Shell/ITMA

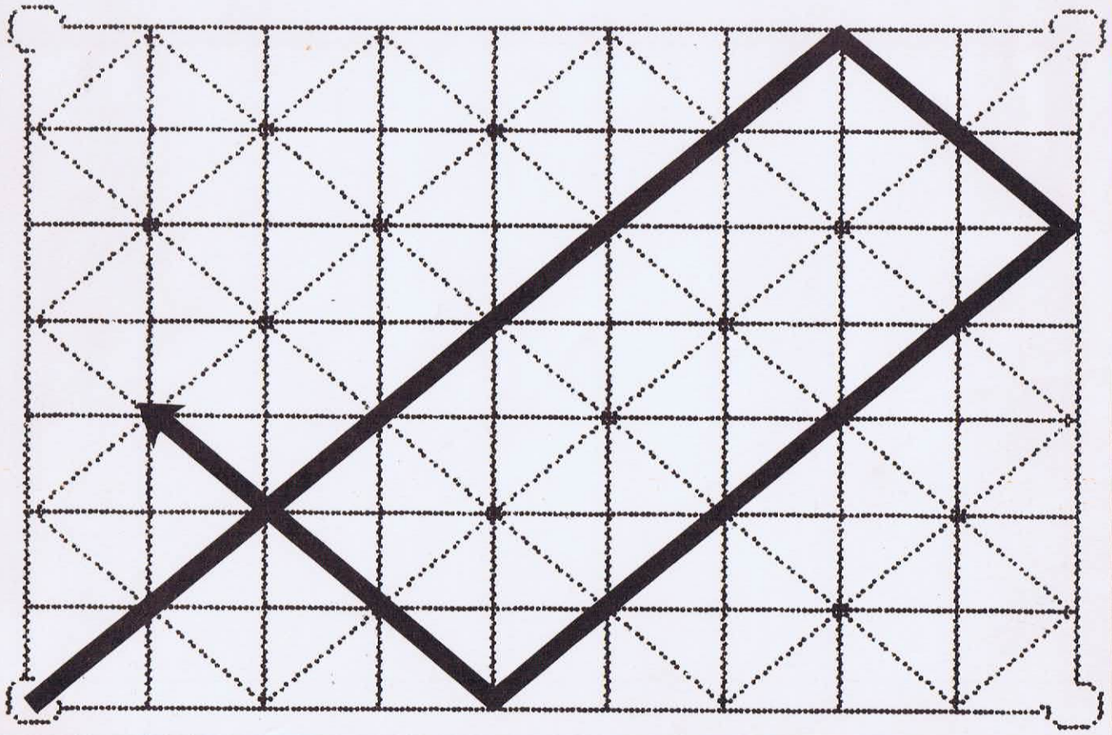
Micros in the Mathematics Classroom

14 teaching units

Longman Micro Software

CEDAR

University of London
Institute of Education
SOFTWARE LIBRARY
Part of SLIN: P163/1



Micros in the Mathematics Classroom

Contents

LONGMAN GROUP LIMITED
Longman House,
Burnt Mill, Harlow, Essex, UK.

© Shell Centre for Mathematical Education, Nottingham University, 1982.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the copyright owner.

The contents of the cassette and disc are copyright, but one back-up copy of each program on the cassette or disc may be made as a precaution against accidental damage to the original.

First published 1982.

Printed in Great Britain by Longman Group Resources Unit, York.


```
10 REM ***VERSION MMC 1.1***
20 REM************************************
30 REM            NUBASE
40 REM************************************
50 INPUT A, B, C, D, E, X, Y
60 LET T=A*X^4+B*X^3+C*X^2+D*X^1+E*X^0
70 T=INT(T+0.1)
80 PRINT T - Y*INT(T/Y)
90 IF T = 0 THEN 120
100 LET T = INT(T/Y)
110 GOTO 80
120 PRINT "BASE";X;"TO BASE";Y;"WITH DIGITS REVERSED"
130 END
140 REM************************************
```

Figure 12: Program NUBASE

```
run
? 0,0,3,2,0,  4,  5
 1
 1
 2
 0
BASE 4 TO BASE 5 WITH DIGITS REVERSED

Ready:
```

Figure 13: Output from NUBASE (for T = 320_4 to base 5)

```
run
? 90
 2
 3
 5
```

Figure 10: Output from FACTORS (for M=90)

Long division

Problem: divide one number by another, and express the answer to as many decimal places as required.

Purpose: To overcome the computer's limitation of seven significant figures, and to encourage careful thinking about the long-division algorithm.

```
5 REM ***VERSION MMC 1.1***
10 REM*********************************
20 REM            DIVALG
30 REM*********************************
40 INPUT N,D
50 LET Q=INT(N/D)
60 PRINT Q;".";
70 LET N=(N-D*Q)*10
90 PRINT Q;
100 GOTO 70
110 END
120 REM*********************************
```

Figure 11: Program DIVALG

Base conversion

Problem: write a program to convert a numeral in base X, to its equivalent in base Y, as a general conversion.

Purpose: to practise the design of algorithms and to reinforce the properties of a numeration system. (This generally follows work on converting from some base X to base 10, and vice versa.)

Program: two possible approaches to this problem are as follows:

1. We can convert from base X to base 10 and then from 10 to X.
2. We can utilize the algorithm which enables one to divide the number in base X by the base X equivalent of Y and consider the successive remainders in the reverse order (this is the common means of converting from base 10 to some other base). The program given below utilizes the first approach to the problem. (NOTE: this program considers a base X numeral of up to five places, that is, $ABCDE_x$: this is arbitrary.)

Introduction

This collection of programs provides interested mathematics teachers with material that will enable them to explore the range of possibilities and potential roles of the microcomputer as an aid in the teaching and learning of mathematics. The programs, which originate from a variety of sources, designers and programmers, are deliberately varied in their approach. In some cases there are detailed teaching notes to suggest ways in which the programs may be used; you will undoubtedly find further possibilities yourself. In others there is no such information; you may explore a new and undocumented program (and there are only too many of these about) to discover for yourself something of its potential. Teachers have found each of the units included in this pack useful as an aid to their teaching.

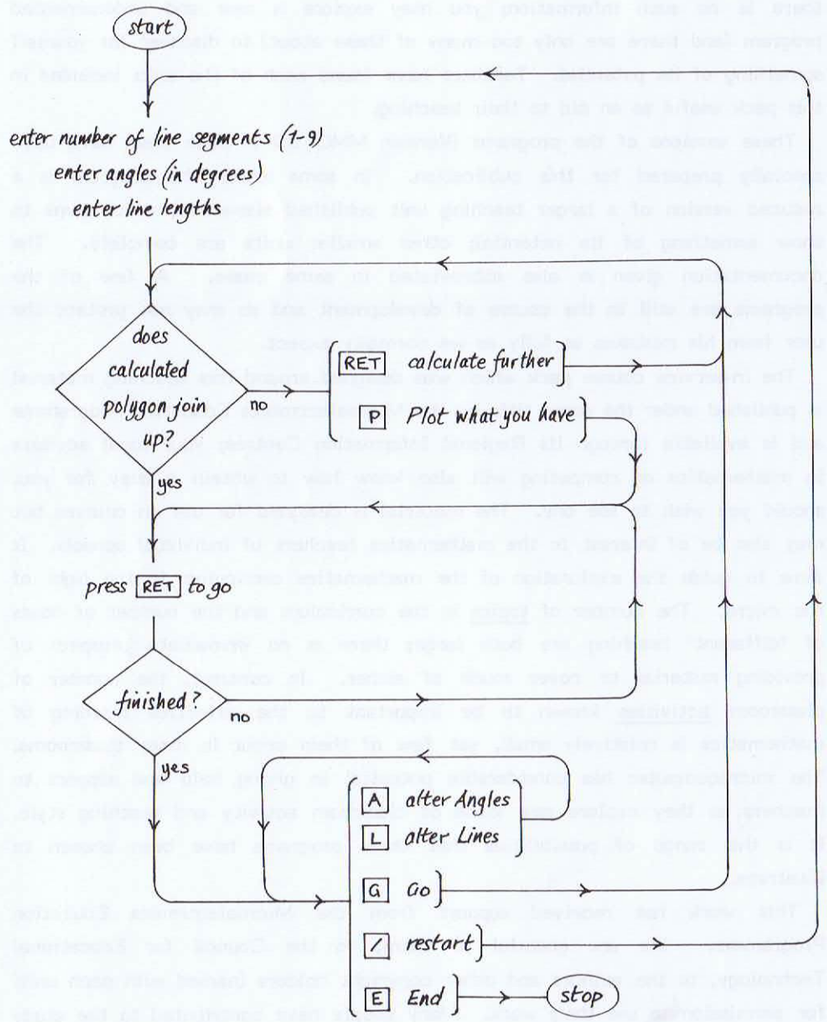
These versions of the programs (Version MMC 1.0 in each case) have been specially prepared for this publication. In some cases the program is a reduced version of a larger teaching unit published elsewhere, which aims to show something of its potential; other smaller units are complete. The documentation given is also abbreviated in some cases. A few of the programs are still in the course of development and so may not protect the user from his mistakes as fully as we normally expect.

The in-service course pack which was designed around this teaching material is published under the same title by the Microelectronics Education Programme and is available through its Regional Information Centres; your local advisers in mathematics or computing will also know how to obtain a copy for you, should you wish to see one. The material is designed for use on courses but may also be of interest to the mathematics teachers of individual schools. It aims to guide the exploration of the mathematics curriculum in the light of the micro. The number of topics in the curriculum and the number of hours of 'different' teaching are both large; there is no immediate prospect of providing material to cover much of either. In contrast, the number of classroom activities known to be important to the effective learning of mathematics is relatively small, yet few of them occur in many classrooms. The microcomputer has considerable potential in giving help and support to teachers as they explore new kinds of classroom activity and teaching style. It is this range of possibilities that these programs have been chosen to illustrate.

This work has received support from the Microelectronics Education Programme. We are grateful to them, to the Council for Educational Technology, to the authors and other copyright holders (named with each unit) for permission to use their work. Many people have contributed to the study of the mathematics curriculum that guided our selections; in particular we thank Mike Aston, Peter Holmes, David Johnson, David Sturgess, Donovan Tagg and the members, in Nottingham and in Plymouth, of the ITMA Collaboration. Andrew Nash and Doreen Johns with the assistance of Susan Botterill prepared the manuscript for publication.

Hugh Burkhardt and Rosemary Fraser May 1982

Drivechart for ASPIR



Note that when entering an angle, you may press RET *without having typed a number: the program will then use the value of the previous angle; in the case of the first angle, it will use 90. Line lengths work similarly, except that if no value is given, the first line length is set to 1. The program automatically scales polygons to use the full area of the screen, so the scale used is not constant.*

the points are distinct and whether the slope is 0 (horizontal) or infinite (vertical).

```
10 REM ***VERSION MMC 1.1***
20 REM*********************************
30 REM                SLOPE
40 REM*********************************
50 READ X1, Y1, X2, Y2
60 PRINT (Y2 -Y1) / (X2 - X1)
70 GOTO 50
80 DATA 0,0,1,2,3,5,11,9,7,12,6,72,-3,4,1,-2
90 END
100 REM*********************************
```

Figure 7: Program SLOPE

```
run
 2
 .5
-60
-1.5

Out of data at line 50
Ready:
```

Figure 8: Output from SLOPE

Prime factors

Problem: find the prime factors of a positive integer. (The program can easily be extended to prime factorisation.)

Purpose: this algorithm involves the application of two number theory concepts - finding divisors of a number, and testing to see if a number (an exact divisor, in this situation) is prime.

```
10 REM ***VERSION MMC 1.1***
20 REM*********************************
30 REM                FACTORS
40 REM*********************************
50 INPUT M
60 IF M > 2 THEN 90
70 PRINT "M=";M
80 GOTO 50
90 FOR N = 2 TO M
100 IF (M/N) - INT(M/N) = 0 THEN 130
110 NEXT N
120 GOTO 50
130 IF N = 2 THEN 170
140 FOR T = 2 TO (N - 1)
150 IF (N/T) - INT (N/T) = 0 THEN 110
160 NEXT T
170 PRINT N
180 GOTO 110
190 END
200 REM*********************************
```

Figure 9: Program FACTORS

```
 10 REM ***VERSION MMC 1.1***
 20 REM*********************************
 30 REM       SUMDIS
 40 REM*********************************
 50 DIM Q(100),P(100),R(100)
 60 READ R,N
 70 FOR I=0 TO R
 80 READ P(I)
 90 LET Q(I)=P(I)
100 NEXTI
110 FOR K=2 TO N
120 FOR J=0 TO R*K
130 LET X=0
140 FOR I=0 TO J
150 IF I)R THEN 170
160 LET X=X+P(I)*Q(J-I)
170 NEXT I
180 LET R(J)=X
190 NEXT J
200 FOR I=0 TO R*K
210 LET Q(I)=R(I)
220 NEXT I
230 NEXT K
240 FOR I=0 TO N*R
250 IF Q(I)=0 THEN 280
260 PRINT I;
270 PRINT TAB(150*Q(I));"*"
280 NEXT I
290 DATA 6
300 DATA 2
310 DATA 0,.16667,.16667,.16667
320 DATA .16667,.16667,.16667
330 END
340 REM*****************************

run
 2  *
 3      *
 4          *
 5              *
 6                  *
 7                      *
 8                  *
 9              *
10          *
11      *
12  *
```

Figure 6: Program DISTRB

Learning through programming

The four problems posed here challenge the pupil to write algorithms for common mathematical processes.

Gradient of a straight line

Problem: Write an algorithm to give the slope of a line through any two given points.

Purpose: To use an easy programming setting to emphasize the importance of the general representation of two points in a coordinate system. The program SLOPE also forces the student to use the definition of slope. Note that a more complete algorithm would also test the X and Y values to make sure

ASPIR

Design: Gordon Haigh, Sheldon Heath School, Birmingham
Program: Gordon Haigh and Richard Phillips
Source: Gordon Haigh

Program copyright © Gordon Haigh, 1982

ASPIR runs on a Research Machines 380Z in 32K RAM with high-resolution graphics and BASIC Version 5. It may be used with disc- or cassette-based systems.

Summary of ASPIR

ASPIR is an investigative program about polygons. Although it is useful in teaching the geometry of regular polygons, its real value becomes apparent when investigating the whole range: concave polygons, irregular polygons, polygons with crossing lines, and combinations of these. The user draws an image on the screen by means of a sequence of instructions given in terms of angles and lines. This sequence may be repeated by the program until the pattern joins up. Pupils may be encouraged to see the relationship between such instructions and the polygon which results, and so learn to predict the one from the other.

