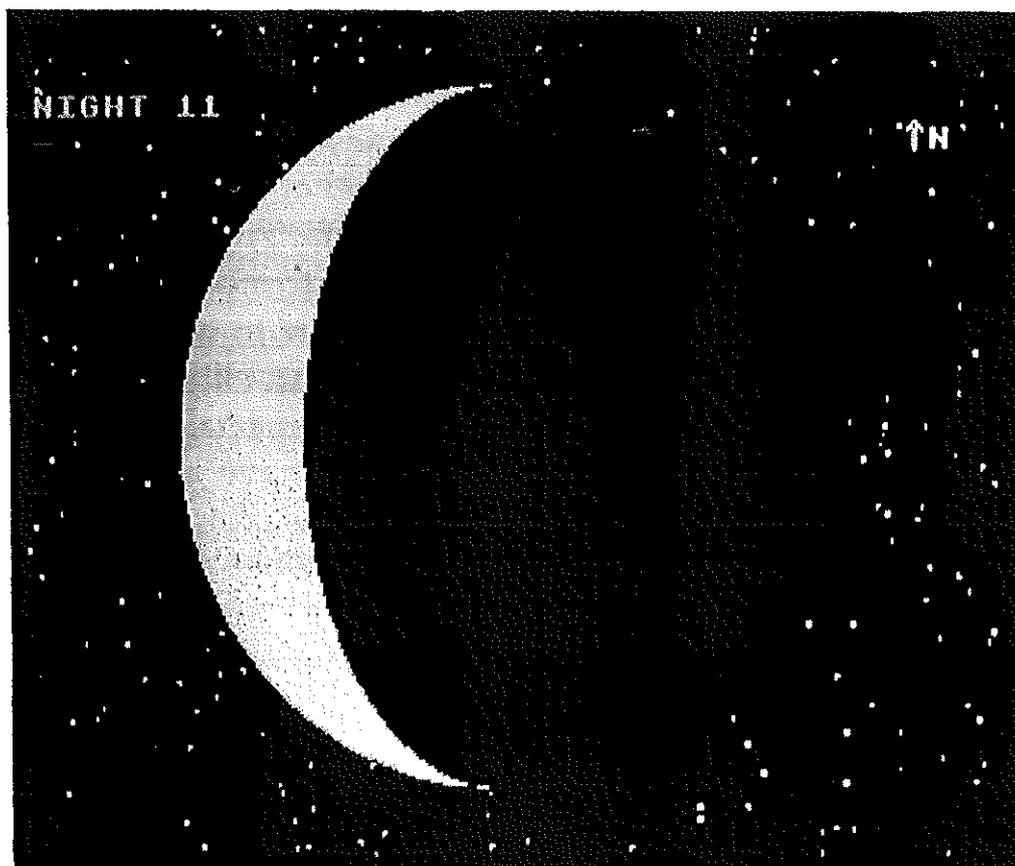
The moon's phases are mathematically easy to model. One side of the lighted part of the moon always appears to us as a semi-circle, and the other side as half an ellipse. (An ellipse is like a circle seen from an angle; the circle itself is a special case of the ellipse. A spinning coin will at any instant have an apparent shape which is elliptical.) When the moon is waxing (growing), the changing ellipse is on the eastern side; when it is waning (shrinking), the ellipse is to the west.

How to use the program

This program requires nothing of the user once it is running. Type it in, type RUN, and sit back and watch. The flashing cursor is turned off (it would be about as welcome in the night sky as a



The program starts with a full moon . . .



. . . and shows the phases night by night

jumbo jet in its landing pattern), the stars are drawn, the full moon appears; and then the night-by-night progression of the phases begins.

Program listing

```

100 REM Phases of the moon, by Simon.
110 MODE4
120 VDU 23,1,0;0;0;0;: REM Cursor off
130 xcentre=640: ycentre=512: radius=432: steps=-8
140 PRINT TAB(11,14);"PHASES OF THE MOON"
150 PRINT TAB(11,17);"A program by Simon"
160 PROCwait(3)
170 REM A new character.
180 VDU23,240,24,24,24,24,24,24,24,24
190 VDU5: MOVE 1140,900: PRINT"^"
200 MOVE 1140,892: PRINT CHR$(240);"N": VDU4
210 REM Plot some stars.
220 FOR I=1 TO 150+RND(150)
230   PLOT 69,RND(1280),RND(1024)
240   PLOT1,0,RND(4): PLOT1,RND(4),0: PLOT1,0,-RND(4)
250 NEXT
260 PROCwait(5)
270
280 PRINT TAB(0,3);"NIGHT 1"
290 MOVE xcentre,ycentre+radius: PLOT0,0,0
300 FOR Y=radius TO -radius STEP steps
310   REM The x-coordinate of a circle.
320   X=SQR(radius*radius-Y*Y)
330   PLOT 85,xcentre-X,ycentre+Y
340   PLOT 85,xcentre+X,ycentre+Y
350 NEXT
360 PROCwait(5)
370
380 REM The moon wanes for 14 nights.
390 FOR N=1 TO 14
400   PRINT TAB(6,3);N+1
410   MOVE xcentre,ycentre+radius: PLOT0,0,0
420   FOR Y=radius TO -radius STEP steps
430     REM The x-coordinates of last night's and
           tonight's ellipses.
440     X1=SQR(radius*radius-Y*Y)*COS(N*PI/14)
450     X2=SQR(radius*radius-Y*Y)*COS((N-1)*PI/14)
460     REM Take away the difference.
470     PLOT 87,xcentre+X1,ycentre+Y
480     PLOT 87,xcentre+X2,ycentre+Y
490   NEXT