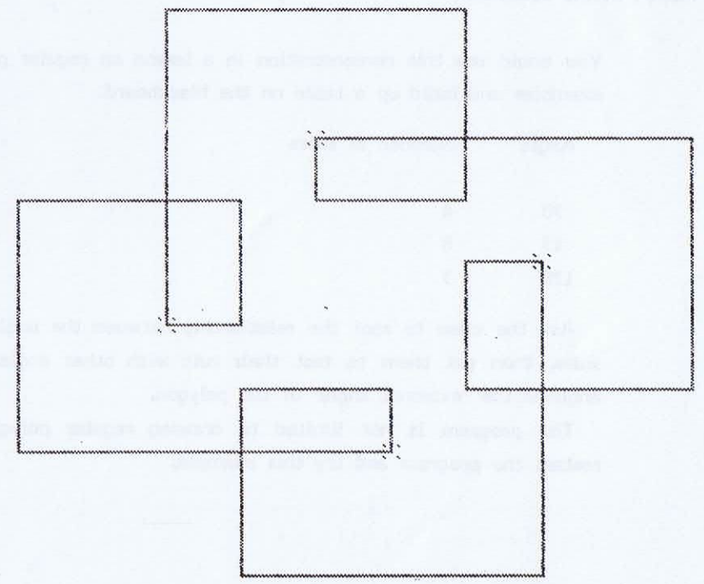


Figure 1

Running ASPIR

Imagine that you are a fly standing on the bottom of the screen facing towards its right-hand edge.

You have been given some instructions which tell you how to move across the screen. These tell you the angles through which you should turn and the number of paces you should walk. Once you have executed these instructions, you repeat them again and again until you get back to your starting point.

Suppose that your instructions are 'Turn anticlockwise through 45 degrees, go forward 5 paces.' If you repeated this eight times, you would return to your original point having traced out a regular octagon.

Now do the same with the program.

Computer: Number of line segments (1-9)?
You: [1] [RET]
Computer: Enter angle 1
You: [4] [5] [RET] (Angles are always in degrees and anticlockwise.)
Computer: Enter line length 1
You: [5] [RET]

After a few moments, the computer will show your first move on the screen and will pause. Press [RET] to display the next move, and go on pressing it until your pattern joins up.

Now press [A], to alter the angle, and type [3] [6] [RET]. Then press [G], to go; again press [RET] repeatedly until the pattern joins up. Choose a different angle and repeat this procedure.

ASPIR in the classroom

You could use this demonstration in a lesson on regular polygons. Try a few examples and build up a table on the blackboard.

Angle	Number of sides
90	4
45	8
120	3

Ask the class to spot the relationship between the angle and the number of sides, then get them to test their rule with other angles. Explain that the angle is the 'external angle' of the polygon.

The program is not limited to drawing regular polygons. Press [/] to restart the program and try this example.


```
10 REM ***VERSION MMC 1.1***
20 REM*********************************
30 REM          COIN1
40 REM*********************************
50 RANDOMIZE
60 FOR I=1 TO 20
70 IF RND(1) < 0.5 THEN 100
80 PRINT"T";
90 GOTO110
100 PRINT"H";
110 NEXT I
120 END
130 REM*********************************

run
THTTTHHHTTTTHTHHTHTT
Ready:
```

Figure 4: COINA

```
10 REM ***VERSION MMC 1.1***
20 REM*********************************
30 REM        COIN2
40 REM*********************************
50 RANDOMIZE
60 LET S=0
70 FOR I=1 TO 1000
80 IF RND(1) < 0.5 THEN 100
90 LET S=S+1
100 NEXT I
110 PRINT"Proportion of heads was";S/1000
120 END
130 *********************************

run
Proportion of heads was .504

Ready:
```

Figure 5: Program COINB

Probability

Probability theory facilitates the calculation of the probabilities of complex events in terms of their elementary constituents. The computer can handle and display the often messy algebra involved, and can compare the results with data from stochastic simulations based on the same underlying probabilities.

The program DISTRB is a probability calculation for the expected distribution of the sum of the results of n experiments, each of whose outcomes is one of a finite set of events whose members have known probabilities - for example, the sum of the results of rolling n dice. It provides a nice algorithmic illustration, though the structure would be more clearly visible with a computer that allowed indention in the BASIC code. It could well be compared with a simulation of the same exercise.

Solution of equations by iteration

The program HALVINT finds an approximation to a solution of f(x) = 0 between a and b, where f(a) and f(b) have opposite signs. You might like to try the program for speed and accuracy using a simple linear equation, a quadratic, an equation of order 5 which could not be done algebraically, and finally perhaps a transcendental one like $e^x = 3x$.

```
10 REM ***VERSION MMC 1.1***
20 REM*********************************
30 REM       HALVINT
40 REM*********************************
50 PRINT CHR$(12);"This program finds a root of F(X)=0"
60 PRINT"between A and B if F(A) and F(B) have"
70 PRINT"opposite signs and F(X) is continuous"
80 PRINT"in (A,B)."
90 PRINT:PRINT:PRINT:PRINT
100 PRINT"The error is to be less than H."
110 PRINT"Input your function at line 170."
120 PRINT"Then type 'RUN 150'."
130 PRINT:PRINT:PRINT:PRINT
140 END
150 PRINT:PRINT"Input A,B,H."
160 INPUT A,B,H
170 DEF FNC(X)=EXP(X)-3*X
180 IF FNC(A)*FNC(B)>0 THEN PRINT"Unsuitable A,B.":GOTO 150
190 C=(A+B)/2
200 IF FNC(C)=0 THEN 240
210 IF FNC(C)*FNC(A)>0 THEN A=C
220 IF FNC(C)*FNC(A)<0 THEN B=C
230 IF ABS(B-A)>H THEN 180
240 ?"The root is ";C
250 END
260 REM*********************************
```

Figure 3: Program HALVINT

Statistics

Simulations

These delightfully simple programs COINA and COINB can be used to stimulate discussion of a range of obvious statistical points; they are obviously at a level which could be managed by the student - or even a brave teacher in front of his class.

Number of line segments?	5	RET
Enter angle 1?	90	RET
Enter angle 2?	90	RET
Enter angle 3?	90	RET
Enter angle 4?	90	RET
Enter angle 5?	90	RET
Enter line length 1?	1	RET
Enter line length 2?	2	RET
Enter line length 3?	3	RET
Enter line length 4?	4	RET
Enter line length 5?	5	RET

You have defined the five-line pattern shown in Figure 1. Press RET, and continue to press RET until the pattern joins up to make a polygon.

There are many ways this program could be used in a lesson. For example, it could be used to teach pupils to visualize geometric patterns. Ask the class to work out in their heads rules for drawing a rectangle, a cross or a star shape. Then use the program to test the rules they have produced.

It is also useful to try this exercise in reverse. Give them a rule such as:

2 line segments
Angles -60, 120
Lines 1, 1

and ask them to visualize the shape.

Ideas and questions about ASPIR

1. When a shape joins up, what can you say about the total degrees turned through? Can you make a shape that will travel across the screen and never join up?
2. If you want the class to visualize geometric shapes, it is a good idea to ask them to close their eyes; give them plenty of time to build up a visual image.
3. Compare the approach in this program with the LOGO turtle geometry described in Seymour Papert's book, Mindstorms (Harveston Press, Brighton, 1980).

AN40

Integration

Numerical integration provides one of the few numerical topics in traditional syllabuses. The program SIMPSON should help the development of intuition about Simpson's rule, and about the underpinnings of its accuracy and stability.

It may be best to start with an integral which can be checked, e.g.

$\int_0^1 x^2 dx$, with N equal to 1, 5, 10 and 50. This can be repeated with an example like $\int_0^{\frac{1}{2}} 6dx/\sqrt{(1-x^2)}$ which gives π . One can then apply the method to an integral which can only be done numerically, e.g.

$\int_0^a e^{-x^2} dx$ $\qquad\qquad$ $a = 0.1, 0.2, \ldots$

Pupils can suggest the number of steps to be tried on each occasion.

```
10 REM ***VERSION MMC 1.1***
20 REM*********************************
30 REM          SIMPSON
40 REM*********************************
50 REM N.B. The FNC(X) given needs to
60 REM       be handled with care!
70 REM*********************************
80 PRINT CHR$(12);"This program calculates the integral"
90 PRINT"of FNC(X) from A to B using N steps"
100 PRINT"(2N+1 ordinates),"
110 PRINT:PRINT:PRINT"Input your chosen function at"
120 PRINT"line 160 and then type 'RUN 150'."
130 PRINT:PRINT:PRINT:PRINT
140 END
150 PRINT"Input A,B,N":INPUT A,B,N
160 DEF FNC(X)=6/(SQR(1-X*X))
170 S=0
180 H=(B-A)/N
190 FOR J=0 TO N-1
200 S=S+FNC(A+J*H)+2*FNC(A+(J+0.5)*H)
210 NEXT J
220 S=2*S-FNC(A)+FNC(B)
230 S=H*S/6
240 PRINT:PRINT:PRINT"The value of the integral is ";S
250 REM*********************************
```

Figure 2: Program SIMPSON

help the development of strategic skills via the use of educational games.

Programming by pupils

Writing simple programs is accessible to pupils of a wide range of ability and a number of benefits can accrue:

Programming is one of the few areas in the school curriculum where pupils are required to master a task completely.
Programming can provide a semi-abstract bridge to formal reasoning.
Constructing algorithms and making them work is an important mathematical activity which can provide pupils with valuable insights.

Using programs

Graph plotting

The computer is well suited to the display of graphical material. Sophisticated graph-plotting programs such as FGP offer a wide range of facilities and provide a good deal of help to the user in terms of an appropriate choice of scales etc. The code, however, is likely to be opaque to the inexperienced programmer.

SCATTER is a very simple point-plotting program which pupils should be able to understand and to extend.

```
10 REM ***VERSION MMC 1.1***
20 REM*********************************
30 REM          SCATTER
40 REM*********************************
50 PUT12
60 PRINT"Press RETURN to start.":PRINT:PRINT
70 PRINT"To finish input x=99, y=99."
80 PRINT:PRINT:PRINT:PRINT
90 A=GET()
100 PUT12
110 GRAPH
120 PLOT 79,8,1:LINE 12,8:LINE 12,58
130 Y$="y values"
140 FOR I=1 TO 8
150 PLOT 0,48-3*I,MID$(Y$,I,1):NEXT I
160 PLOT 40,0,"x values"
170 PRINT"Input co-ordinates x,y."
180 INPUT X,Y
190 IF (X<1 OR X>15) THEN GOTO 220
200 IF (Y<1 OR Y>15) THEN GOTO 220
210 GOTO 250
220 IF X=99 THEN GOTO 270
230 PRINT"Co-ordinate out of range."
240 GOTO 170
250 PLOT X*3+12,Y*3+8,2
260 GOTO 170
270 TEXT
280 END
290 REM*********************************
```

Figure 1: Program SCATTER

AUTOFRAC

Design: Jon Coupland
Program: Jon Coupland
Source: ITMA, Plymouth

Program copyright © Jon Coupland, 1982

AUTOFRAC runs on a Research Machines 380Z in 32K RAM with BASIC Version 5. It may be used with a disc- or cassette-based system. It does not require high-resolution graphics.

Summary of AUTOFRAC

AUTOFRAC provides a continuous film-like display of equivalent fractions, in the form

$$\frac{8}{6} = \frac{4}{?}$$

Having been initiated, the program runs indefinitely without intervention from the user. Each new pair of equivalent fractions, chosen at random, is displayed in its incomplete form for a specified time, after which the missing number is shown. The program then moves on to the next pair.

The user may specify the level of difficulty required, and the delay time (in seconds) for which the unsolved problem is displayed.

Running AUTOFRAC

AUTOFRAC is very simple to set up. The program is loaded from the disc or cassette and then run.

At the outset, you must choose the level of difficulty, in the range 1-10: this number defines the maximum number that can occur in the fractions. Secondly, you must choose the length (in seconds) of the delay before the answer appears; this may be from 0 to 20 seconds.

Typical screen displays are shown below. Each item of input must be completed by pressing the RET key.

```
****************************************

   CONTINUOUS DISPLAY OF FRACTIONS
          by J.COUPLAND

LEVEL OF DIFFICULTY (1-10)? 5

DELAY BETWEEN ANSWER (SEC)? 4
```

Figure 1

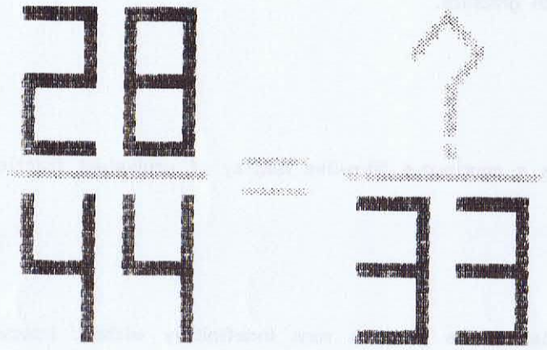


Figure 2

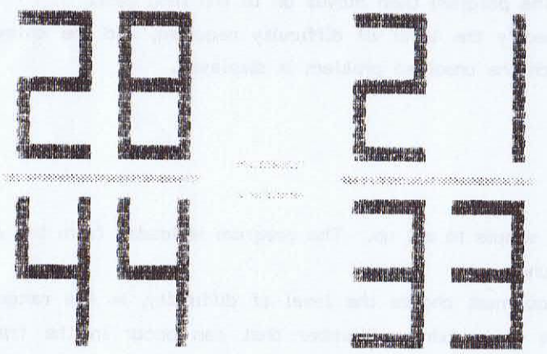


Figure 3

AUTOFRAC runs continuously until you 'break in' to the program. To do this, hold down the CTRL key and press Z. This stops the program and returns to the 'Ready' state, when the program can be run again.

Notes on AUTOFRAC

The levels of difficulty are inclusive, and are dictated simply by the range of numbers which can appear in the fractions. Given a level of difficulty 'D', numbers used are in the range of 1 to 9.9 x D. This means that even if a level of difficulty of 10 is chosen, problems of difficulty 1 will be

In this section, we give a few simple programs. You will undoubtedly want to amend them before use. A copy of each program listed here exists on disc, so there is no need for you to key the programs into the computer.

The main purpose of this section is to illustrate the role of the computer as a teaching aid for the individual. The user can benefit both from using programs and from the process of programming itself.

General warning

Most of the programs earlier in this handbook have been carefully developed. They protect novices against a number of errors which they might make. This is not the case here.

The programs here offer no protection against user error. The nature of the error must be deduced from the computer's somewhat cryptic run-time comments (such as 'Undefined statement at line X' or 'Can't divide by zero at line Y').

This lack of protection can, in itself, be instructive: it may demonstrate the need to safeguard naive users, and may cast light upon the underlying mathematics. In any case, these simple programs are typical of many programs written for specific use by individual users.

We are grateful for contributions from David Johnson and Donovan Tagg, and to Jim Ridgway for help in compiling this section.

Using programs

A large number of mathematical topics are appropriate to the development of programs by and for individual users. These topics include:

graph plotting;
numerical integration;
solving simultaneous equations;
matrix manipulation;
solving equations via iteration;
differential equations;
displaying data;
simulations and games.

Thus a range of mathematical and statistical topics can be illustrated via programs; and many of these can easily be written by pupils.

Individual learning

Computers used by individual pupils serve to:

motivate drill and practice;
encourage problem solving activities;

This program was written for primary schools, but could easily be used in secondary schools in remedial work. Its operation is self-explanatory.