```

```

500 PROCwait(5)
510 NEXT
520
530 REM The moon waxes for 14 nights.
540 FOR N=15 TO 28
550 PRINT TAB(6,3);N+1
560 MOVE xcentre,ycentre+radius: PLOT0,0,0
570 FOR Y=radius TO -radius STEP steps
580 REM The x-coordinates of last night's and
           tonight's ellipses.
590 X1=SQR(radius*radius-Y*Y)*COS(N*PI/14)
600 X2=SQR(radius*radius-Y*Y)*COS((N-1)*PI/14)
610 REM Plot in the difference.
620 PLOT 85,xcentre-X2,ycentre+Y
630 PLOT 85,xcentre-X1,ycentre+Y
640 NEXT
650 PROCwait(5)
660 NEXT
670 VDU23,1,1;0;0;0;: REM Cursor on
680 END
690
700 DEF PROCwait(n)
710 TIME=0: REPEAT UNTIL TIME>100*n
720 ENDPROC

```

Comments on the program

1. VDU 4 and VDU 5.

To make a north-pointing arrow, I have combined the standard arrowhead character with a user-defined shaft (character 240, defined in line 180). The correct relative positioning of these two is somewhat beyond the capabilities of the text cursor, restricted as it is in mode 4 to 32 lines. So we use VDU 5 (shorthand for PRINT CHR\$(5)) to enable text to be written at the position of the graphics cursor; position that cursor with the MOVE command; print the characters; and use VDU 4 (shorthand for PRINT CHR\$(4)) to ensure that any further text will be written at the text cursor.

2. Playing with random numbers.

The random number generator is used in two different contexts in this program. First, used alone, RND(x) is simply used to generate a random number between 1 and x. That's fine for the circumstances in which I want a random number between 1 and

x. But when deciding on a random number of stars to plot (no more than about 300), the same device doesn't work: a random number between 1 and 300 might well be a mere one or two. And one or two stars do not a night sky make, except perhaps in London or Melbourne. So I chose a minimum number of stars, too, and simply randomised the difference. Hence the use of $150 + \text{RND}(150)$. I did actually toy with the idea of making a genuine northern night sky, but I doubt that anyone would want to type in all the data if I did.

3. Notional and actual pixels.

The idea of what I choose to call notional pixels is one which didn't come across to me very clearly when I read the user guide, so I shall expound it here. Pixel, by the way, is a word used to indicate the smallest meaningful element of a picture.

The graphics screen is addressable in 1280×1024 points. The address of a point on the screen has a first coordinate which can be any number between 0 and 1279, and a second coordinate which can be any number between 0 and 1023.

We know, though, that different modes have different graphics 'resolutions'. Mode 5 has 160×256 graphics points; mode 4 has 320×256 . So each of these graphics points, or plottable points, corresponds to several screen address points. In mode 5, for instance, a graphics point corresponds to 8×4 screen points ($1280/160 \times 1024/256$); in mode 4, a graphics point corresponds to 4×4 screen points. These graphics points are the actual pixels in a given mode. But so far as addressing is concerned, the computer always believes that it has 1280×1024 pixels. These screen addresses I shall call notional pixels. Any graphics effects which we create will use actual pixels; but any addressing that we do will be to the notional pixels — a fact which confused me at first.

Let us concentrate on mode 4 for the time being. One of its actual pixels, the second from the left at the bottom of the screen, consists of the 16 notional pixels whose addresses are (4,3), (4,2), (4,1), (4,0), (5,3), (5,2), (5,1), (5,0), (6,3), (6,2), (6,1), (6,0), (7,3), (7,2), (7,1), and (7,0). All of these addresses refer to exactly the same point on the mode 4 graphics screen.

In plotting my stars, I decided to give the appearance of different intensities by plotting different numbers of adjacent points. One point (or actual pixel) would be the dimmest star, and four points,

in the shape of a square, the brightest. But given that one has gone to a random notional pixel (which must of course be in an actual pixel), how far must one move in order to get to the next actual pixel? Consider that the graphics cursor is in the notional pixel whose address is (7,2). This is in the actual pixel described above. To get to the next actual pixel to the right, we need only move one notional pixel, to (8,2). To move one actual pixel to the left, though, we must move four notional pixels, to (3,2). Likewise, to move up we must move two notional pixels, to (7,4); and to move down we must move three notional pixels, to (7, -1) (yes, it does exist, even though it isn't on the screen). Clearly, the furthest we need move to ensure a place in the next actual pixel is four notional pixels; but unless we know where we are in the actual pixel, we don't know how many fewer than four will do the same trick.

What I have done, then, for the stars, is first to plot a random point. That takes care of the one pixel. I then move upward a random number of notional pixels, four at the most, which might or might not take me to the next actual pixel. I do the same sort of move to the right, and the same sort of move downward. And at the end of it all I have just what I wanted: a star consisting of one, two, three, or four plotted pixels.

4. The PLOT command.

The PLOT command is extremely versatile, and so takes quite some getting used to. It's worth the effort. This program uses it in five different ways — five of the 64 presently available. Let us look at those five, by way of starting.