Ideas and questions about TABCAR

1. The problems are generated using a random number generator; all integers between zero and twelve occur equally often. Watch the program working for several minutes, then compare the advantages and disadvantages of using a random number generator in a program with those of a fixed set of arithmetic problems stored in the program.
2. It is usually true that flexible programs are difficult to use, whereas inflexible programs are easy to use. TABCAR is easy to use and relatively inflexible - for example, you have no choice about which multiplication tables are practised. Consider what features you would want in a more flexible multiplication practice program, and how this would affect the ease in using it.
3. How motivating is a program of this sort?
4. Compare it to the CWORDS program.
5. Would you use the 'racing cars' to motivate other drill and practice exercises?
6. How would you organize the classroom use of these programs?

encountered. This is a very crude method of deciding how difficult problems will be to solve. For example, 25/50 = 1/? should prove straightforward, though the size of numbers here means this problem is possibly of difficulty level 6.

The delay before the answer is given can in practice be chosen only in the range 0 to 20 seconds. A delay of 20 seconds will probably result in total boredom; a delay of 0 seconds is probably best used only when it is the working of the computer, and not the nature of fractions, which is being demonstrated to pupils: they can then see that the computer is not really as slow as otherwise appears.

In considering the working of the program itself, it will be found that (not surprisingly) AUTOFRAC cheats in working out its equivalent fractions: it knows the missing number from the beginning each time.

The program code appears long and tricky, but the majority of it is involved in producing the large digits on the screen. The generation, at random, of the actual problems will be found to be quite short.

Possible changes to AUTOFRAC

There are two main areas in which changes should be considered.

1. The user might be allowed to stop the program at any stage to permit discussion of a particular problem. Such a change can be implemented easily as follows: add these two lines to the program and save the new version.

```
3295 L=GET(0) : IF L=0 THEN 3300
3297 PRINT "WAITING - Press any key to continue": L=GET( )
```

   This addition will cause the program to wait after a problem has been set if any key has been pressed, and to continue only when any key is pressed.

   The author considers such a change undesirable. Its effects are discussed below.

2. The coding which decides the level of difficulty could be improved so that the number chosen does actually determine the level of difficulty encountered in solving the fractions (rather than merely the range of numbers that can appear). Technically this poses some problems as there is little general agreement about which aspects of the solution of equivalent fractions are found to be most difficult.

AUTOFRAC in the classroom

In its original form, AUTOFRAC waited after setting each problem and displayed the answer only when a key had been pressed. This meant that the

user (teacher or pupil) was in control of the program: the user decided on a suitable time to move to the next problem. This detracted from one aspect of the computer's potential - its role as an additional member of the group, seemingly independent and actually in control of the pace of the activity. It was found that the pupils (and teacher) much preferred the computer to be in charge so that the teacher was in the same position as the pupils, struggling to solve a problem before the computer relentlessly presented the solution and moved on to the next.

A constraint on the operation of AUTOFRAC during its development was having only one computer in a suite of mathematics classrooms. The computer itself was difficult to move, and sometimes more than one group wished to use it at the same time. The solution was to wire up rooms with television screens which could all be driven simultaneously from the same computer. Naturally, programs broadcast on this network could not allow interaction. This constraint on software is highly significant; parameters for the program must all be set at the beginning.

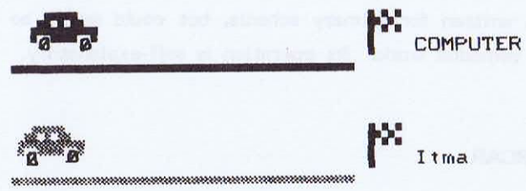
AUTOFRAC has been used in formal lesson time and left running during breaks, lunchtimes and registration in a somewhat subliminal mode. In lessons it provides a rare opportunity for children to display their prowess at mental arithmetic. It also demonstrates dramatically the surprising range of techniques required in solving equivalent fractions efficiently. It has been used with a purely verbal response, with children competing to volunteer the correct answer. If a suitably long delay is set, the children can write down their answers for consideration later. In this case the teacher must do likewise: the program has no facility to play back a set of problems.

The program may appear repetitive and boring, but children seem very attracted to the novelty of the task. Perhaps this is a tribute to how interesting and stimulating mathematics has become.

AUTOFRAC is not intended to support a full lesson's activity, but a few ten-minute bursts through the school year may prove useful. Certainly, it has been amazing to leave the program running in the lunch-hour and to watch children glued to the windows of the classroom, desperately trying to find the correct solutions!

Ideas and questions about AUTOFRAC

1. Does the program promote some useful mathematical activity?
2. The program relies on mental arithmetic. Is it therefore a help in identifying pupils with particular abilities or problems?
3. Can the program be used well with any child responding, or are the best results obtained if the class is split into teams?



QUESTION 2

What is 4 x 4 ?

Figure 3

As soon as one car reaches the winning post, the program stops.

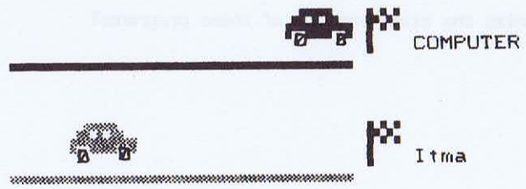


Figure 4

To interrupt the program, hold down the CTRL key and type Z.

TABCAR in the classroom

This simple multiplication testing program is probably best used with individuals or small groups, although it has been used successfully with whole classes.

If the output is sent to a printer also, the cars will not be printed but you will have a permanent record of the sums set and whether they were right or wrong. Pupils could be asked to correct wrong answers shown on the print out.

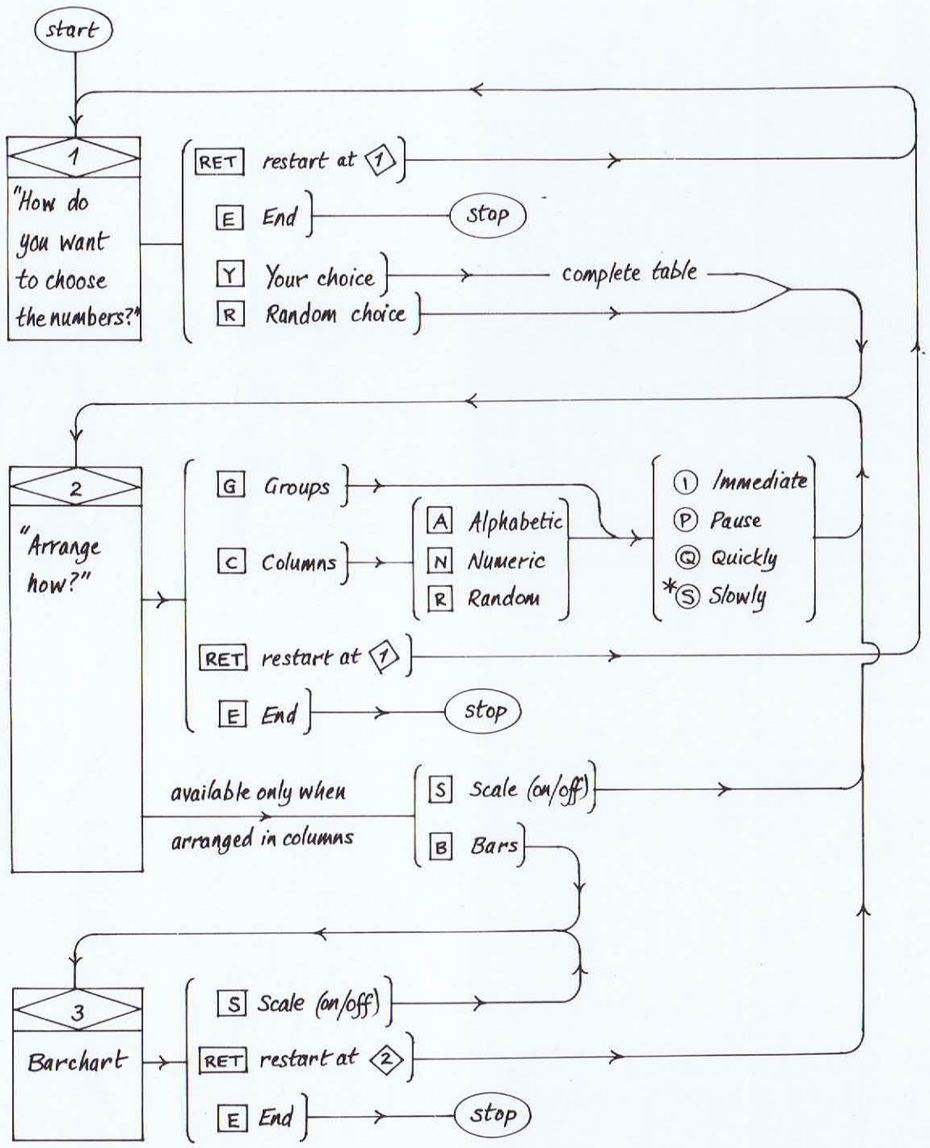
```
RUN
Hello. What is your  name? ITMA
Hi ITMA. This is a test of your
tables.
Type your answer to each question
followed by the 'RETURN' button.
Your car will move if you get a
question right and mine  will move
if you get one wrong.
GOOD LUCK
QUESTION 1
What is  7 x 12  ? 74
WRONG
QUESTION 2
What is  1 x 1  ? 1
RIGHT
QUESTION 3
What is  8 x 6  ?
```

Figure 2

Now type your answers to the questions posed. The screen modifies the picture of the car race according to the accuracy of your answers.

4. Would you like more control as the program is running, such as the ability to stop the program at any time or to key in attempts at a correct solution?
5. Does the ease of running the program outweigh limitations to its flexibility?
6. Is the lack of control by the teacher undesirable in the classroom?
7. Do the pupils (and teacher) actually enjoy the activity?

Drivechart for BARSET



TABCAR

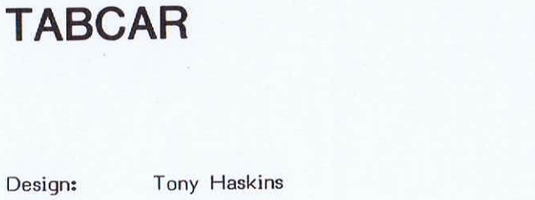
Design: Tony Haskins
Program: Tony Haskins
Source: Birmingham Educational Computing Centre

Program copyright © Tony Haskins, 1982

TABCAR runs on a Research Machines 380Z in 32K RAM with BASIC Version 5. It may be used on a disc- or cassette-based system. It does not use high-resolution graphics.

Summary of TABCAR

TABCAR provides practice in multiplication tables in the context of a car race. A correct answer moves your car forward; a wrong answer advances the computer's car. Problems are chosen randomly, using integers between zero and twelve. Note that if you get the answer wrong, you are not told the correct answer.




```
QUESTION 7

What is  12 x 10  ?
```

Figure 1

Running TABCAR

Load TABCAR; then type 'RUN'.

BARSET

Design: Graham Field, Rosemary Fraser, Jane Petty, Jan Stewart, Laurie Tate
Program: Graham Field
Source: ITMA, Plymouth

Program copyright © CET/College of St. Mark & St. John, 1982

BARSET runs on a Research Machines 380Z in 32K RAM with one of the following versions of BASIC:

BASICS Version 5 (but not 5.0G, which has a fault in the TAB function)
BASICS Version 4 (tested with 4.0D)
DBAS9
BASG

Note: If the program is transferred to BASIC Version 5 using OLDLOAD, you must ensure that lines 9040 and 10300 are properly translated. Type

[E][D][I][T] [9][0][4][0] [RET] [RET]
[E][D][I][T] [1][0][3][0][0] [RET] [RET]

Then save the program.

BARSET may be used with a disc- or cassette-based system. It does not require high-resolution graphics.

Please note this program is still under development and later published versions may differ from this one.

Summary of BARSET

The program handles sets of pictograms of five objects prepared in advance (using BARPIC, a separate program) and of pets. There may be up to 5 of each object, with an overall maximum of 20 objects. These objects are first displayed at random and may then be moved into various arrangements.

Running BARSET

The first decision is prompted by:

How do you want to choose the numbers?

Permitted alternatives are [R] (at random), [Y] (your own choice) or [E] (end).

[R] will produce a random number of pictures of the defined set of objects: this number will be in the range 8 to 20.

[Y] leads to a table which is completed by the user. The machine lists the names of the five possible objects and the maximum allowed number of each: this must be less than 5 and give a grand total of 20 or less. When the table is complete, the chosen pictures are displayed on the screen in random positions (Figure 1).

The second decision follows: Arrange how?

The program continues to return to this point from now on, until the question is answered by pressing the [RET] key (when it goes back to the previous decision point).

The user has initially two choices:

[G] Groups, with one of the five types of object in each of the four corners of the screen, and the fifth type in the centre; or

[C] Columns, with the five types arranged in columns.

In the latter case, the order of the columns may be indicated by typing [A] (alphabetic order), [N] (numeric order, decreasing in frequency) or [R] (randomly ordered). When arranged in columns, there are two further options: [S] instructs the computer to draw or remove a scale at the left-hand side; and [B] converts the diagram into a barchart. Having reached the barchart stage, you must type [E] to end the program, or [RET] to go back to the previous decision point.

Speed controls in BARSET

At any time during the rearrangement of the pictures, one of four speed controls may be used:

(S) Slow. This is the default condition. The picture to be moved is enclosed by a border: the picture flashes, a similar border is drawn at the point where the picture is to be moved to, the picture is transferred and flashes again, and the borders are removed.

(Q) Quick. Transfers occur without bordering and flashing.

(P) Pause.

(I) Immediate. In this mode, the final position is displayed at once.

BARSET in the classroom

Here is a typical route through the program.

[R] Random choice of numbers. Count how many there are of each type.

[G] Rearrange into groups, since this makes counting easier. Are the numbers still the same?

[C] Rearrange into columns...

[N] ...in numeric order.

(P) Pause, with one picture flashing - where do you think it will go?

Pupils can be given this strategy and then encouraged to suggest improvements, and to test their ideas against the computer.

To conclude, SUBGAME can provide a useful and stimulating environment in the classroom, promoting class discussion, individual pupil work and development of a simple gaming strategy.

SUBGAME in the classroom

SUBGAME has promoted many interesting lessons in mathematics. On first acquaintance, SUBGAME appears to be simply a game best suited to a single user; SUBGAME displays its true potential only when used in the classroom with a full group participating.

The initial problem can be simply to beat the computer either with the whole class contributing or (better) with each student working individually and recording his own attempts. As with many topics, the structure of students' answers can be improved by providing duplicated sheets with boxes already drawn for them; they can then enter values and keep the record of their work neat and accurate.

Once the program has been used solely as a game (perhaps ten times), the problem can be widened to consider the strategy being used by the computer, and whether this can be bettered. Pupils may well allege that the computer cheats; how can we be sure that it is generating random numbers and not carefully adjusting the values it produces to make sure that it wins. It might even lose games to encourage the user to try again (the 'one-armed bandit' syndrome). In one lesson when pupils then accused the computer of cheating, the teacher assured them that it was only a machine and so did not know how to cheat. However when they tried to place a digit in a square already occupied, the computer accused the students of cheating! This multiplicity of roles played by the program, and the pupils response to it, is an interesting facet of SUBGAME.

The strategy used by the computer is very simple, merely a hierarchy of places for each digit depending on its value.