PLOT 85,x,y draws a solid triangle in the foreground colour (white in this instance) between the point (x,y) and the last two actual pixels (this is most important) visited by the graphics cursor. As this is the only easy way of filling in areas on the screen, it is what we use for circles, ellipses, squares, and any other shapes we want filled in. As we have already seen in the pie-chart program, it is not difficult to envisage a circle as being composed of triangles — we need merely approximate the circle's curve with lots of little straight edges. If you look very closely at any of the curves produced by this program, you will see the edges of the triangles. You could even put in an extra delay after each triangle (345 PROCwait(0.5), for instance) to see how the triangles combine to produce the curves.

PLOT 87, x,y is almost the same. It draws solid triangles too, but in the background colour — black in this program. It is used here to 'undraw' bits of the moon, night by night.

PLOT 69, x,y is used to plot a single point in whichever actual pixel contains the notional pixel (x,y). This is used for the first (and possibly only) point in a star.

PLOT 1, x,y is used in a significantly different context from the three versions of PLOT looked at so far. It draws a line from wherever the graphics cursor is, not to the point (x,y), but to the point which is (x,y) away from the starting point; that is, to the point which is x units away horizontally, and y units vertically. This is what the User Guide means when it says 'draw line relative' as opposed to 'draw line absolute'. This is clearly what I want to do when, having plotted a random point, I want to move so many notional pixels up, so many across, and so many down.

PLOT 0, x,y is another of the relative movements, and in fact it is just a movement, with no plotting. When building a 'ladder' of triangles of the sort you will see if you put in that extra delay in line 345, the last two pixels visited are automatically taken care of — they are the closest side of the last triangle drawn. But we must have two appropriate starting points for the first triangle. The first one is easy: it is the top centre of the moon, to which we MOVE the graphics cursor (and MOVE, by the way, is simply PLOT 4 in disguise). The second point might be easy, too, but the way I have calculated the subsequent triangles makes it difficult for me to make it another point of the first triangle. So I cheat, and visit the same pixel again, with PLOT 0,0,0. Thus the first triangle will actually be the straight line formed between these two identical points and the next one plotted — a little bit of cheating which will go quite unnoticed in the overall picture.

You might wonder why I have made two separate operations of plotting foreground and plotting background, instead of plotting them both as the inverse of what is already there (PLOT 86, x,y). The easiest answer is 'try it and see'. The point is that the triangles plotted are not mutually exclusive; any two adjacent triangles share a common edge. Consider the case in which the dark side of the moon is replacing the light side. The common edge of two triangles will first be inverted from white to black, as expected, when the first triangle is inverted. Then the second triangle, including the black edge, is inverted — and the edge becomes white again. Pretty, but not quite the desired effect.

5. Dragging out a computation.

Lines 440, 450, 590, and 600 could have been dispensed with, and their calculations performed in the PLOT lines. For example

```
470 PLOT 87, xcentre + SQR(radius*radius - Y*Y)
   *COS(N*PI/14), ycentre + Y
```

Why wasn't this done? Because I feel that it would have unnecessarily confused two operations — the calculation of the appropriate x-coordinate, and the plotting of the point relative to the centre. It seemed to me that the separation of these two operations would make the program's intentions easier to grasp.

Many programmers, though, go much too far in this dragging out of computations. All too often one sees a computation like the one in question rendered

```
440 X1 = radius*radius
441 X1 = X1 - Y*Y
442 A1 = N/14
443 A1 = A1*PI
444 A1 = COS(A1)
445 X1 = SQR(X1)
446 X1 = X1*A1
470 X1 = xcentre + X1
471 Y1 = ycentre + Y
472 PLOT 87,X1,Y1
```

To my way of thinking, this achieves nothing but confusion. Do people do it because they are unable to grasp the overall mathematical expression? Hardly! They must have understood the expression to translate it into the form used. Why, then, do they do it? I can only see it as an evil practice passed down from the days of assembly-code programming (in which such dragging out is indeed necessary). I can only hope that this paragraph helps persuade some people that the dragging out of computations should be practised only so far as is needed to make the program more comprehensible; when it starts to make it less comprehensible, it becomes reprehensible.

6. No integer variables.

Why haven't I used integer variables for the x- and y-coordinates of the pixels? Because to me, curve-plotting is always something

involving real numbers. This might well be regarded as a throw-back to my days as a physicist and mathematician, but I certainly don't feel that the use of integer names would have made the program the slightest bit easier to understand.

It would, however, improve the speed. If all of the variables in the program are made integer, it takes some 20% less time to run, despite the deliberate delays. But why should anyone want it to run faster? Like a proud father on sports day, I think it runs at just the right speed.

7. No stars behind the moon.

Some people in England think I'm rather rude in suggesting that they don't see enough of the moon. They then immediately support my thesis by asking why I haven't put any stars in the 'undrawn' area. If you're wondering the same thing, think about what that undrawn area actually represents.

Suggested amendments to the program

1. If you haven't already done so, try the suggestions made in comment 4 above: putting in a delay between triangles, so as to see their shape and size; and changing the PLOT 85 and PLOT 87 commands of lines 470, 480, 620, and 630 to PLOT 86. You should notice an interesting effect with this second one: while one pass of inversion will produce the lines in the wrong colour, the second pass will look fine. That's because on the second pass, lines of the wrong old colour are inverted twice, leaving them in the right new colour.

Sadly, what you should observe and what you do observe are different. I haven't quite worked out why yet.

2. The program can be made to run faster, at the expense of resolution, by altering the step size (the variable 'steps'). The picture is still quite reasonable with steps at -16 — try it. You can even go the other way — better resolution and a slower program. But steps should always be a multiple of -4 , otherwise it won't be moving a whole number of actual pixels.

You could also speed things up by skipping more than one night at a time; the rest of the program won't mind.

3. Feeling ambitious? Try giving the moon some craters, or at least some shade. You might be able to do this with random black dots, as I used random white dots for the stars. But to give an effect anything like that of craters, the dots will have to be clustered. Not difficult, once you see how.

4. I have quite deliberately used black for the night sky and white for the moon and stars. If you want other colours, try working out the appropriate VDU19 commands. You will need to change logical colour 0 to actual colour 4 if you want a blue sky, and logical colour 1 to actual colour 3 if you want yellow moon and stars. I don't recommend it.

Tachistoscope

For most people reading is essentially a matter of pattern-recognition. We read a word by looking at the whole thing in one glance, rather than by looking at each letter in turn. Having taken in the overall pattern of letters, we refer to a table in our memory to see what word it corresponds most closely to. Whether we recognise the word depends on two things: how accurately we have registered the pattern for checking in the table; and how comprehensive the table is. It is generally the case that we don't register the pattern particularly accurately. We get a rough idea of what it was, and then rely on the table-checking area of our minds to find something which is close, if not exactly the same.

Did you notice that the word 'recognition' was misspelt in the previous paragraph? The chances are that you didn't. You saw a pattern of letters, sent it off for processing, and your mind replied 'Ah yes, that was the pattern for "recognition"'. Why bother you with the information that the pattern was a little wrong? That sort of thing happens all the time; and in fact the ability to recognise different patterns as instances of the same thing is one of the positive features of the mind which computer scientists are having great trouble trying to emulate with machines and programs.

Some people read phrases, and even sentences, in the same way: all at once, rather than word by word or letter by letter. Others go to the opposite extreme: each letter is read in its own right, and the mind is then asked to string the letters together to spell a word. If a pattern isn't recognised by the table-checker, we all resort to this letter-by-letter reading.

A tachistoscope (the stress is on the second syllable, and the 'ch' is pronounced 'k') is an instrument which can be used to test and improve reading habits. It flashes a word or phrase on a screen for a very short time, and you are then asked to say what it was. Those who read letter by letter are unlikely to answer

correctly: the display is too brief for the whole word or phrase to be absorbed. Those who read word by word or phrase by phrase are much more likely to take in the whole display.

If you are a letter reader, you can use the tachistoscope to help you become a word or phrase reader. Keep trying it at low speeds and you will soon become familiar with the 100 or so phrases in its vocabulary. Then you can start working on recognising a phrase rather than reading it. Improvement can be drastic.

If you already read a phrase in one gulp, you shouldn't have much difficulty getting the tachistoscope to its top speed. (But be careful: I have deliberately included several pairs of similar words or phrases, so you do still have to absorb the whole of each display.) You will probably also be able to take in the smaller numbers in one go. But try yourself on bigger numbers — up to seven digits. You might be surprised at how much harder it is to read a 7-digit number than a 20-letter phrase. The trouble is that numbers, while making perfect sense to read, can't be stored in a table for checking: there are too many of them. Besides, there's no need to keep mental tables of things whose meanings relate so closely to their form. So if we can't read numbers by recognition, we have to read them a digit at a time, or perhaps a group of digits at a time, which makes them much slower than words.

Still not convinced? Try yourself on 'jumbles of letters' — meaningless collections of letters, of about the size of a long word or a short phrase. These jumbles are almost totally unreadable (i.e. unrecognisable as patterns) and I've yet to meet anyone who can get the tachistoscope anywhere near maximum speed on them. If you are very good on the phrases and numbers options, the jumbles could prove a very useful exercise in pattern memorisation and recall.

How to use the program

The program is menu-driven. It will present you with a list of five options, and ask you to choose one by typing its number. Pressing ESCAPE at any time will return you to the menu. To finish, you must choose the 'finish' option from the menu.

You will probably find it easiest to start with words and phrases. Choose option 1, and the tachistoscope will be ready for

you. An arrow will appear in the upper left of the screen, and you will be told to press any key to trigger the display. Clearly, when you are being given only a few hundredths of a second to see a phrase, you must know exactly when it is going to appear; or, as in this case, you must be able to tell it when to appear. Watch the space to the right of the arrow and press a key. A word or phrase will appear for one second, and you will be asked what it was.

When typing what you thought you saw, you must be sure to get it exactly as it was. If the program displays 'THIS WON'T DO' and you type 'THIS WONT DO' you will be marked wrong. The program doesn't try to assess how close you were — it compares the two strings of characters, and marks you right only if they are exactly the same. A particular point worth noting in this respect is spaces. If you have too many or too few spaces in your guess, you will be marked wrong. An extra space on the end of your guess will be thoroughly confusing, producing a dialogue which looks like this:

What was it?