Digit	Order of search for an empty place
1, 2	d e c b a
3, 4, 5	e d c b a
6, 7	c b a e d
8, 9	a b c e d

Figure 5

(S)	Continue slowly.
(P)	Pause again. Why did that picture move there?
(Q)	Quickly (if and when sufficient discussion has taken place).
[B]	Put boxes around the pictures to complete the conversion to a barchart.
[SPACEBAR]	Does the picture still say the same thing? Can we count now? What do we need to be able to count?
[S]	Add a scale. Does this help?
[E]	End the program run.

The program provides sufficient variation for a large number of teaching situations. It can promote counting, discussion and re-ordering of objects, as well as an introduction to concept of barcharts.

Since there may be no more than five of each type of picture, the use of true pictograms - in which each picture represents more than one item - may be explored with the pupils.

Rearrangement of the pictures into columns in one of three possible orders may raise the question of the 'best' or 'most natural' order for the columns. Does the order matter?

BARSET has been designed primarily for young children, however the algorithm that is employed for the sorting and grouping processes will be of interest to computer studies groups in Secondary Schools.

Ideas and questions about BARSET

1. Is the driving system satisfactory or too complex?
2. Are the low-resolution graphics diagrams satisfactory for the purpose?
3. Does the program cover all the likely teaching requirements in this area?
4. Do the restrictions on numbers (maximum of 5 of each picture and total less than 20) limit the program too severely?
5. To what extent would it be possible for the program to be used by pupils, individually or in small groups, with minimum intervention from the teacher?

(BARSET as published in this collection of programs restricts the user to one set of data. A later version of BARSET is to be published with a sister program BARPIC which allows you and/or the pupils to create your own sets of data. This extends the use of BARSET considerably.)

The following figures illustrate the variety of arrangements that are possible.

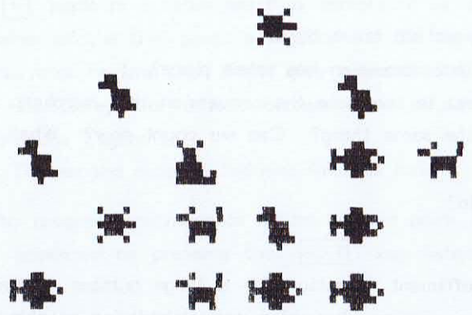


Figure 1

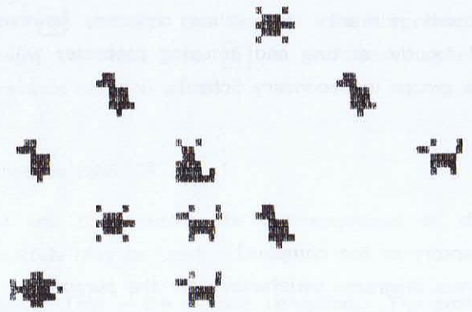


Figure 2

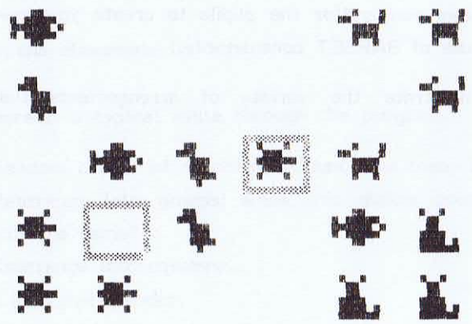


Figure 3

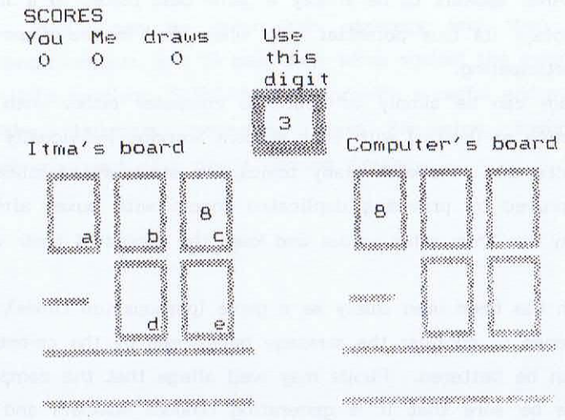


Figure 3

If you attempt to place a digit in a space already occupied, the computer will prevent this and accuse you of cheating. When both sums are complete, the computer will calculate its answer and then prompt you for the units, tens and hundreds answer for your sum, accepting correct values only. The score board is updated, and you are invited to have another try.

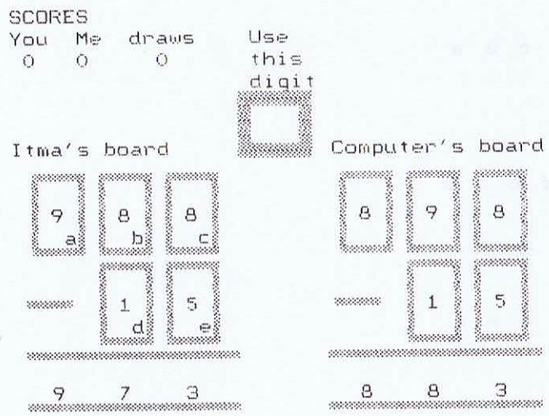


Figure 4

Running SUBGAME

SUBGAME is relatively easy to run. When the program has been loaded from disc or cassette, type 'RUN' as usual. The first prompt is for a name: after typing this, press the [RET] key.

```
        SUBTRACTION     GAME

You will be given five digits to place
in a subtraction sum. The computer will
play against you.

You have to make the answer to your sum
as large as possible!
See if you can beat the computer.

Please type your name and press RETURN.
```

Figure 2

The screen displays the two blank subtraction sums (a two-digit number to be taken from a three-digit number), and the first digit to be placed in your sum. To place the first digit, press [A], [B], [C], [D] or [E] on the keyboard. The computer then places the same digit in its own sum, and proceeds to the next digit.

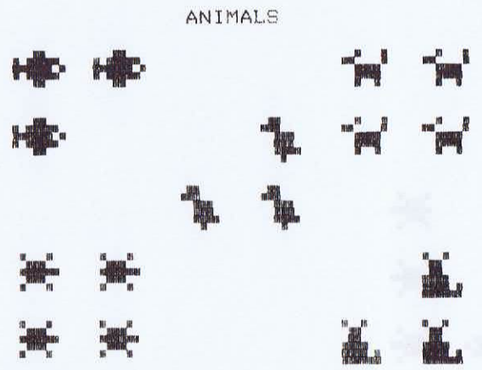


Figure 4

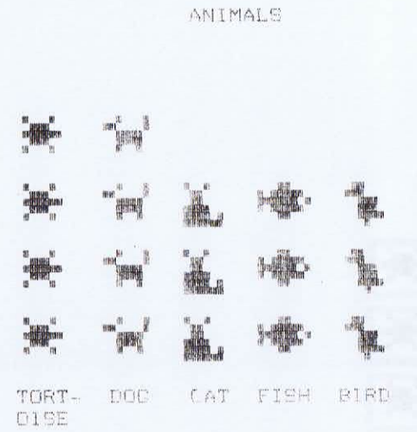


Figure 5

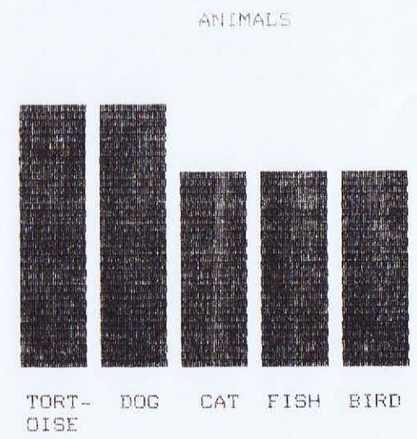


Figure 6

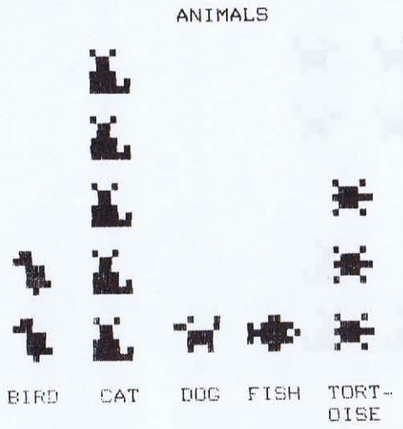


Figure 7

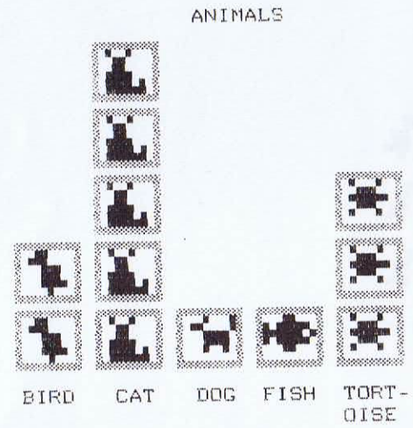


Figure 8

SUBGAME

Design: Brian Ives
Program: Brian Ives
Source: Brian Ives, Microcomputer Coordinator for North Yorkshire County Council

Program copyright © Brian Ives, 1982

SUBGAME runs on a Research Machines 380Z in 32K RAM with BASIC Version 5. It may be used on a disc- or cassette-based system. It does not use high-resolution graphics.

Summary of SUBGAME

SUBGAME is a useful example of the different roles that a computer can play in the classroom. The program consists of a competition between the user and the computer to allocate digits to a subtraction sum so as to give the largest answer. The digits are generated randomly by the computer; the class decides where to place each in turn while the computer competes alongside with its own sum. (Note that the same digit may recur.) When all digits have been allocated, the computer calculates its own result, the class types its result and the winner, is the player with largest answer. The computer displays a record of won, lost and drawn.

A typical screen display is shown here in Figure 1.

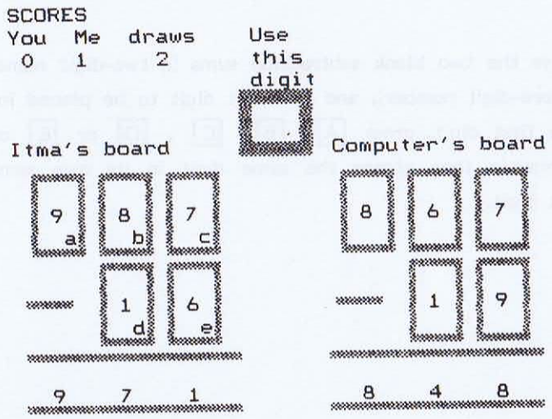


Figure 1

COUNTERS

Design: Anita Straker
Program: Anita Straker and Graham Field
Source: Anita Straker, Wiltshire Education Authority

Program copyright © Anita Straker, 1982

COUNTERS runs on a Research Machines 380Z in 32K RAM with BASIC Version 5. It may be used on a disc- or cassette-based system. It does not use high-resolution graphics.

Summary of COUNTERS

COUNTERS is a game for two players, intended to strengthen knowledge of number bonds and to develop strategic thinking.

The program displays a row of squares marked with the numbers 1 to 9. Players take turns to place their counters on uncovered squares. A player wins if he or she is the first to have three counters on squares whose numbers total exactly 15.

The game may be repeated with the same or with new players. If the same players continue, they take turns to begin.

Running COUNTERS

After an introductory title-sequence, the prompt

Do you want the rules?

appears. The operator(s) must reply by pressing either of the keys [Y] (yes) or [N] (no). If [Y] is pressed, a description of the game is displayed.

The players are then invited to type their names. The program automatically uses a capital letter at the beginning of the name and lower-case letters thereafter. If the [RET] key is not pressed within a given time, a warning is printed.

The boxes used for the game are displayed with the numbers 1-9 above them. Players are prompted by name to enter the number of a box that they wish to occupy. Although this entry is of a single digit, it must be completed by pressing [RET]. A grey or white counter, as appropriate, is displayed in the required box (Figure 1). Totals are checked automatically and the winner advised both by flashing the three numbers in their boxes and displaying their sum below (Figure 2). If all boxes are occupied without either player having won, a draw is declared.

The game may be played again with the same players or new ones. In each case the option is selected by typing [Y] (yes) or [N] (no) in answer to a simple question.

```
Michael- grey   George - white.

   1   2   3   4   5   6   7   8   9
 \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
 |   |   |   |   |   |   |   |   |   |
 |   |   |   |   |   |   |   |   |   |
 |   |   |   |   |   |   |   |   |   |
 \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

George, your turn

Which box for your counter ?
```

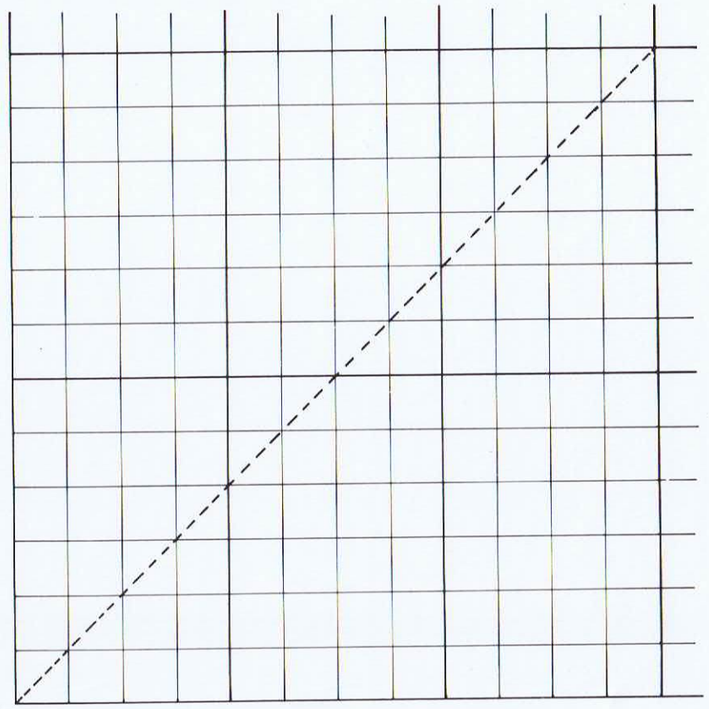
Figure 1

```
Michael- grey   George - white.

   1   2   3   4   5   6   7   8   9
 \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
 |   |   |   |   |   |   |   |   |   |
 |   |   |   |   |   |   |   |   |   |
 |   |   |   |   |   |   |   |   |   |
 \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

Well played, a draw!

Michael - 5 games, George - 4 games.


Another game ? Type Y for yes, N for no

    Warning - PRESS Y OR N PLEASE.
```

Figure 2

Ideas and questions about COUNTERS

1. Does the title sequence add to or detract from the program?
2. Are the instructions provided sufficiently clear?
3. What is your strategy for play?
4. What strategies would you expect children to develop?
5. Could the program be used for a whole class, under teacher control?
6. For what age and ability range is the difficulty level appropriate?



Horizontal and vertical grid lines crossed	3
(Add on for start and finish)	+2
Total number of hits recorded	5

From the diagram it is 'clear' that the ball will finish in the bottom right corner.