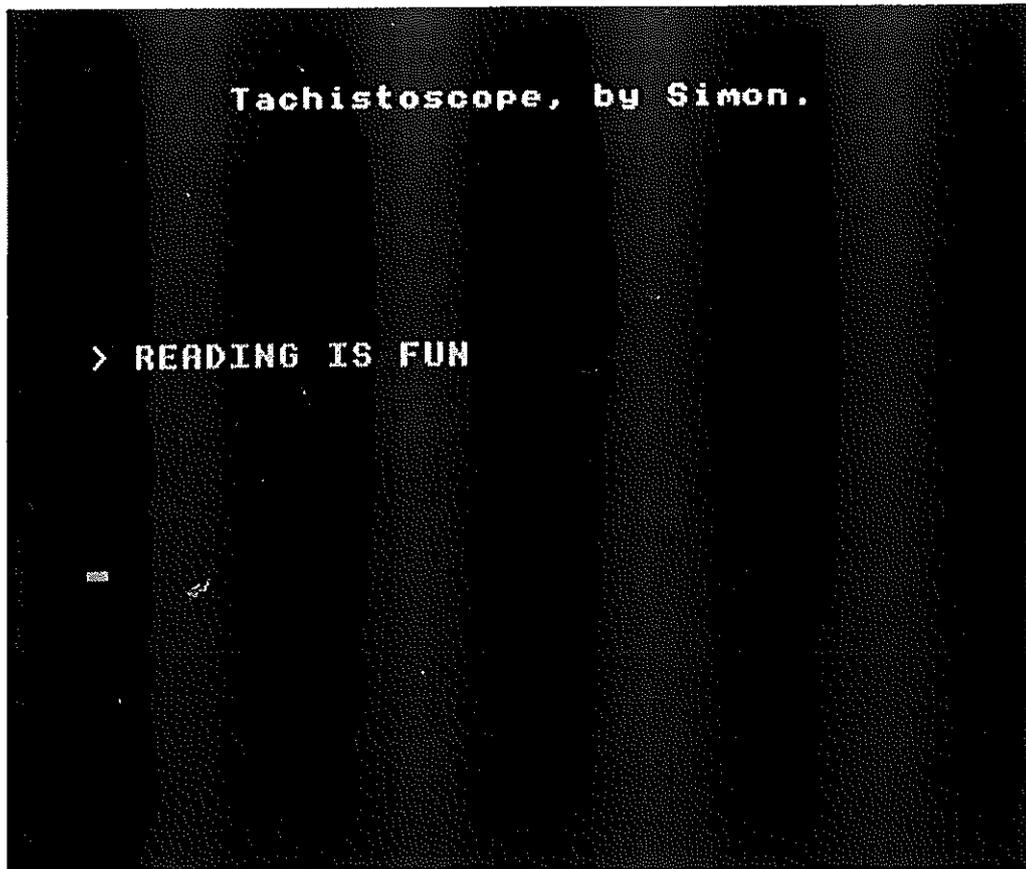
COMPUTERS ARE FUN

No. It was COMPUTERS ARE FUN.

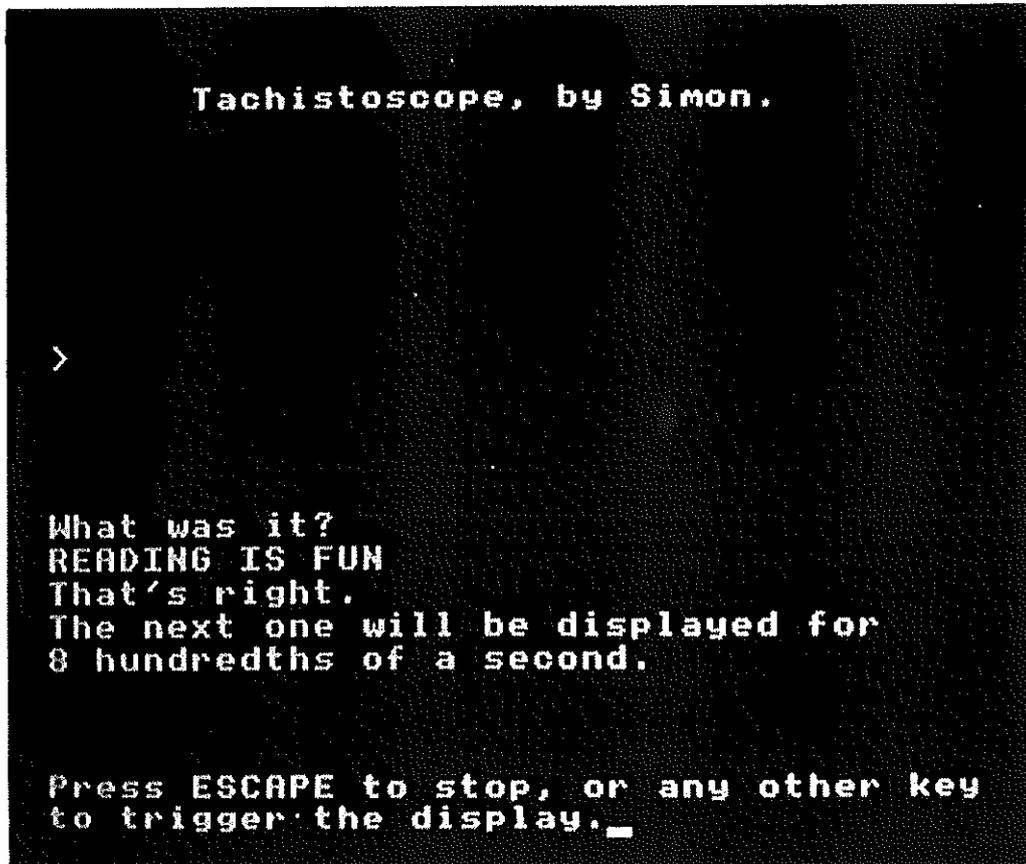
So be sure to put just one space between two consecutive words, and no spaces before or after your answer. You will of course have to press RETURN at the end of your answer, to send it to the program for checking.

If your answer was right, the next phrase will be displayed for two-thirds as long. If you were wrong, the next phrase will show for one-and-a-half times as long. There is a maximum display time of three seconds (nobody should have trouble reading phrases at that speed) and a minimum of one hundredth of a second (any faster and there isn't time to show the whole phrase on a normal television before it's wiped off again). When you want to finish a particular option, either to finish the program or to try another option, press ESCAPE to return to the menu.

If there are other people trying to read the phrases with you, don't feel too smug if they can't keep up with you — you're the only one who knows exactly when you're going to press the trigger!



A phrase is flashed on the screen . . .



. . . did you register it correctly?

Program listing

```

100 REM Tachistoscope, by Simon.
110 MODE6
120 PROCinitialise
130 PROCinstruct
140 REPEAT PROCmenu: UNTIL finish
150 MODE6: PRINT""Bye for now.""
160 END
170
180 DEF PROCinitialise
190 ON ERROR GOTO 140
200 PRINT""      Tachistoscope, by Simon."
210 VDU28,0,24,39,5: REM Set a text window.
220 maxphrases=100: DIM phrase$(maxphrases)
230 DIM recent(4)
240 FOR cycle=0 TO 4: recent(cycle)=0: NEXT
250 cycle=0: numphrases=0
260 REPEAT
270   numphrases=numphrases+1
280   READ phrase$(numphrases)
290 UNTIL phrase$(numphrases)="*" OR numphrases=maxphrases

300 IF numphrases=maxphrases THEN PRINT""There are too ma
ny phrases. Reset the""value of maxphrases in line 220."": P
ROCwait(2) ELSE numphrases=numphrases-1
310 ENDPROC
320
330 DEF PROCinstruct: LOCAL dummy
340 PRINT"This program can be of use in improving"
350 PRINT"your pattern-recognising speed."
360 PRINT""I'll display a word, phrase, or number"
370 PRINT"in the upper left of the screen, and"
380 PRINT"you try to recognise it."
390 PRINT""You will type what you think it was,"
400 PRINT"and I'll tell you whether you're right."
410 PRINT""If you are right, the next word,"
420 PRINT"phrase, or number will only be shown"
430 PRINT"for two-thirds as long (unless you've"
440 PRINT"reached the maximum speed)."
450 PRINT""If you are wrong, I'll increase the"
460 PRINT"display time by 50% (unless you've"
470 PRINT"reached the minimum speed I'll allow)."
480 PRINT""Press any key to continue.";
490 dummy=GET
500 ENDPROC
510
520 DEF PROCmenu
530 CLS: finish=FALSE: words=FALSE
540 smallnums=FALSE: bignums=FALSE: jumbles=FALSE

```

```

550 PRINT "What would you like to do now?"
560 PRINT "1> Test your speed on words and phrases"
570 PRINT "2> Test your speed on smallish numbers"
580 PRINT "3> Test your speed on bigger numbers"
590 PRINT "4> Test your speed on jumbles of letters"
600 PRINT "5> Finish"
610 PRINT "Please type the appropriate number. ";
620 choice=VAL(GET$)
630 IF choice=1 THEN words=TRUE ELSE IF choice=2 THEN smal
lnums=TRUE ELSE IF choice=3 THEN bignums=TRUE ELSE IF choice
=4 THEN jumbles=TRUE ELSE finish=TRUE: ENDPROC
640 PROCtest
650 ENDPROC
660
670 DEF PROCtest: LOCAL delay,dummy,string$,guess$,unit$
680 CLS: delay=100
690 REPEAT
700 PRINT TAB(0,5); ">"; TAB(0,18);
710 PRINT "Press ESCAPE to stop, or any other key"
720 PRINT "to trigger the display.": dummy=GET
730 string$=FNstr(choice): CLS
740 PRINT TAB(0,5); "> "; string$; TAB(0,10)
750 PROCwait(delay)
760 CLS: PRINT TAB(0,10) "What was it?"
770 INPUT "guess$"
780 IF guess$=string$ THEN delay=delay*2/3: PRINT FNrepl
y(guess$) ELSE delay=delay*3/2: PRINT "No. It was "; string$;
". "
790 IF delay<1 THEN delay=1 ELSE IF delay>300 THEN delay
=300
800 IF delay<2 THEN unit$=" hundredth" ELSE unit$=" hund
redths "
810 PRINT "The next one will be displayed for"
820 PRINT; INT(delay); unit$; " of a second.";
830 IF delay<2 THEN PRINT " (The maximum""speed.)" ELSE
IF delay=300 THEN PRINT " (And I'm not going any slower!)"
840 UNTIL FALSE
850 ENDPROC
860
870 DEF FNreply(x$)
880 IF x$="I MISSED IT" THEN ="But you didn't, did you?"
890 IF x$="WHAT WAS IT?" THEN ="I can't fool you, can I?"
900 IF x$="I DON'T KNOW" THEN ="Oh yes you do!"
910 ="That's right."
920
930 DEF FNstr(class): LOCAL num,I,good,jumbl$
940 REM Chooses the next string for display, according to
class. Class 1 is words and phrases, 2 is small
numbers, 3 is big numbers, 4 is jumbles.
950 ON class GOTO 1040,980,1010,1150
960