4. How can the various hypotheses be written down? This can lead to a discussion of mathematical research, and to the need to exchange on paper details of work done.
5. Using the [W] (wait) key at some stage, can we tell from the present pattern what the complete pattern will look like? Must the ball produce a complete symmetrical pattern before it hits a corner?
6. What combination of height, width and gradient (or angle) causes the ball to run across the grid and finally hit the original corner (bottom left)?
7. The move from gradients to angles will introduce concepts of irrationals. What effects does an irrational gradient have on the path of the ball?
8. If drawing has been used, you can investigate other shapes of table. Of particular interest is a circular table with a hole in the centre and the ball sent from the edge.
9. What relationship is there between height, width and the number of regions into which the ball has split the grid when its path is complete?

Hypotheses may also be formed concerning the relationship between height, width and the particular corner pocket into which the ball finally falls. This relationship should not prove too demanding if the earlier one was successfully discovered.

SNOOK can be used to introduce the concept of generalizing results. By changing the gradient of the ball's path, setting it to 0.5 or 2 as well as to the default setting of 1, pupils can quickly appreciate that the problem is fundamentally the same; they simply need to factor the height or width accordingly.

Finally, pupils can study the results of changing the angle of the ball's path. This is set initially to 45^{o}, but in the full version can be set to any angle. The results may provoke much interesting discussion and thereby reveal the well-defined mathematical structure of the problem as opposed to that of the real situation.

The height and width can be keyed at the keyboard, or the program can be run in film-mode. In this setting the height and width are generated randomly and the teacher can sit with the children, not typing at the keyboard but busy recording results and sifting ideas.

Using a program such as SNOOK frequently promotes wide-ranging discussion with pupils. Here are some points that have emerged in our trials. These questions either posed directly or subtly encouraged, are of interest:

1. To what extent is the situation displayed an artificial one? (Bounce is perfect; the ball never stops; a point ball falls into a point pocket.) Is it a necessary feature of computer simulations that in order to be implemented, they must simplify the situation so that it becomes unrepresentative?
2. How many examples of the application of a rule must be considered before deciding that the rule is correct? What is the difference between the mathematical proof for a well-defined problem and the testing of a scientific hypothesis concerning observed phenomena?
3. The proof is quite difficult. One interesting approach is to draw a grid of rectangles of the chosen shape 'allowing' the ball starting in the corner in the chosen direction to cross each grid line it meets - instead of being reflected it moves from 'table' to 'table' in a straight line. The ball enters a pocket when the line meets another corner; the sum of horizontal and vertical 'table' lines crossed plus two gives the number of hits. You can then work out which pocket the final corner represents. For example, on a table width 4, height 6 the diagram would be:

CWORDS

Design: Anita Straker
Program: Anita Straker and Graham Field
Source: Anita Straker, Wiltshire Education Authority

Program copyright © Anita Straker, 1982

CWORDS runs on a Research Machines 380Z in 32K RAM with BASIC Version 5. It may be used on a disc- or cassete-based system. It does not use high-resolution graphics.

Summary of CWORDS

CWORDS ('counting words') is a drill-and-practice program intended for use by individual children or small groups. The program offers a test of verbal description of numbers within a predetermined range of values. If an incorrect answer is given, the child is invited to try again. This time, he or she is prompted with information as to the number of digits required. Should the second answer be wrong also, a special tuition page is displayed and the child prompted to complete the number digit by digit (Figure 3). At the end of the program, the teacher may have a report of the scores of all children who have used the program. (Answers are taken as right if they were correct at either the first or the second attempt.)

Running CWORDS

After an introductory title sequence, the prompt:

Do you want the teacher's notes?

appears. The operator must type [Y] (yes) or [N] (no) as desired. The notes give a brief explanation of the program.

The operator (who may be presumed at this stage to be the teacher) may select one of three options:

[1] for numbers up to 999;
[2] for numbers up to 9999;
[3] to end the program.

Selection of [1] or [2] specifies the range of numbers which may be presented to the child or children for identification. In either case the teacher will next specify the number of questions which are to be put to each child. This input must be terminated when complete by pressing [RET].

The program now requests the name of the first child. The name is used both for an initial greeting and during subsequent prompts.

Each question is presented to the child on a new 'page' (that is, a fresh screen). This display consists of the number of the question, the current score and, below a dividing line, the question itself with this prompt:

What is this number in figures?

(See Figure 1.) An incorrect response is translated into words and the child is told how many digits are expected. He is then given a second opportunity to answer (Figure 1). If the answer is again wrong, a different 'tuition' page is displayed (Figure 2). The child is taken through the various 'place-values' in a decreasing sequence. In the event of a wrong answer at this stage, the correct digit is inserted.

When this has been completed, the pupil is required to type the full number before moving on to the next question (Figure 3). When all the questions have been completed, the child is given a page showing his or her score and is congratulated. The name of the next child is then requested as above.

At this stage the teacher may, by pressing the [*] key, obtain a report on the scores of the children who have attempted the exercise. This report may be sent to a printer if hard copy is required.

The questions are generated at random and are different for each child.

Notes on the use of printers

To attach a printer, change lines 20100 and 20110 as follows to accommodate your own printer:

20100 PT = printer type for your printer
20110 BR = baud rate for your printer (may be zero for some)

These provide the two parameters in the command 'PRINTER PT, BR'. As provided, all printer output is sent to the screen, so selection of this option will cause the report to appear twice.

```
 1.                          SCORE: 0
========================================

 eight hundred and seventy

What is this number in figures ?807

You have typed in
eight hundred and seven

Try again.  Your answer should have
three  Figures.
```

Figure 1

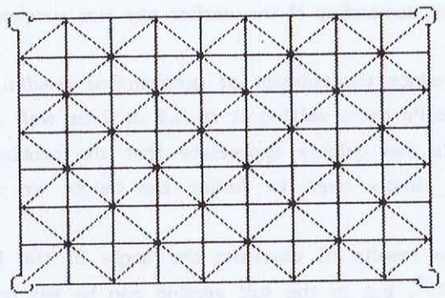


Figure 2

Enhanced version of SNOOK

An enhanced version of SNOOK will be published separately. The full version will allow the graphic display of results for heights and widths up to 999999. Any gradient can be used and also angles can be selected to a chosen number of decimal places to support hypothesis testing. The program can also produce graphics printouts of the results from the screen.

SNOOK in the classroom

The main line of an initial investigation is to find a relationship (given orally or in writing, verbally or algebraically) between the height, the width and the number of hits. The next problem is to find a suitable method of testing the hypothesis (or hypotheses). Pupils should be encouraged to find their own suitable test data; most incorrect hypotheses can be eliminated by choosing heights and widths such as 2 by 8 or even 6 by 6. Pupils, quite correctly, may well suggest that there is no single relationship but many sets of special cases (as with square tables, for instance). This can promote much useful discussion of the extent to which the problem is defined: can we expect a solution? This version of the program will display results graphically only for heights and widths less than ten, but the final results for other dimensions can be obtained by pressing [N] for numbers only (that is, no graphics).

Once a suitable relationship has been found, pupils may consider how many different dimensions of table should be tested before this relationship can be assumed to hold true for all dimensions. An attempt should be made to discuss the difference between testing observable natural phenomena (as with scientific experiments) and providing a tight mathematical proof.

In this example, in which the ball moves across a 7 by 4 table, the final result displayed is that there were 11 hits and that the ball finished in the bottom right-hand corner.

You are then invited to have another go:

Again Y/N?.

If you press [Y] , you are returned to the initial height prompt; if you press [N] , you exit from the program, and are left with the 'READY': prompt. (If you press [N] accidentally, you can type RUN [RET] to start again.)

Options available in SNOOK

(Options marked * are set automatically when the program is run.)

At the height, width prompt:

Digits - set the height or width as appropriate.

* [E] Evenly: sets the gradient to 1.

[G] Gently: sets the gradient to 1/2.

[S] Steeply: sets the gradient to 2.

[A] Angle: choose the angle for the ball's path.

[N] Numbers only: results only are displayed; tables whose height or width exceeds 10 can be investigated.

[F] Film mode: height and width are generated by the computer.

* [K] Keyboard control: height and width are typed at the keyboard

[H] Help: display the list of options.

As the ball is moving:

[Q] Quit: the current problem.

[W] Wait: stop the ball to allow discussion. Pressing [W] again resumes the path.

[R] Result: give the final result immediately, without completing the picture.

A typical screen display is shown in Figure 2.


```
 You were asked to type

eight hundred and seventy

                H ! T ! U
                ----------

How many hundreds have you got ?8


                That's right Mary
```

Figure 2

```
 You were asked to type

eight hundred and seventy

                H ! T ! U
                ----------
                8   7   0

Now the units ?0

 Now type

 eight hundred and seventy

 in figures:- 870

                Well done!
```

Figure 3

Ideas and questions about CWORDS

1. Are the teacher's notes included in the program necessary? Are they sufficient?
2. Is the program easy to use? For the teacher? For the pupil?
3. Compare CWORDS with TABCAR. What do you think of the pupil-machine interface?
4. Do you like the tutorial mode of operation? Should the teacher deal with this? If so, how?
5. What proportion of a pupil's time should be spent in using a computer in this way?
6. What are the advantages and disadvantages of using a computer in this way as compared with worksheets or a blackboard?

7. Do you like the display of the teacher's record? Does it give sufficient information?
8. Should the request for print-out be made at this stage rather than earlier?
9. Would large letters improve the program? Are there any other design features you would like to see added?
10. Programmers may like to compare CWORDS and COUNTERS for similar structure. If you delete lines 71 to 800 and 2500 to 27900, you are left with a framework which can be used in similar programs. Would such a framework be of use to you?

SNOOK

Design: Jon Coupland
Program: Jon Coupland
Source: ITMA, College of St. Mark & St. John, Plymouth

Program copyright © CET/College of St. Mark & St. John, 1982

SNOOK, in its reduced trailer version, runs on a Research Machines 380Z in 32K RAM with BASIC Version 5. It can be used on a disc- or cassette-based system. The program uses high-resolution graphics if available, and low-resolution graphics otherwise: it checks for itself when running.

An enhanced version of SNOOK is available separately.

Summary of SNOOK

SNOOK depicts a ball moving round a rectangular table until it falls into one of the pockets placed at each corner.

The simulation encourages pupils to form hypotheses and to test these in a systematic manner. The path traced by the ball can be displayed for tables whose height and width, chosen each time by the user, are less than ten: results for larger sizes can be obtained in a text mode.

Running SNOOK

SNOOK is designed for classroom use: it provides help if any errors are made and displays the range of options available at each stage.

The ball starts from the bottom left-hand corner. As it moves across the table, the number of collisions with the sides are counted, and this count is displayed. To ease the solution of problems, the count is initialised to 1 and is also incremented when the ball falls into a pocket. As well as counting the hits, the system shows which pocket the ball fell into at the end.

A typical run of SNOOK would be as follows:

	Effect required	Keys pressed
1.	Load SNOOK into the computer from disc or cassette (Wait for 'Ready': to be displayed)	[L][O][A][D] ["][S][N][O][O][K]["] RET
2.	Set SNOOK running	[R][U][N] RET
3.	Select height 7	
4.	Select width 4	
5.	Draw the table and start the ball moving	SPACEBAR

Figure 1

4. Clues given as instructions in the form 'GO NORTH EAST'
5. The skull and tree picture is displayed

Full version of PIRATES

See the published unit for the full version of PIRATES. As the drivechart shows, this has many more options available than the trailer version, and includes further information, detailed examples and teaching notes.

Commands in DICECOIN

Select any option by typing the appropriate keys in response to the prompt symbol >>>> .

Simulations

S 1	One coin; frequency diagram
S 2	One coin; proportion of heads
S 3	Two coins: frequency of 2H, 1H, 0H
S 4	Two coins; frequency of HH, HT, TH, TT
S 5	One die; frequency diagram
S 6	One die; proportion of sixes
S 7	Two dice; frequency diagram

Options for a preselected simulation

R	Repeat
C	Repeat continuously (use CRTL Z to stop)
P	Change parameters
T	Go into tabular form
SPACEBAR	Pause/start

Other options

O	Give a list of options
E	End the program and return to BASIC

PIRATES

Design: Rosemary Fraser, David Lee, Colin Wells
Program: Colin Wells and Mike Allnut
Source: ITMA, College of St. Mark & St. John, Plymouth

Program copyright © College of St. Mark & St. John, 1982

PIRATES in its reduced trailer version runs on a Research Machines 380Z in 32K RAM with BASIC Version 5. It may be used on a disc- or cassette-based system. It does not use high resolution graphics.

The full version of PIRATES is available separately.

Summary of PIRATES (trailer version)

PIRATES is a treasure hunt. The treasure is hidden by the computer at a mesh point on an integral grid, and the treasure seeker approaches its hiding place by supplying 'guesses' expressed in coordinates. Each new 'guess' yields information which may be used in making next guess.

Running PIRATES

In this trailer version, the grid is two-dimensional. Its range in each dimension can be set, and the computer is free to place the treasure anywhere. There are two different ways (clues) in which the computer can report the relation between the treasure's location and the current guess.

For each new search, the initial amount and type of treasure is selected by the computer. During an investigation the pirates steal more of 'the loot' with each unsuccessful guess. A search continued beyond ten tries will locate an empty chest!

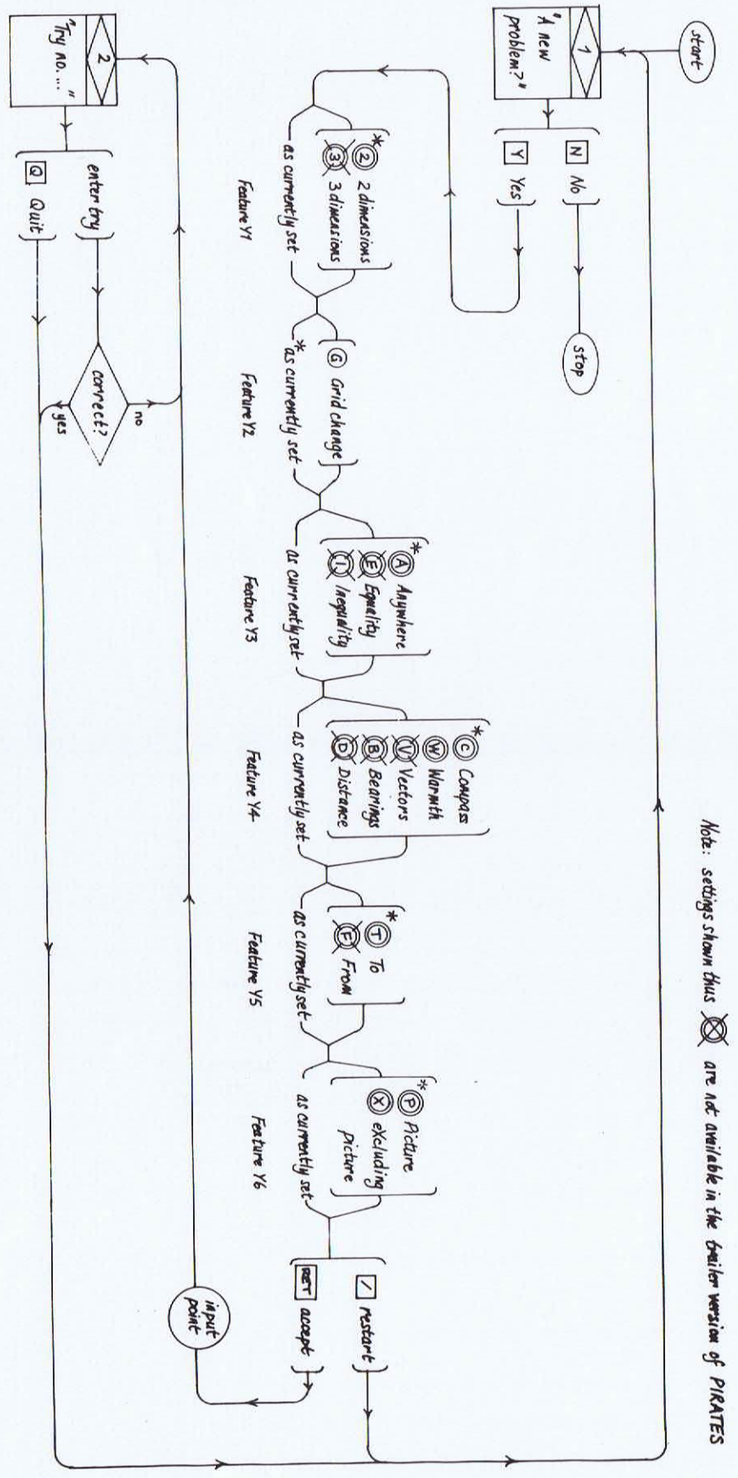
A number of themes emerge from this rich combination of search space and clues. These include practice in the use of two-dimensional coordinate representations of position; compass work; the plotting of data; the construction of hypotheses; and the formulation of best strategies for an investigation.