```

```

970 REM Small numbers.
980 =STR$(RND(9999))
990
1000 REM Big numbers.
1010 =STR$(RND(9999999))
1020
1030 REM Words and phrases.
1040 REPEAT good=TRUE
1050   num=RND(numphrases)
1060   FOR I=0 TO 4
1070     IF num=recent(I) THEN good=FALSE
1080   NEXT
1090 UNTIL good
1100 cycle=(cycle+1) MOD 5
1110 recent(cycle)=num
1120 =phrase$(num)
1130
1140 REM Jumbles of letters.
1150 jumbl$=FNrandletter+FNrandletter
1160 FOR I=1 TO RND(3): jumbl$=jumbl$+FNrandletter: NEXT
1170 IF RND(2)=1 THEN jumbl$=jumbl$+" ": num=5 ELSE num=3
1180 FOR I=1 TO RND(num): jumbl$=jumbl$+FNrandletter: NEXT
1190 =jumbl$
1200
1210 DEF FNrandletter=CHR$(64+RND(26))
1220
1230 DEF PROCwait(n)
1240 REM To wait for n HUNDREDTHS of a second.
1250 TIME=0: REPEAT UNTIL TIME>=n
1260 ENDPROC
1270
1280 DATA GREETINGS, GET FLICKED, THE BABY CRIED, THE BABY
CRAPPED, UP THE CREEK, OMNISCIENT, TO THE SHOPS, TO THE SHOR
E, PROGRAMMERS
1290 DATA I MISSED IT, I DON'T KNOW, WHAT WAS IT?, MY BRAIN
HURTS, HE GOES FIRST, HE GOES FAST, IF YOU CAN READ THIS YO
U'RE SITTING TOO CLOSE
1300 DATA WATCH IT!, WATCH OUT!, FLY UNITED, FLY UNTIED, STU
FF THE TURKEY, IN THE MOOD, IN THE NUDE, THIS IS NICE, THIS
IS NAUGHTY, AT THE TOP
1310 DATA COUP DE GRACE, CUT THE GRASS, MOI AUSSI, I'M AUST
RALIAN, SUIVEZ LA PISTE, FOLLOW THE INTOXICATED WOMAN
1320 DATA COMPUTERS ARE FUN, COMPUTERS ARE FINE, READING IS
FUN, READING IS GOOD, I LIKE READING, NUMBERS ARE HARDER, A
FIRM DATE, DONNA IS GORGEOUS
1330 DATA COME AND SEE ME, THROUGH THE NIGHT, THROUGH A GLA
SS, I THOUGHT SO, IN ALL WAYS, LOOK AT THIS, NOT A CHANCE, T
HEY ARE GREEN, THEY ARE GREAT
1340 DATA ON THE MOVE, IT'S YOUR ROUND, NEVER ON A SUNDAY,
OMNIPOTENT, ARTHUR, ALEXIS, A DREAM COME TRUE, TROUBADOUR, C
ONSCIENCE, GAS BILLS, DUCK BILLS

```

1350 DATA SIMPLE SIMON, FURTHER READING, SOFT ACORNS?, CURT
AINS, CRETINS, LAGER, LARGER, ROTUND, ROTUNDA, I LIKE TEACHE
R, SUPERCALIFRAGILISTICEXPIALIDOCIOUS

1360 DATA IT'S YOUR FAULT, IT'S YOUR TURN, GRATEFULLY ACKNO
WLEDGED, DEVIANT, DEFIANT, DEFOLIATED, ROOM FOR MORE HERE, P
ROGRAMS, PROGRAMMES, WORKS OF ART

1370 DATA SO MANY WORDS, SO LITTLE TIME, TELEVISION, SOCKET
TO ME, ATROCIOUS, INFERIOR QUALITY, GRATEFULLY ACCEPTED, TR
Y AGAIN, I CAN'T THINK OF ANY MORE, *

Comments on the program

1. More on text windows.

The program sets up a heading at the top of the screen, and then sets a new text window (using VDU28) to exclude the area with the heading in it. As we have already seen, this ensures that any further text display commands will leave the heading untouched. PRINT TAB(0,5) now means print at the beginning of the sixth line of the text window, not the sixth line on the screen; CLS means clear everything within the text window, not everything on the screen.

2. GET as a trigger.

There are three obvious ways of using a touched key as a trigger: GET, INKEY, and INPUT. The disadvantage of INPUT (as used in the telephone costs program) is that if it is to respond to a single key-press, the key must be RETURN. I find that I do use RETURN as a trigger in this program — my finger is still resting there after inputting my answer — but I see no reason to restrict users to this key.

The disadvantage of INKEY is the time limit it imposes. Users should be able to leave a program like this for as long as they like, and still expect to find it as they left it. There wouldn't seem to be much point in insisting that the display be triggered within the next so many seconds.

So we use GET. It will wait for ever (barring power failures), and will respond instantly to a touch on any key.

3. Multiple returns from functions.

This point was touched on in comment 6 to FNwhichday. When a

function is being evaluated, the first time a dangling '=' is encountered (i.e. an equals sign without a variable name to its left) the value of the expression to its right is made the value of the function, and control is returned to the statement which called the function; none of the following statements in the function is interpreted. It is thus possible to have many different returns from a function, each to be used in different circumstances. This program shows two different ways of arranging this.

FNreply is used to decide what reply to make if the user has answered correctly. It is quite definitely a frill to the program. There would be nothing wrong with replying 'That's right' every time; it's just a little more fun to have a response which is relevant to the phrase being tested. (The idea came to me when I saw somebody correctly type 'I MISSED IT', and watched the program reply 'That's right.' My pedantry insisted that I step in and change something.) The function makes three separate tests for specific phrases. If the first of these tests succeeds, the first reply is returned and the function ends. If it fails, control simply passes to the next line of the function, and the second test is made. Things proceed in this way either until one of the tests succeeds, or until we reach the last line of the function — a dangling '=' with no test — at which point the general reply is returned and the function ends.

FNstr is used to produce the next string of the appropriate class (note that even the numbers are treated as strings in this program). Depending on the class, the function has four completely different things to do, so an ON GOTO is used. The function then consists of four distinct blocks of code, each ending with its own dangling '='. As I remarked in the note to FNwhichday, this is the ideal situation for an ON GOTO, in that we don't have to use GOTOs at the end of each block to return control to some common point.

4. The cycle of recent phrases.

The array called 'recent' is used to ensure that a selected word or phrase is not one of the five most recently used. It is used only for the words and phrases; the other classes of string are quite random, and their duplication is so unlikely that it isn't worth testing for.

The commonest way of organising such an array is to keep the most recently used phrase in position 5, the next most recent in

position 4, and so on. This entails five assignments in each pass; something like

```
FOR I = 1 TO 4
  recent (I) = recent (I + 1)
NEXT
recent (5) = current
```

A little thought dispenses with three of the assignments. We use a counter which cycles around the elements of the array; on each pass we need only work out the correct value for this counter, and change the one array element to which it points.

The easiest way to have a value cycling round a given set of numbers (like the hours on a clock — one to twelve and then starting again at one) is to use MOD. But because MOD gives us a number from zero to some top value, we take advantage of the fact that an array DIM(n) actually has $n + 1$ elements, from the 0th to the nth. We give recent a dimension of 4, and use the 5 elements recent(0) to recent(4).

We often ignore the zeroth element of an array. This is not just a bad habit: look at the array 'phrase\$', whose main use is in providing a randomly chosen phrase. We use RND(n), which gives us a random number between 1 and n. Certainly we could use $\text{RND}(n) - 1$ to get a number between 0 and $n - 1$, but it's more cumbersome. So we just ignore the zeroth element, and pretend that the array goes from phrase\$(1) to phrase\$(maxphrases). Whether we use the zeroth element in an array will almost always be dictated by the sort of operations (like MOD and RND) we will be using on it.

5. A different PROCwait.

Many of the programs in this book use a PROCwait which waits a given number of seconds. Because the whole essence of this program involves timing in hundredths of a second, it makes sense to use a suitably modified procedure rather than play around with delays like 0.04. For one thing, integer delays are a lot easier to print nicely.

It should be pointed out that a stated delay of one hundredth of a second is in fact significantly longer than that. The procedure delays for one hundredth, but the actions of calling the procedure, interpreting the BASIC, clearing the screen, and so on add perhaps a few more hundredths. You should learn to

interpret the specified delays with a pinch of salt when they reach the region of a few hundredths of a second.

6. A question on the layout of the menu.

If you have copied the program correctly (or if you have it on tape) you will notice that option 5 on the menu is printed with only one apostrophe, while all of the other options are printed with two. But the options are evenly spaced when the program runs. Why?

7. How the jumble works.

Some moderation must be exercised in the formation of a jumble of letters for presentation to the user. A 'word' of 20 random letters would be impossible for all but the gifted few. I have settled on a minimum length of four letters. If the jumble is split by a space, the first part is three to five letters long and the second part is one to five letters long. If there is no space, the whole jumble is from four to eight letters in length. Given that information, you should find it fairly easy to understand the relevant segment of the function.

8. VAL(GET\$).

In line 620 the expression VAL(GET\$) is used to interpret a value typed in by the user. What does this expression do?

GET returns the number of a pressed key. GET\$ returns the string containing the character of a pressed key. VAL produces a number from a string. So isn't VAL(GET\$) the same as GET? No, because VAL and GET produce different types of number. GET produces the ASCII number of the character, while VAL assumes that a string consists of digits, and produces the number for which those digits are the written form. So if, say, the '3' key is pressed, GET will return a value of 51, GET\$ will produce a value of "3" (the string, not the number), and VAL(GET\$) will produce a value of 3.

Wouldn't it be easier just to INPUT the number? Yes, but then the user would have to remember to press RETURN after typing the number — something which isn't really necessary in this context.

9. Offensive words and phrases.

Not wanting to offend anyone, I have deliberately omitted certain common phrases which are in my own version of the program.

That works to an extent, but I disclaim all responsibility if anyone is offended by a word or phrase produced as a random jumble of letters.

Suggested amendments to the program

1. If you don't like the words and phrases I have used, replace them with your own. If you do like them, add some more of your own. Don't forget the advisability of having some similar pairs of phrases, to ensure a certain accuracy of recognition.
2. As a fairly major project, turn the program into a spelling checker. You will need to replace the data with half a dozen variants (one of them the right one) of various words you have difficulty with, set the display to a fairly fast constant rate, and amend the program so that it asks 'Did I spell that correctly?' Of course it will then have to make the necessary checks on the answer. It might also keep a score of right and wrong answers; and perhaps even ask for the correct spelling of an incorrectly displayed word.
3. If you want the jumble of letters to reflect English a little more closely, change FNrandletter so that it has different chances of producing different letters. It might, for instance, produce E 20% of the time, A 15%, I 12%, and various other probabilities ranging to 1% or less for Z, X, and Q. If you're not sure how to do this, look at FNhowmany in the GROAN program.
4. If you always run this program in the same circumstances, i.e. without having used the random number generator since turning the computer on, you will find that it always comes up with the same sequence of words and phrases. To avoid this, you must 'randomise' the random number generator; one way of doing this is used in the Artillery program, and mentioned in comment 1 to that program.