Because of the wide choice of puzzle settings afforded by the program's flexible 'driving' system, the teacher may specify a mathematical activity suitable for the pupils and may control the level of demand made of them by the task.

For example, the following Feature settings are vey suitable for young children:

1. Both x and y range between 0 and 9
2. The treasure can be at any mesh point on the grid
3. Compass directions used to describe position relative to the treasure

Drivechart for PIRATES



Note: settings shown thus ⊗ are not available in the basic version of PIRATES

DICECOIN

Design: Alan Wigley
Program: Alan Wigley
Source: South Wolds School, Nottinghamshire
Program copyright © Alan Wigley, 1982

DICECOIN runs on a Research Machines 380Z in 32K RAM with BASIC Version 5. It may be used on a disc- or cassette-based system. It does not use high-resolution graphics.

Summary of DICECOIN

This program could be the basis of many interesting investigations into probability. It simulates the throwing of dice and tossing of coins, and uses graphs, histograms and tables to illustrate the patterns which emerge from random events.

Figure 1

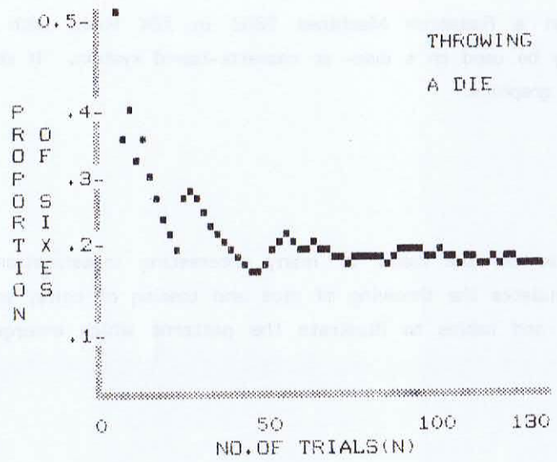


Figure 2

Running DICECOIN

Load and run the program. Try these examples (Note that the computer prints '>>>>' when it is ready for you to type a command.)

Computer: >>>>
You: [S] [3]

Simulation 3 shows how often you get 0, 1 or 2 heads when throwing two coins. Before continuing, try to imagine what the histogram would look like. At this point you might ask your class to write down what they think would happen.

Computer: How many trials (<170)?
You: [1] [0] [0] [RET]
Computer: >>>>
You: [R]

The simulation is repeated, though the result will of course be slightly different.

Computer: >>>>
You: [C]

Ideas and questions about FGP

1. What are the merits of active and passive uses of this program? With passive uses, pupils simply watch the screen. Active uses include 'guess the function' games and fitting curves to data.
2. Algebraic notation teaches us to write 'y=ax', but in most programming languages we must write 'y=a*x'. This program allows either expression. Which would you encourage pupils to use here?
3. Using the [I] option, plot 'y=x n' for all integers between -4 and +4. Discuss the significance of the common points.

The simulation is repeated continuously. This option of repeating many times is useful in giving the class a feel for the variability which occurs. You can let this display run for a long as you like. To stop it, hold down the [CTRL] key and type [Z].

Computer: >>>>
You: [S][4]
Computer: How many trials (<320)?
You: [1][0][0] RET
Computer: >>>>
You: [E]

[E]nds the program and returns control to BASIC.

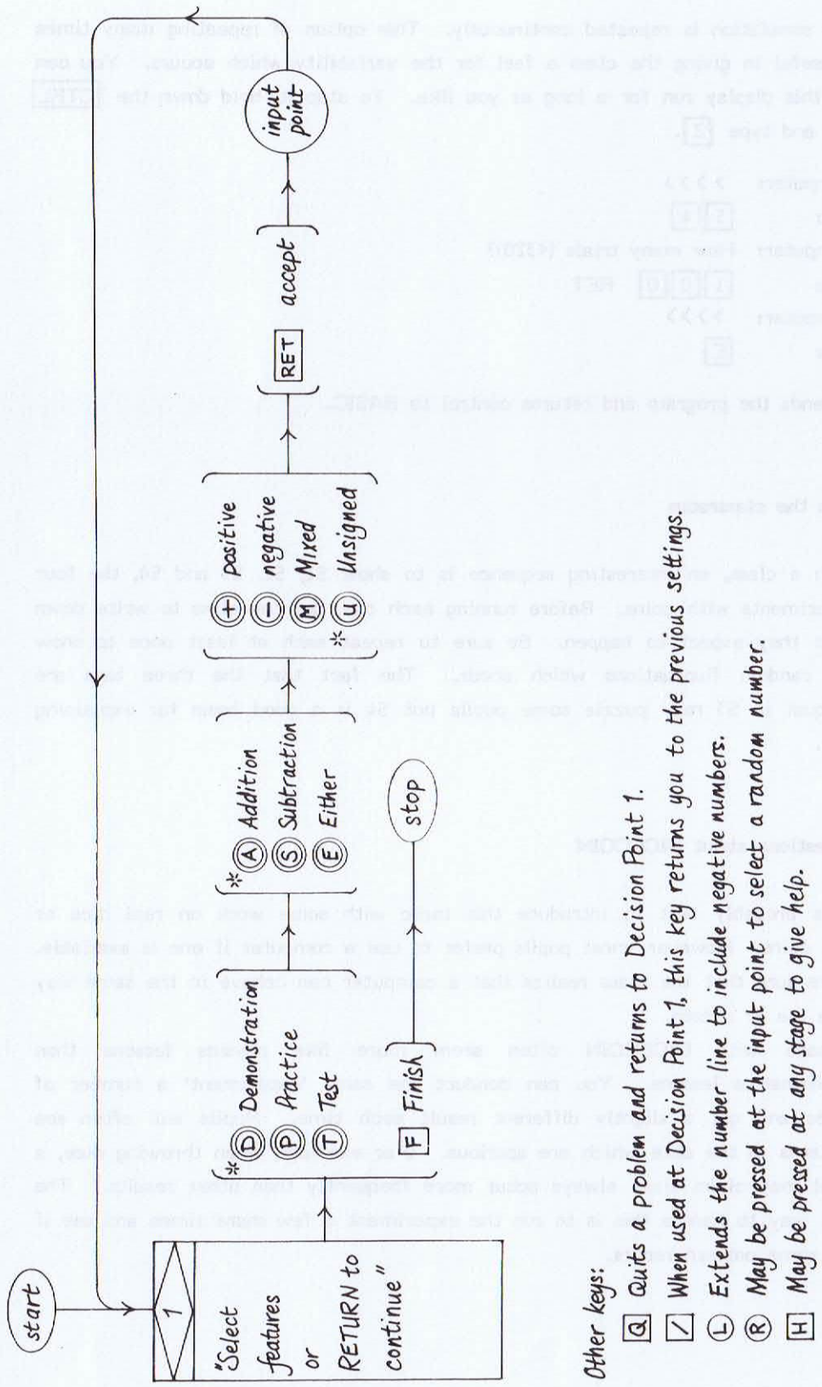
DICECOIN in the classroom

With a class, an interesting sequence is to show S1, S2, S3 and S4, the four experiments with coins. Before running each one, get the class to write down what they expect to happen. Be sure to repeat each at least once to show the random fluctuations which occur. The fact that the three bars are unequal in S3 may puzzle some pupils but S4 is a good basis for explaining why.

Ideas and questions about DICECOIN

1. It is probably best to introduce this topic with some work on real dice or real coins. However, most pupils prefer to use a computer if one is available. Make sure that the class realize that a computer can behave in the same way as a die or a coin.
2. Lessons with DICECOIN often seem more like physics lessons than mathematics lessons. You can conduct the same 'experiment' a number of times and get a slightly different result each time. Pupils will often see patterns in the data which are spurious. (For example, when throwing dice, a pupil may claim sixes always occur more frequently than other results.) The best way to handle this is to run the experiment a few more times and see if the same pattern recurs.

Drivechart for DIRECTED



Other keys:

- Q Quits a problem and returns to Decision Point 1.
- / When used at Decision Point 1, this key returns you to the previous settings.
- L Extends the number line to include negative numbers.
- R May be pressed at the input point to select a random number.
- H May be pressed at any stage to give help.

Options

- A Angles - degrees or radians. When the program is loaded it will work with degrees, but can be switched to radians by calling this option. Subsequent uses of the option switch backwards and forwards between radians and degrees. Invoking this option has no effect until a new function is defined.
- D Double replot. This does everything done by the replot option (see R below), but also replots the current function which is unchanged.
- E End the program and return to BASIC.
- I Increment and superimpose. This option provides an easy way of displaying a family of curves which vary along one parameter. Before the option can be used, a graph must be displayed on the screen. The option will increment or decrement a parameter and superimpose the new graph; it will continue doing this until the space-bar is pressed.
- L Load and display data. This option allows point data stored in a disc file to be displayed on the screen. Lines can be plotted on top of the data points using the X option. We have provided sample data file, WEIGHT, based on work by D. Burghes and R. Blackford.
- O Display a list of these options on the screen.
- P Plot a graph. Clear the screen and plot a graph of the current function with a choice of scale on each axis. Before a graph is plotted, the program asks you for values of 'xmin', 'xmax', 'ymin' and 'ymax'. The RET key should be pressed after each value. If RET is pressed without typing a value, the previous value will be unchanged.
- Q Quick graph. Clear the screen and plot a quick graph of the current function. The x-axis is chosen automatically to show most or all of the graph.
- R Replot. This option replots whatever was originally plotted, without any of the superimposed lines.
- S Superimpose a graph of the current function on top of the graph or graphs shown on the screen.
- U Unchanged scales. Clear the screen and plot a graph of the current function with the scales unchanged. For this option to work, a graph must be displayed on the screen.
- V Variables. Set the letters used for the dependent and independent variables. Initially these are y and x, but any letters may be used. This option has the effect of cancelling the current function and clearing the screen.
- W Write the current function on the screen, with the values of any parameters.
- X Secrecy option. Calling this will cause most characters typed on the keyboard to be echoed as asterisks on the screen. Subsequent calls switch between normal echoing and echoing with asterisks.
- Z Superimpose the first derivative of the current function on the graph or graphs shown on the screen. The line is drawn in grey. Note that the derivative does not become the current function, and there is no way of plotting the second derivative.

at about 45 degrees. If it is not, experiment with the y scale until the angle is satisfactory. Make a note of the range which works best, and use this whenever you require equal scales.

'Guess the function' games can take a number of forms. A simple version might start by displaying a straight line on the screen and inviting the class to discover its equation. Each guess could be superimposed on the screen, and the first person to superimpose exactly on the original line wins.

With your class, start by typing [X] so that they cannot read which function you are entering. Then type y=x/2, and type [X] again to cancel the secrecy option. Plot the function using the [P] option and choose scales which will show gradients correctly.

Now invite the class to guess the function. Suppose someone suggests 'y=2x'. Type this in and then type [S] to superimpose this suggestion. It is clearly wrong. Next someone might suggest 'y=x+2' and you can superimpose this in the same way. If the screen gets cluttered with lines, type [D] which will replot your original line and the last suggestion which was superimposed. When someone finally suggests 'y=x/2' or 'y=.5x' or 'y=1/2x', this is exactly superimposed on your original line.

To summarize, when '>>' appears in the bottom left-hand corner of the screen, you can type either an option (such as [O]), or function (such as y=log(x)), or an assignment (such as x=4). Whichever of these you type, you must end with the [RET] key.

Functions

Functions are typed using a notation which is similar both to the BASIC programming language and to normal algebraic notation. The program is quite versatile and you will often find that the same function can be entered in several different ways.

Assignments

An assignment statement with the independent variable (such as x=5) is used to evaluate the function for that value. The program will reply with the corresponding value of the dependent variable (such as y=3.474). When an assignment is used with a parameter, it sets the parameter to that value.

DIRECTED

Design: Trevor Greenslade, Falmouth School, Cornwall
Program: Trevor Greenslade, Colin Wells, Maggie Anderson
Source: ITMA, College of St. Mark and St. John

Program copyright (C) CET, 1982

DIRECTED runs on a Research Machines 380Z in 32K RAM with BASIC Version 5. It may be used on a disc- or cassette-based system. It does not use high-resolution graphics.

Summary of DIRECTED

DIRECTED uses the movement of a robot up and down a number line to help in the teaching of addition, subtraction and directed numbers. This is the first draft of the program which is under development - the final version will be available as a complete teaching unit later.

Running DIRECTED

In DIRECTED, a robot moves up and down a number line. By following simple rules, the robot provides the answer to a sum. The rules are these:

1. Addition - face to the right of the line
   Subtraction - face to the left of the line
2. Positive number - walk forwards
   Negative number - walk backwards

The magnitude of the numbers in the sum is always less than 10 and the teacher may select the type of sum that the robot does:

(A) Addition sum, such as a + b. (This is the default setting.)
(S) Subtraction sum, such as a - b.
(E) Either addition or subtraction sum, chosen at random by the computer.
(U) Sum with unsigned numbers, such as 3 + 2 or 4 - 3. (This is the default setting.)
(+) Sum with positively signed numbers only, such as $^{+}3 + ^{+}2$ or $^{+}3 - ^{+}2$.
(-) Sum with negatively signed numbers only, such as $^{-}3 + ^{-}2$ or $^{-}3 - ^{-}2$.
(M) Sum with a mixture of positive and negative numbers. The sign must be chosen when the number is input.

When unsigned numbers are selected, only the positive side of the number line is drawn on the screen, with an unsigned scale.

When signed numbers are selected, the full number line from -18 to +18 is drawn, with a signed scale.

There are two ways of operating the robot:

(D) Demonstration mode. The computer asks for small numbers to be input. After each input has been concluded with RET , the space-bar must be pressed to move the robot. Thus the movement of the robot occurs in steps under the control of the teacher. When the second number in the sum has been entered, repeated depressions of the space-bar cause an 'equals' sign to appear and the robot to 'walk' to the correct answer, which is then displayed on the screen. (This is the default setting.)

(P) Practice mode. When the first number has been input, the robot moves automatically to the correct place on the line. When the second number has been input, the robot does not move; instead, a question mark flashes on the screen while the computer waits for an answer to the sum. If an incorrect answer is entered, the robot stays where it is and 'says' a suitable phrase. If the correct answer is entered, the robot finishes the sum and 'says' a (different) suitable phrase.

There is a third mode of operation of the program, in which the robot and number line do not appear.

(T) Test mode. A number of randomly selected sums are produced on the screen, with a suitable delay for pupils to write down the numbers. (No answer is given.)

Other keys available in the program are as follows:

[/] This key may be used when selecting a new type of problem or mode or operation, and resets the program to the previous setting.

[Q] This key can be pressed at any time when a problem is being investigated, and returns the program to Decision Point 1, 'Select features or RETURN to continue'.

[H] This key can be pressed at any point in the program and prints a list of the keys that are available.

(L) This key extends the short number line to include negative numbers and produces a signed scale. It is only available when the settings are (D), (S) and (U), and the answer is less than 0. At this point the robot will walk off the end of the number line and say 'Help'. When the line is extended by pressing (L), it says 'Thanks'.

(R) This key may be pressed instead of a number key when entering any number, and causes the computer to generate a number, at random.

[F] This key finishes the program.

If you try the last example, the program will ask you to provide values for a, b and c before the graph is plotted. When you use the [Q] option, your graph will show the range on the x-axis from approximately -10 to 10. But this range is not always satisfactory. Try typing:

y=sin(x)

and then type [Q]. (It is assumed that you will remember to press [RET] after each line.) You may have expected to see a sine wave but because x is in degrees you are seeing only a small part of a wave, namely that part lying between -10 and 10 degrees. One possibility is to instruct the program to work in radians rather than degrees. But suppose we want to keep degrees but see more of the graph. To do this, type [P]. This will plot a graph with your own choice of scales. Before the graph is plotted, four parameters must be provided: the minimum value of x the maximum value of 26, the minimum value of y, and the maximum value of y. Try asking for x from -300 to 300, and y from -1 to 1. When the graph is plotted, give it a title by typing [W].

Now try typing the quadratic :

x=ax↑2+bx+c

Notice how the '↑' sign is used for exponents. On the keyboard, '↑' is the [^] symbol on the top row of keys. Set a to 1, b to -2 and c to -35, and type [Q] for a quick graph. When the '>>' prompt returns, type:

b=-1

to change the value of b from -2 to -1. Now type [S]. This will superimpose a graph on top of the graph shown on the screen. Try setting b to other values and superimposing each graph. You should get a good idea of how varying b affects the shape of the graph. You can, of course, change a and c as well, although it is usually best to vary one parameter at a time.

When you have superimposed a number of graphs, the screen will look quite cluttered. To return to your original graph, and erase all the superimposed lines, type [R] to replot.

Now type [Z]. The [Z] option superimposes the first derivative (dy/dx) of the current function on the screen. As the current function is a quadratic, the first derivative will be a straight line.

Guessing a linear function

If the program is used with second- or third-year secondary classes, it is often useful to have equal scales on the two axes so that a gradient of one appears on the screen as 45 degrees. Enter the function 'y=x', type [P], set xmin=-10, xmax=10, ymin=-7 and ymax=7. On most screens the line should be

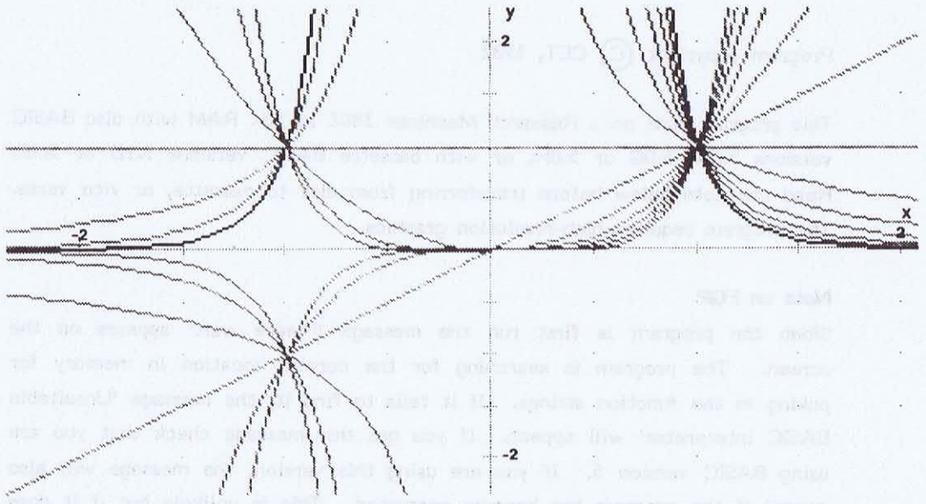


Figure 1

FGP in the classroom

Electronic blackboard

The program may be used as an 'electronic blackboard' whereby you can draw functions quickly and accurately. When you have successfully loaded and run the program, the '>>' prompt should be in the bottom left-hand corner of the screen. Type a simple function:

y=(x-8)(x+3)

You should end each line by pressing the RET key. Plot a graph of your function by typing Q and pressing RET. This plots a 'quick graph': it is quick because you do not have to decide the scale range you want; the program makes this decision for you. When plotting has finished the '>>' prompt will return.

Now type a different function and plot a graph of this in the same way. Here are some functions you might try:

y=1/x
y=log(x)
y-(x-a)(x-b)(x-c)

DIRECTED in the classroom

Simple sum practice with unsigned numbers

With unsigned numbers, the program can provide a simple but motivating way of practising sums, both addition and subtraction. In the 'demonstration' mode, pupils may become familiar with the rules operating the robot. In the 'practice' mode, they are asked to work out their own answers.

Introducing directed numbers

By investigating the robot's rules using unsigned numbers, and by producing a sum which takes the robot over the end of the line (3 - 5, for example), directed numbers may be investigated and sums involving them practised.

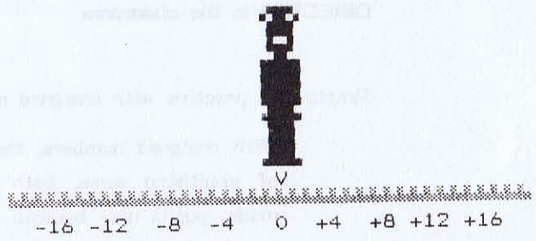
Generating examples

The 'test' mode can be used to generate random examples of a certain type of sum, such as unsigned subtraction, or subtraction with negative numbers only. This provides a quick and easy way for the teacher to set examples during a lesson.

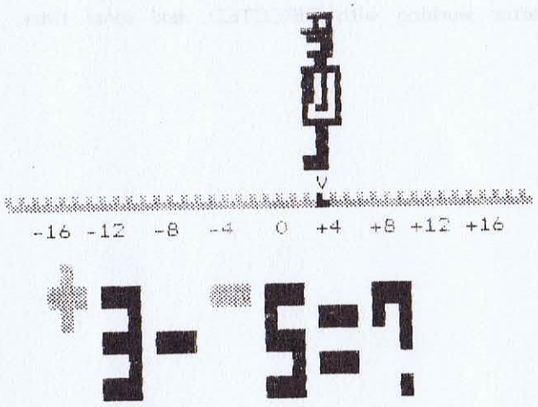
Ideas and questions about DIRECTED

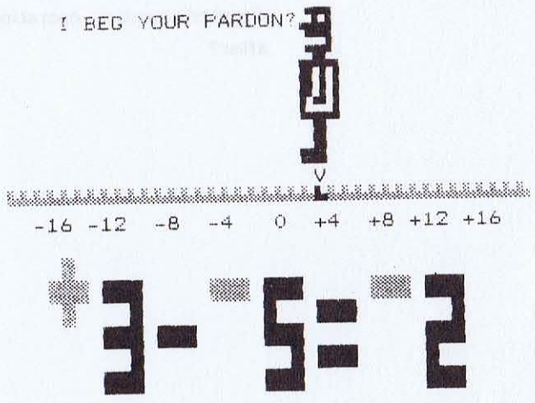
1. What are your views on the 'test' mode - should answers appear automatically after a time lapse, or on pressing a key? Or should the computer remember all the sums generated, and be able to 'replay' them with answers?
2. Can you think of any more ways in which the program could be used?
3. This is a difficult topic - do you in fact find your pupils more competent with directed number operations after working with DIRECTED, and some time after?

Figure 1



Input a number)-10 and (+10 or R





FGP

Design: Richard Phillips, David Rooke and Alan Wigley
Program Richard Phillips
Source: ITMA/Shell Centre for Mathematical Education

Program copyright © CET, 1982

This program runs on a Research Machines 380Z in 32K RAM with disc BASIC versions 5.0A, 5.0G or 5.0H, or with cassette BASIC versions 5.1D or 5.1E. Read the note below before transferring from disc to cassette, or vice versa. The program requires high-resolution graphics.

Note on FGP

When the program is first run the message 'Please wait' appears on the screen. The program is searching for the correct location in memory for poking in the function strings. If it fails to find it, the message 'Unsuitable BASIC interpreter' will appear. If you get this message check that you are using BASIC version 5. If you are using this version, the message will also appear if the program has become corrupted. This is unlikely but if it does happen it is best corrected by reloading the program.

DC is a variable set in the first few lines of the program. With Version 5 disc BASIC, DC should be set to 200. But with Version 5 cassette BASIC, DC should be set to 199.

Summary of FGP

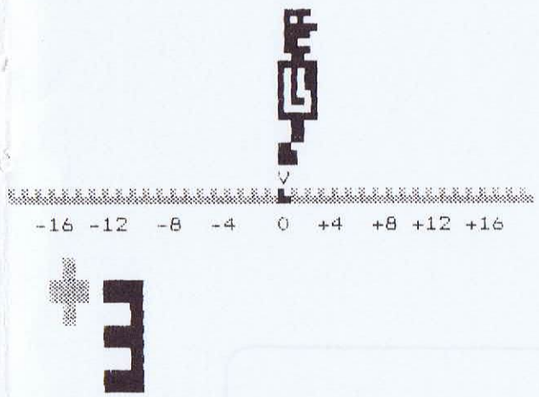
This is a general-purpose program to plot Cartesian graphs of most easily written functions. The user can choose the scales or opt for limited automatic scaling. Functions are entered in normal algebraic notation. Parameters can be changed, graphs can be superimposed and first derivatives can be plotted. Besides its use as an 'electronic blackboard', the program can be used to play 'guess the function' games and to fit curves to point data.

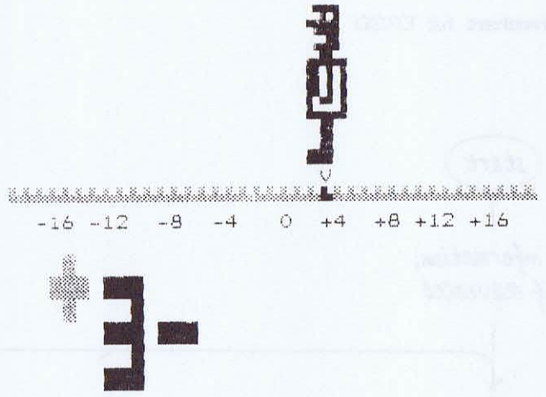
Commands in FGP

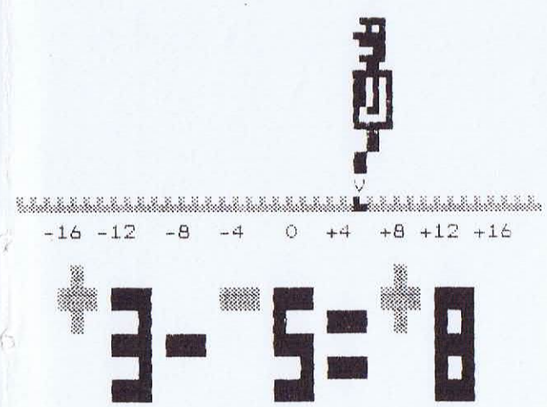
After », you may type:

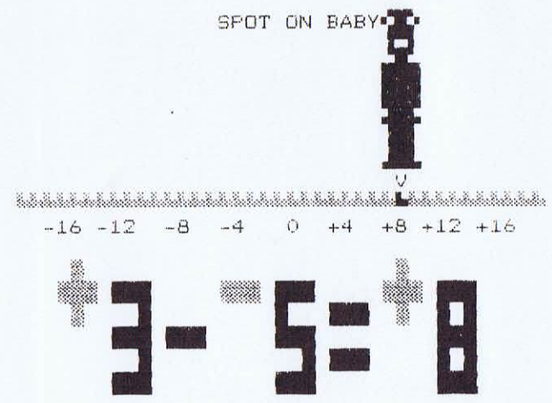
either a function, such as
y=(x-a)(x-b)
or an assignment, such as
a=3 (sets a to 3)
x=2 (program responds with value of y)
or one of these options:

- [A] Angles - degrees or radians
- [D] Double replot
- [E] End and return to BASIC
- [I] Increment and superimpose
- [L] Load and display data
- [O] Display option list
- [P] Plot graph with choice of scales
- [Q] Quick graph, automatic scaling
- [R] Replot
- [S] Superimpose graph
- [U] Plot graph with unchanged scale
- [V] Set dependent and independent variables
- [W] Write function on screen
- [X] Echo asterisks or echo normally
- [Z] Superimpose first derivative

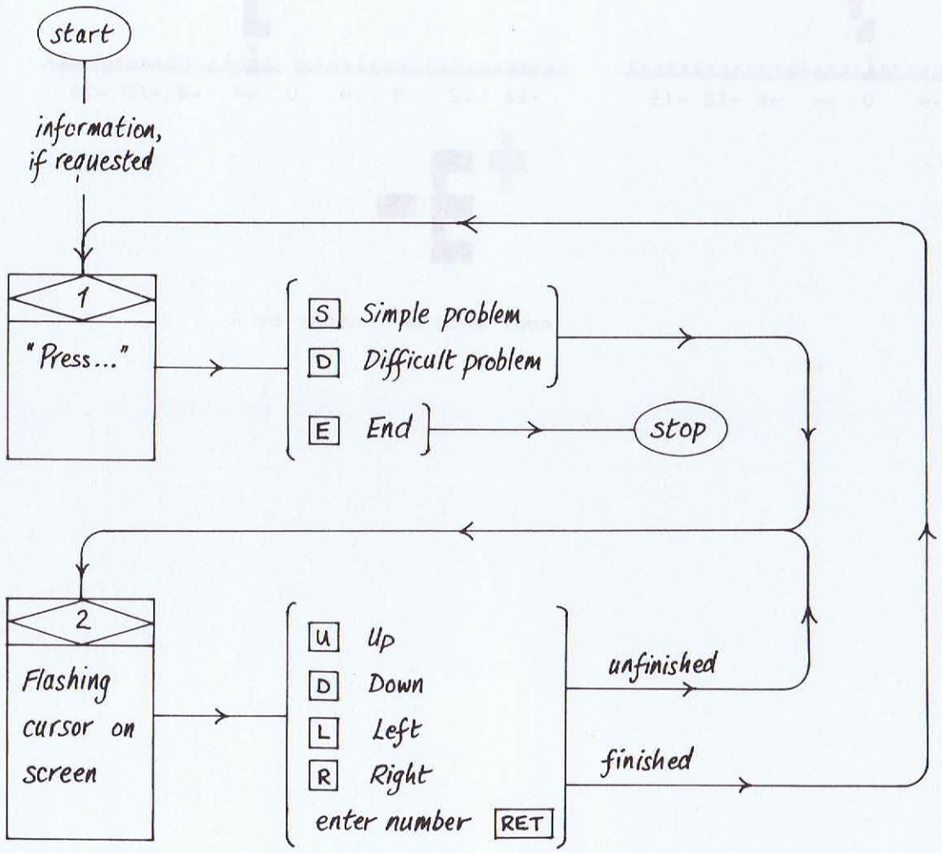








Drivechart for ERGO



For the next stage, the television or monitor is switched off and everyone is asked to think of a different sequence of events. There are many possibilities. For example, the man may get into the bath before the taps are turned on, or he may let the bath overflow. Everyone is asked to write down their sequence in words and then to sketch a graph on a separate piece of paper. When this is done, the children are asked to swap graphs and try to interpret what sequence of events is shown on their neighbour's graph. Finally, the computer screen is switched on again, and a number of children are invited to verify their graphs by typing the events themselves at the keyboard.

A common mistake in all graph work is for children to confuse graphs with pictures. Be sure to emphasize the differences between the graph and the picture above it, and look out for signs of confusion - for example, a child who when asked to sketch a graph insists on drawing taps on it!

Ideas and questions about EUREKA

1. Give the class a graph (such as Example 6) and ask them to write a short story about it.
2. What difference would it make if you plotted water volume rather than water level? Does the man ever affect water volume?
3. With a small remedial class, everyone could take a turn at experimenting with the program. Get each child to recall their sequence of events using the graph as a reminder.

Press [/] to restart the program and this time press the number (2), then the letter (G), then the [RET] key. The (2) selects pre-stored example number 2, and the (G) instructs EUREKA to display the graph only. Think about how this could serve as a useful basis for classroom discussion.

Press [E] to end the program. This returns you to the BASIC interpreter, but since EUREKA is still present in the computer's memory, you can run it again simply by typing [R] [U] [N] [RET].

EUREKA in the classroom

Before trying the program with a class, it is important for you to understand the program's limitations. For example, filling and emptying a real bath would not produce straight lines on the graph. It is especially important to realise that the gradients produced by filling and emptying the bath are unchanged when the man is put in, so you should avoid this comparison when discussing gradients with the class. If you wish to discuss gradients it is best to compare the bath emptying with the taps on and off. It is not recommended that you discuss the program's limitations with a class unless their understanding of graphs is especially good.

There are several ways of using EUREKA in a lesson. A typical lesson is described here together with some possible variants.

The program is introduced to the class by working through the normal sequence of events when someone has a bath: plug in, taps on, taps off, man in, man out, plug out. This is done with relatively little talk so that the class can concentrate on the visual images.

At the end of the sequence, the teacher presses (F) to freeze the action and discusses what is happening. This includes an analysis of the graph, with the teacher asking questions about each step.

After this introduction, the teacher goes on to some interpretation exercises using the examples built into the program. As Example 1 is very similar to the first example, the teacher starts with Example 2. Typing [/] (2) (G) and pressing [RET] displays the graph of Example 2 without the picture. The class are asked to explain what is happening. Explanations at various levels are encouraged. For example, 'the level of water has dropped suddenly', 'the man has got out', and 'he has had to get out to answer the telephone' are all different facets of interpreting the same event on the graph.

When the class has agreed about what is happening in the graph displayed by Example 2, the teacher types [/] (R) and presses [RET] to check this interpretation by watching the graph and picture together. When events reach a point which has caused some confusion, the teacher freezes the action to gain more time to talk. Some of the other examples are treated in the same way; it is strongly recommended that Example 6 should be used at some point in the lesson.

Design: Richard Phillips
Program: Richard Phillips
Source: ITMA/Shell Centre for Mathematical Education

Program copyright (C) CET, 1982

ERGO runs on a Research Machines 380Z in 32K RAM with BASIC Version 5. It may be used on a disc- or cassette-based system. It does not use high-resolution graphics.

Summary of ERGO

Twenty-five numbers, arranged in a five-by-five grid, form a pattern; only two of them are shown on the screen. The task is to discover the pattern and so fill in the missing numbers. The program is intended for the lower half of the secondary school. Each problem is chosen at random from more than 1000 patterns. If the computer judges that you have discovered a pattern, it will save you work and complete the problem for you. There are two levels of difficulty.

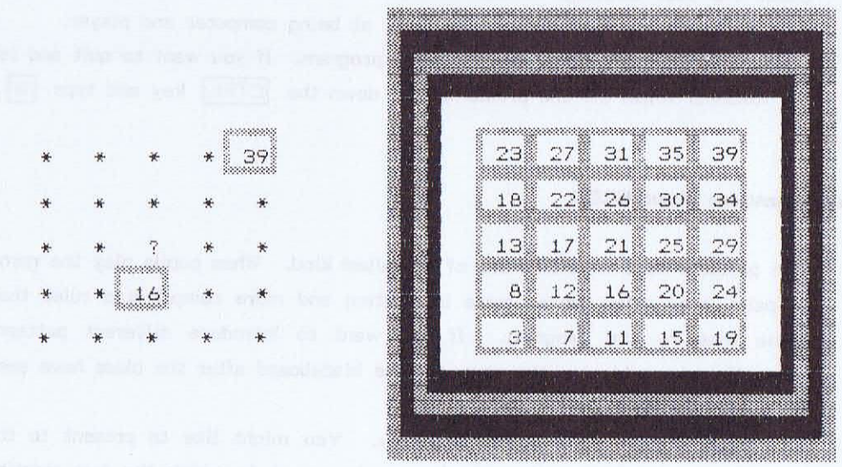


Figure 1

Running ERGO

These are the instructions displayed on the screen:

The computer will think of 25 numbers arranged in a five-by-five square. The numbers form some kind of pattern and your task is to discover what they are. All the numbers lie between 1 and 999 inclusive.

Your position in the square is shown by a blinking spot. You can move about by typing [U] for up, [D] for down, [L] for left, and [R] for right.

To guess a number, type it in and press the [RETURN] key. If you are wrong, either try again or move to another position.

To help you, two numbers are filled in at the start. You score points for getting numbers right first time.

Press [S] to start, [E] to end.

ERGO in the classroom

ERGO was designed for use by individuals and small groups, but it can be used with a whole class. Although you should familiarise yourself with the program, the best strategy with the class is to pretend you know no more about the program than they do. Work round the class getting pupils to guess numbers. Allow wild guesses at the start, but gradually get them to think about the best strategy to use. Switch the computer off and get them to play the game in pairs, each taking turns at being computer and player.

There is a useful secret option in the program. If you want to quit and let the machine finish off the problem, hold down the [CTRL] key and type [W].

Ideas and questions about ERGO

1. The patterns used in ERGO are of a limited kind. When pupils play the game in pairs, they often devise more interesting and more complicated rules than those used by the program. If you want to introduce different patterns yourself, you could play the game on the blackboard after the class have seen it on the computer.
2. The program uses positive numbers only. You might like to present to the class a grid which includes negative numbers, but in which the two starting numbers are positive.
3. Try teaching two lessons with different classes. In one, play ERGO on the computer; in the other, play it on the blackboard setting the questions yourself. Analyse the roles you play and the tasks you carry out in these contrasting lessons.

the man get in and out a dozen times. However, the man will not always sing. For this to work he must be in the bath with the taps off and the plug in. Once you start him singing, you will find that none of the other commands work until you silence him by pressing (S).

Press the [/] key (the same key as [?], but don't use [SHIFT]) to restart the program. Try working through a different sequence of events. Alter the sequence to your own taste, and to create changes in the graph which you think might stimulate discussion in a classroom. Notice how the graph faithfully follows the actions you perform. It is interesting to compare the gradients which are produced by the bath filling, by it emptying, and by it emptying with taps left on.

Press (F) to 'freeze' the action. Graph plotting will stop, and any animation in the upper part of the screen will be frozen. Press (F) a second time to unfreeze the action, and the program will continue as if nothing had happened. This key allows you to hold the display constant during classroom discussion.

Press [H] for help in using the program; it will remind you which keys you can press at the stage you have reached.

Here is a summary of the commands you have used so far:

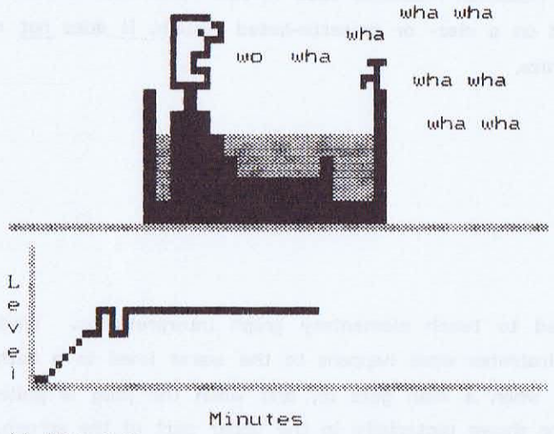
(P) Plug in or plug out
(T) Taps on or taps off
(M) Man in or man out
(S) Start singing or stop singing
(F) Freeze or unfreeze
[/] Restart the program
[H] Give help

Press [/] to restart the program again. The words 'Press RETURN to start' should be on the screen. At this stage there are a number of hidden options that you can select by pressing keys before pressing [RET]:

(1)-(6) Examples 1 to 6
(R) Replay events
*(K) Keyboard control of events
(P) Picture only
(G) Graph only
*(B) Both graph and picture

If you press none of these but press [RET] only, you will get the two options marked with an asterisk (*). These options are said to operate 'by default'.

Here are two examples to explain some of these options. First press the number [1] and then press [RET]. EUREKA will now display the first of six pre-stored examples. Press (F) to freeze or unfreeze this pre-stored sequence.




```
)) Man out.
)) Man in.
)) Sing.
))
```

Figure 1

Running EUREKA

Load and run EUREKA.

When you see the message 'Press RETURN to start', press the [RET] key and look at the screen. In the top part there is a picture of a bath; in the bottom part there is a graph. Watch the graph for a few moments and you will see that it is being drawn continuously. Every two seconds an extra point is plotted on the graph to show the current level of water in the bath.

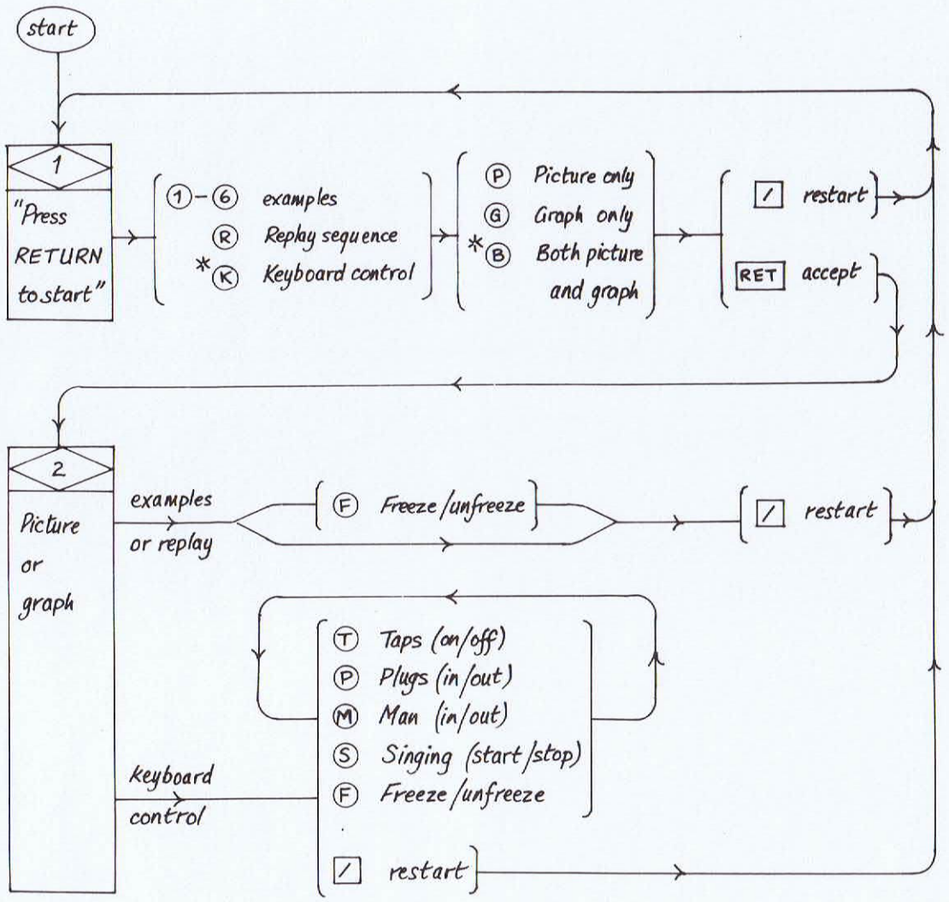
Press this sequence of keys, pausing for a few second after each one to see its effect. You do not need to press [RET] after any of these characters.

(P) (T) (Wait at least 15 seconds) (T) (M) (S) (S) (M) (P)

You should have discovered that (T) turns the taps on and off, (P) puts the plug in and pulls it out, (M) puts in the man and takes him out, and (S) starts and stops him singing.

Spend a few moments experimenting with these keys. Unlike a real bath, you can do no damage by letting the water overflow! The keys can be pressed any number of times and in any order, so you may, if you wish, make

Drivechart for EUREKA



Notes:

At any stage in the program,

- [H] *gives Help,*
- [/] *returns to Decision Point 1,*
- [E] *Ends the program and returns to BASIC.*

To start the man singing, he must be in the bath with the plug in and the taps off. Once he is singing, the only options which work are Ⓢ, Ⓕ, [E], [H] *and* [/].

EUREKA

Design: Hugh Burkhardt, Richard Phillips and Malcolm Swan
Program: Richard Phillips
Source: Shell Centre for Mathematical Education/ITMA

Program Copyright © Shell Centre for Mathematical Education, University of Nottingham, 1982

EUREKA runs on a Research Machines 380Z in 32K RAM with BASIC Version 5. It may be used on a disc- or cassette-based system. It does not require high-resolution graphics.

Summary of EUREKA

EUREKA is designed to teach elementary graph interpretation. Under the user's control, it illustrates what happens to the water level in a bath when the bath is filled, when a man gets in, and when the plug is pulled out. These operations are shown pictorially in the upper part of the screen, while in the lower part a graph is plotted of water level against time.

The program can be used to teach a number of aspects of graph interpretation and sketching. The idea of gradients can be discussed by considering the different gradients which result from filling the bath, from emptying it, or from emptying it with the taps left on. Step functions are illustrated by the man entering and leaving the bath.